Chair for Computer Science 10 (Media Computing and Human-Computer Interaction)

RWTH AACHEN UNIVERSITY

Chisv: User-Centered Design of a Conference Volunteer Management System

Master's Thesis
submitted to the
Media Computing Group
Prof. Dr. Jan Borchers
Computer Science Department
RWTH Aachen University

by
Florian Busch

Thesis advisor:
Prof. Dr. Jan Borchers

Second examiner:
Prof. Dr.-Ing. Ulrik Schroeder

Registration date: 05.02.2020
Submission date: 31.08.2020

Zentrales Prüfungsamt/Central Examination Office

RWTH AACHEN UNIVERSITY

Eidesstattliche Versicherung
Statutory Declaration in Lieu of an Oath

___________________________	___________________________
Name, Vorname/Last Name, First Name	Matrikelnummer (freiwillige Angabe) Matriculation No. (optional)

Ich versichere hiermit an Eides Statt, dass ich die vorliegende Arbeit/Bachelorarbeit/ Masterarbeit* mit dem Titel
I hereby declare in lieu of an oath that I have completed the present paper/Bachelor thesis/Master thesis* entitled

____

____

____

selbstständig und ohne unzulässige fremde Hilfe (insbes. akademisches Ghostwriting) erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.
independently and without illegitimate assistance from third parties (such as academic ghostwriters). I have used no other than the specified sources and aids. In case that the thesis is additionally submitted in an electronic format, I declare that the written and electronic versions are fully identical. The thesis has not been submitted to any examination body in this, or similar, form.

___________________________	___________________________
Ort, Datum/City, Date	Unterschrift/Signature

*Nichtzutreffendes bitte streichen

*Please delete as appropriate

Belehrung:
Official Notification:

§ 156 StGB: Falsche Versicherung an Eides Statt
Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.
Para. 156 StGB (German Criminal Code): False Statutory Declarations
Whoever before a public authority competent to administer statutory declarations falsely makes such a declaration or falsely testifies while referring to such a declaration shall be liable to imprisonment not exceeding three years or a fine.
§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt
(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.
(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.
Para. 161 StGB (German Criminal Code): False Statutory Declarations Due to Negligence
(1) If a person commits one of the offences listed in sections 154 through 156 negligently the penalty shall be imprisonment not exceeding one year or a fine.
(2) The offender shall be exempt from liability if he or she corrects their false testimony in time. The provisions of section 158 (2) and (3) shall apply accordingly.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:
I have read and understood the above official notification:

___________________________	___________________________
Ort, Datum/City, Date	Unterschrift/Signature

Contents

List of Figures

List of Tables

Abstract

At many conferences, various organizational tasks exist. For instance, rooms need to be set up for sessions, traffic might have to be coordinated, or attendance to be verified. These, and many more other tasks, are usually taken care of by (student) volunteers (SVs). For that to happen, conferences have so-called Student Volunteer Chairs (SV Chairs) that assign volunteers to certain tasks or shifts. As many large conferences can quickly have hundreds of tasks per day, SV Chairs usually rely on a system that keeps track of assignments of volunteers and also their completed hours.

One of these applications is called CHISV and has been used for more than 10 years. We present a new version of CHISV that we developed closely together with its users to not only include the existing features but also to improve in terms of usability and experience. Before we will take a look at how we constructed our application, we will evaluate how other conferences approach the management of SVs. While others plan the volunteer's assignment before the conference, we will see how CHISV's interactive task bidding approach is more flexible and also more transparent for the volunteer.

We show in detail how we built CHISV's back-end structures and why the design eases maintainability and expandability. For the front-end application, we focus more on the interaction with the user and the compatibility with modern standards. In our evaluation, we will see how CHISV is performing from different geographical regions and with different conference sizes. Lastly, we will understand why CHISV's publicly and well-documented API is opening up the entire system for new interaction concepts and form factors, and how we focused on building third-party integration into its core.

Überblick

Bei vielen Konferenzen gibts es verschiedene organisatorische Aufgaben. Zum Beispiel müssen Räume für Veranstaltungen vorbereitet, der Verkehrsfluss gesteuert oder die Anwesenheit überprüft werden. Diese und viele andere Aufgaben werden üblicherweise von EhrenamtlerInnen (SVs) erledigt. Damit dies geschieht, haben Konferenzen sogenannte Student Volunteer Chairs (SV Chairs), welche die Zuweisung von EhrenamtlerInnen zu Aufgaben oder Schichten erstellen. Da viele große Konferenzen oft schnell mehrere hunderte Aufgaben pro Tag haben können, nutzen SV Chairs üblicherweise ein System um die Zuweisungen und abgeschlossenen Aufgaben nachzuverfolgen.

Eine dieser Anwendungen heißt CHISV und ist seit mehr als 10 Jahren im Einsatz. Wir präsentieren eine neue Version von CHISV, die wir eng mit seinen Benutzern entwickelt haben, um nicht nur die vorhandenen Funktionen einzubeziehen, sondern auch die Benutzerfreundlichkeit und Erfahrung zu verbessern. Bevor wir uns ansehen, wie wir unsere Anwendung entwickelt haben, schauen wir, wie andere Konferenzen das Management von SVs angehen. Während andere die Aufgaben der EhrenamtlerInnen vor der Konferenz planen, werden wir sehen, wie CHISV's Ansatz zum interaktiven Bieten auf Aufgaben flexibler und auch für die EhrenamtlerInnen transparenter ist.

Wir zeigen detailliert, wie wir die Backend-Strukturen von CHISV erstellt haben und warum mit diesem Design die Wartbarkeit und Erweiterbarkeit erleichtert wird. Bei der Frontend-Anwendung konzentrieren wir uns mehr auf die Interaktion mit dem Benutzer und die Kompatibilität mit modernen Standards. In unserer abschließenden Auswertung werden wir sehen, wie responsiv CHISV ist, wenn wir die Konferenzgrößen und die geografischen Regionen, von denen aus zugegriffen wird, variieren. Zuletzt werden wir zeigen, warum die öffentliche und gut dokumentierte API von CHISV das gesamte System für neue Interaktionskonzepte und Formfaktoren öffnet und wie wir uns darauf konzentriert haben, die Integration mit Drittanbietern in dem Kern der Anwendung zu verankern.

Acknowledgements

At first, I would like to thank Prof. Dr. Borchers and also Prof. Dr.-Ing. Ulrik Schroeder for examining this thesis.

I also want to thank my supervisor Christian Cherek, who was always available and provided me with constructive feedback during my work, which helped me improve this thesis. Also, without him pushing, this text would not exist yet. Thank you!

Additionally, I would like to thank my family and the Student Volunteer Chairs of CHI 2019/2020, MobileCHI 2020, and UIST 2020 for their feedback and ideas. Furthermore, I am very thankful for every single volunteer of the SV community who helped me to understand the needs and duties which come along with volunteering.

Especially, I would like to thank my girlfriend, Jennifer, for your advice at any time of the day and your strong support whenever I was not sure how to continue.

Conventions

Throughout this thesis we use the following conventions.

Although all the work in this thesis has been done by myself, I will use the first-person plural pronoun when referring to things that have been done.

The terms "student volunteer", "volunteer", "student", "SV", or "user" will be used synonymously referring to a person interacting with the system.

We will use the terms "system", "application", and "CHISV" synonymously when it is obvious to where the term refers to.

We will often refer to a "user's tasks" or a "user's assignments". Both express a user's association with a task. We will use these terms synonymously.

When we link to web pages of CHISV, we usually link to the domain where it will reside later on ("chisv.org"). At the time of this thesis, the actual application is still running under the subdomain of "new.chisv.org". Thus, some references might require adaptation.

Text conventions

Definitions of technical terms or short excursus are set off in colored boxes.

Definition: *Excursus*

Excursus:
Excursus are detailed discussions of a particular point in a book, usually in an appendix, or digressions in a written text.

Source code and implementation symbols are written in typewriter-style text.

`myClass`

References to a specific requirement from the requirements chapter will be prepended with the pound sign (e.g. #1, #54).

The whole thesis is written in American English.

Chapter 1

Introduction

Many conferences in the sphere of Human-Computer Interaction (HCI) usually consist of multiple events, talks, sessions, or workshops. Several of these events need some preparation or assistance. Thus, for a conference, there are also always organizational tasks accompanying the conference program. These tasks are usually undertaken by so-called (student) volunteers (SVs). They not only help with preparing events of the conference's schedule but also with supporting various other activities, like managing car traffic, checking attendance, or helping out attendees.

Organizational tasks around the actual conference events

Large conferences, for instance, the Conference on Human Factors in Computing Systems (CHI), tend to have hundreds of tasks per day and more than one hundred volunteers on-site. At that scale, it becomes very hard, if not impossible, for the Student Volunteer Chairs to manually schedule and manage task-volunteer associations. Thus, they usually rely on a system that keeps track of all important factors.

Large conferences usually also have many organizational tasks

Conferences like CHI, UIST[1], and many others use a system called CHISV. It's actively used by many conferences every year to help manage volunteers, their enrollment, and task assignments. Over the last ten years that the

Task and volunteer management can be done with CHISV

[1]Conference for User Interface Software and Technology

system ran, many requirements changed, new technologies appeared, and old technologies vanished what made keeping the application available and performant harder every day.

Redesign CHISV to adapt new requirements

We saw that the time had come to adapt CHISV to the new requirements, new and more modern technologies, and to overall include its users into every step of the design and implementation process. We will start in chapter 2 “Related Work” by getting a broad overview of various solutions that are used by other conferences in the same field but also take a look at how conferences of other disciplines organize volunteers.

Designing for the users

Before we were able to start our design process, we collected ideas and requirements from a large pool of users. During chapter (3 “Requirements Analysis”) we will see how our studies, interviews, and surveys guided us in creating a great experience and functional tool for the SV community.

Split in two entities

In chapter 4 “CHISV”, we will then dive into CHISV’s design and implementation and understand how all structures, extensions, and components interface with each other to form two separate entities: A RESTful back end and a modern and responsive front-end application.

Evaluate new version of CHISV

Our evaluation in chapter 5 “Evaluation” will present the results from our performance and scalability tests and conclude with the users’ experience, which we acquired through surveys. We will also shortly compare impressions of the previous version of CHISV and ours.

Extendability and third-party integration

Finally, after summarizing our findings in 6 “Summary and Contributions”, we will give an outlook into the possibilities the community now has thanks to the open design and implementation (chapter 7 “Future Work”). We will show how new interactions can be created by extending CHISV through its API and how a simple core extension could bring realtime calendar support.

Chapter 2

Related Work

In this section, we will take a look at other student volunteer management systems other conferences use. While we were collecting insight into other applications, which provide similar features as CHISV, we noticed that many conferences do not use any volunteer management system at all.

Many different ways of volunteer management exist

Some of them utilize tools like Google Forms to collect the required information online. They use a fixed set of questions that every volunteer has to submit. We found this technique at IMS, SC, ICSE, ACL, ICSA, AEA, ICFP, CITT, POPL, HRI.
Others published PDF or plain text files for their volunteers to download, fill, and send back via e-mail (e.g. CVPR, SSWR, PASC, ECAI).
We also saw some conferences using the conference management system where participants submit their papers for volunteer registration (e.g AAMFT, HCII).

Google Forms is often used

These different approaches are mostly taken due to the expected number of volunteer applications. Another factor is the additional work and the financial aspects of building an application for the sole purpose of managing volunteer applications.

Management of tasks and assignment might not always be required

In the following, we will take a look at volunteer management systems that have been custom-built to register and manage SVs as well as tasks. The Special Interest Group on Computer Science Education (SIGCSE) and the Special Interest Group on Computer Graphics (SIGGRAPH) are using web applications for their student volunteer and task management. We also compared our system to its predecessor, which is hosted at the same facility. To understand these systems and their abilities in detail, we got in touch with the people who have built the system or are currently keeping it operating.

Name	Built for	Multi Conf.	Permission system	Schedule creation	Internal messages	Cal. export
Previous version of CHISV	Manage volunteers, task bidding and lookup of schedule	Yes	Super Admin Conf. Admin Moderator SV	Reactive, progressive	No	No
SIGCSE	Manage volunteers and lookup schedule	No	Coordinator SV	Prior to the conference	No	No
SIGGRAPH	Manage volunteers and lookup schedule	No	Team Lead SV	Prior to the conference	No	No
CHISV	Manage volunteers, task bidding and lookup or export of schedule	Yes	Admin SV Chair Day Captain SV	Reactive, progressive	Yes	Yes

Table 2.1: Comparison of four systems for student volunteer management in regard to functions and their context of use. Two applications are build for multi-conference use (Multi Conf.), all use a permission management. Only CHISV allows for internal messaging and calendar export (Cal. export).

This gave us a solid overview of the features and needs these systems have been built against. We compared them with respect to the features we find most important in CHISV but also to the rather technical aspect of hosting the application. We found that all systems we looked into are web-based applications hosted on a publicly available server. However, we also discovered a clear division into two groups of systems. While the systems of SIGCSE and SIGGRAPH implement a procedure where the SV is mostly using the application to look up a schedule that has been generated weeks before the conference, we see a different approach with the reimplementation of CHISV and also it's predecessor. For these two the schedule is generated based upon bids submitted to the system. This allows the SV to express a level of preference. The schedule is then generated fresh for every day rather than being static (except for minor changes), as we see it with SIGCSE and SIGGRAPH (see column "Schedule creation" in table 2.1).

Name	Hosting	CDN	Software stack
Previous version of CHISV	Computer science institute at RWTH Aachen University	No	Ruby (Rails)+MySQL and PrototypeJS
SIGCSE	Computer science Department at University of British Columbia	No	PHP+MySQL and Bootstrap+JS
SIGGRAPH	Student Volunteer Sub Committee via Amazon Web Services	Yes	Node.js+PostgreSQL and Vue.js
CHISV	Computer science institute at RWTH Aachen University	No	Laravel+MySQL and Vue.js

Table 2.2: Comparison of four systems for student volunteer management in regard to hosting and framework related aspects. All but one application are hosted at facilities of universities. Each approach has been built with a different framework.

Apart from this major difference, we found all systems solve the similar issue of scheduling shifts or assignment of student volunteers such that the platform provides the schedule afterward accessible via a web browser. All systems keep track of the hours the volunteer has worked. This allows the SV Chairs/Coordinators to later refund the conference fee for each volunteer that did complete the expected amount of hours.
We will now explore the different characteristics of each system one at a time.

2.1 Previous Version of CHISV

Hosting and web frameworks

The predecessor (see SV-Portal) of our reimplementation is hosted by the Media Computing Group at RWTH Aachen University (see table 2.2). Its development started in 2008 by Jonathan Diehl[1] and Christopher Gretzki[2]. The application is written in Ruby and uses the web framework Rails. This enables the application to implement the Model–View–Controller paradigm. The system interfaces with a MySQL database server and incorporates the JavaScript framework Prototype to provide interactivity related aspects of the web interface.

SV Interface	Management Interface
SV Role	Super Administrator Conference Administrator Moderator

Table 2.3: Roles in the previous version of CHISV

Two different web interfaces for different use cases

The web interface for volunteers is separated from the administrative interface, which can be used to create conferences and tasks. We will call the interface used by volunteers "SV Interface" and the interface used by conference administrators "Management Interface". While the system supports four different roles (see roles table 2.3) with different access abilities, we observed that only the SV and Conference Administrator roles are actively used. A Super Administrator usually creates a conference and assigns a newly created Conference Administrator for that conference.

Shared login credentials for SV Chair and Day Captain

The login of the Conference Administrator is then passed to the SV Chair, which enables them to log in to the management interface. Any volunteer who is accessing and enrolling for a conference via the SV interface holds the SV role. We noticed that the job of managing assignments of volunteers is usually done by a so-called Day Captain. They check the volunteers' badge and mark

[1]https://hci.rwth-aachen.de/diehl
[2]https://hci.rwth-aachen.de/gretzki

them as "checked-in". When the volunteer has completed the task the associated assignment will be marked as "done". For the task of managing the assignments the Day Captains usually receive the login of the Conference Administrator to log in to the management interface.

We will focus on the SV interface of CHISV first. This is the interface most users will encounter and use. After that we will take a look at the management interface, which is used to manage the system, its conferences, users, and tasks.

2.1.1 SV Interface

Accounts for the SV interface are stored separately

All volunteers have to register themselves on the SV interface. Once logged in to this account a volunteer can enroll in conferences that are open for enrollment. There is no specific role for the SV interface. However, as the volunteer's account is represented by the system in a different datastore as the administrative accounts, we find it easier to speak of the SV role for any user that can log into this interface. Logically we have a separation between all four roles the system provides (see table 2.3), albeit the accounts that can log into the SV interface have no definitive role assigned. The SV interface displays and explains the current state of the conference. This gives the volunteers a chance to understand in which phase the conference is in and if it is possible to enroll or bid on tasks. A conference can be in one of six states:

- **Planning** – The conference is not displayed in the SV interface but can be edited in the management interface
- **Enrollment** – Volunteers can enroll in the conference
- **Registration** – Volunteers have been accepted and will now have to register for the conference themselves

- **Bidding** – The volunteers can now bid on tasks with their preference
- **Running** – The conference is running and bids can be placed
- **Over** – The conference is over and not visible in the SV interface

After successful authentication the volunteers can navigate between these different pages (views) of the SV interface:

- News
- Profile
- Enrollment Status
- Bidding
- Assignments

We will now take a look at each of these pages and see what they are used for.

News

Get a short introduction to volunteering

When a volunteer navigates to the News view, information from the Conference Administrators will be shown. This can contain additional information for enrollment or may also give a short introduction to the conference.

Profile

Update profile information

The Profile view gives the volunteers the ability to change the essential attributes of their account. This includes the full name, password, e-mail address, their home country, university, and their desired T-shirt size and fit. In addition to these, the SV can also give some insight into optional fields like spoken languages or city of residence.

Enrollment Status

To enroll in a conference, the volunteer would navigate to the Enrollment Status page and submit answers to a fixed set of questions. In case the volunteer is already enrolled this view will show the current status of the enrollment for the volunteer. An enrollment can be in one of seven states:

Enroll or see the status of a pending enrollment

- **Unenrolled** – The volunteer is not enrolled
- **Enrolled** – The volunteer is enrolled and waiting to be accepted
- **Waitlisted** – The volunteer has a position on the waitlist and is waiting to be accepted
- **Accepted** – The volunteer is accepted to the conferences for volunteering
- **Registered** – The volunteer did successfully register as an attendee for the conference
- **On-site** – The volunteer is on-site and awaiting assignments
- **Dropped** – The volunteer is rejected and cannot bid on tasks

When the conference is in the state "Registration" or "Bidding" the volunteer will have the possibility to display the waitlist. This shows all student volunteers who could not yet be accepted due to the limitation on the amount of SVs.

Bidding

Limited number of bids for "high" and "medium"

On the bidding page, the volunteer can express a preference for a task by placing a bid with the desired preference. The preference can be "high", "medium" or "low" which are internally represented by the numbers 1, 2, and 3. How many bids of one type can be placed is limited. A volunteer is able to submit up to three "high" bids per day and up to ten per conference. Up to ten bids with "medium" preference per day are possible. A "low" preference bid can be placed arbitrarily often.

Bidding on tasks is restricted by the SV's and conference's state

To be able to bid on tasks, the volunteer and the conference have to be in a specific state. The SV has to be enrolled with the state of "Registered", while the conference the task is associated with will have to be in the state "Bidding" or "Running" (see 2.1.1 "SV Interface"). Tasks will be shown with their start and end time as well as their name, description, hours, and slots. For task bidding, the SV would first have to pick the desired day. This will show all tasks of the day sorted by their start time.

State of a bid

A placed bid is in an initial state called "Open". The auction (covered in 2.1.2 "Auction"), which is later run by the SV Chair, will evaluate all bids and assign them a new state such that the volunteer can see why or why not a bid was successful. Whenever a bid was successful it will yield an assignment. When a task has already been filled with SVs or the desired task is conflicting with any of the volunteer's assignments, the bid will not be successful.

Assignments

See all associated assignments with their corresponding state

When the conference is in the state "Running", the assignment page will show all assignments of the volunteer sorted by day. This view will feature the same columns as the bidding page but will, in addition, also show the state of each assignment and the SV's total accounted hours.

An assignment can be in one of three states:

- **Assigned** – The volunteer has not started working on the assignment
- **Checked-in** – The volunteer is currently working on the assignment
- **Done** – The volunteer completed the assignment

2.1.2 Management Interface

Role Management

Dedicated portal for management purposes with three roles

The management interface is used by the Super Administrator, Conference Administrators, and Moderators. These three roles (see roles table 2.3) are exclusive to each other and give the users different permissions on resources. The abilities the Super Administrator has are total, being able to modify every aspect of the system. This role is held by the institution that is hosting the application. The Super Administrator will create a new conference and a corresponding Conference Administrator on request.

Conference Administrators can change details about their conference as well as manage tasks, volunteers, and their enrollment. This role is usually held by the SV Chairs. They can also send e-mails to the volunteers, manually assign tasks to them and run the lottery (see 2.1.2 "Lottery") as well as the auction (see 2.1.2 "Auction").

We noticed that the Moderator role was not actively used during any of our interviews. Furthermore, the Super Administrators and Conference Administrators were not even aware of this role, which would give the Day Captains just enough permission to manage assignments. Day Captains were always relying on credentials from the Conference Administrators.

Conference Details

Maintenance mode

The Super and Conference Administrators can change the parameters of the conference. These include the conference's name and year as well as a start and end date, the number of SV slots the conference has to be filled with and the state it is in. This is also the place where the SV interface can be temporarily disabled by switching the conference into maintenance mode. This prevents SVs from bidding but also takes their ability to check on their assignments.

Enrollments

Changing the enrollment state of SVs

The Enrollments page is used by SV Chairs to check or manually change the state of a volunteer's enrollment. This is often used to accept volunteers before running the lottery. All enrolled volunteers are listed with the corresponding attributes (see 2.1.1 “Profile”) and their lottery number when set. This view also allows the authorized administrators to see the enrollment history, remove the enrollment altogether or block the user from logging in. This place can also be used to create new accounts for volunteers and enroll them in the conference.

Lottery

Algorithm automatically accepts SVs

The lottery view is used to run the lottery on the server. A user with an administrative role (e.g. SV Chair) can start the lottery. The lottery's algorithm assigns a new lottery position to every user with an enrollment that is in the state "enrolled". To give the SV Chairs the ability to automatically accept more SVs who have specific criteria, the system uses a scoring system that is based on so-called tickets. Each question of the enrollment form can yield a positive or negative ticket.

These questions are:

- Have you lived in the conference city, or are you very familiar with it?
- Have you attended this conference before?
- Have you been an SV at this conference before?
- Do you need to apply for a travel visa in order to attend this conference?

Moreover, the SV Chair can also configure if the degree program, the SV is currently on, should yield a ticket.

Positively evaluated SVs get processed first

Each question requires a binary answer. The SV Chair can decide if a positively answered question should yield a positive or negative ticket – or none at all. This can be configured in the lottery configuration. A ticket is a number that ranges from -3 to 3. The score, the so-called lottery score, is then calculated by summing up all tickets. All enrollments with a positive number (excluding zero) will be processed first. How likely it is that an enrollment with a positive score is processed early on, also depends on the lottery score. The higher the score, the more likely it is that the enrollment is processed earlier by the algorithm.

Whenever a volunteer's enrollment gets processed the algorithm marks it as "Accepted" if there are free slots available for the conference (see states in 2.1.1 "Enrollment Status"). Should there be no more free slots available, since all have been filled, the enrollment, and thus the volunteer, is put on the waitlist.

Negatively evaluated SVs might not be considered in the lottery

When all volunteers with a positive lottery score have been processed, volunteers with a negative score (or equal to zero) will get processed. When there are more enrollments with a positive score than there are SV slots available, this means that volunteers with a negative score cannot get a seat. In that case, all negatively scored SVs will be put on the waitlist.

Task Management

Task can be created, modified, and transferred to different systems and conferences

Tasks are jobs that need to be done during the conference. Tasks can be assigned to student volunteers manually or by running the auction. Within the system all existing tasks can be edited, deleted, or exported via CSV. New tasks are creatable from scratch or based on another task. The system also includes a CSV import to enable the SV Chairs to import tasks from different external applications (e.g. Microsoft Excel or Apple Pages). It is also possible to clone all tasks of entire conference days or even conferences. Tasks are shown on a per conference day basis with their attributes:

- Name
- Description
- Priority: "High", "normal" or "low" (0, 1, or 2)
- Conference day
- Start time
- End time
- Awarded hours: This can differ from the hours resulting from start and end time
- Number of volunteers required

Auction

Algorithm assigns tasks to volunteers

When all SVs have bid on the tasks of a conference day, which is usually a day before that day, the conference administrators run the auction. The auction tries to create all assignments such that all tasks are filled with SVs. The algorithm (see figure 2.1) takes the task's priority and also the volunteer's preference into account. How many hours a volunteer has already worked will not influence the results. We noticed that this design can cause many assignments and work for some SVs who have already completed many hours but are nevertheless still

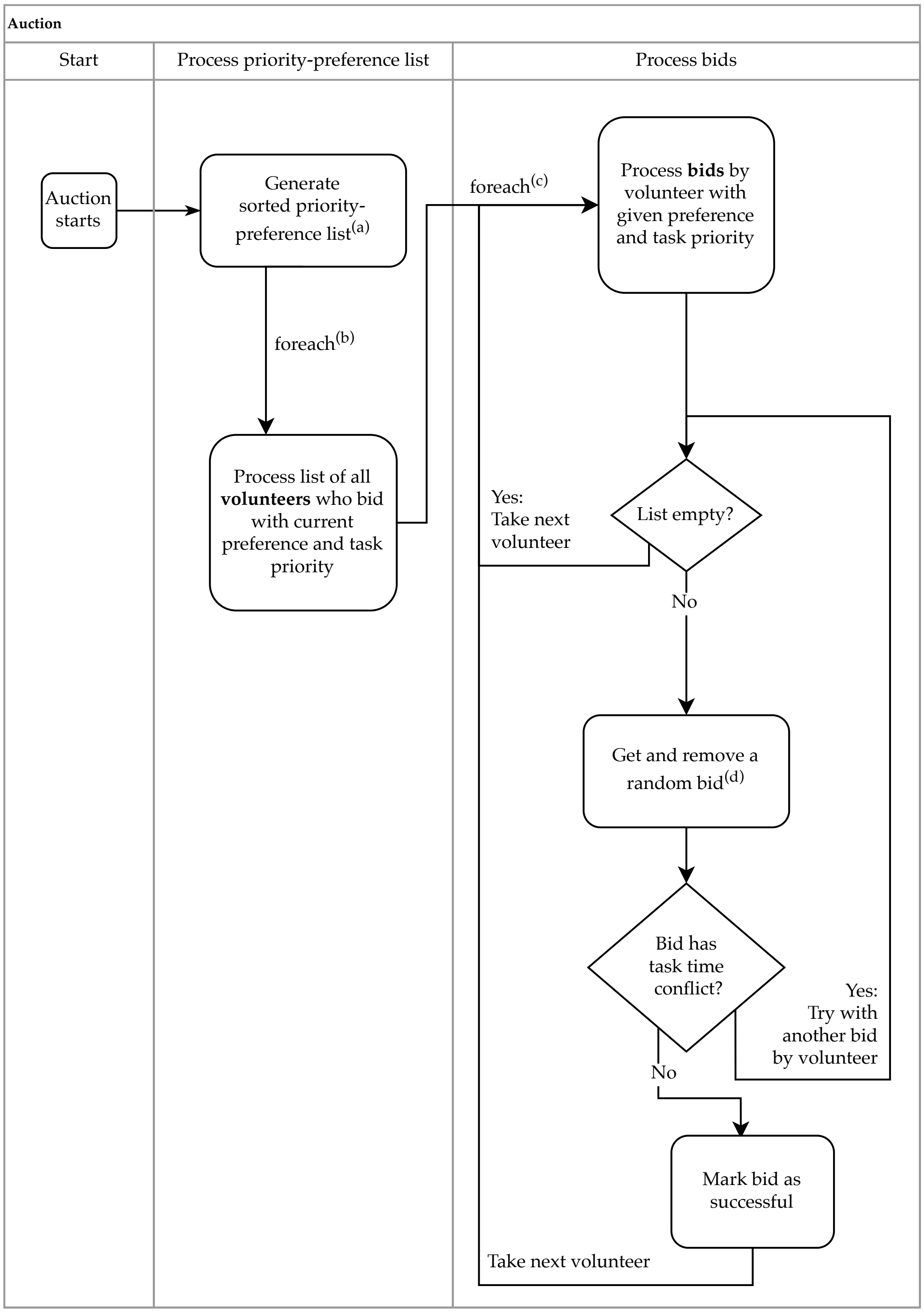

Auction
Start
Process priority-preference list
Process bids
Auction starts
Generate sorted priority-preference list(a)
foreach(b)
Process list of all volunteers who bid with current preference and task priority
foreach(c)
Process bids by volunteer with given preference and task priority
List empty?
Yes: Take next volunteer
No
Get and remove a random bid(d)
Bid has task time conflict?
Yes: Try with another bid by volunteer
No
Mark bid as successful
Take next volunteer

Figure 2.1: Algorithm running the auction to evaluate bids and assign tasks to volunteers. See page 16 for an in-depth example of (a), (b), (c), and (d).

placing bids with preference "high". Each bid, which could be accepted, will yield an assignment, therefore binding the task to the SV.

Deeply nested data objects

The auction algorithm loops over cascaded dictionaries to eventually get to an array that contains the volunteer's bids. To get a better overview of the data that is passed from one loop to another, we will now cover a sample data object in detail. In reality, this object would be by far larger, containing more volunteers and bids. However, how many bids are in the actual object depends on how many and how often SVs have bid. Only bids that have been placed will end up in the data object.

Our example data object has very verbose names for its keys. The actual implementation uses a more efficient approach. We now pretend that the step (a) in figure 2.1 has produced the following object:

```
1  {
2      task-priority0_bid-preference1: {
3          volunteer5: [bid1,bid2,bid3,bid4],
4          volunteer3: [bid5,bid6]
5      },
6      task-priority0_bid-preference2: {
7          volunteer8: [bid7],
8          volunteer5: [bid8,bid9]
9      }
10     task-priority1_bid-preference3: {
11         volunteer5: [bid10, bid11]
12     }
13 }
```

Groups of bids with same preference

This object contains all bids with a specific preference X for a task with a specific priority Y under one key, which represents the priority and preference: "task-priorityY_bid-preferenceX" (see line 2,6 or 10). The keys are sorted such that tasks and bids with higher priority or preference are iterated over first. Task priorities can be 0 ("high"), 1 ("normal"), or 2 ("low") (see attribute list at 2.1.2 “Task Management”). The bid preference can take the values 1 ("high"), 2 ("medium"), or 3 ("low") (see

2.1.1 “Bidding”).

In the next step we will be looping through all these keys. In every iteration we will be working with an object (b) of this structure:

```
task-priority0_bid-preference1: {
    volunteer5: [bid1,bid2,bid3,bid4],
    volunteer3: [bid5,bid6]
}
```

In this particular example, any key of this object represents the bids of a volunteer with preference 1 ("high") which have been placed on some task with priority 0 ("high"). We will now loop over all the keys, each of which representing a volunteer. The corresponding value (c) of each key is a list of bids. Each bid has the same preference but has been placed on different tasks. All these tasks however have the same priority of 0.

```
volunteer5: [bid1,bid2,bid3,bid4],
```

Should the key (in this case "volunteer5"), which is currently processed by the loop, not have any bids in the list we simply skip to the next key (in this case "volunteer3"). In our example "volunteer5" has bids in the list.

In the next step, we randomly take and remove one bid (d) from the list.

```
bid3
```

Check task time conflict

We then check if this bid is in a task time conflict with another of the volunteer’s already assigned tasks. We don’t want to assign tasks to volunteers that take place at the same time. If there should be a time conflict we will try to get another random bid from the volunteer’s bid list. If the list is empty we will continue with the next volunteer in the parent list ("volunteer3"). However, when we encounter no time conflict we assign the task, to which the bid points to, to the volunteer. After a successful assignment, we continue with the next volunteer in the parent list (in this case "volunteer3").

Iterates over the data object multiple times

The entire auction algorithm is bound in a while-loop, which only ends when no assignments could be made anymore. This is important since the algorithm is designed in such a way that it will only create one assignment for a randomly chosen bid (d) in every loop for each volunteer (c). This design, however, tries to fill tasks with a high priority with high preference bids first. The algorithm does not take the hours into account that have already been awarded to the SV. We also noticed that tasks for which no volunteer had bid will not be assigned to any volunteer. These would have to be filled manually after the auction has been run.

Can leave tasks unassigned

Assignments

Tracking assignments and their completion

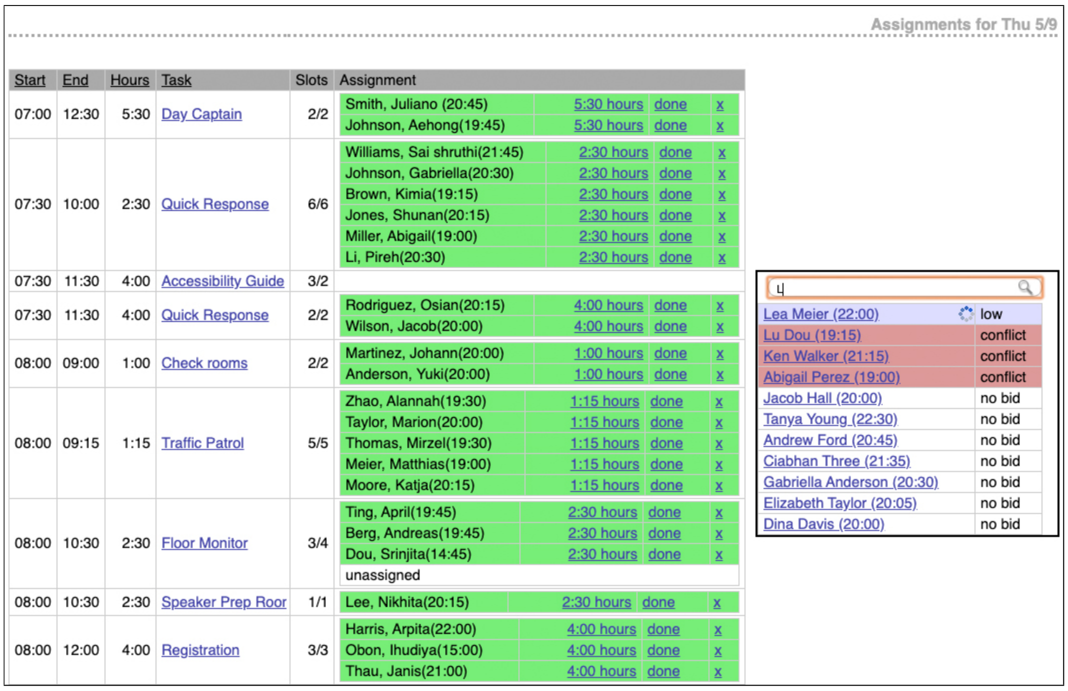
During the conference, the SV Chairs and Day Captains track the status of the volunteers' assignments. Figure 2.2 shows the assignment view for all the assignments of a selected day and their assigned volunteers. The interface allows for adding new volunteers to a task and for removing SVs from an assignment. How many SVs can be manually assigned to a task is not bound to the slots the task has. The system also allows the SV Chair and Day Captain to modify the awarded hours a volunteer gets for completing a task.

Changing the status of an assignment

The status of an assignment can be cycled between the three states of "Assigned", "Checked-in" and "Done" (see 2.1.1 “Assignments”). It is also possible to add a comment to an assignment. This is often used to give a short explanation when the awarded hours have to be adjusted. If the SV went over the suggested hours, a warning will be shown next to the name of the volunteer.

Manually adding SVs

To add a new volunteer to a task manually, one can click on the task's name to open a popup (see figure 2.2). This will show all SVs and mark those who are unavailable due to a task time conflict, red. The list will present the first and last name, the bid preference and also the completed hours. By clicking the name the volunteer will be assigned to the selected task.



Assignments for Thu 5/9

Start	End	Hours	Task	Slots	Assignment			
07:00	12:30	5:30	Day Captain	2/2	Smith, Juliano (20:45)	5:30 hours	done	x
					Johnson, Aehong(19:45)	5:30 hours	done	x
07:30	10:00	2:30	Quick Response	6/6	Williams, Sai shruthi(21:45)	2:30 hours	done	x
					Johnson, Gabriella(20:30)	2:30 hours	done	x
					Brown, Kimia(19:15)	2:30 hours	done	x
					Jones, Shunan(20:15)	2:30 hours	done	x
					Miller, Abigail(19:00)	2:30 hours	done	x
					Li, Pireh(20:30)	2:30 hours	done	x
07:30	11:30	4:00	Accessibility Guide	3/2				
07:30	11:30	4:00	Quick Response	2/2	Rodriguez, Osian(20:15)	4:00 hours	done	x
					Wilson, Jacob(20:00)	4:00 hours	done	x
08:00	09:00	1:00	Check rooms	2/2	Martinez, Johann(20:00)	1:00 hours	done	x
					Anderson, Yuki(20:00)	1:00 hours	done	x
08:00	09:15	1:15	Traffic Patrol	5/5	Zhao, Alannah(19:30)	1:15 hours	done	x
					Taylor, Marion(20:00)	1:15 hours	done	x
					Thomas, Mirzel(19:30)	1:15 hours	done	x
					Meier, Matthias(19:00)	1:15 hours	done	x
					Moore, Katja(20:15)	1:15 hours	done	x
08:00	10:30	2:30	Floor Monitor	3/4	Ting, April(19:45)	2:30 hours	done	x
					Berg, Andreas(19:45)	2:30 hours	done	x
					Dou, Srinjita(14:45)	2:30 hours	done	x
					unassigned			
08:00	10:30	2:30	Speaker Prep Roor	1/1	Lee, Nikhita(20:15)	2:30 hours	done	x
08:00	12:00	4:00	Registration	3/3	Harris, Arpita(22:00)	4:00 hours	done	x
					Obon, Ihudiya(15:00)	4:00 hours	done	x
					Thau, Janis(21:00)	4:00 hours	done	x

ч

Lea Meier (22:00)	low
Lu Dou (19:15)	conflict
Ken Walker (21:15)	conflict
Abigail Perez (19:00)	conflict
Jacob Hall (20:00)	no bid
Tanya Young (22:30)	no bid
Andrew Ford (20:45)	no bid
Ciabhan Three (21:35)	no bid
Gabriella Anderson (20:30)	no bid
Elizabeth Taylor (20:05)	no bid
Dina Davis (20:00)	no bid

Figure 2.2: In the management view of the application SV Chairs and Day Captains can alter the status of assignments that have been created manually or by the auction. Authorized users can also change the awarded hours and add a comment to assignments. This view is also used to assign new volunteers to tasks.

Messaging

Keep volunteers informed about changes

To keep the students up to date with changes in the system, the application will generate e-mails for changes in the enrollment status, the volunteer's profile or any associated assignment. These messages are generated automatically but only sent when an administrator (e.g SV Chair) decides to send them to the SVs. Each e-mail contains information about the type of change, the changed data and also some additional information, which can be customized in the conference details.

2.2 SIGCSE Volunteer Registration Portal

Another system, which was specifically build to help with organizing SV tasks, is the application that the Special Interest Group on Computer Science Education (SIGCSE) uses. We had the chance to get some insight into the application stack and learn about the purpose it was built for. This insight was provided to us by the Student Volunteer Co-Coordinator and the person who oversaw the initial development.

Exclusively used for managing volunteers at SIGCSE

The SIGCSE Volunteer Registration Portal is a web application hosted at the University of British Columbia (BCU) (see table 2.1). It is reset every year to a clean state during which all tasks and volunteers get wiped. Since this system is exclusively used by SIGCSE, there is a clear schedule to when the system is in use and when not. This means that at any point in time there will only be one conference. From a technical perspective, there have been approaches to use the application with multiple conferences simultaneously. Due to the introduced overhead and the resulting need for more maintenance, these approaches were not evaluated further. Giving multiple conferences access to the system simultaneously would also have required adapting the user interface.

2.2.1 Area of Application

The volunteers' schedules are created beforehand – no task bidding

We learned that the SIGCSE Volunteer Registration Portal was built for a different purpose. While the previous version of CHISV we looked at before focuses on task bidding with preferences on-site during the conference, the SIGCSE application was built to generate a schedule before it. The volunteers can submit their availability beforehand. There is no option to give any preference for tasks. Furthermore, a volunteer does not even know how many and what kind of tasks exist.

As soon as the schedule has been generated (see 2.2.4 “Scheduler”) and published within the application, the volunteers can log in and look at the final schedule. This is when they get to know when their shift starts and ends. At this point, it is still not apparent what kind of tasks they will be working on. The volunteers have no option to export this schedule to an external application. Thus, they have to rely on the web interface. When it is time to start the shift, the volunteers report to the so-called SV Coordinators. These are also the ones who award the hours of the shift after the volunteer has reported it to be complete. This is verified by having the volunteers sign a sheet of paper that they got from the SV Coordinators when starting their shift. The paper gives a short introduction to the task or session and also allows for later attendance tracking of the event.

Schedule generation and access

What this application does also include, and is missing in the CHISV we looked at, is the ability to record the attendance count of each session or task. The volunteers will usually get a short introduction to the task/shift and more details to it on a printed sheet of paper. When they return, they will not only have signed the sheet to confirm their attendance but have also added how many people did attend this task or session. This data is very valuable for the Program Chairs later on.

Attendance tracking is done by the SV

When a volunteer knows that it will not be possible to do the shift it is possible to swap the shift and the tasks with it. It is important to notice that not the SV Coordinators nor the application will do any match-making. The only way a volunteer can swap shifts is by finding another volunteer to swap shifts with. The actual swap is done by the SV Coordinators when two volunteers can present a match. They will then update the schedule in the application such that the two volunteers can see it.

Manual task swapping is possible if the SV finds another one to swap with

2.2.2 Web Technologies

Software stack and user experience

When we look at the software stack, we see the back-end application consisting of PHP connected to a MySQL database. The front end is driven by CSS and JavaScript from the Bootstrap framework. This rather conservative approach provides a clear and well-known structure to most web developers. At the same time, however, using this software stack cannot make use of new web technologies, which could improve the user experience further. For every page change in the web interface, the server at BCU needs to send the entire view with all the data to the user. These responses are usually larger, as they include not only the data. We discovered in our survey in 2019 (see 3.2.2 "Survey") that many volunteers are using the system that provides their schedule on a wirelessly connected device. Due to the nature of wireless connections, they are sometimes confronted with bad coverage or an overloaded access point.

2.2.3 Messaging

Sending e-mails to volunteers and subsets

The system can send e-mails to all volunteers and also to certain subsets. Messages can be based upon templates (e.g. details about and when the SV dinner will be) but modified to fit the current target. There is no internal system to hold a copy of e-mails such that the volunteer can, later on, refer back to them in case of an unreceived e-mail.

2.2.4 Scheduler

Tasks are rarely overlapping

To ensure that all volunteers can work the expected amount of hours but not many more, the scheduler tries to find an optimal distribution between shifts. A shift is a time slot with start and end time. There can be multiple tasks in one shift. Most of the time tasks are aligned to the conference schedule with breaks in between. Only a few are overlapping between shifts. This makes it easier for the scheduler to avoid collisions and fragmentation within the volunteers' schedule.

Task attributes

To create all volunteers' schedules, the scheduler can make use of these attributes of a task:

- Start date and time
- End date and time
- Location
- Description
- Minimal number of required SVs
- Desired number of SVs
- Maximum number of allowed SVs

Range for the amount of SVs helps scheduling

Apart from these attributes, the scheduler also includes a range of hours volunteers should at least and at most work at the conference. A significant feature, however, is the option to specify a range of volunteers per task. This gives the scheduler the flexibility to meet the constraints of getting SVs to work the hours they need while making sure that all tasks get the optimal number of assigned SVs.

Some tasks require special abilities

Some tasks may require the volunteer to have some special ability. This could, for instance, be a task that can only be done by some very experienced SVs. It might also happen that the volunteer has to meet some other criteria. Another example is given by the so-called kids camp. An SV would have to fit some extended criteria to be able to be selected to work on this task.

Since the scheduler is not able to evaluate these extended requirements, the tasks that require special attention are usually filled first. This is done manually and will block the task from being used in scheduling.

Schedule overlapping tasks first

When the manual assignment is complete, the scheduler is run, which will try to fill as many tasks while ensuring an even amount of hours among all volunteers. Tasks that are considered to be difficult to fill (due to overlapping) are processed first. This makes it easier to get these tasks filled since at the start of the algorithm many SVs are available. Before the scheduling is done, the volunteers have also the ability to specify shift time slots in the web portal where they will not be available. This is also the place where they can specify their arrival time (see 2.2.5 "Step Three"). This will be considered by the scheduler as well.

After the scheduler has been run, the SV Coordinators evaluate the output and make some last adjustments. Once the schedule is found to be sound it is published and students get notified via e-mail.

2.2.5 Getting Enrolled

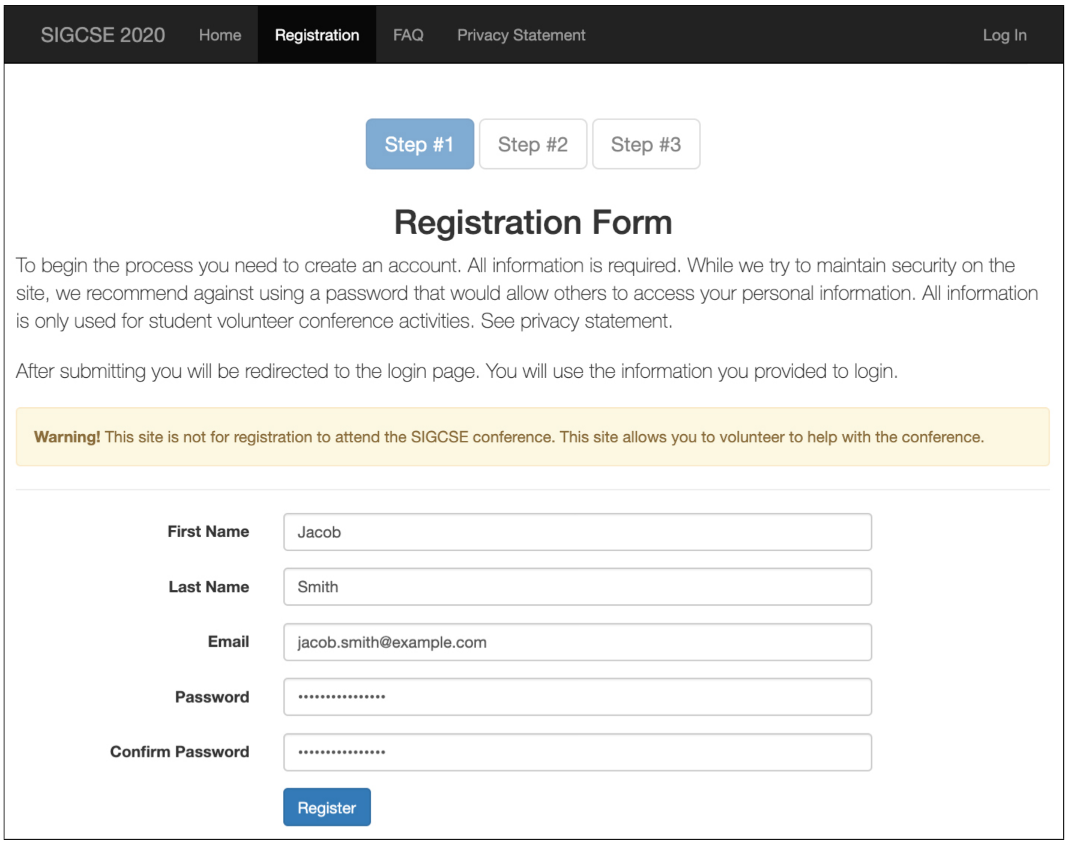
Step One

Three steps to get registered and submit availability

The portal greets a new user with a simple three-step process to get registered (we would call this "enrolling" after successful registration in CHISV). In the first step (see figure 2.3) the volunteer needs to give some general information, like the name and e-mail address. Since this step will create the account that the SV can later use to login to the system, the volunteer has to set a password at this stage as well.

Making step one required and all others optional

This is equivalent to registering in CHISV. Yet, the CHISV we looked at earlier would also ask for more personal details in this step, which SIGCSE is collecting in steps two and three, as we will see later. As this first step creates the account it cannot be skipped. However, steps two and three can be skipped. The information the volunteer would have provided can later be appended after logging in.



Registration Form

To begin the process you need to create an account. All information is required. While we try to maintain security on the site, we recommend against using a password that would allow others to access your personal information. All information is only used for student volunteer conference activities. See privacy statement.

After submitting you will be redirected to the login page. You will use the information you provided to login.

Warning! This site is not for registration to attend the SIGCSE conference. This site allows you to volunteer to help with the conference.

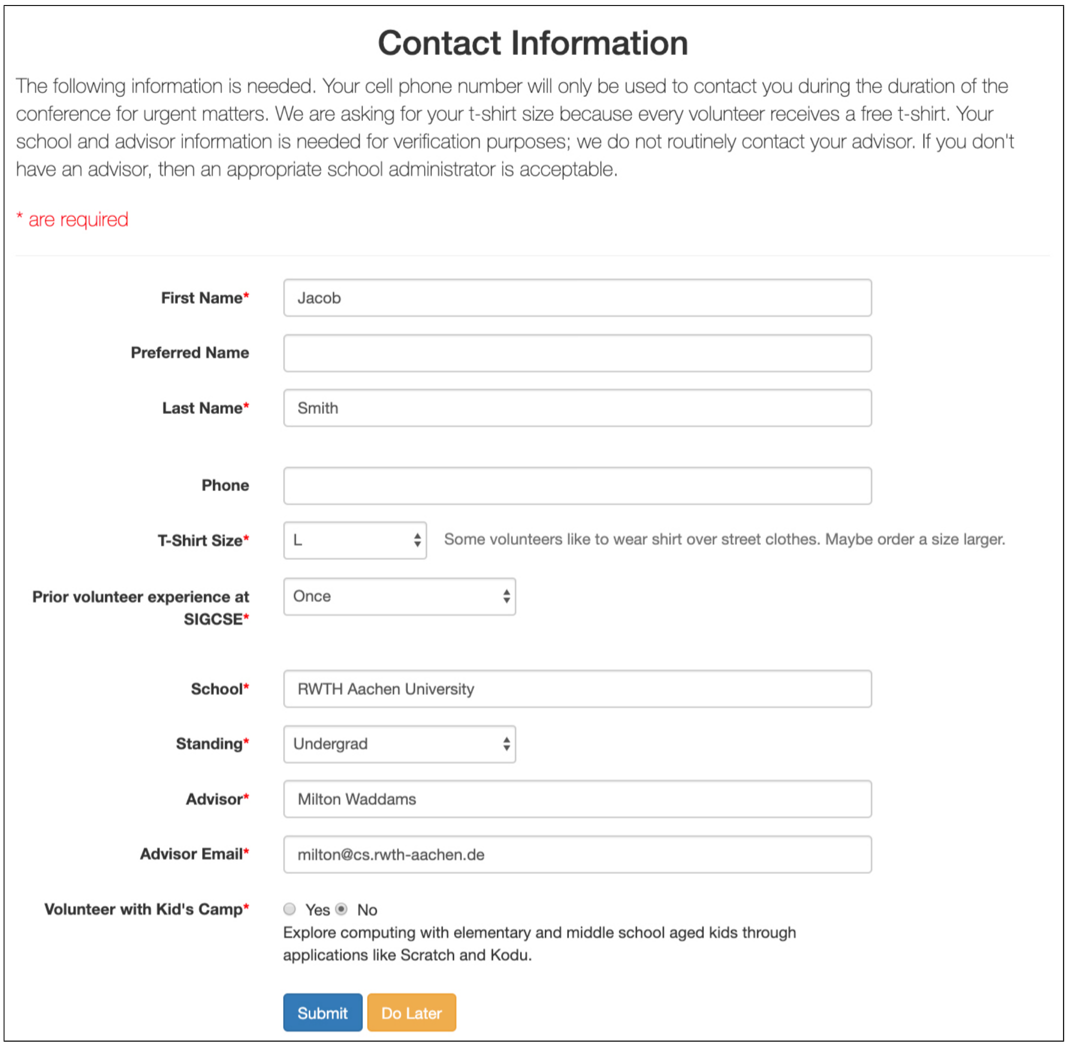
First Name	Jacob
Last Name	Smith
Email	jacob.smith@example.com
Password	••••••••••••••••
Confirm Password	••••••••••••••••

Figure 2.3: Step one will have the potential volunteer create a new account for the SIGCSE Volunteer Portal by requesting the SV's first and last name, an e-mail address and a password.

Step Two

Personal details can also be submitted later on

Step two (see figure 2.4), which is skippable, is all about more personal details and contact information of the SV. The system asks volunteers about their earlier experience with volunteering at SIGCSE, their T-shirt size and a phone number. Privacy-wise we see SIGCSE doing a pretty good job by leaving any but the essential pieces of information that are required for volunteering optional. To be able to verify the validity of the request, the application will require potential student volunteers to submit a contact of the advisor. Besides, volunteers will also have to specify their school and current degree program.



Contact Information

The following information is needed. Your cell phone number will only be used to contact you during the duration of the conference for urgent matters. We are asking for your t-shirt size because every volunteer receives a free t-shirt. Your school and advisor information is needed for verification purposes; we do not routinely contact your advisor. If you don't have an advisor, then an appropriate school administrator is acceptable.

* are required

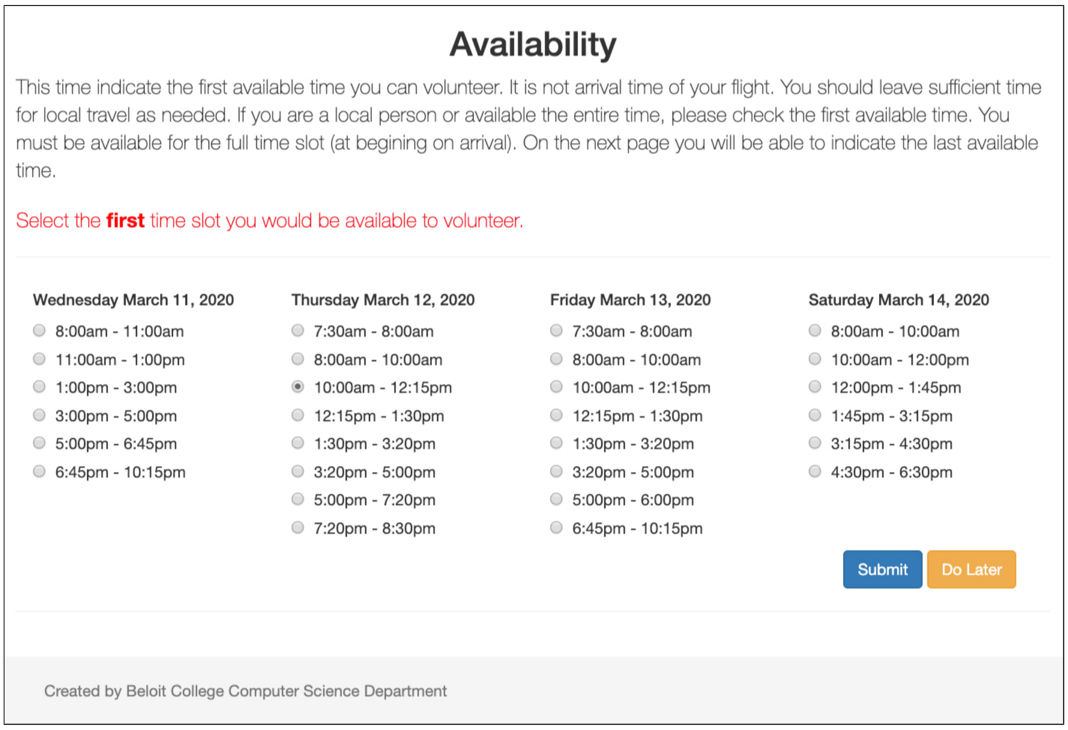
First Name*	Jacob
Preferred Name	
Last Name*	Smith
Phone	
T-Shirt Size*	L — Some volunteers like to wear shirt over street clothes. Maybe order a size larger.
Prior volunteer experience at SIGCSE*	Once
School*	RWTH Aachen University
Standing*	Undergrad
Advisor*	Milton Waddams
Advisor Email*	milton@cs.rwth-aachen.de
Volunteer with Kid's Camp*	Yes [✓ No] Explore computing with elementary and middle school aged kids through applications like Scratch and Kodu.

Figure 2.4: Step two will have the volunteer submit more personal details and contact information of the volunteer and the associated advisor.

Step Three

Provide earliest availability

Step three (see figure 2.5), which is the last, is also optional. The required data can also be submitted later on after logging in. In this step, the volunteers will submit the first possible time slot they are available to take a shift. The system will consider this input and only schedule possible shifts after this first time slot. The scheduler will also not assign time slots that the volunteer declared to be unavailable for, which can be set after logging in.


Availability

This time indicate the first available time you can volunteer. It is not arrival time of your flight. You should leave sufficient time for local travel as needed. If you are a local person or available the entire time, please check the first available time. You must be available for the full time slot (at begining on arrival). On the next page you will be able to indicate the last available time.

Select the **first** time slot you would be available to volunteer.

Wednesday March 11, 2020	Thursday March 12, 2020	Friday March 13, 2020	Saturday March 14, 2020
8:00am - 11:00am	7:30am - 8:00am	7:30am - 8:00am	8:00am - 10:00am
11:00am - 1:00pm	8:00am - 10:00am	8:00am - 10:00am	10:00am - 12:00pm
1:00pm - 3:00pm	[✓ 10:00am - 12:15pm]	10:00am - 12:15pm	12:00pm - 1:45pm
3:00pm - 5:00pm	12:15pm - 1:30pm	12:15pm - 1:30pm	1:45pm - 3:15pm
5:00pm - 6:45pm	1:30pm - 3:20pm	1:30pm - 3:20pm	3:15pm - 4:30pm
6:45pm - 10:15pm	3:20pm - 5:00pm	3:20pm - 5:00pm	4:30pm - 6:30pm
	5:00pm - 7:20pm	5:00pm - 6:00pm	
	7:20pm - 8:30pm	6:45pm - 10:15pm	

Created by Beloit College Computer Science Department

Figure 2.5: Step three will ask for the first possible time slot the volunteer will be on-site to receive a shift.

Should the SV have selected a date for the first available time slot that makes it's unlikely to gain all required hours, which are needed to wave the conference fee, the system will warn the user. The availability the volunteers specify is crucial to the scheduler, which tries to create a fair distribution of shift time slots among all volunteers. However, the volunteer has no further opportunity to specify preferences for when the own shift will be scheduled, let alone any preference regarding the tasks of a shift.

2.3 SIGGRAPH Student Volunteer System

Purpose-built web application similar to SIGCSE system

The Special Interest Group on Computer Graphics (SIGGRAPH) is, just like SIGCSE and CHI, using a web application. This system was specifically built to register and manage student volunteers and their tasks or shifts. It was developed in 2017 by five members of the Student Volunteer Sub Committee (SVSC). SIGGRAPH's so-called SV Portal is more similar in terms of features to SIGCSE's system than to CHISV at which we looked at earlier. The SV Portal is reset every year and seeded with the data and dates for the upcoming conference by the SVSC. It is specifically built for the needs of SIGGRAPH and not intended to provide service to multiple conferences at a time.

Volunteers can propose at which venue they would like to work at

Just as with SIGCSE's approach, student volunteers do not work on tasks directly but rather on a shift. A shift is a time slot that was assigned to the SVs either manually or by the scheduler. When we saw that with SIGCSE student volunteers could block a time slot where they would not be available, SIGGRAPH's SV Portal is handling this differently. Before the conference, student volunteers can tell the SVSC about their preferred venues they would like to work at. This information is then considered by the scheduler.

Modern SPA with AWS intergration

SIGGRAPH uses two distinct applications for handling student volunteer-related matters. All details about SVs and shifts are held within a PostgreSQL database. This database is accessed by a Laravel application for administrative tasks and a single-page application (SPA), the SV Portal, which is used by the student volunteers. This application is built using Vue.js and Vuex interacting with the database via Node.js microservices. These are run via Amazon Web Services Lambda (AWS Lambda). The Vue.js application is also hosted on AWS servers.

Serverless approach with AWS CDN infrastructure

Due to this architecture SIGGRAPH can not only achieve outstanding reliability but also use AWS content delivery network (CDN) which can reduce the initial page load time for all volunteers worldwide. Since the SV Portal is built as a SPA, it can not only dynamically react to connectivity issues but also provide much richer feedback to the users than a common request-driven approach.

The SV Portal is mainly used by three parties:

- Student Volunteer Sub Committee (SVSC)
- Team Leaders (TLs)
- Student Volunteers (SVs)

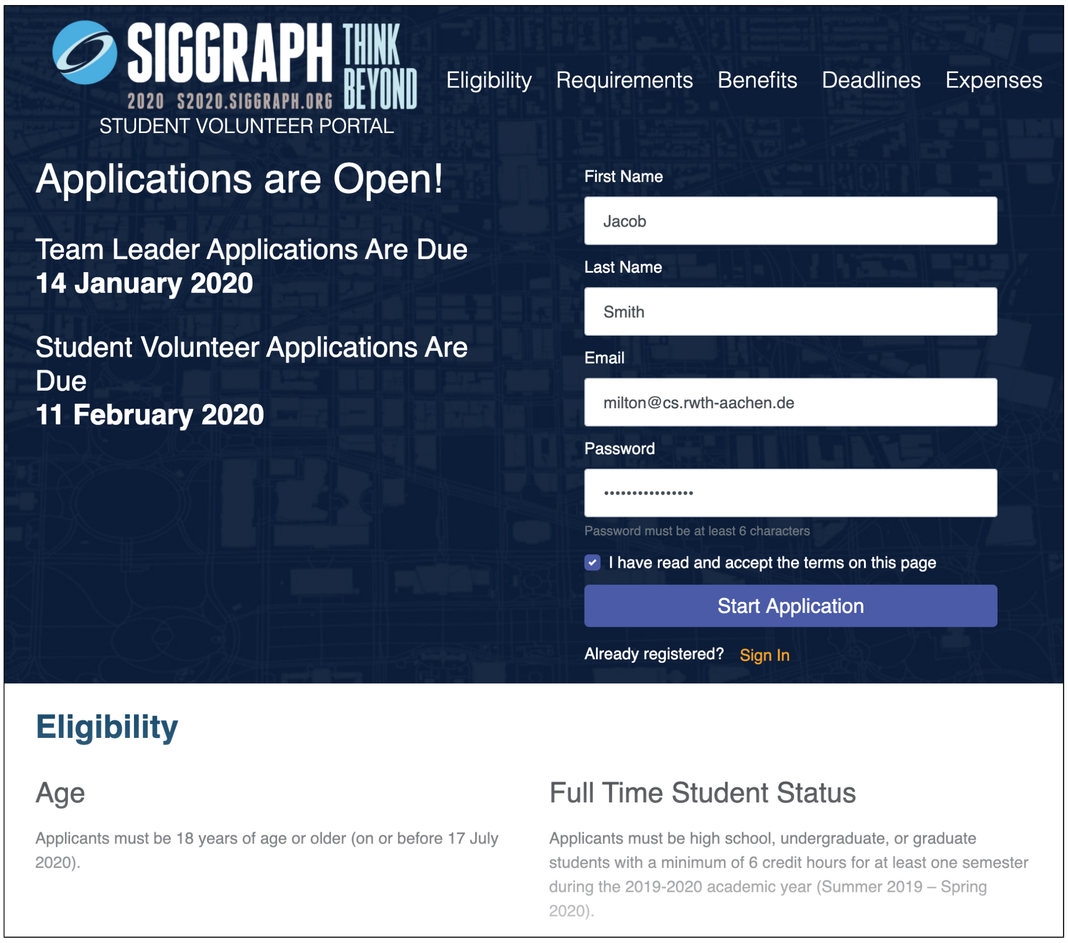
SVSC appoints Team Leaders, which can score SV applications

The members of the SVSC can be seen as the administrators of the SV Portal. Team Leaders and SVs will apply through the SV Portal (see 2.3.2 “Applying as Team Leader or Student Volunteer”) after registration (see 2.3.1 “Registration”). The members of the SVSC can appoint Team Leaders by accepting their application and assign Student Volunteer applications to them. The Team Leaders will then score those applications and make the results available for the SVSC. They will then accept or reject the SV applications, which have been scored by the Team Leaders before. After being accepted, SVs, and Team Leaders will have to confirm or decline their attendance.

2.3.1 Registration

Create new account every year

New student volunteers, who want to apply as an SV or a Team Leader (TL), will first have to create a new account. This is even necessary when the applicant has had an account some years before since the application is reset every year – wiping all user accounts. The account creation requires specifying a first and last name, appending an e-mail address and also a password (see figure 2.6). Submitting the registration will create and log the user into the new account.


SIGGRAPH THINK BEYOND
2020 S2020.SIGGRAPH.ORG
STUDENT VOLUNTEER PORTAL

Eligibility Requirements Benefits Deadlines Expenses

Applications are Open!

Team Leader Applications Are Due
14 January 2020

Student Volunteer Applications Are Due
11 February 2020

First Name: Jacob
Last Name: Smith
Email: milton@cs.rwth-aachen.de
Password: ••••••••••••••••
Password must be at least 6 characters
☑ I have read and accept the terms on this page

Start Application

Already registered? Sign In

Eligibility

Age

Applicants must be 18 years of age or older (on or before 17 July 2020).

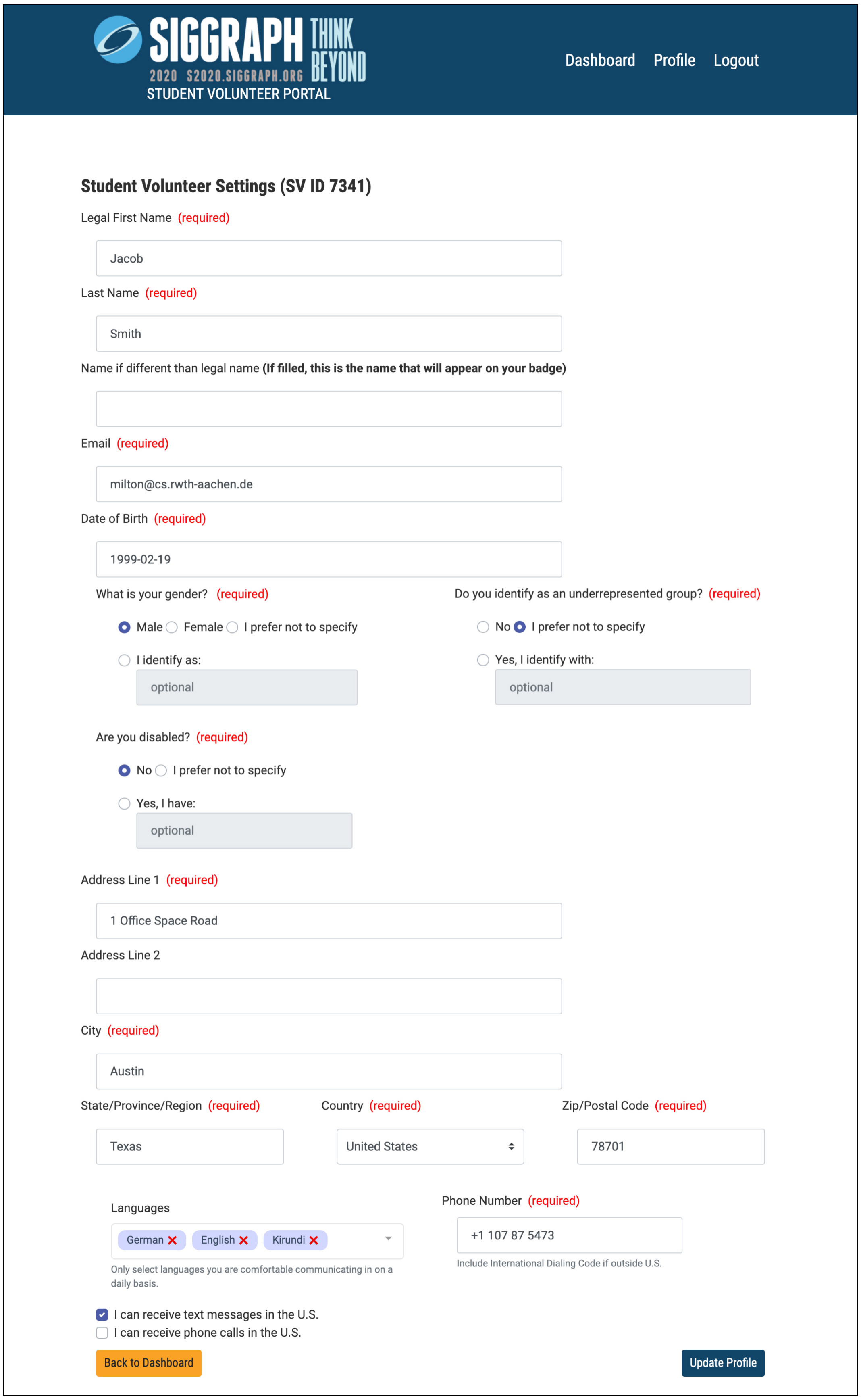
Full Time Student Status

Applicants must be high school, undergraduate, or graduate students with a minimum of 6 credit hours for at least one semester during the 2019-2020 academic year (Summer 2019 – Spring 2020).

Figure 2.6: New volunteers will start with creating an account for the system. This requires them to specify their first and last name and to provide an e-mail address as well as a password.

Personal details will later be used for the conference registration

After registration and login, the volunteer will have to provide some additional personal details (see figure 2.7). These also include the date of birth, precise address, phone number, spoken languages, and the preferred badge name. This is important as these details will later be used for registering the volunteer for the conference. This is unique to SIGGRAPH's SV Portal. Neither SIGCSE nor the version of CHISV we have looked at provides such an integration.



Student Volunteer Settings (SV ID 7341)

Legal First Name (required): Jacob

Last Name (required): Smith

Name if different than legal name **(If filled, this is the name that will appear on your badge)**:

Email (required): milton@cs.rwth-aachen.de

Date of Birth (required): 1999-02-19

What is your gender? (required)
[✓ Male] Female I prefer not to specify
I identify as: optional

Do you identify as an underrepresented group? (required)
No [✓ I prefer not to specify]
Yes, I identify with: optional

Are you disabled? (required)
[✓ No] I prefer not to specify
Yes, I have: optional

Address Line 1 (required): 1 Office Space Road

Address Line 2:

City (required): Austin

State/Province/Region (required): Texas

Country (required): United States

Zip/Postal Code (required): 78701

Languages: German ✕ English ✕ Kirundi ✕
Only select languages you are comfortable communicating in on a daily basis.

Phone Number (required): +1 107 87 5473
Include International Dialing Code if outside U.S.

[✓] I can receive text messages in the U.S.
[ ] I can receive phone calls in the U.S.

Back to Dashboard

Update Profile

Figure 2.7: Student volunteers can edit their preferences and details after logging in to the portal.

2.3.2 Applying as Team Leader or Student Volunteer

Team Leader application is optional and done as part of the SV application

To be able to volunteer at the conference, volunteers will have to apply. During this three-step process, the applicant will have the option to also apply as a Team Leader. The deadline for Team Leader applications is roughly one month before the deadline of student volunteer applications. This ensures that the SVSC has a good number of Team Leaders to score the SV applications once the deadline passed.

The application process will pose multiple questions to the potential volunteer:

- Step 1
  - University/School/Institution name
  - City, State, Country
  - Expected graduation date + proof of enrollment
  - Professional reference (e.g. Teacher, Advisor, Supervisor)
  - Earlier SIGGRAPH experience as attendee or SV
  - Areas of interest (e.g. Arts, Web, Gaming)
  - T-Shirt size
  - Dietary Restrictions
- Step 2
  - Explain motivation and conflict handling and how to approach individuals from a different background
- Step 3
  - Financial assistance for transportation and lodging
  - Explain motivation and experience when applying as Team Leader

In addition to the questions of steps one and two, potential Team Leaders will also have to express their motivation and level of qualification, which can include earlier mentorship or other relevant leadership experience. This can be done in step three together with the request for additional financial assistance. Some of the details of the application will later also be used for the conference registration.

Team Leader duties and financial assistance can be requested in step three

Once the deadline for all applications has passed, the SVSC will assign each Team Leader a set of SV applications to evaluate and score. Their work helps the SVSC to not get overwhelmed by the amount of SV applications.

Team Leaders evaluate the SV's application

2.3.3 Shift Swapping

An outstanding functionality is found in the SV Portal's shift swapping procedure. It was added in 2018 and shortly after disabled again. Students were able to find other students to swap the shift with. To swap a shift, students would mark their shift as swappable. Other SVs would then see this shift in the overview of available swappable shifts. To then swap the shift, both parties would accept the swap request, which modified the schedule of both parties involved. As the whole process required no participation of the SVSC, these unmonitored changes made it more difficult for them to keep track. Thus, task swapping was disabled shortly after its introduction.

Shift swapping was introduced in 2018 and disabled shortly after

The current procedure to swap shifts is similar to how it is done in our new version of CHISV. Requests and the actual change to the schedule are not done within the application but in external tools (e.g. Slack or Google spreadsheet). SIGGRAPH uses a shared Google spreadsheet to have SVs list their shift swap requests. The SVSC will evaluate the request, verify that the shift is swappable and that no restrictions are violated. The change is then made in the (Laravel) admin portal. This approach makes shift swapping more structured such that the schedule is only changed manually by the SVSC's hand.

Task and shift swapping not done as part of the application

2.3.4 Schedule Creation and Accounting

Volunteers can state their preferred venue

After volunteers have been accepted, they will have to confirm or decline their attendance. During this phase, they also have the possibility to state their preferred venue. Depending on the venue's requirements, this preference can be considered by the scheduling tool. All the scheduling is done before the conference starts, usually a few months ahead and not on-site.

Special shifts are manually filled

Shifts that require special training are manually filled with certain student volunteers. Since the scheduler will only fill free shift slots, these SVs will remain assigned when the scheduler is run. The scheduler tries to distribute shifts among all student volunteers such that their hours are usually kept below 30.

Team Leaders check-in/out volunteers

At the venue, Team Leaders will check-in SVs who arrive for their shift. When the Team Leader checks the SV out again the hours the volunteer has worked will be awarded to the volunteer's account. We see that this job of assignment management is quite similar to the duties of a Day Captain in the version of CHISV we have looked at so far. Volunteers should check their schedules frequently on their smartphones. This is part of the orientation from the SVSC. There is no option for the volunteers to export their schedule to an external application. However, this is in any case not advisable, as the schedule tends to change daily. Thus, the volunteers should rely on the web application, which will show them their shifts.

Chapter 3

Requirements Analysis

We started getting in contact with the Student Volunteer (SV) Chairs for the Conference on Human Factors in Computing Systems (CHI) 2019 in fall 2018. We intended to exchange ideas and feedback about the previous version of CHISV while the CHI 2019 took place in Glasgow (UK) between the 4th and 9th of May.

Interviewing CHI 2019 SV Chairs

Before we started reaching out to the participants at the conference, we defined our target groups of which we assumed participants would fall in most likely.

3.1 Target Audience

To identify our target audience, we used knowledge of the CHISV maintainers, previous CHISV users and the Student Volunteer Chairs of CHI 2019. We identified three groups of users of the application:

- SV Chairs
- Experienced student volunteers
- Novice student volunteers

3.2 First Iteration

3.2.1 Interviews

First interviews conducted at CHI 2019

During the CHI 2019, we conducted a study with six participants of which two fell into each of our defined audience groups (see 3.1 “Target Audience”). The study was done as part of a video call with the participant individually. The participants performed all the tasks they would usually perform during the conference. During the execution, the participants were describing what they were using the system for and reflecting on the series of tasks they had to perform to accomplish their goal (Think-Aloud).

Distinct feedback from each group

This helped us a lot to understand not only the ways of interaction the previous version of CHISV offered but also to identify some of its shortcomings. Having the participants split into three groups based on their earlier experience with the system, gave us the unique possibility to precisely look into the different domains the application is used for.

SV Chairs

Tasks of the members of the SV Chair and their requirements

We were able to see the perspective of the members of the organizing SV Chair. We got insights into how they manage tasks and student volunteers and make sure that all SVs are getting the tasks they bid for while keeping an eye on fair assignments. The SV Chairs also use the application to contact individual or groups of SVs. The report generation is an important field for them such that they can keep an eye on all the different fields intersecting with the volunteer’s experience. This helps them to fill tasks evenly and distribute the workload among all student volunteers more even and fair. The reports are subsequently also often used by other chairs of the conference to give additional insight into demographics and attendance. In the next year this may help SV Chairs to accept a fair amount from each region or institution.

Experienced Student Volunteers

Day Captains are experienced student volunteers

The feedback we got from the study with the experienced student volunteers gave us more insight into the Day Captain duties. Experienced student volunteers often get selected to be a Day Captain for some time. As such, they need to know the application in more depth. For instance, they have to use several more advanced features of the application, which are hidden for the average SV.

Day Captains help student volunteers

Solving SV related issues is always a time-critical task since all tasks are tightly scheduled. Volunteers will check in with the Day Captain on every task start and end. This can result in queues at the Day Captain's counter. Any implementation that aims at the tasks a Day Captain does has to keep this in mind.

Day Captains are in close contact with the SV Chairs – sometimes only via direct messaging. Both parties are using the system to note down issues and create remarks to assignments of volunteers. This gives them a defined way on how to keep track of any assignments or SV related occurrences.

Day Captains require special permissions

In the previous version of CHISV, Day Captains had the same permissions and abilities in the system like the SV Chairs (they were sharing the credentials). Any permission system in the new version of the application should thus precisely evaluate the required permissions for all created roles.

Novice Student Volunteers

Focus on the interface and feedback

Our third group of novice student volunteers helped us to understand which needs may arise when SVs are new to a system with which they have never worked before. This can be especially difficult in a time-sensitive and often unfamiliar environment. Student volunteers come from all around the world. Thus, the application should make it easy to get started and adapt to different requirements (e.g locales and timezones). While we saw a lot of need for advanced functions with the other groups, we noticed that for this third group we would have to focus more on a user interface with a great amount of feedback and visually aiding elements.

Novice student volunteers have various qualities of connectivity

This group of volunteers was mostly interacting with the system from a mobile device, like a tablet or smartphone. Their internet connection wasn't always the best as they were staying in various locations with different connectivity options. Many volunteers we spoke with were interacting with the system on the go.

3.2.2 Survey

Survey was filled by 69 volunteers

During CHI 2019 we also encouraged the student volunteers and SV Chairs to partake in a survey. We used this survey to get a good overview of what the needs of the users are. Of all 200 available student volunteers, 69 provided us with details into their experience with the previous version of CHISV.

Focus on the user experience (UX)

We asked questions to determine their level of experience with the system and what they are mostly using the system for. We were very open for requests for new features or changes of the existing system. We laid out a couple of questions to understand what works great for them and what they think should be changed or adapted to new requirements. We noticed that the user interface of the previous version of CHISV was not particularly suited for various form factors. This made us go into greater detail about the user interface in the survey. It confirmed our theory in which we saw volunteers mostly using the application on a mobile end device, like a smartphone.

3.3 Second Iteration

Additional feedback from CHI, UIST, and MobileHCI

After CHISV had been reimplemented, we evaluated it with the organizing SV Chairs of CHI 2020, UIST 2020 and MobileHCI 2020. Furthermore, some CHI 2020 SVs agreed to evaluate registration, enrollment, filtering for desired tasks, placing bids, and using the new calendar UI. The feedback from the SVs and the SV Chairs was then later used to revise some features or to provide missing functionality, which was initially not part of the requirement set.

3.3.1 Interviews

In-depth group interview

For each SV Chair, we conducted an in-depth group interview. We structured the interview to simulate the tasks an SV Chair would have to do before and at the conference. Since they were seeing and interacting with the application for the first time we were able to collect unbiased feedback. While we were going through our prepared sequence of tasks, the participants were interacting with the system and simultaneously providing feedback about their experience.

UX knowledge

Fortunately, all participants have experience in the domain of Human-Computer Interaction (HCI) and were also able to provide feedback about what they think could be improved to be better suited for student volunteers.

3.3.2 Survey

To be able to compare the results of the survey from the first interaction, we asked the SV Chairs to fill out a new survey. For this, we copied the first survey and adapted its questions to target the new system. Yet, the content of the questions was mostly kept in place such that comparison was possible.

Demo conference for SV experience testing

The same survey was also filled by volunteers who agreed to evaluate the new system as an SV. For that task, and as CHI 2020 was canceled, we set up a demo conference for these participants. We seeded the conference with real data from CHI 2019 and had them go through the steps of registration, enrollment, and finally task bidding. No guidance was given to them that could influence their first time experience with the interface or the application. After having gone through all steps, some volunteers filled the survey and gave us insight into the SV experience.

3.4 Requirements

Sorting feedback into two sets

Since the requirements from the group of novice and experienced volunteers overlap, we bundled them into one group. The feedback we got from the SV Chairs and the Day Captains, which was mostly organizational, was put into the requirements set of "SV Chairs". Requirements from the Day Captains, which focused on the topics student volunteers come into contact with while bidding and planning their schedule, was put into the "Student volunteers and Day Captains" set of requirements.

Since the feedback of the people we spoke with mostly covers changes or additions to the system, we will begin each of the six domains (introduced on page 41) by describing the basic requirements, which were obvious for all participants, therefore not mentioned to us directly. We call this set of requirements "General aspects".

This gives us three requirement sets[1]:

- General aspects ■
- SV Chairs 🟥
- Student volunteers and Day Captains 🟪

[1]The colors in the listing correspond to the colors used in the concept map 3.1.

Each set of requirements has a different focus. While "General aspects" tend to sketch more the demand for structure and functionality, we see that requirements from the set of "SV Chairs" fall more into the domain of manageability and accountability of resources and volunteers. Finally, the requests we got for the "Student volunteers and Day Captains" group are more referring to topics that volunteers have while volunteering and working on their tasks. This also includes their schedule management and task bidding.

Distinct requirements

After filtering and aggregating all requests, improvements, and feedback we got from the three interest groups, we evaluated their feasibility also with respect to technical limitations and visual esthetics. We also took into consideration the factor of maintenance. This is an important aspect to ensure that the system can be kept up to date and provide service. For all requirements, we also considered data protection and privacy following the General Data Protection Regulation (GDPR).

Feedback was filtered, aggregated, and evaluated

The following section will present all requirements for each of the three requirement sets. This includes the feedback that we were able to collect in the second iteration (see 3.3 “Second Iteration”). This makes this section a complete list of all desired functions the system should implement.

We will evaluate the requirements for each set in the following six domains:

- 3.4.1 “Web Application”
- 3.4.2 “Enrollment of Volunteers”
- 3.4.3 “Accepting Volunteers and Lottery”
- 3.4.4 “Task Bidding”
- 3.4.5 “Task Assignment and Auction”
- 3.4.6 “Notifying Volunteers”

To get a better overview of all requirements and the associated domains, we organized them into one concept map in figure 3.1. The different domains are presented in the middle, whereas each requirement is shown in its requirement set's color (see the listing on page 40).

3.4.1 Web Application

General aspects

Use modern standards

1. **Web application built with modern frameworks** CHISV should be an application that is accessible from a web interface. It has to be compatible with all major web browsers. The implementation has to put special emphasis on the maintainability of the entire product. Thus, it should only rely on well-known frameworks and tools.

Hosting at RWTH Aachen University

2. **Integrate with given infrastructure** Since the aim is to host the application with the infrastructure of the Media Computing Group of RWTH Aachen University, it has to be compatible with the stack in use there. To tackle these requirements most efficiently, the implementation should be coordinated with one of the administrators of this facility.

Compatible with third-party applications

3. **Extendability with third-party applications** To make further extension possible without any additional work, the application should provide a native application programming interface (API) for third-party applications. This should be realized by applying commonly used approaches, like publishing a RESTful[2] API with token authentication.
   Furthermore, the application should be back-end independent, meaning that the back end could adapt to the environment where it is deployed. Since this is likely a framework-related requirement, this has to be evaluated when the framework is chosen.

Be able to scale out

4. **Performance and scalability** Since CHISV is used by various users accessing it from different regions of the world, the required network bandwidth

[2]Standardized API to access resources

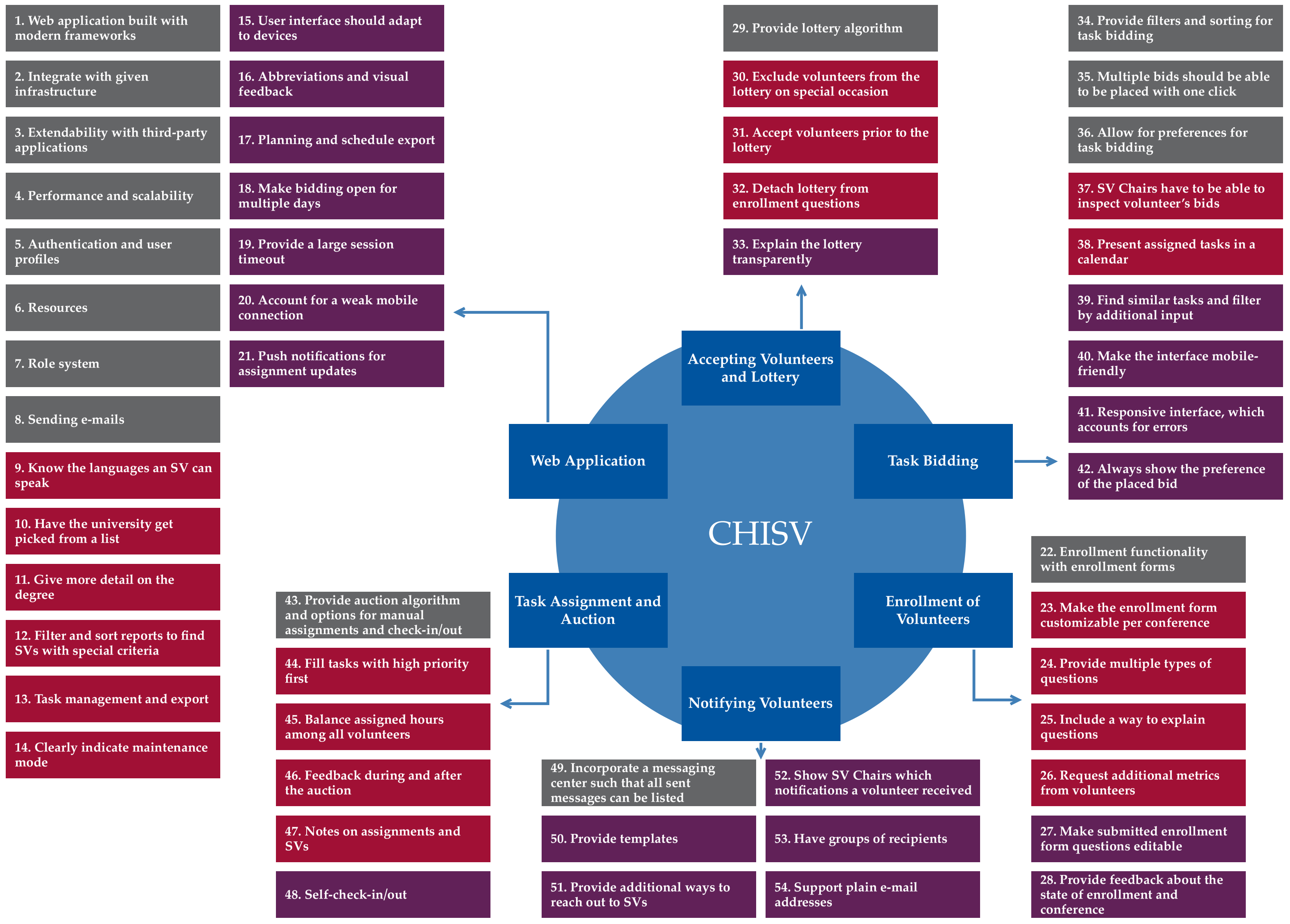

1. Web application built with modern frameworks
2. Integrate with given infrastructure
3. Extendability with third-party applications
4. Performance and scalability
5. Authentication and user profiles
6. Resources
7. Role system
8. Sending e-mails
9. Know the languages an SV can speak
10. Have the university get picked from a list
11. Give more detail on the degree
12. Filter and sort reports to find SVs with special criteria
13. Task management and export
14. Clearly indicate maintenance mode
15. User interface should adapt to devices
16. Abbreviations and visual feedback
17. Planning and schedule export
18. Make bidding open for multiple days
19. Provide a large session timeout
20. Account for a weak mobile connection
21. Push notifications for assignment updates
29. Provide lottery algorithm
30. Exclude volunteers from the lottery on special occasion
31. Accept volunteers prior to the lottery
32. Detach lottery from enrollment questions
33. Explain the lottery transparently
34. Provide filters and sorting for task bidding
35. Multiple bids should be able to be placed with one click
36. Allow for preferences for task bidding
37. SV Chairs have to be able to inspect volunteer's bids
38. Present assigned tasks in a calendar
39. Find similar tasks and filter by additional input
40. Make the interface mobile-friendly
41. Responsive interface, which accounts for errors
42. Always show the preference of the placed bid
Accepting Volunteers and Lottery
Web Application
Task Bidding
CHISV
Task Assignment and Auction
Enrollment of Volunteers
Notifying Volunteers
22. Enrollment functionality with enrollment forms
23. Make the enrollment form customizable per conference
24. Provide multiple types of questions
25. Include a way to explain questions
26. Request additional metrics from volunteers
27. Make submitted enrollment form questions editable
28. Provide feedback about the state of enrollment and conference
43. Provide auction algorithm and options for manual assignments and check-in/out
44. Fill tasks with high priority first
45. Balance assigned hours among all volunteers
46. Feedback during and after the auction
47. Notes on assignments and SVs
48. Self-check-in/out
49. Incorporate a messaging center such that all sent messages can be listed
50. Provide templates
51. Provide additional ways to reach out to SVs
52. Show SV Chairs which notifications a volunteer received
53. Have groups of recipients
54. Support plain e-mail addresses

Figure 3.1: CHISV requirements listed by their domain 🟦 (see listing on page 41). Boxes in the color grey ⬛ denote general requirements ("General aspects"). Bordeaux boxes 🟥 present requirements of SV Chairs. Violet boxes 🟪 list requirements of student volunteers and the Day Captains

and latency should be kept as low as possible. When many users engage with the application this should not be noticeable for the user. The system should be able to scale to the required size. One could for example evaluate the option to deploy multiple versions of the application and then load-balance the requests. This has to be supported with minimal adaptions to the configuration or code.

User authentication and management

5. **Authentication and user profiles** Student volunteers should be able to register on the website, provide some personal details about themselves, and log in with their e-mail address and password. Users have to be able to reset their password and manage their details with the help of the system. Users with special roles need be able to view role-specific subsets of users.

Manage and assign resources to users

6. **Resources** CHISV has to provide a way to create and modify resources like conferences, tasks, reports, and associations of volunteers to some of these resources.

Assign and revoke permissions to manage access rights of users

7. **Role system** A role system should be used to manage the view and access levels of users. Special features (like assigning tasks) may only be available for specific roles. Permissions have to be used to permanently or temporarily grant access to users. These roles have been proposed:
   - **Administrator** – Has access to all views and resources
   - **SV Chair** – Has access to one specific conference and all its associated resources
   - **Day Captain** – Has access to task-related resources of one specific conference and can modify the assignments of SVs of this conference
   - **Student Volunteer** – Has access to task bidding and basic schedule management

Connect with mail servers to deliver notifications

8. **Sending e-mails** The application has to be able to connect to e-mail gateways and use those to deliver e-mails to student volunteers or other users of the

system. E-mails should be able to be sent by SV Chairs and Day Captains.

SV Chairs

9. **Know the languages an SV can speak** Student Volunteers should be able to specify the languages they speak. The choices should come from a predefined list of languages from the back end. This enables the SV Chairs to find SVs with a special language ability whenever it is needed for a specific task. This field should also be available in a report such that demographic reports can include this metric.

Collect characteristics of the volunteer

10. **Have the university get picked from a list** Having the SVs pick their university from a list of predefined universities can help to ensure that SVs from all known universities are equally present. This would make it possible to filter for institutions that have a lot of enrolled SVs and the ones with only a few. Additionally, holding this data can also help SVs to find other SVs from the own university or one nearby.

11. **Give more detail on the degree** Extending the data collected on the degree can help to filter for SVs. Especially the options for PhDs can be extended to also show and give an option for the year.

12. **Filter and sort report to find SVs with special criteria** The application should provide a way to list SVs based on specific criteria. Sorting and filtering the collection would greatly improve the options an SV Chair has to find students with a special ability. For example, this would provide an option to filter out SVs who need additional information to get a VISA or all SVs who speak a specific language.

Find volunteers by criteria

13. **Task management and export** One of the key features of the application is task management. A simple and fast user interface is inevitable. Additionally, the system has to provide a routine to create

Task batch creation and modification

or update tasks via comma-separated values (CSV) import.

Task management has to provide a way to interactively update modified tasks. It should also be possible to only update a small subset of attributes of all tasks (e.g the location). The import has to be compatible with the results exported from the reports section.

Sorting and filtering of tasks

The interface has to provide a way to filter and sort for tasks of a given day, name, location, or priority. This requires the view to provide ways to input a name, location, and other criteria. Having the ability to see tasks of multiple days could help while comparing tasks. It would also give an overall better overview of the tasks of a common category.

Clearly indicate the state of the system

14. **Clearly indicate maintenance mode** The previous version of CHISV made use of a maintenance mode. A conference was put in said mode while the auction was running and the manual assignment was being done. While the conference was in maintenance mode SVs were unable to access the application, their assignments, or the task overview. The maintenance mode did not clearly explain why it was there, leaving many SVs confused. The new version of CHISV should clearly indicate in which mode it is and what this mode is for.

Student volunteers and Day Captains

Adapt to different device sizes

15. **User interface should adapt to devices** While we saw a lot of the SV Chairs using the system on a laptop many SVs were interacting with the user interface from a smartphone or tablet. These devices not only make use of smaller screens with different aspect ratios but also are often used in different environments than a laptop. The interface of the web application has to account for the different display sizes and aspect ratios.

Design for errors

Smartphones and tablets are usually used by touch input. When showing interface components the user interface will have to account for this as well.

These components will have to be touchable – even when the user is in a busy environment. Any error that might occur has to be correctable on the spot.

16. **Abbreviations and visual feedback** Whenever abbreviations have to be used in the interface, they have to be explained beforehand or get explained when used. To not clutter the interface on mobile devices, it could be beneficial to have one central place in the application where users can read up about abbreviations and features that are not instantly reasonable. This is especially required for the lottery and auction (see 4.4.3 “Lottery” and 4.4.4 “Auction”).

Explain abbreviations and describe algorithms

17. **Planning and schedule export** To help the volunteers organize their day and to ease the planning of tasks they like to bid for, it requires an appropriate visual representation. To create a more visually appealing interface, which is also easier to grasp, the integration of a calendar should be considered. This gives the SVs a better understanding of how tasks and assignments are placed to each other. To bridge the gap between the SVs’ private calendar and the schedules in the application, schedules from it should be exportable. This could either be per event or for an entire day, week, or month. For the SV, this could heavily simplify the planning of their day. The export has to be in a well-known format.

Calendar export

18. **Make bidding open for multiple days** Student volunteers want to be able to plan ahead. The ability to bid on multiple days (as long as the SV Chair allows for it) should be included in the system. In combination with additional filters for the task selection, the SV can then create a schedule before the conference begins.

Allow bidding on arbitrary time intervals

19. **Provide a large session timeout** The previous version of CHISV featured a rather short session timeout. The SVs reminded us that having to log back in to the application every few hours interrupts the interaction with the system heavily. The session timeout should be greatly increased.

Users connectivity might be limited

20. **Account for a weak mobile connection** Our survey from 2019 showed us how and where student volunteers interact with the system. SVs also interact with the system while walking or commuting. The system has to account for slow and disruptive connections. As all information that gets transferred back and forth between the user and the system might also count against a user's mobile data quota, the latency and request size has to be kept to a minimum.

Keep the user informed about changes in the system

21. **Push notifications for assignment updates** The survey also showed us that SVs want to get announcements whenever something task or assignment related changes in the system. This could give the SV feedback on a successful task completion without having to manually peek into the application. There could be multiple channels like E-mail or Slack to quickly push notifications to the user.

3.4.2 Enrollment of Volunteers

General aspects

22. **Enrollment functionality with enrollment forms** The application will have to provide a function to associate student volunteers with a conference. This process is called "enrolling". During this enrollment, the SVs answers a set of questions to give the SV Chairs more insight about their abilities, disabilities, and additional needs.

SV Chairs

Customizable enrollment forms

23. **Make the enrollment form customizable per conference** Since each conference has different requirements towards the SVs, the set of questions has to be customizable. A set of questions is the basis for every enrollment form. These forms should be able to be manually weighted to enable the SV Chairs to filter for student volunteers with specific needs or criteria. This can also be used to provide

additional information back to the volunteer. For example, this often happens to volunteers who need additional information to get a VISA.

24. **Provide multiple types of questions** There should be different types of questions. SVs will have to be able to answer simple binary questions, input a predefined amount of text or provide a number as an answer. All quantifiable questions have to be considered when the enrollment form weight is calculated.

Different types of questions

25. **Include a way to explain questions** Some questions might require additional information to better understand them. Every question should be able to contain a hint, which is shown to the user on request.

26. **Request additional metrics from volunteers** Apart from the enrollment form, the student volunteer should also be able to provide nationality and country of residence. This information may only be available once for each user, rather than on a per conference level. The SV must also be able to pick the current degree and the spoken languages from a list with predefined choices.

Collect additional metrics for each volunteer

Student volunteers and Day Captains

27. **Make submitted enrollment form questions editable** Once an enrollment form is submitted the information should still be editable. This is especially important as the answers a volunteer has provided might change between the time of enrollment and the start of the conference – which can be multiple months.

Allow for modification of submitted forms

28. **Provide feedback about the state of enrollment and conference** As long as unenrolling is allowed by the SV Chair, a student volunteer should be able to do so without further consequences. The application should also provide feedback about the current status of enrollment to the user. This could additionally also tell the user about the position on

Show state of enrollment and conference

the waitlist if needed. To be able to understand when enrolling and unenrolling is possible, the application will have to precisely provide the state each conference is in.

3.4.3 Accepting Volunteers and Lottery

General aspects

Lottery is random with exception of SVs on the waitlist

29. **Provide lottery algorithm** The lottery (see 4.4.3 “Lottery”) is an algorithm that is used to automatically change the state of volunteers from the initial "enrolled" to "accepted". The algorithm should be, with exception of SVs on the waitlist, completely random. This means that all volunteers in the initial state of "enrolled" have the same chance to be accepted. No questions from any enrollment form are taken into consideration. Any volunteer who could not be accepted due to the limited slots for volunteers of each conference should be put on the waitlist. The order on the waitlist should be the same random order the algorithm calculated when started. Whenever the algorithm is run, volunteers from the waitlist will be accepted first before randomly accepting new volunteers that are in the state "enrolled". On the first run of the lottery, the waitlist has to be empty.

SV Chairs

Be able to exclude volunteers from the conference

30. **Exclude volunteers from the lottery on special occasion** Some student volunteers might have reported themselves as not being able to participate while still being enrolled. The application will have to provide a way to manually drop enrolled users to not participate in the lottery. This could either be done by removing the association altogether or by setting the association to a predefined state ("dropped").

31. **Accept volunteers prior to the lottery** Before the lottery is run for the first time, it should be possible to accept some volunteers based on some criteria. These criteria should be able to be expressed by weighting the enrollment form or by filtering for SVs by any of the available attributes of the user.

Manually accept volunteers

32. **Detach lottery from enrollment questions** To make the algorithm of the lottery more transparent, it should only rely on randomness rather than on any of the answers of a given enrollment form. How this algorithm works will have to be explained in a way that is available within or through the application.

Make lottery unbiased

Student volunteers and Day Captains

33. **Explain the lottery transparently** We learned from the interviews and the survey that one major issue with the lottery of the previous version of CHISV is its transparency. This could be improved by simplifying the algorithm to not take any enrollment form data into consideration. This could also mean cutting back on the features and questions the algorithm evaluates.

    Being able to comprehend why a student volunteer was accepted or not would greatly improve the trust in the system and overall make the application feel more robust. This is why the lottery algorithm should be reconstructed and precisely explained such that everyone interacting with the system can develop a mental modal of what is happening in the background when the lottery is run.

Evaluate the lottery algorithm in terms of transparency to the users

3.4.4 Task Bidding

General aspects

34. **Provide filters and sorting for task bidding** Task bidding is one of the major features of CHISV. It requires the SV to make complex decisions to create a suitable schedule. Student volunteers have

Allow for filtering and sorting before bidding to obtain all the required hours while evaluating for themselves if they like to apply for a specific task. Tasks might conflict with other events or other tasks.

Bid on multiple tasks

35. **Multiple bids should be able to be placed with one click** Student volunteers should be able to create a selection of tasks by one or more days, a timespan as well as by specifying the name or location. It should be possible to bid on the resulting collection of tasks by placing a single bid that will be applied to all the filtered tasks.

Provide option to express a preference for a task

36. **Allow for preferences for task bidding** The application should provide a way to have volunteers bid on tasks with a specific preference. These preferences need to also include a way to express unavailability at the task's time. Apart from that, there should be three additional levels of preference. Overall the application may provide these four preferences, which need to be pickable per task:

    - Unavailable
    - Low
    - Medium
    - High

SV Chairs

Give SV Chairs insight to volunteer's bids

37. **SV Chairs have to be able to inspect a volunteer's bids** The application should provide a way to check on all bids of an SV. This is important to understand why tasks have or have not been assigned by the auction. There should also be a report that gives an overview of all SVs and their placed bids.

Student volunteers and Day Captains

Show tasks in calendar

38. **Present assigned tasks in a calendar** To furthermore assist the SV while making the task selection, the application should incorporate some sort of calendar. Seeing how tasks are aligned to one another

could help the SVs to avoid collisions and help them to get a better understanding of the duration of a task.

39. **Find similar tasks and filter by additional input** Student volunteers also told us that they like to be able to find similar tasks in the system. It should also be possible to limit a given selection down by the day and time. This could for example help the SVs to only bid on tasks in a specific interval, which is giving them more flexibility in planning.

Allow bidding for time interval

40. **Make the interface mobile-friendly** The interface of the previous version of CHISV was not optimized for any mobile form factor. Thus, we got a lot of feedback pointing out the urgent need for an adaptive interface – especially for task bidding. The volunteers are often using their smartphones to bid on tasks. This has to be considered when laying out the interface components of the application.

Interface for bidding has to adapt to various form factors

41. **Responsive interface, which accounts for errors** When the user bids on tasks, the interface has to provide immediate feedback. Due to the environment, the connection to the application can be disrupted or drastically slowed down during bidding. The application has to present a consistent and up-to-date representation of a bid that was placed.

Account for connection disruption

42. **Always show the preference of the placed bid** After the auction has been run and tasks have been assigned, SVs should be able to still see their submitted bids regardless if they were successful or unsuccessful. This can help when understanding why they have or have not been assigned. Additionally, some student volunteers like to refer back to their choice in retrospect.

Always show placed bids

3.4.5 Task Assignment and Auction

General aspects

Algorithm to automatically fill tasks

43. **Provide auction algorithm and options for manual assignments and check-in/out** The application should provide an interface to assign tasks to volunteers. For each potential SV, it should show a small summary of the number of bids, hours, and the selected task's bid preference. Any assignment should be instantly visible to the volunteer. Another option to assign volunteers to tasks should be the auction (see 4.4.4 "Auction"). That is an algorithm that will automatically assign tasks to volunteers based on preference, completed hours, and task priorities. Just like the lottery algorithm, the auction's algorithm has to be explained in detail in a separate section of the website.

Explain the auction algorithm

SV Chairs

Fill important tasks first

44. **Fill tasks with high priority first** Whenever tasks get filled by the auction, they should be filled in descending order of priority. This will ensure that tasks that are critical to the conference can be filled more securely.

Help create an even distribution of workload among all volunteers

45. **Balance assigned hours among all volunteers** Assigning tasks to SVs will account the task's hours for the SV when the task has been marked as done. Any task assignment, manual, or automatic, has to account for the hours the SV has already done. It is important to ensure that on average all SVs work the same amount of hours.

Display summary of the auction

46. **Feedback during and after the auction** Feedback is also inevitable for the auction, which automatically assigns tasks to SVs. This algorithm should also report any tasks that could not be automatically filled and also provide feedback while it is running to give the SV Chairs a feeling for how good tasks are being filled.

47. **Notes on assignments and SVs** Another important feature is the ability to provide feedback via a note on any volunteer's assignment to explain manually added hours or report the absence of the volunteer. These notes have to be manageable for SV Chairs as well as for Day Captains. Additionally, notes could also be extended to cover volunteers.

Notes for volunteers and assignments

Student volunteers and Day Captains

48. **Self-check-in/out** Our survey showed the desire for a function where student volunteers can check themselves in and out of their assignments. However, this request conflicted with the usual way how assignments are being tracked. This might thus only be an optional feature that has to be explicitly enabled by the SV Chair. Once enabled the student volunteer can change the state of associated assignments. Since this function will mostly be directly used by the volunteer before and after the task, the interface has to account for the specific environment and the device's form factor.

Evalute a method for self-check-in/out

3.4.6 Notifying Volunteers

General aspects

49. **Incorporate a messaging center such that all sent messages can be listed** Staying in contact with student volunteers is crucial. This is especially true for the time before the conference has begun. To inform the volunteers about important steps they have to take or to provide additional information, the system should feature a section where messages can be sent to the volunteers. Apart from any external channels where the notification will be delivered, it would be beneficial to also keep a copy of every message. This would enable all users to lookup messages even when they have missed them on other channels. They could always rely on the notification section of the application to get all the important messages.

Provide central place for all messages

SV Chairs

Enable SV Chairs to choose pre-composed messages

50. **Provide templates** Messages should be able to get composed based on templates. The templates are stored within the system's database and have to be manageable by the SV Chairs. Additionally, text building blocks could simplify the creation of messages. These could also contain placeholders, for instance, to automatically fill in the correct year and location based on the conference.

Build the application to support multiple channels for delivery

51. **Provide additional ways to reach out to SVs** The system should not only rely on e-mail messages to reach the user. Having alternative channels where users can be contacted could be beneficial. This could, for example, be a notification center that holds all messages that have been sent by the notification system.

52. **Show SV Chairs which notifications a volunteer received** Every conference associated notification that has been sent to the volunteer should be visible to the SV Chair to ensure the SV got the message.

Have groups with multiple recipients

53. **Have groups of recipients** The system should provide predefined groups of recipients such that a user would not have to select all recipients of a group manually. There should be groups for the different enrollment states of volunteers as well as internal groups, for example, to reach all Day Captains. It could later also come in handy when the interface enables the user to fill the recipient list with users from other parts of the application.

Deliver notifications to users who are not known to the system

54. **Support plain e-mail addresses** SV Chairs should be able to append plain e-mail addresses to the list of recipients. The message should be delivered just like all other messages with the difference that no copy will be kept by the notification system (as there is no associated user).

Chapter 4

CHISV

4.1 Overview

CHISV is open-source with its code residing on GitHub

In this chapter, we will take a look at CHISV, our reimplementation of the CHISV we looked at earlier as part of the “Related Work”. We built CHISV in multiple feedback cycles where we always asked our users (SV Chairs and SVs) about the functionality and feature first and then later on re-evaluated the outcome with them (see 3 “Requirements Analysis”). During these cycles, we went over 190 distinct features, requirements, and bugs. In the end all requested functionality was in a state such that all parties involved were happy with them. Our development started in February 2019. At the time of this writing CHISV is at version 1.0.7 and has undergone multiple automatic security audits. In CHISV’s GitHub repository[1] everyone can track all 664 commits of CHISV development history and easily contribute to the project. We think open-sourcing the entire application and having its code accessible to everyone is very important for transparency and maintainability in the long run. If the reference to the repository above is no longer valid an updated location can be requested from the administrators[2].

[1]https://github.com/dwhoop55/chisv
[2]https://hci.rwth-aachen.de/contact

In this chapter we will use the notation for referencing to requirements from the chapter 3 "Requirements Analysis" as explained in the "Conventions".

Identifying previous CHISV's shortcomings

To build the successor of CHISV (see 2.1 "Previous Version of CHISV") we first evaluated its shortcomings related to the software foundation it is built on. We found that some major issues are related to the maintainability and the hardware it runs on. Furthermore, due to the need for some customizations, we quickly learned how important a well maintained software ecosystem will be. This also includes the language the application is written in. Another aspect we will have to consider is the scalability and availability at all times (#4).

For our search of a well-suited software ecosystem we were quickly able to identify the criteria we think are most important:

1. Maintainability
2. Written in a widely-used programming language
3. Built with a renowned framework

We will now have a look at our choices for the back-end and front-end application, and why using these technologies will not only benefit our reimplementation of CHISV but also tackle issues that might arise in the future as requirements may change (#3).

4.2 Back End

Node.js and PHP are often used for backend applications

When looking at the landscape of server-side frameworks and programming languages one does quickly see that two languages stand out: PHP and JavaScript (Node.js). There are several frameworks available built with both languages.

Node.js has seen a strong uptake in recent years due to its performance (Lei et al. [2014]) and language, which could also be used for web front-end applications (Tilkov and Vinoski [2010]). Having to switch between multiple languages can be difficult at times. Using one language for the server-side application as well as for the client application (front end) can overcome this issue. Lei et al. [2014] also found that Node.js is especially suited for I/O intensive back ends. Given their numbers, we see that the performance increase in choosing Node.js over PHP is negligible in our scenario.

Node.js is fast and performant

Maintainability

Looking into frameworks built using PHP we find various solutions, like CakePHP, Yii, Symfony or Laravel. We found that Laravel is especially suited. Reconsidering our three crucial points (see page 58), especially maintainability, we see that we have to be able to rely on good documentation. Furthermore, this also includes good integration with other frameworks or plugins. We found that Laravel has great documentation and does also provide good integration for the features we are especially targeting. Given that Laravel follows a very strict principle of how code is organized and how entities interface with each other, we find that this framework gives us a solid foundation that can be reliably maintained in the long run.

Laravel is best suited for our long-term support requirements

Laravel:
Laravel is a free and open-source PHP framework for creating web applications. It's intended for the development of applications that follow the model-view-controller (MVC) design pattern. Laravel is using many of the components of Symfony and provides a modular packaging system to extend functionality in various ways without shipping with too much overhead for fresh installations. Laravel provides different drivers for relational databases and helps with syntactic sugar and wrappers to streamline development with PHP.

Definition: *Laravel*

Written in a widely-used programming language

PHP is the most commonly used scripting language on the internet

We stated that we want to use a framework that is written in a widely-used programming language. Laravel is built with PHP. Recently Yadav et al. [2019] found that PHP is still the most commonly used scripting language (82%) on the internet. Tatroe and MacIntyre [2020] found a similarly high number of 79% for March 2019. We think this is an indicator of it being around for some more years. Interestingly about 70% of the installations are using an old version of PHP 5. PHP itself has continued to improve syntax and functionality with the releases of PHP 7.2+. We are requiring a minimal version of PHP 7.2 for CHISV.

Built with a renowned framework

Large community

Laravel has a very large community (laravel.io) and there are numerous places to find material to study (Laracasts, Udemy). Additionally, Laracon is the related conference for Laravel developers. With all that we can say that Laravel is a renowned framework (#1). It is used in numerous business-critical environments and has developed over the years to a very capable and highly stable platform for server-side application processing. We noticed there are also many job offerings especially targeting developers with expertise in this framework. Laravel's ecosystem has many first-party extensions and services, which, for example, help with hosting, authentication, payment processing, or real-time applications.

Has all required functionality already included

Laravel utilizes many other smaller frameworks to provide all its functionality. This includes solutions for job queues, mailing, notification handling, authentication, and authorization. Furthermore, with additional authentication extensions, we have the ability to give access to third-party applications easily. Laravel provides all the small building blocks a developer would need to build various applications. While not having to use and know about them all, they are available should requirements change. Given that these tools and features are already integrated into the framework their compatibility and stability are far better than if one would have to collect different frameworks and extensions manually.

4.2.1 Database

Models built with Eloquent ORM

Most of our data will be stored and fetched from a central database. Laravel uses a highly capable database model, called Eloquent ORM, which makes it easy to interact with models in an MVC environment. We built our application on top of a MySQL database. However, all of the code we wrote is database independent. Any database driver available in Laravel would be able to store and retrieve our models. This makes it possible to move the entire application to a different hosting facility in the future (#2).

Database independent

We embraced the way how things are done in Laravel. This means we always tried to go with the framework as long as possible for any functionality. This can, for example, be seen in database handling. We wrote database-independent migrations[3] which make it possible to spin up the application on any other Laravel supported database without any modifications. Laravel supports all PHP Data Objects (PDO) compatible databases, like MySQL, PostgreSQL, SQLite, and SQL Server (see Documentation).

Session handling with multiple instances

We are also using the database for session handling for logged in users and as our queue driver. This enables CHISV to scale-out[4] such that multiple CHISV instances can be run simultaneously as long as they are connected to the same database. Traditionally one would use a load balancer (e.g. HAProxy or Nginx) to improve the overall reliability and performance of the application by dis-

[3]A database migration is a small set of instructions to create, alter, or delete tables in various database implementations.

[4]Scaling-out (in contrast to scaling-up) describes the process of providing additional and new instances to handle an increase in demand.

tributing the load among multiple instances (nodes) of the same application [Yusuf and Nuha, 2018, Pramono et al., 2018]. For this approach to function we had to make sure that sessions are either accessible to all nodes or design the application such that it can work in a stateless manner (#4). One could for example use JWT[5], a concept we will later pick up again when we look at the third-party client integration.

4.2.2 Job Queue

Prevent mutual execution of jobs

Laravel uses a queue to schedule jobs and have them be processed by a queue worker. The queue worker is a separate process, waiting for scheduled jobs in the queue by continuously checking for updates. Usually, there are multiple queue workers per installation (node). Before a job is executed it is locked to prevent mutual execution. There are multiple drivers available for the queue's store. We chose the database driver to persist our queue. Hence, any scheduled job is accessible to any node. Due to the job being locked before execution we ensure that only one queue worker can process a job at a time.

Thin wrapper around Laravel's job model

Our long-running jobs (lottery, auction) are also processed by one of the queue workers. Laravel provides a very simple implementation of jobs, having them vanish when succeeded. For better UX, we also wanted the jobs to produce feedback while running, giving the user an estimate of how long the job will continue to run. Thus, to be able to use Laravel's job queues we had to implement a thin layer on top of the original job model. We go into more detail in this in 4.4.2 "Job Extension". This allows us to have a very simple interface for developers to implement their functionality, yet also provide rich feedback to the user interface while a task is running.

[5]JSON Web Token are stateless since they rely on a cryptographic signature of the server.

4.2.3 Authentication

Laravel's authentication scaffolding only provides Cookie-based authentication

Laravel provides great scaffolding for any authentication and authorization related tasks (see Documentation). We used this to generate all required controllers and to ensure that the `User` model can authenticate with a password against the login endpoint. Using this approach configures the Laravel installation to use Cookies for authentication. Cookies are session-based and not suited for third-party applications. Since our initial aim, to keep the system maintainable, also includes having the application be open for third-party clients, we felt the need to also provide token-based authentication.

Add token-based authentication with Laravel Passport

To implement this requirement, we opted for Laravel Passport[6]. Passport is a first-party extension and developed together with Laravel. It provides OAuth authentication and authorization. With the help of this package, we were then able to also provide token-based (JWT) authentication and authorization to our application (#3).

Authentication handled internally by Laravel independent of the method

To interface with the OAuth API of CHISV, one would first have to request a `client_id` and `client_secret` from the administrators. If the application does not require an OAuth compliant endpoint, authorization can also be done via a different endpoint that does not require the developer to own a client_id and `client_secret`. We provide a small example for authentication and API access in 7.1.1 "Authentication".

Equal API endpoints for all clients

After successful authentication, a client may then interact with any of the available API endpoints (see 7.1.2 "Endpoints") the same way our first-party web application does. While a single-page application (SPA) is usually also built to make use of token-based authentication, we used Cookie-based authentication as this allowed us to overcome certain security-related attack vectors. Nevertheless, as all authentication is handled by Laravel be-

[6]It should be noted that Laravel Airlock/Sanctum was not released when we decided to use Passport. Laravel Airlock/Sanctum would minimize the footprint for token-based authentication and remove all the logic of OAuth that is not required for simple client applications.

fore the request hits any controller, the implementation does not have to account for any Cookie- or token-based authentication. Whenever a request is authenticated, Laravel will prepare a `User` object of the currently authorized user.

4.2.4 Model Relations

ER model omits non-essential attributes and relations

Laravel makes it extremely easy to link model entities. Thus, we have specified numerous bidirectional relations between models. Expressing all of them in an entity-relationship model would overfill any figure. We made two simplifications: First, we will focus on a smaller section of the application at a time. Second, we have omitted any non-essential attributes and bidirectional relations. For example, the `User` model used to have more attributes than we express in figure 4.1. Since these are simple attributes and not contributing to any relation between models, we found it sufficient to only cover attributes that play an important role in connection with other models.

Uniform vs polymorphic relationships

In our application we made use of simple uniform relations, such as between a `User` and a `Task` model but also connected some models via polymorph relationships. This can, for example, be seen in the relation between a `User` and a `Note`. Users and assignments (`Assignment`) can have many notes. Thus, the `Note` model has to have a polymorphic relation to the model it is for. In Laravel this is accomplished by providing the model id (`for_id`) and the model class (`for_type`). In this example the `for_type` of a note would either be `App\User` or `App\Assignment`. Using this approach it is very easy to, later on, extend notes to be pointing to any desired model. Let's say we would have to extend CHISV to also allow for notes on tasks then this would require no structural change in the database nor the code.

Images have polymorphic owners

Our `Image` model has also a polymorphic relation to its owner. Images can be owned by a user or a conference. Furthermore, conferences can have two different images, one for the artwork and one for the icon. To accomplish the same relation, one could also have created something like a `UserImage` and `ConferenceImage`. We found this approach to introduce a lot of overhead since we would have to duplicate a lot of our logic for each model. As we are strongly focused on the maintainability of the entire application we always tried to reuse our models as much as possible.

State model represents every state needed in the application

Take the `State` model for instance. It expresses any state of multiple models and relations. It's used by the `Conference`, `Permission`, `Job`, `Bid` and `Assignment` models. Every state object combines a name with a short description. The description explains what the state means for the object it is attached to. Since we will always return the associated state object with any resource we send to the front-end application, we can improve the experience of the application drastically. Whenever a user is not sure about the meaning of a state, the description will try to help clear any confusion (#16). Since the structure of how such a state object will look like is always the same, implementing front-end components is getting easier. To see how the state object is playing an important role in the user-permission relation, we will now take a deeper look into the relations between a conference, it's users and the associated permissions, which bind those users to the conference.

Conference, Users, and Permissions

Permission objects associate users to conferences

Figure 4.1 shows our conference-user relations in CHISV. This figure focuses on the conference and user-related aspects but leaves out any other relations (e.g. the `Task-Bid` relation). A conference may have multiple users. This relation is defined by a `Permission` object, which acts as the mapping model between users and conferences. It is an integral part of CHISV. Any ability and association of a user is expressed by a `Permission` object (see figure 4.1 "Permission" at the lower-left corner). As explained, this object maps users with certain roles to conferences. Some roles go together with a state.

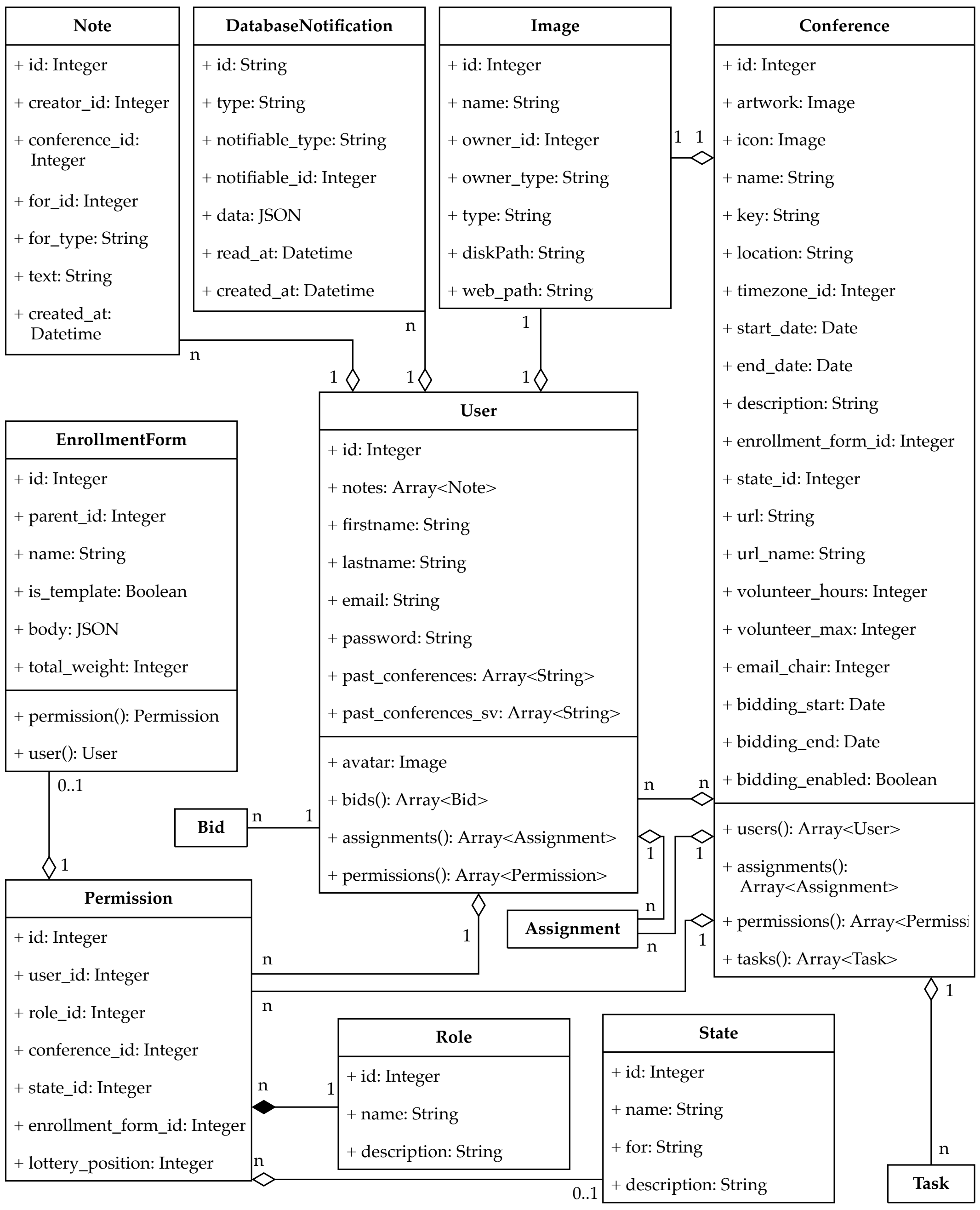


Figure 4.1: Relations between conferences, users, permissions, enrollment forms, notes, and images. We see that a user can have multiple notes, notifications, and an avatar. A User does also have multiple enrollment forms, permissions, bids, and assignments. A permission has a specific role and is in a certain state. A conference can have multiple users (associated by a permission), permissions, tasks, and assignments. Like the permission model, a conference is in a certain state.

Based on our requirements (#7), we have defined four roles:

- **Administrator** – The user can modify any aspect of any resource and can grant and revoke any permission.
- **SV Chair** – The user is a member of the SV Chair for a specific conference. This role allows granting the "Day Captain" permission and to modify any aspect and resource about the conference it is bound to.
- **Day Captain** – This role is usually granted only for a few days and allows the user to modify assignments of SVs and to edit tasks. However, modifying an SV's enrollment state is not possible.
- **Student Volunteer** – The user is associated with the conference as an SV, can bid on tasks and check the calendar for any assignments.

To express an SV's current state of enrollment we also defined four states:

- **Enrolled** – The user is enrolled and waiting to be accepted, waitlisted, or dropped
- **Accepted** – The user is accepted to the conference as SV
- **Waitlisted** – The user is waiting to be accepted when other SVs are dropped
- **Dropped** – The user has been manually dropped from the conference

"SV Chair" and "Day Captain" only get assigned manually

When users register with CHISV they have no permission object at all. While the SV Chair role (2.) has to be manually granted by an administrator to a user, the SV role is created as soon as a user enrolls for a conference. A user with the SV Chair permission can (like the administrator) grant or revoke Day Captain permissions to or of a user. Figure 4.1 shows the relation of a role to a permission object. Any permission has to have a reference to a user (`user_id`) and a role (`role_id`). We left the relation to a state object optional (`state_id`) since only permissions with the "SV" role need to express a state. The same is true for the relation to a conference (`conference_id`) and the enrollment form (`enrollment_form_id`). A reference to the conference is not needed in the case of the "Administrator" role and Enrollment forms can only exist for permissions expressing the "SV" role.

For the user to be able to enroll for a conference, the conference has to be in a certain state. We have defined five states for a conference:

- **Planning** – The conference is invisible for users without an associated "SV Chair" or "Day Captain" role. This state allows for setting up the conference with details and images before it is opened for enrollment.
- **Enrollment** – The conference is open to the public for volunteers to enroll. During this phase, any SV can unenroll or edit the enrollment details.
- **Registration** – Volunteers can no longer enroll. SVs have been accepted, notified, and are now required to register for the conference.
- **Running** – The conference is taking place and tasks are given out to SVs.
- **Over** – The conference is over and has the same visibility as in the "Planning" state.

Enrollment forms can be modified

Bids (`Bid`) and assignments (`Assignment`) are, other then an enrollment form, directly bound to a user. They express their relation by an `user_id` attribute. The enrollment form is also associated with a specific user, which is done by using the permission object as a proxy. This way we can also ensure that a user has to have the "SV" role for a specific conference to be able to have an enrollment form (#22, see `enrollment_form_id` on Permission model). The enrollment form can be customized on a per conference basis (#23). Any SV who has enrolled can still edit the enrollment form as long as the conference is in the "Enrollment" state (#27).

All sent messages are visible in the internal messaging center

Users of CHISV have access to all notifications that have been sent through the system (#49). These are represented as `DatabaseNotifications`. A user may have an arbitrary amount of notifications – or none at all. Each notification is marked read when the owner reads it by setting `read_at` to the current UTC date (#52). The content of the notification is stored in the `data` attribute as a JSON string. We use this `data` attribute to render an optimized version of the notification based on the channel it is delivered over.

Store subject, greeting, and valediction separated from the actual content

To do this, we have split the information into subject, greeting, valediction, and a message elements array. This array can consist of multiple markdown message pieces and one call to action button. Laravel can send special call to action e-mails. It ensures that the button with the primary action is always correctly displayed, even in text-only e-mails. Since we have separated the notification's message data into multiple entities, messages can be displayed more efficiently (see figure 4.18).

Seperating the content of a notification helps when working with different channels

CHISV can deliver notifications as e-mails and through the internal messaging system (#51, #54). However, Laravel also supports delivering messages over additional channels, like Slack, Telegram, or SMS. When we consider SMS, for instance, we see a need to limit the transmitted characters to a minimum. In this example, we could only send the call to action and prevent any unnecessary content (greeting, valediction). Having this additional information (like URLs) already stored in a computer-readable format (rather than plain text), we can also make use of additional API features of Slack or Telegram when delivering messages via them. We could, for instance, render a button and improve the user experience (UX) by not making the recipients scroll through a long text with non-clickable URLs.

Notification system is extendable to allow notifying any kind of ressource

We implemented the default notification handling, which is provided by Laravel (see Documentation). This scaffolds the required table, model (`DatabaseNotification`) and controller. However, this enables any model that uses the `Notifiable` trait to own and receive notifications. This helps with our goal of keeping the CHISV system open for changes and new requirements in the future. One could, for example, add notifications for groups of users (e.g. Day Captains), rather than having to send notifications to each group member manually. We will go into more detail about the notification system in 4.4.7 "Notifications and Reports".

Polymorphic note system for assignments and users

In our second iteration (see 3.3 "Second Iteration") we used the feedback we got from SV Chairs and other user groups of the application. In particular, our attention was drawn to the ability to store short notes associated with assignments. This feature was requested by the SV Chairs of CHI 2020 during the interviews we did (#47). As we explained in 4.2.4 "Model Relations", we used a notes model with a polymorphic relation to its owner to implement this requirement. Since we were also focused on extendability for future requirements, we took the chance and extended this requirement to also cover notes on SVs. While we will focus on notes on assignments later in 4.2.4 "Tasks, Bids, and Assignments", we will now take a look at how we designed the notes model to also cover an additional need for data privacy when associated to the user model.

Need to limit access to notes

Notes on SVs (which are `User` models) can be created, viewed, and deleted by the SV Chairs and Day Captains of a conference. We quickly saw the need to limit access to a specific note to the conference in which context it was created. We cannot prevent an SV Chair member or Day Captain to create a note containing negative impressions about an SV. We can, however, ensure that this information is contained and only visible for SV Chairs and Day Captains of the conference in which context it was posted in. As the SV can enroll for multiple conferences over time, we think it is best to not show internal notes that were created associated with an SV at other conferences. This might negatively affect the SV selection process of conferences in the future. Thus, we saw the need to associate any note to a conference.

Using an additional relation to the conference for enhanced privacy

A note that is bound to a specific assignment can be easily mapped to a conference. The assignment has a relation to a task, which again belongs to a conference. We check if the accessing user is an SV Chair member or Day Captain for the conference and grant access. A note that is associated with a user had no further association with a conference in the early design we approached. We introduced a relation to a conference, which is expressed via the `conference_id` on the `Note` model. This allows us to only show notes on the "SVs" view (see figure 4.3) which have been created in the context of the currently selected conference. For notes on assignments we won’t need this additional relation, thus leave it unset.

Tasks, Bids, and Assignments

Tasks are bound to conferences

Apart from conferences and users one of CHISV’s integral features is the ability to manage tasks, bids, and associated assignments. Each task is bound to a conference via the `conference_id` attribute (see figure 4.2). While a conference may have many tasks, one task can only be assigned to one conference. Tasks themselves have relations to bids, assignments, and the associated users. The users are not directly connected to the task but rather through the assignment model. Since a task may have multiple assignments there can be multiple users assigned to one task.

Bids have five states

There may also be multiple `Bid` objects associated to one task and one user via the `task_id` and `user_id` attributes on the `Bid` model. Bids and assignments are in one specific state.

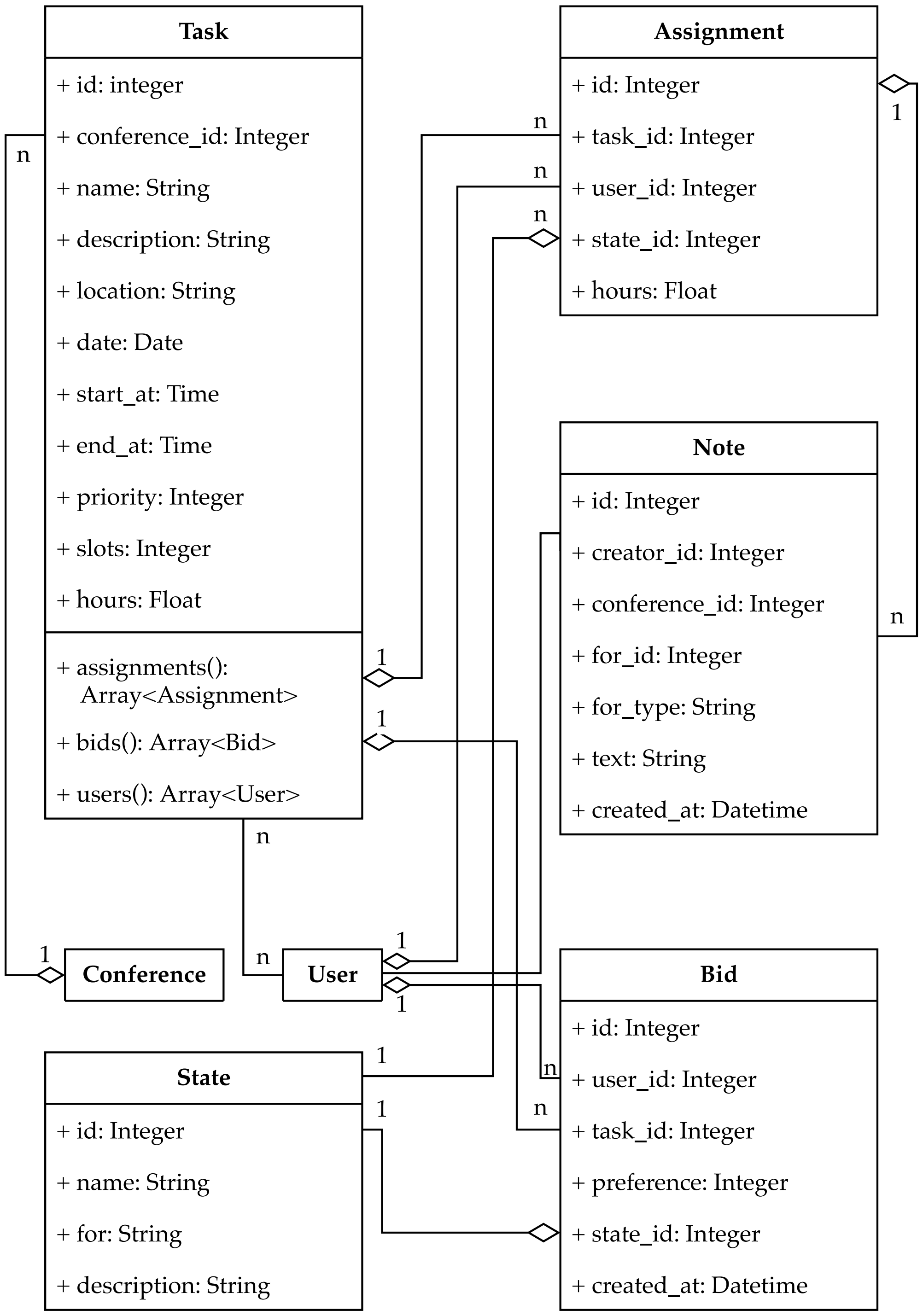


Figure 4.2: Relationships between tasks, assignments, and bids. Each task is bound to a conference via the `conference_id`. A task may have multiple assignments and bids. Both are bound to the task via the `task_id` property. A bid and an assignment are furthermore also bound to a user via the `user_id` attribute. Both models are in a certain state, associated via the `state_id` value. An assignment may have notes. These are, on the one hand, linked to the assignment via the `for_id` and, on the other hand, to the creator (`creator_id`). The `Note` model also features a `conference_id` attribute, which is used to limit the visibility of notes on users for SV Chairs and Day Captains.

We have defined these five states for bids:

- **Placed** – The bid was submitted by the volunteer and will be evaluated by the auction.
- **Successful** – The bid won the auction. An assignment was created.
- **Unsuccessful** – The bid did not win the auction as all slots for the associated task are already filled.
- **Conflict** – The bid was skipped due to a time conflict with another task.
- **Unavailable** – The bid was skipped since it signals SV's unavailability.

Assignments can be in one of three states

For assignments we adopted the states from the previous version of CHISV (see 2.1.1 "Assignments"):

- **Assigned** – The assignment is scheduled. The SV is currently not working on the task.
- **Checked-In** – The SV is working on the task at the moment.
- **Done** – The task has been completed by the SV.

State helps SVs to understand the situation better

These states are expressed by pointing to the desired state via the `state_id` property on the assignment or bid. Keeping track of the states allows the volunteer to precisely understand how many hours have already been completed and why a certain bid might not have yielded an assignment. Improving transparency and explaining the algorithms was one of the requirements of SV Chairs and volunteers (see #33, #42 and #46).

Priority of tasks

Assignments can be created manually or automatically by the auction (see 4.4.4 "Auction", #43). The auction will use two key attributes for each task: `Priority` and `slots`. The `priority` can be one of three options, which are internally stored as a number: "Low" (1), "medium"

(2) or "high" (3). A priority of "low" would be chosen for tasks that should be filled by the auction but are not as important as tasks with preference "medium" or "high". Since tasks with "high" (then "medium") priority are processed before any "low" priority task is evaluated (#44), it can happen that these tasks cannot be filled with volunteers. This means that all available volunteers have already been assigned to tasks with higher priorities. This is very likely to happen when tasks with a higher priority have many slots.

Slots express how many volunteers are needed for the task

The amount of volunteers needed for a task is expressed by the `slots` attribute and stored as a number on the task model. This number expresses how many volunteers the SV Chair thinks are required for this task. While a task may be manually under or overfilled, the auction will respect the precise value. An option for expressing a minimum, desired, and maximum amount of volunteers (like SIGCSE's application, see 2.2.4 "Scheduler") was not implemented and may be re-evaluated for future development with an adjusted set of requirements.

Assignments can have notes

We looked at notes on users earlier in this chapter. Just like users, assignments may also have notes (see figure 4.2). We incorporated this feature (#47) in the second iteration. It was part of the feedback from SV Chairs of CHI 2020 (see 3.3 "Second Iteration"). SV Chairs and Day Captains can create notes on assignments on the "Assignments" view (see figure 4.3). These will be shown on the detail view of the SV on the "SVs" view and are only visible to SV Chairs and Day Captains. Another hint to the note will also be present on the assignment itself on the "Assignments" view.

Using notes on assignments to give an additional explanation of a change

Usually, Day Captains use the note-taking system to let the SV Chairs know about some specific change concerning the assignment. This can, for example, be that the SV did work for more hours than scheduled. While the hours can be easily adjusted on the same view, a note may help to understand why the change had to be introduced. While the `Note` model has a `conference_id` attribute to reference the conference on which this note is visible, this attribute is only used for notes on users.

Notes on assignments do not require the `conference_id` to be set since the conference can be referenced from the assignment-task-conference relation.

Summary of back-end logic

We now know about the user-conference relation and learned that it is expressed through a permission object, which binds together user and conferences with a certain role and state. Just as relevant as the structures we build to handle user-conference associations is the domain of task-conference management and assignment accounting. We saw how tasks are bound to conferences and how the relation of assignments and bids enable the application to keep track of which SV is assigned to which task and how many hours have been completed. We also saw how the note-taking system for assignments and volunteers enabled more efficient communication between Day Captains and SV Chairs.

With all this background knowledge we can now shift our attention to the front-end application. For most users interacting with the application, the front end "is" the application. They don't know about all these back-end structures, and little do they need to.

4.3 Front End

4.3.1 Frameworks

Laravel Blade returns static, pre-filled websites to the client

As we have seen in 4.2 "Back End" we are using Laravel for our back-end logic. Laravel also brings concepts to the table with which one can create user interfaces for the web. With Laravel, web pages are defined in plain Hypertext Markup Language (HTML) utilizing Blade templates (see Documentation). Each URL path (route) would usually be matched to a template view by a controller handling this specific route. It would then be the responsibility of the controller to prepare the Blade view and fetch any required data from the data store/s. Blade templates (views) provide much functionality one could hope to find for implementing large applications.

This includes branch logic (if/else) and markup for iterating through data structures (for each/while). Since each Blade template is also a valid PHP document, even more advanced logic could be inlined. Usually, this would be moved to the controller and only passed to the view.

Server-side rendering

It is important to notice that these views are generated and prepared server-sided. Any data that hits the client's device is already formatted (sorted if required) and will only be parsed and rendered by the browser. Without the use of any additional front-end framework, we would get a static website for any request we make without the need to sort, filter, or manipulate results in the client's browser with the use of JavaScript. The returned result only consists of plain HTML markup, which can be interpreted by any browser[7]

Returning server-side rendered websites requires the user to interact with the UI to induce changes

Laravel does not dictate which front-end framework the developer uses. In fact, none may be used at all. Given these techniques, one can build an application that requires no JavaScript on the client's end device. All Blade templates are filled with the desired HTML elements based on the request on the server-side. This approach, where we pre-render each view server-sided, is with respect to the current state of JavaScript and it's adoption in the web (Wirfs-Brock and Eich [2020], Eich [2005]) a rather conservative approach. It follows a request flow where data is retrieved only when the user interacts with the interface. While, over the years, we have seen a lot of improvement by enhancing the underlying components, like HTTP (Grigorik [2013], Stenberg [2014], Yamamoto et al. [2017]), the control flow remains the same.

We started development with Vue.js

When we started development on CHISV we were using Blade templates. We noticed in the very beginning that we would need to use a front-end framework that can handle changing form factors of end devices and provide generally improved responsiveness (#15, #41) to users interacting with the graphical user interface (GUI). Since Laravel and it's community provides great support for Vue.js (#1), we settled for Vue.js for our front-end

[7]We expect that the template does not contain any modern HTML5 or advanced CSS, which would again limit the compatibility.

framework. Our Laravel installation (pre 7.0) also came with the popular UI framework Bootstrap[8]. However, we opted to use a different UI framework, which we will present later.

Vue.js:
Vue.js is an open-source and community maintained model-view-viewmodel JavaScript framework. It helps with building user interfaces and single-page applications (SPA).

Definition: *Vue.js*

We see Vue.js's approach for reactively updating the Document Object Model (DOM) perfectly fitting our requirements for a reactive UI (#15). Furthermore, re-evaluating our three crucial points for choosing a back-end and front-end framework (see 4.1 "Overview") we noticed that Vue.js is one of the most popular front-end frameworks of the time, standing right next to Angular and React (Medium.com). At its core, Vue.js focuses on only keeping the UI in sync with the data model. Thus, it provides a very shallow learning curve, which helps with keeping the application up to date and maintainable.

Bulma:
Bulma is a free and open-source Cascading Style Sheets (CSS) framework based on Flexbox. It contains HTML and CSS based design templates for typography, forms, buttons, tables, grid systems, navigation, and other surface design elements. Bulma's appearance can be customized via SASS to create a distinct look.

Definition: *Bulma*

Bulma as CHISV's UI framework

With Vue.js we found a valuable framework to help with keeping our UI in sync with our data models. As we also intended to provide great support for different form factors, we picked a UI framework appropriate for this task. We chose Bulma (Documentation) as it is detached from any design trends, keeps the markup clean and gives us a lot of space to craft a distinct look for CHISV. Bulma

[8]A fresh Laravel 7.0+ installation will no longer pre-install Vue.js and Bootstrap.

does not dictate any JavaScript framework and only concentrates on providing the CSS markup to create a responsive UI with a modern look.

Binding Vue.js and Bulma with Buefy

As Bulma does not ship with any JavaScript, it integrates very well with Vue.js. One can use plain HTML and decorate the elements with Bulma directives. Since we are using Vue.js we can use a wrapper around Bulma, called Buefy (Documentation), to integrate its features into Vue.js and greatly improve responsibility.

Definition: *Buefy*

Buefy:
Buefy is a user interface component library created on top of Vue.js and Bulma. Buefy can be seen as the JavaScript layer for Bulma. It extends Bulma design by integrating reactivity and bindings to Vue.js while respecting both frameworks' design principles.

Buefy provides great table components with lots of space for customization

We chose Buefy as it provides a very feature-rich and responsive table user interface component. CHISV is mostly about organizing resources (like tasks, users, or assignments) inducing the need to handle multiple rows of data. We found that especially Buefy's approach of responsive and intuitive integration of table components was a perfect fit for CHISV. It can not only display, sort, and filter data that is present in the DOM but also asynchronously fetch more data from a back-end service. This became inevitable to handle a large set of tasks or SVs where we use pagination to only show and display a small subset of data for each page. Yet, the user can make the assumption that all selectable rows are available and already loaded – while in reality, they are fetched from the back end on request.

Persist UI state during page loads

Our initial approach towards the reimplementation of CHISV was based, as we presented earlier, on a request-driven approach where any major interaction with the UI would trigger a page load in the browser. While Vue.js applications can contain and hold a certain state and data when loaded and used in the client's browser, a page refresh or navigation triggers an entire page load. During this reload, the Vue.js application is stopped and then started again once the new page finished loading. Any data or state that may have been present on the previous page is lost as a result of the page load. To counter this issue, we integrated Vuex into our application to hold any data and UI state that we want to persist.

Vuex:
Vuex is a library bringing the state management pattern to Vue.js applications. It provides a centralized data store for all components of an application. As Vuex is a first-party plugin for Vue.js, it integrates with the official Vue.js development tools and allows the developer to inspect, backup, or restore certain states of an entire application.

Definition: *Vuex*

We equipped Vuex with the plugin "vuex-persistedstate" (Documentation) which persists and rehydrates the Vuex state store between page reloads. The Vuex store is stored as JSON in the browser's local storage. This allowed us to have a persistent appearance of UI components between any navigation, which was done by a user. This included search inputs, tab, and dropdown selections or a shortcut to the last viewed conference.

Persist Vuex store to browser's local storage

Earlier we have been talking about Laravel's Blade template system and how we initially used it for CHISV to render out our views. During the course of development, we noticed that to fulfill our requirement of accounting for weak or limited connectivity (#20), we were forced to redesign our request handling approach. From our interviews in 2019 (during CHI 2019, see 3.2.1 "Interviews") we got an insight into the available connectivity volunteers and SV Chair members might encounter when on-site.

Surveys help understand connectivity issues

While the connection might be good in most places on-site, we saw the need to also consider that the volunteer might have bad connectivity at the hotel or hostel. 70% of all 69 volunteers told us that they are bidding on tasks on hotel or hostel WiFi. Furthermore, we learned that 25.4% of all volunteers were also interacting with the bidding system while commuting (bus, train, or subway). Even

Connection quality might fluctuate

more, 38.8% noted that they were using the system while walking in an open area. With these numbers, we noticed the clear need to also design the application and how requests to the back end are made in a fail tolerant way. To implement this, we need to abandon the request-driven approach, which Blade's template system offers, and instead use an approach where we can use JavaScript code in the front end to update the UI accordingly. This would then also allow us to provide richer feedback in the case of a failed request.

Conversion to single-page application

To accomplish this, we reverted our Blade view to a bare minimum where one view would only return the Vue.js Javascript bundle. The entire front-end application would then run in the user's browser while already incorporating all views of the application. This approach is called single-page application or SPA.

Single-page application:
A single-page application (SPA) is a web application that interacts with the user's input by dynamically rewriting the active page rather than loading entire pages from a web server. As there are no requests necessary to fetch the structure (user interface) of the requested page, the application avoids interrupting the user experience between page load, creating a more responsive experience similar to a desktop application.

A SPA fetches only the desired data from a back end and dynamically adds it to the application's views. The page does never reload at any time, nor does the actual page in the browser change. Modern SPAs can, however, alter the browser's URL to make it look like a different page is being viewed. Overall this approach can minimize network load and provides numerous ways to more precisely control the user's experience.

Minimized network requests

As our application is now a SPA it only needs to talk to the back-end service to fetch new data. Rendering the according view and presenting it to the user is handled entirely by Vue.js and does not require any interaction with the back end. By this, we drastically limit the number of requests to the back end, while also minimizing the transferred data.

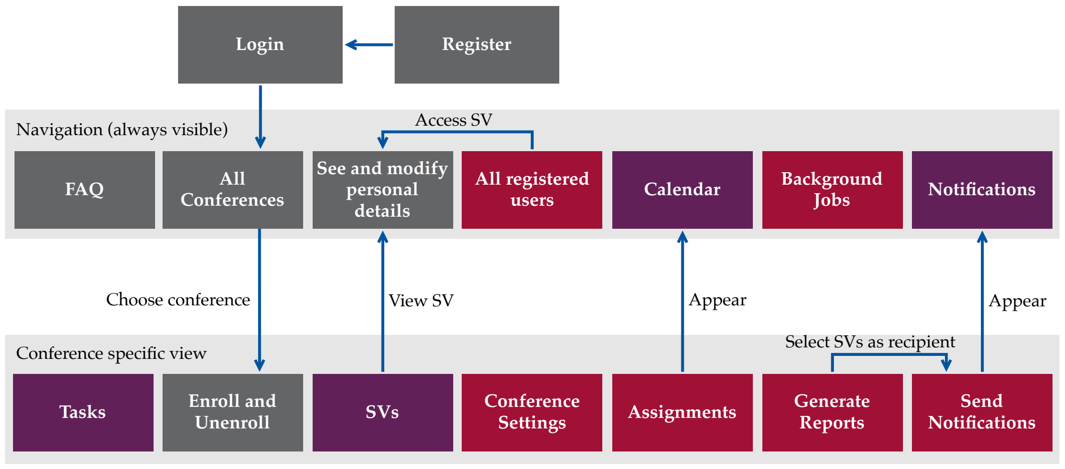


Figure 4.3: CHISV's user interface structure: After the users logged in, they are greeted with the conference selection. Selecting one specific conference will drop the user into the conference-specific view where one can interact with the resources of the conference. Grey boxes ■ denote views that can be accessed by any registered user. Bordeaux boxes 🟥 present views visible for SV Chairs and Day Captains. Violet boxes 🟪 mark the views that are shown to SVs.

Provides new ways to enhance UX

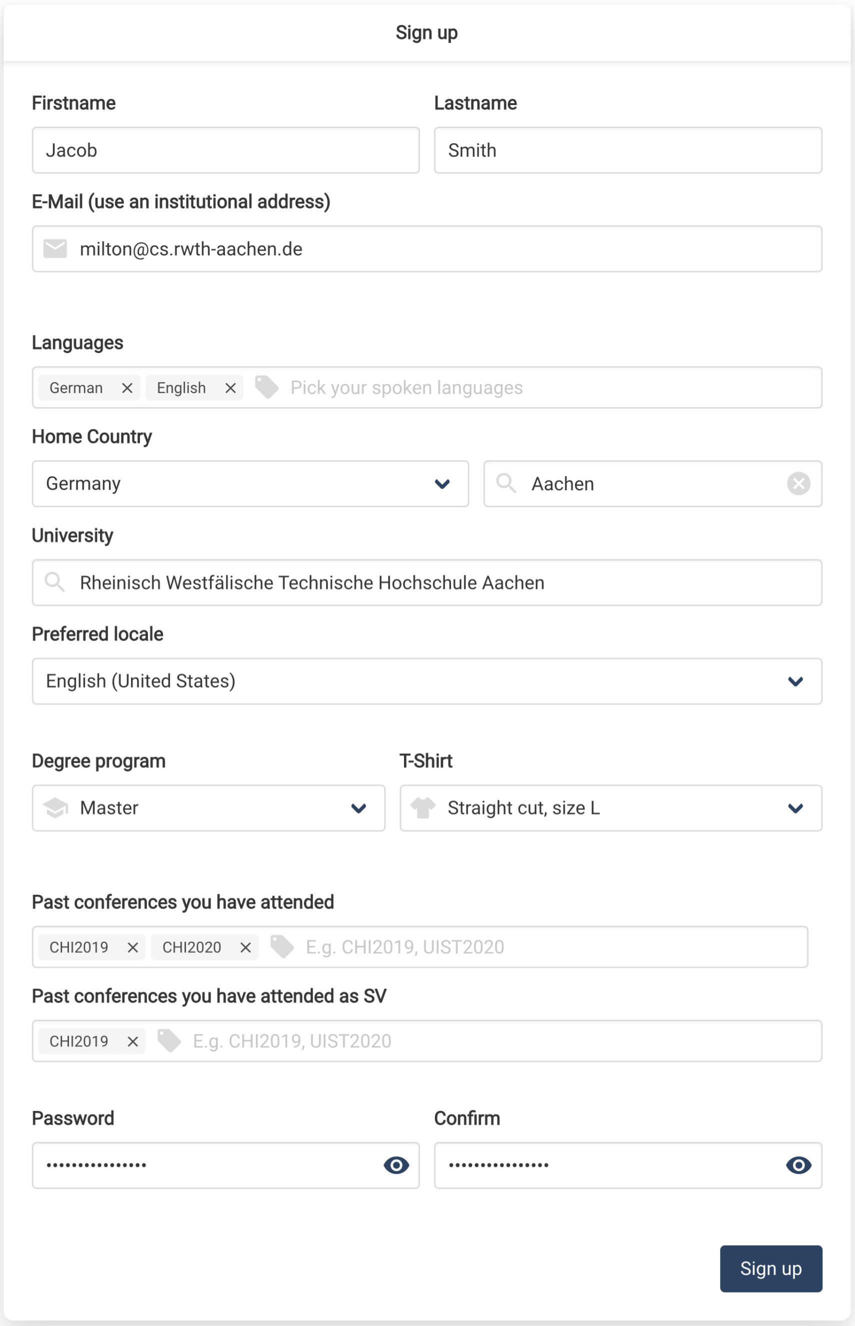
This helps us in case of weak connections (#20) and also enables us to provide a much better user experience. For instance, we can show distinct loading indicators before we actually proceed to load a view to the DOM and can also provide a much better explanation in case of a back-end timeout. As views won't have to be loaded from the back end, the entire application feels more responsive to the user. For example, a view can be shown even before any data is present. We can then use skeleton elements to visually hint where data is going to appear – without having even talked to the back end.

4.3.2 User Interface Structure

Register

Using CHISV requires an account

To be able to access the application, users have to register first (see figure 4.4). Without a valid login, no page, except the authentication and register pages, can be ac-


Sign up

Firstname
Jacob

Lastname
Smith

E-Mail (use an institutional address)
milton@cs.rwth-aachen.de

Languages
German × English × Pick your spoken languages

Home Country
Germany
Aachen

University
Rheinisch Westfälische Technische Hochschule Aachen

Preferred locale
English (United States)

Degree program
Master

T-Shirt
Straight cut, size L

Past conferences you have attended
CHI2019 × CHI2020 × E.g. CHI2019, UIST2020

Past conferences you have attended as SV
CHI2019 × E.g. CHI2019, UIST2020

Password
••••••••••••••••

Confirm
••••••••••••••••

Figure 4.4: New volunteers, which have no account yet, need to register and provide some personal details to use CHISV. We collect personal details like the full name and e-mail and are also asking for the spoken languages, the associated institution as well as the home country. To present dates in the user's appropriate format we collect the preferred locale. To get the SV Chairs the required details, we also ask for the T-Shirt, attended conferences, and the current degree program. Lastly, the user needs to set a password to use with the new user account at CHISV.

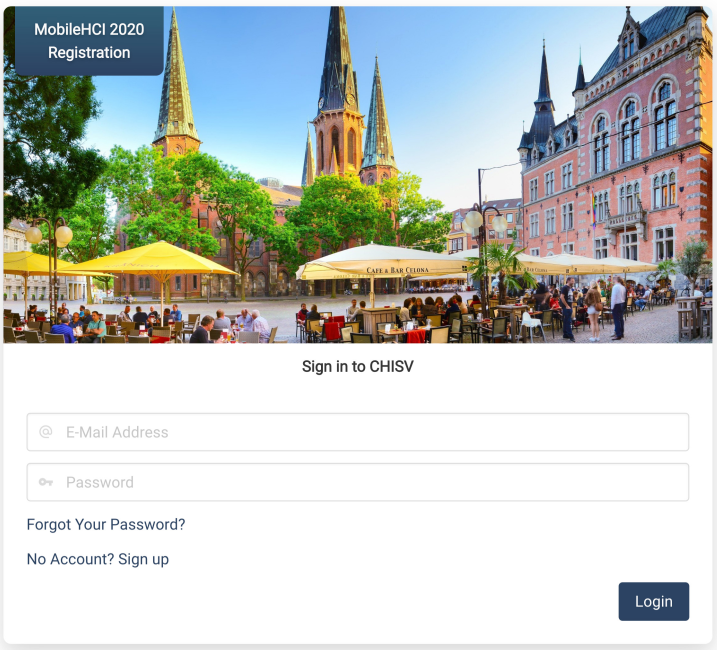


Figure 4.5: CHISV's Login: An image carousel shows currently open conferences the user can enroll for or bid on tasks.

cessed. The registration process collects some details about the user, which is required to create the account and make it uniquely identifiable. This info will later also be used by the SV Chairs to selectively accept volunteers with special criteria. Users can later always update the provided details at the personal profile (see "See and modify personal details" in figure 4.3).

Login

When users have created an account they can log in. CHISV's login page greets the users with an image carousel of currently open conferences (see figure 4.5). This way the users who have no account can quickly verify that they are on the correct website. After logging in, we use a Cookie to store the session such that our

CHISV uses Cookies for authentication to prevent leakage of the session

SPA can always make requests to the back end as the browser automatically appends it. This approach has the benefit that we are protected against Cross-site scripting (XSS) attacks as the JavaScript SPA cannot retrieve the Cookie and thus cannot leak it. We go into more detail on this in 4.4.1 “Cross-Site Scripting (XSS) and Cross-Site-Request-Forgery (CSRF) Mitigation”.

Some views and features are only visible and accessible for certain users

After the login, the user can always refer to the navigation bar at the very top of the page. From there the user may jump directly to different parts of the application (see "Navigation" in figure 4.3). Based on the user’s roles, different parts of the application are accessible. A student volunteer would, for instance, not be able to see the "Background Jobs" queue. Some views, like the "All Conferences" view, may be accessible for all users but will reveal less or more information based on the role the user has. An SV Chair member can, for example, see the own conference, which is in the "Planning" state. The same conference would not be shown to SVs even if they can access the "All Conferences" page.

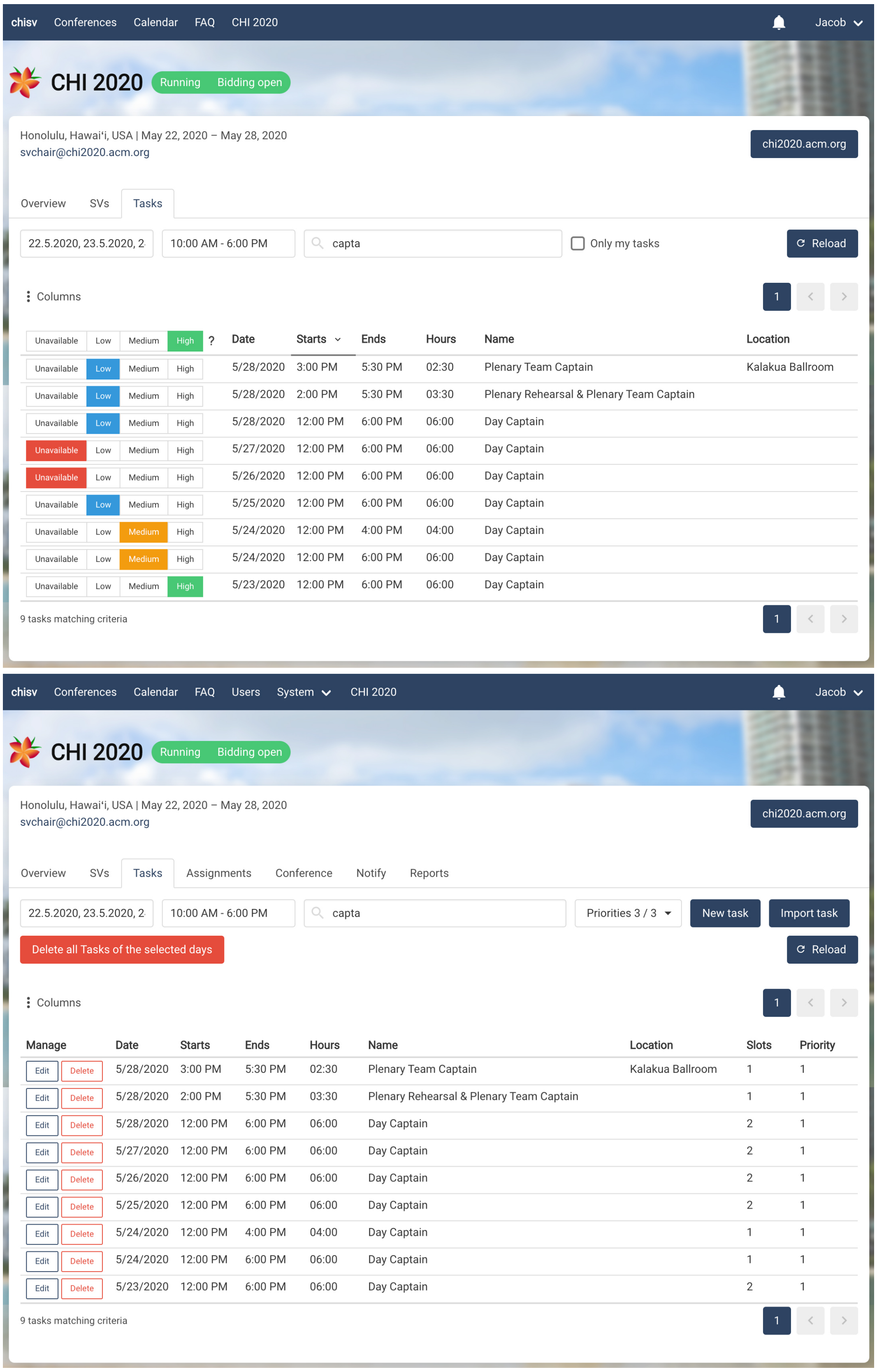
Navigation within a conference

Entering a conference by selecting it in the overview

After a user has entered a specific conference by selecting it in the "All Conferences" view, the application will present all conference-related features in the main content area. The navigation bar stays visible all the time. We introduced an additional navigation level where the user can view different aspects of the conference by changing the active tab (see figure 4.6).

Enrolling is required to see other SVs and to bid on tasks

Before student volunteers can fully interact with the conference, they need to enroll for the conference on the "Overview" tab (see figure 4.7). This will require them to answer a few questions, which later on help SV chairs to optimize the SV selection. For example, SV Chairs could be interested in accepting at least a bunch of volunteers who are local to where the conference is. This might, later on, help to tackle issues related to the location. We will take a look at custom enrollment forms later in 4.4.5



Honolulu, Hawaiʻi, USA | May 22, 2020 – May 28, 2020
svchair@chi2020.acm.org

22.5.2020, 23.5.2020, 2 | 10:00 AM - 6:00 PM | capta

Unavailable / Low / Medium / High ?	Date	Starts	Ends	Hours	Name	Location
Unavailable [✓ Low] Medium High	5/28/2020	3:00 PM	5:30 PM	02:30	Plenary Team Captain	Kalakua Ballroom
Unavailable [✓ Low] Medium High	5/28/2020	2:00 PM	5:30 PM	03:30	Plenary Rehearsal & Plenary Team Captain	
Unavailable [✓ Low] Medium High	5/28/2020	12:00 PM	6:00 PM	06:00	Day Captain	
[✓ Unavailable] Low Medium High	5/27/2020	12:00 PM	6:00 PM	06:00	Day Captain	
[✓ Unavailable] Low Medium High	5/26/2020	12:00 PM	6:00 PM	06:00	Day Captain	
Unavailable [✓ Low] Medium High	5/25/2020	12:00 PM	6:00 PM	06:00	Day Captain	
Unavailable Low [✓ Medium] High	5/24/2020	12:00 PM	4:00 PM	04:00	Day Captain	
Unavailable Low [✓ Medium] High	5/24/2020	12:00 PM	6:00 PM	06:00	Day Captain	
Unavailable Low Medium [✓ High]	5/23/2020	12:00 PM	6:00 PM	06:00	Day Captain	

9 tasks matching criteria

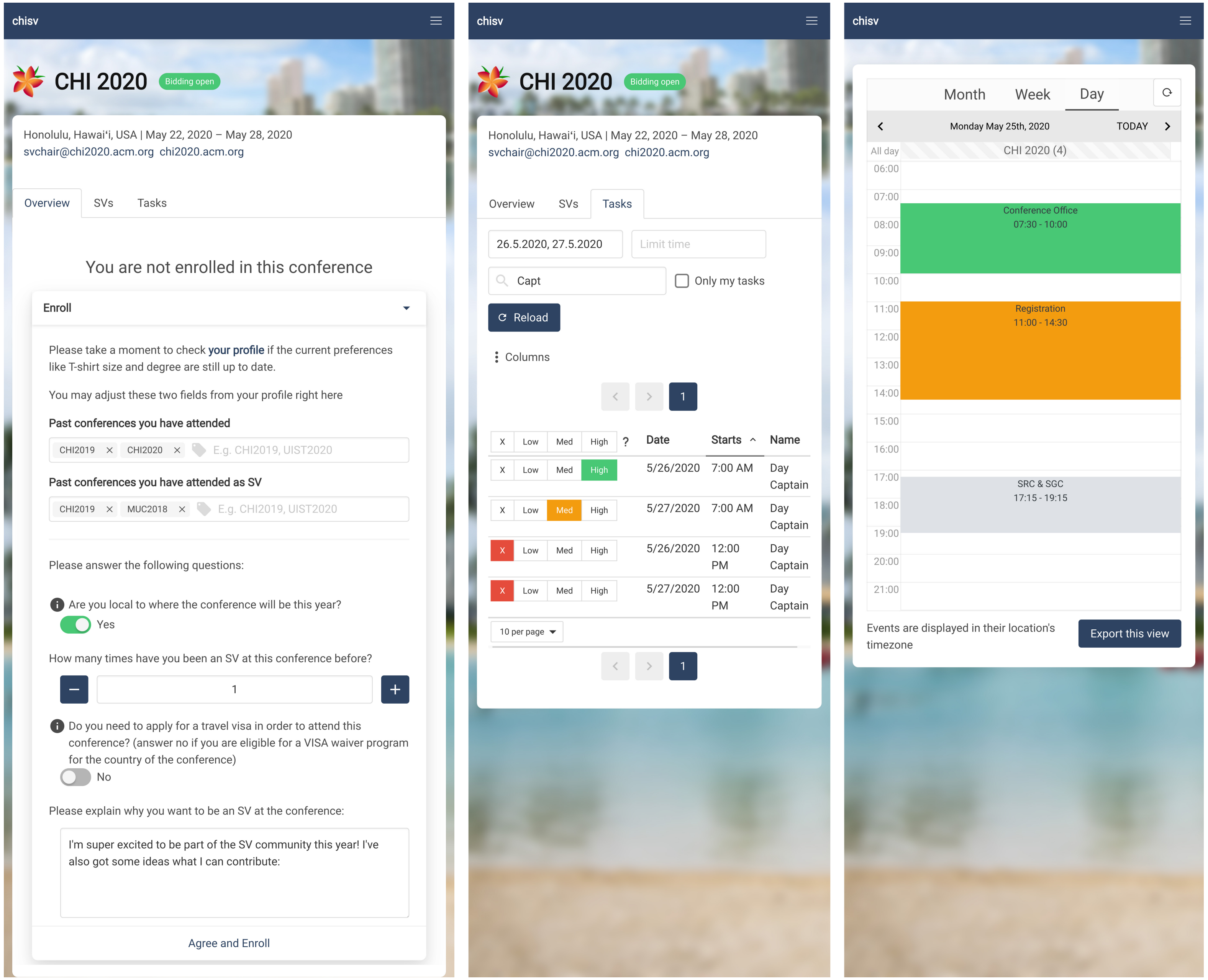
Honolulu, Hawaiʻi, USA | May 22, 2020 – May 28, 2020
svchair@chi2020.acm.org

22.5.2020, 23.5.2020, 2 | 10:00 AM - 6:00 PM | capta

Manage	Date	Starts	Ends	Hours	Name	Location	Slots	Priority
Edit Delete	5/28/2020	3:00 PM	5:30 PM	02:30	Plenary Team Captain	Kalakua Ballroom	1	1
Edit Delete	5/28/2020	2:00 PM	5:30 PM	03:30	Plenary Rehearsal & Plenary Team Captain		1	1
Edit Delete	5/28/2020	12:00 PM	6:00 PM	06:00	Day Captain		2	1
Edit Delete	5/27/2020	12:00 PM	6:00 PM	06:00	Day Captain		2	1
Edit Delete	5/26/2020	12:00 PM	6:00 PM	06:00	Day Captain		2	1
Edit Delete	5/25/2020	12:00 PM	6:00 PM	06:00	Day Captain		2	1
Edit Delete	5/24/2020	12:00 PM	4:00 PM	04:00	Day Captain		1	1
Edit Delete	5/24/2020	12:00 PM	6:00 PM	06:00	Day Captain		1	1
Edit Delete	5/23/2020	12:00 PM	6:00 PM	06:00	Day Captain		2	1

9 tasks matching criteria

Figure 4.6: Comparison of the task view for SVs (top) and SV Chairs/Day Captains (bottom). The UI adaptively shows and hides the required UI components.



Honolulu, Hawai'i, USA | May 22, 2020 – May 28, 2020
svchair@chi2020.acm.org chi2020.acm.org

You are not enrolled in this conference

Please take a moment to check your profile if the current preferences like T-shirt size and degree are still up to date.

You may adjust these two fields from your profile right here

Past conferences you have attended: CHI2019 × CHI2020 × E.g. CHI2019, UIST2020

Past conferences you have attended as SV: CHI2019 × MUC2018 × E.g. CHI2019, UIST2020

Please answer the following questions:

Are you local to where the conference will be this year? Yes

How many times have you been an SV at this conference before? 1

Do you need to apply for a travel visa in order to attend this conference? (answer no if you are eligible for a VISA waiver program for the country of the conference) No

Please explain why you want to be an SV at the conference:

I'm super excited to be part of the SV community this year! I've also got some ideas what I can contribute:

26.5.2020, 27.5.2020 | Limit time
Capt | Only my tasks

				?	Date	Starts	Name
X	Low	Med	High				
X	Low	Med	🟩 High		5/26/2020	7:00 AM	Day Captain
X	Low	🟧 Med	High		5/27/2020	7:00 AM	Day Captain
🟥 X	Low	Med	High		5/26/2020	12:00 PM	Day Captain
🟥 X	Low	Med	High		5/27/2020	12:00 PM	Day Captain

Monday May 25th, 2020
All day: CHI 2020 (4)
06:00 07:00 08:00 09:00 10:00 11:00 12:00 13:00 14:00 15:00 16:00 17:00 18:00 19:00 20:00 21:00
Conference Office 07:30 - 10:00
Registration 11:00 - 14:30
SRC & SGC 17:15 - 19:15

Events are displayed in their location's timezone

Figure 4.7: Mobile view for student volunteers showing the three most used features for SVs: Enrolling for a conference by answering the customizable enrollment form (left), task bidding with preferences (middle), and the calendar with month, week, and day view (right).

“Custom Enrollment Forms”.

After enrolling, SVs can unenroll as long as they have not been dropped from the conference (#28). They will be greeted with their current enrollment state whenever visiting the "Overview" tab. After unenrolling, they could re-enroll as long as the conference is in the "Enrollment" state.

Unenroll as long as not dropped

To be able to see other student volunteers on the "SVs" tab the volunteer has to be accepted first. This can either be done manually or automatically when the enrollment phase ends and the lottery is run by the SV Chair. Accepted SVs can then also see other SVs, their name, university, and home country.

Only see other SVs when "accepted"

SVs

The "SVs" tab will list and show all associated student volunteers. Based on the permissions and roles of the currently authenticated user, we adapt the available information dynamically. For SVs, the view features a simple table with every other volunteer’s name and home country. This view will present a lot more detail for SV Chairs and Day Captains.

Information is limited for SVs

They can see every important detail about any SV by clicking on the table row. This will expand the row for the selected user to show:

Shows certain details about a volunteer

- **Key profile details** – E.g. all spoken languages and the current degree program
- **Assignments** – Shows accounted hours and any associated note
- **Notes** – Lists all notes posted for the user or on associated assignments
- **Bids** – Shows all bids placed by the user including their state (e.g. "Won" or "Conflict")

- **Notifications** – Any notification that has been sent to the user will be listed here including the time the message was read
- **Enrollment Form** – The complete enrollment form with all answers
- **Statistics** – Shows the number of completed hours, placed bids, and all assignments

Sort and filter columns

While most of these sections are just presenting facts, SV Chairs and Day Captains may post or delete notes for the volunteer at this place. The lists showing assignments, bids (#37), notifications, and notes can be sorted per column or searched for a specific occurrence (e.g. date). Furthermore, SV Chairs will have the option to look into posted notifications of the user and the active conference.

Changing SVs' enrollment state

The "SVs" view is also the place where SV Chairs can manually drop, accept, or waitlist volunteers (#6, #30). This is independent of the lottery, which can always be run to fill the conference with waiting ("waitlisted") or new ("enrolled") SVs. Since we have implemented custom enrollment forms (see 4.4.5 “Custom Enrollment Forms”), we also made the evaluation of enrollment form questions possible from this view.

Submit new form weights

An SV Chair member can submit new weights for each custom enrollment form question and by doing so update the column "Weighted form" for each SV on this "SVs" view. While we will better understand this process when we later take a deeper look at it, we can now think of this value as a "best suited" index. SV Chairs will have the possibility to sort by this column and then modify the enrollment state of the most suited SVs (according to their submitted weights). While this is most commonly used for accepting SVs, it can also be used to waitlist or drop volunteers.

Tasks and bidding

Interface only exposes interactions that are important for the user

Another tab that is available for SVs is the "Tasks" tab where volunteers can bid on tasks after selecting the desired day, time, and name filter (#34, #39). SV Chair members and Day Captains can use the same view to modify tasks. The interface adjusts dynamically based on the roles the user has. In figure 4.6 one can see the differences in the "Tasks" view between SVs and SV Chairs/Day Captains. The SV's view lacks any UI components for modifying tasks but provides components for task bidding. The view for SV Chairs/Day Captains allows for modifying tasks but hides the elements for bidding.

Mobile view adapts to the screen space

Figure 4.7 gives us a similar view on tasks (view in the middle) but focuses on the experience on a mobile device. We can clearly see how some UI components are truncated (preferences) or left out (conference state) to not clutter the interface and degrade the users' experience. As commonly seen on mobile web applications, the main navigation bar is hidden and visible only when triggered on the upper right hamburger menu[9].

Multi-bid

Multi-bid on all filtered tasks

We added functionality for bidding on all filtered tasks with just one click (#6, #35). We call this "multi-bid" or "multi-bidding". A set of filtered tasks can be determined by expressing the desired days, time, or task name with the UI components of the "Tasks" view (see figure 4.6). To submit a bid for each task in the table (includes all pages), a volunteer would click in the column's header on the desired preference. To make this concept easier to understand, we added an in-place explanation (question mark next to the component). Also, before a multi-bid is submitted, the volunteer is presented with a short overview of the number of bids that are about to be

[9]The hamburger menu is named after it's often used three stacked horizontal lines, which appear like the stacked ingredients of a hamburger.

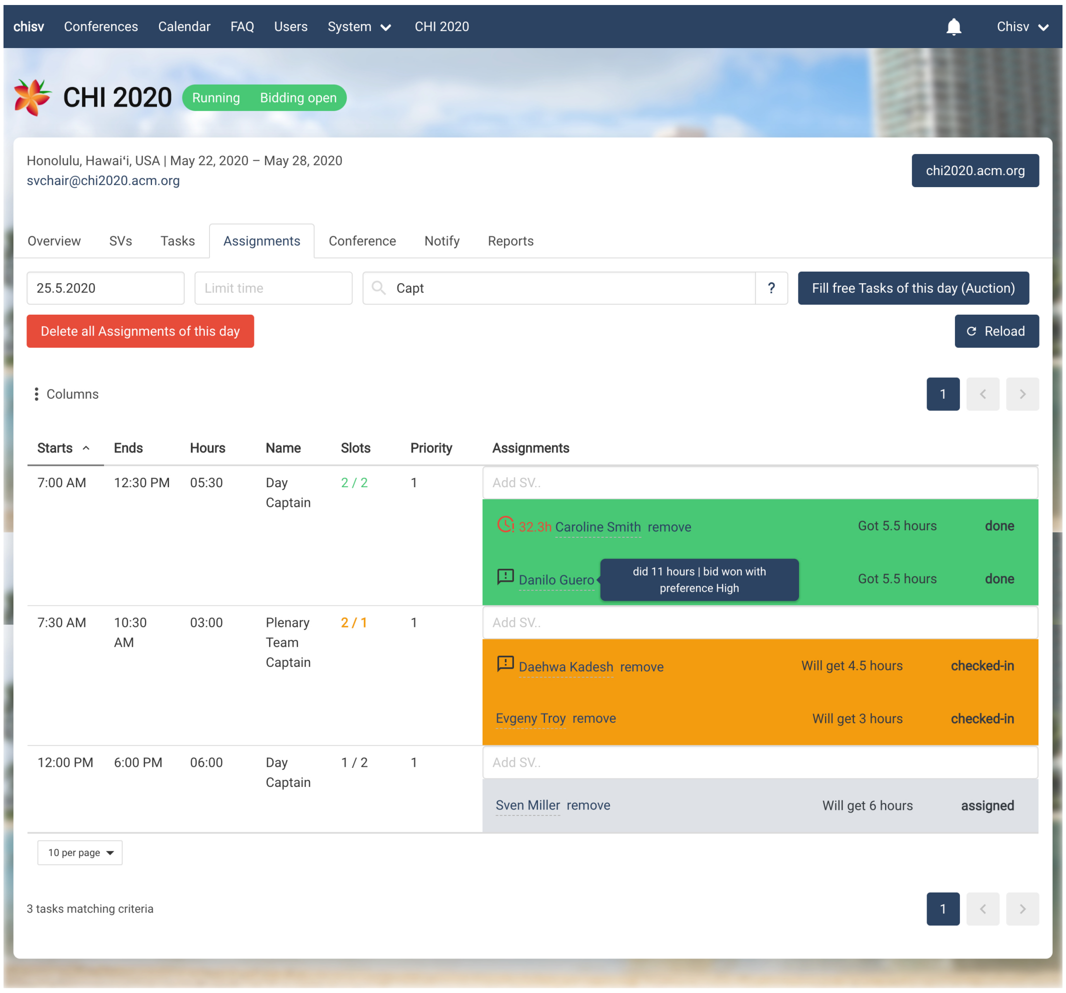
placed. The explanation pop-up, which the SV can trigger by clicking the question mark next to the component, will also offer a link to the FAQ section where we explain multi-bidding in even greater detail. After a multi-bid has been sent to the back end, the state of each placed bid is reflected in the front end automatically.

Assignments

The "Assignments" view is only visible for SV Chairs and Day Captains

A good example for a page that only SV Chair members and Day Captains can access, is the "Assignments" view (see figure 4.3 and 4.8). On this page, SV Chairs and Day Captains can add or remove SVs to or from a task (#6). Clicking on the "Add SV.." input (see figure 4.8 and figure 4.9 for a closer look) will show a dropdown with SVs best fitting the task. We sort the volunteers by their bid preference descending and ascending by their number of completed hours. We show the completed hours as well as the assigned hours such that these manual assignments won't accidentally put too many hours of work on the SV. Furthermore, we show the SV's bid to the task (if there was any) and also a short statistic about the SV's bidding behavior.

While manual assignment creation is always possible, SV Chairs usually only pick some specific SVs for certain tasks (e.g. for a task that requires special skills). After that, the auction is usually run to most efficiently fill any free tasks with SVs (see 4.4.4 "Auction"). The auction can be started for the selected day at the "Assignments" view. If SV Chairs want to redo the schedule, they can delete all assignments of a given day. Any assignment, which is shown in the assignment view, is also present in the associated volunteer's calendar including its current state.



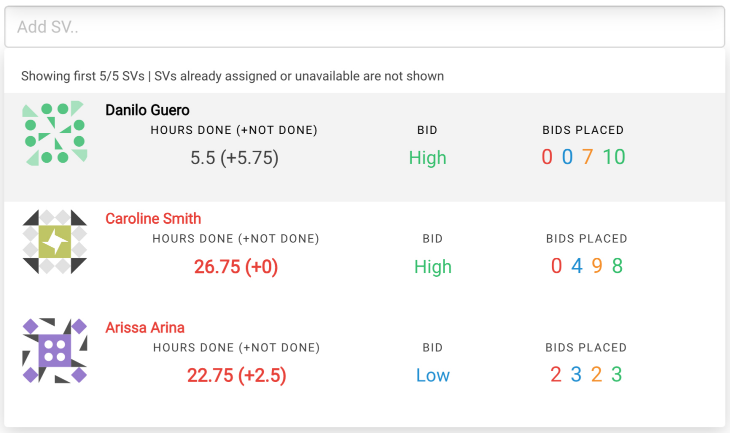
Honolulu, Hawaiʻi, USA | May 22, 2020 – May 28, 2020
svchair@chi2020.acm.org

25.5.2020 | Capt

Starts	Ends	Hours	Name	Slots	Priority	Assignments		
7:00 AM	12:30 PM	05:30	Day Captain	2 / 2	1	Add SV..		
						32.3h Caroline Smith remove	Got 5.5 hours	done
						Danilo Guero (did 11 hours | bid won with preference High)	Got 5.5 hours	done
7:30 AM	10:30 AM	03:00	Plenary Team Captain	2 / 1	1	Add SV..		
						Daehwa Kadesh remove	Will get 4.5 hours	checked-in
						Evgeny Troy remove	Will get 3 hours	checked-in
12:00 PM	6:00 PM	06:00	Day Captain	1 / 2	1	Add SV..		
						Sven Miller remove	Will get 6 hours	assigned

3 tasks matching criteria

Figure 4.8: CHISV's assignments view allows SV Chairs and Day Captains to modify an SV's assignment state (see 4.2.4 "Tasks, Bids, and Assignments") and to remove or add volunteers from/to a task. This is also the place where the SV Chairs start the auction and where they and Day Captains can add notes to assignments (see "Danilo Guero", "Daehwa Kadesh") or adjust the accounted hours (see "Daehwa Kadesh"). When SV Chairs or Day Captains hover over the SV's name, a blue pop-up will give insight about the SV's hours and the bid for the selected task. We used the same filters as we did in the "Tasks" view. However, only one day can be selected at once.



Add SV..

Showing first 5/5 SVs | SVs already assigned or unavailable are not shown

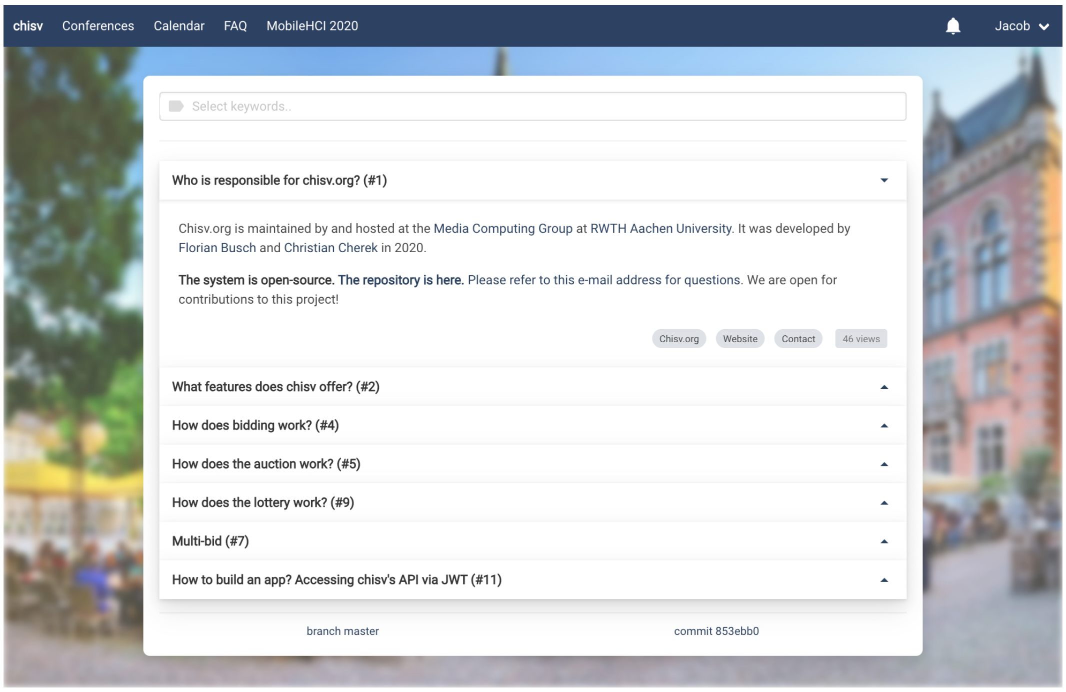
	HOURS DONE (+NOT DONE)	BID	BIDS PLACED
Danilo Guero	5.5 (+5.75)	High	0 0 7 10
Caroline Smith	26.75 (+0)	High	0 4 9 8
Arissa Arina	22.75 (+2.5)	Low	2 3 2 3

Figure 4.9: Creating an assignment with CHISV: Users with a strong preference for the task are shown first. We sort them ascending by the hours they've completed. By typing a name, a specific SV can be found. SVs who would have a time conflict by creating an assignment for this task, are not shown. If an SV has more hours than required for the conference the hours will be colored red.

Calendar

Minimize network load of the calendar view

For CHISV's calendar view (see on the right in figure 4.7) we optimized all required network requests to only transfer as few data as possible. The survey (3.2.2 "Survey") showed us the importance of this feature (#17, #38). We learned that SVs will most likely be on the go, before, or after an assigned task when using the calendar. This could hint at an unstable connection to the application. To ensure that we do not waste any valuable network resources, we will only load the assignment for the currently visible view. Any static content, like the days a conference is scheduled for, will not be loaded. These are available after the Vue.js application booted and are stored in our Vuex store (see page 79). This way we will only load the assignments and ensure an up to date view for the SVs. CHISV's calendar is accessible through the main navigation (see "Calendar" in figure 4.3). The calendar will show all assignments a volunteer has and also the days of all conferences, as the calendar is not bound



Who is responsible for chisv.org? (#1)

Chisv.org is maintained by and hosted at the Media Computing Group at RWTH Aachen University. It was developed by Florian Busch and Christian Cherek in 2020.

The system is open-source. The repository is here. Please refer to this e-mail address for questions. We are open for contributions to this project!

Chisv.org Website Contact 46 views

What features does chisv offer? (#2)

How does bidding work? (#4)

How does the auction work? (#5)

How does the lottery work? (#9)

Multi-bid (#7)

How to build an app? Accessing chisv's API via JWT (#11)

branch master commit 853ebb0

Figure 4.10: FAQs in CHISV: Helping users of the system understand its terminology and algorithms. Keywords can be selected at the top. Topics based on the selection will show up below. Each topic is expandable and will show it’s view count and tags in grey at the bottom. The FAQ view is also the place where we expose CHISV’s current version and branch.

to a specific conference.

FAQs

FAQs and the notification center are results from our user-centered development

Apart from the personal profile, which we covered earlier, student volunteers have access to two more new features we have added to CHISV as result from volunteer’s feedback on the previous version of CHISV. We introduced a section to answer Frequently Asked Questions (FAQ or FAQs), where we got the chance to explain the system in more depth. We also added a notification center. We will look at the FAQ system (see figure 4.3) now and later on present the notification center as part of 4.4.7 “Notifications and Reports”.

FAQ page helps the volunteers to catch up on terms and procedures

When we started looking into the previous version of CHISV we quickly noticed that it uses many abbreviations and unknown terms. As we were approaching the system the same way any other volunteer would, we saw the need to dive deeper into the terminology of CHISV and figure out how transparent and precise the wording and documentation is. We noticed during our interviews and while reading CHISV's code, how imprecise – and often wrong – the assumptions were that have been made about the software.

Interviews with the SVs and SV Chairs showed how inaccurate the assumptions were

Most notably have been the assumptions of how the lottery and auction work. However, we also saw some shortcomings in the representation of states around many instances like conferences and volunteers. We think it is important that users of a system feel not intimidated by it. This may happen when functionality is not explained precisely enough such that the system tends to look complex or when the user might fear to break it. As we have addressed earlier (see 4.2.4 "Conference, Users, and Permissions" and 4.2.4 "Tasks, Bids, and Assignments"), we tried to expose the meaning of each state by adding a description to the model. Whenever a state is shown in the web application, the user can click or touch the state's name to see the description. We see this as a first step. Furthermore, we think, it is also important to revisit the states in the documentation, which covers the relations and backgrounds more broadly than a short pop-up description can.

Introducing a central place for all questions and explainations

To be able to better explain the terminology, like the meaning of states or the algorithms CHISV uses, we introduced a Frequently Asked Questions (FAQ) section (see figure 4.10). FAQs are accessible from the main navigation and are presented like the calendar component by taking all available space in the main content area. The view will show different help topics ranked by their popularity (view count) and is accessible for any registered user. Our primary intention was to help new SVs understand how CHISV works.

During our interviews and continuous feedback, we noticed that there are also a lot of topics only intersecting with tasks of SV Chairs or Day Captains. We, later on, added the ability to limit certain help topics to specific user roles. This was required as some topics may contain internal information or may be confusing for new student volunteers.

Help new SV Chairs and Day Captains to learn more about the advanced features of CHISV

Help topics can contain multiple tags. These are selectable from a dropdown shown on the top of the page. SVs can, for instance, filter for any help topic tagged for "Lottery" and will then only see topics related to the lottery. Furthermore, help topics can be created, edited, or deleted right where they are shown. This functionality is available for all SV Chairs such that we can shorten the paths required for creating new or updating existing help topics. Also, this allows SV Chairs to adapt the documentation of a specific topic when they feel the need without reaching out to the CHISV administrators, making adjustments quick and easy. We see this as an important point for keeping the FAQs up to date.

Topcis can be tagged and modified right on the spot

All FAQs are also available from the API endpoints for third-party applications (like any CHISV resource). Furthermore, we found it helpful to be able to directly link to a topic. This is possible by appending it’s number to the URL making it easy for SV Chairs to reference a specific topic when sending e-mails or other related materials.
This wraps up our overview of CHISV’s user interface and front-end functionality. We will now take a look at some more advanced features, which make CHISV stand out, as they provide a strong foundation for the years to come and are often a response to the requirements and ideas many volunteers shared with us.

FAQs support hard-linking and are stored in the back end

4.4 Selected Features In-Depth

4.4.1 Cross-Site Scripting (XSS) and Cross-Site-Request-Forgery (CSRF) Mitigation

It is important to notice that applications that interface with CHISV can use Cookie-based or token-based authentication. The later one is handled by the OAuth driver of Laravel. Our SPA does only use Cookie-based authentication. For that to work, the SPA and the API endpoints have to share the same domain. As we have control over the domain (chisv.org) we used this Cookie-based approach. While this approach is uncommon for a SPA, we opted for it due to security reasons.

JSON Web Token (JWT)

First, let's take a look at a commonly used method to connect a SPA to an API endpoint. When we think of a Vue.js single-page application that makes use of Vuex we would usually obtain a JSON Web Token (JWT) by sending an HTTP POST request to an API login endpoint. This request needs to contain our credentials, for example, an e-mail address and password. In case the credentials are correct the API would return a JWT. This token is Base64 encoded[10].

Definition:
JSON Web Token

JSON Web Token:
A JSON Web Token (JWT) is a compact and URL-safe claim of possession between two parties. Such a claim is signed and integrity protected with a Message Authentication Code (MAC). The JWT's content can also be encrypted, while for authentication and authorization the token's payload is usually in plaintext. JWTs are prepended with a header, which signals the type and signature. To sign the header and the payload, a simple secret or public/private key pair can be used.

[10]Base64 encodes binary data to ASCII characters

Consider this JWT, for example:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2l
kIjoiMTIiLCJuYW1lIjoiSmFjb2IgU21pdGgiLCJpYXQiOjE
1MTYyMzkwMjIsImV4cCI6MTkxNjIzOTAyMn0.zXgO1Yr8pnt
rDu22xLxzwDgGMaN5k5ZpgWIoYY21OPY
```

First part is the token's header

This token consists of three parts. We set each part in a different color. Each part is delimited by a period. Every JSON Web Token is structured in this way: The first and blue part is the header of the token. It states the type and the algorithm used for its signature. JWT supports a variety of different algorithms (HMAC, RSA, and ECDSA) which differ in terms of speed and architecture (secret or key pair) [Rahmatulloh et al., 2019]. When we decode the blue header (Base64 string) we obtain a JSON object, which shows us that the token is of type "JWT" and uses HMAC with SHA-256:

```
    {
        "alg": "HS256",
        "typ": "JWT"
    }
```

The header is followed by the payload

The next bordeaux colored part is the token's payload. When we decode it we again get a JSON object:

```
    {
        "user_id": "12",
        "name": "Jacob Smith",
        "iat": 1516239022, // 01/18/2018 1:30am
        "exp": 1916239022  // 09/21/2030 4:37pm
    }
```

Token can carry its expiry date

In this example, the payload is not encrypted and can be read by anyone with access to the token. While the payload can contain any arbitrary amount of keys and content, the "iat" and "exp" are reserved to handle the expiration of a token. The "issued at" ("iat") and "expiration time" ("exp") are optional but when set signal when the token was issued and when it will expire. One might think exposing the expiry date editable in the payload induces security-related issues. However, since the header and the payload are used for the signature generation, a modified token can be quickly identified and rejected.

Signature generation

The signature is stored in the last (violet) part of the token. Decoding its Base64 representation yields a binary blob (the output of the signature function) and no JSON object like with the first two parts. For a HS265 JWT token the signature is calculated by using the header, the payload, and the chosen secret passphrase:

```
HMACSHA256(
    base64(header) + "." + base64(payload),
    256-bit-secret-passphrase
)
```

The generated signature is then appended to the Base64 encoded header and payload with a period.

Checking validity

Since every token is signed by the server when handed out to the client, the server does not need to keep a state associated with the authenticated client. As long as the server's secret does not change, the server can verify the validity of the token again by generating the signature with the configured secret. The generated signature and the signature of the requesting token have to match. If they match the server verifies the expiration date ("exp") to see if the token is still valid. If the token has not expired yet the server can safely assume that the request is authentic and check for authorization of the requested resource.

The stateless design of JWT makes it especially suited for distributed environments

Using a JSON Web Token for authentication has the benefit that it is stateless. We will not need to hold a reference to the token in the back end and thus can scale better [Madwesh and Nadimpalli, 2019]. Think of our earlier example where we described how CHISV can scale-out by adding additional instances to a load-balanced group of instances. By exclusively using JWT for authentication we could relinquish providing a shared session store via our database. As long as all instances share the same secret or public/private key pair, every instance can verify the integrity and validity of a request's token.

JWT tokens are stored in memory or the website's local storage

As well suited as JWTs tend to be for our application, we see some strong security issues when integrating token-based authentication into our SPA. Usually, tokens are requested and stored by JavaScript code. For a Vue.js SPA, one would typically tend to store the token in the Vuex data store and persist it to the website's local storage. This way the token can be loaded again from the local storage when the user reload the website (which resets the Vuex store). This approach however is prone to Cross-site scripting (XSS) attacks.

Definition: *Cross-site scripting (XSS)*

Cross-site scripting (XSS):
Cross-site scripting (XSS) is a security vulnerability commonly found in web applications. An XSS attack allows a user of the website to inject client-side scripts into web pages that are viewed by other users (victims). The injected code is then interpreted and run in the victim's local environment, having access to the victim's memory. This is often used to exfiltrate authentication tokens or Cookies.

Carrying out an XSS attack

A malicious user could inject malicious code into an unsanitized field or form. In this example, we assume that the application is not sanitizing inputs from users and allows for injecting a JavaScript script tag into the input field for the first name in the profile settings. We assume that the injected script reads the JWT from the Vuex store (or local storage), and sends it to an external server. All this would happen whenever a web page is shown to any user (victim) that contains the first name of the attacker. The attacker can thereby get hold of multiple valid tokens. By simply overwriting the local storage with the stolen token, refreshing the website, is sufficient to be logged in as the victim the token belongs to. Ahmed and Mahmood showed how the design of JSON Web Tokens can be strengthened. However, we see the possibility that by deviating too far from a standardized path of how authentication with JWT works we make it harder to maintain the application.

JWT revocation and expiry

While the user can simply remove a valid token from the Vuex data store and the local storage to logout, the JWT token itself remains valid. Consider a stolen token, for example. The user could log out (remove the token) without affecting the validity of the stolen token. The JWT is stateless by design und thus not easily revokable. This can be overcome by storing a list of revoked tokens in the back end – but this again induces the problem with the shared back-end database [Jánoky et al., 2018]. Another countermeasure to this is to keep the lifetime of the token short by setting an early expiration date ("exp" field in the payload). This, however, on the one hand, only reduces the timeframe a stolen token can be used and on the other hand, requires additional mechanisms to constantly renew the token without the user noticing or interacting with the SPA.

To summarize, we think CHISV's SPA is more vulnerable to XSS attacks when paired with token-based authentication. To overcome the XSS problematic many web applications try to restrict access to the session secret (token/-Cookie) such that it cannot be exfiltrated or used without the user's knowledge and consent [Grossman et al., 2007]. A rather conservative approach to this is by using a Cookie that cannot be accessed from JavaScript at all.

Cookie-based Authentication

Authentication Cookie is `httpOnly`

As we described earlier in chapter 4.3.2 “Login”, we opted for Cookie-based authentication for CHISV's SPA. We can configure the webserver (Nginx) which runs the Laravel application and also serves the SPA. We ensured that whenever Laravel would issue a Cookie, which can be used for authentication, Nginx also ensures it is appended with the string `httpOnly`[11]. Modern browsers understand this modifier and ensure that the Cookie can never be read from any JavaScript code. This shields our application against any XSS attacks that try to get hold of the Cookie (session). Unfortunately, this causes

[11]We also append the `secure` modifier to ensure the Cookie gets only passed to the client via HTTPS

the application to not know the current status of authentication. We solved this by fetching the API endpoint `/api/v1/user/self` which returns the currently logged in user as JSON or `403 Unauthenticated`. This way the Vue.js application knows when it boots if the Cookie is available and valid. Our JavaScript library for talking to the API endpoints (Axios) is also not aware of the Cookie and is just carrying out the requested calls.

Browser appends the Cookie

The browser will always send the authentication Cookie along with every request to CHISV's domain. This way once logged in, the browser will take care of passing the Cookie to the back end whenever called by our Vue.js application. Each time Laravel issues a Cookie for a user the resulting session is linked to an entry in the session database. Every authenticated user has a session in this database. Different from using JWTs, we can now quite simply end a user's session by removing the session from the session database. This happens whenever the user decides to logout. A HTTP `POST` request to the logout URL (`/logout`) is made. The response clears the authentication Cookie of the user, which has also been invalidated in the session database by the back end. This way a leaked Cookie cannot be used past logout.

Cookies can suffer from CSRF

While with JWT tokens the SPA will always need to manually add the token to the request's header, this does not apply to Cookie-based authenticated session. The browser will always append the Cookie based on the domain. This makes an application prone to Cross-site request forgery (CSRF or XSRF) attacks.

Definition: *Cross-site request forgery (CSRF or XSRF)*

Cross-site request forgery (CSRF or XSRF):
Cross-site request forgery (CSRF or XSRF) is a security vulnerability commonly found in web applications. A CSRF attack allows an attacker to instruct the victim's browser into carrying out arbitrary HTTP requests to a website for which the victim holds an active session. If the attack succeeds the attacker can manipulate data on the target website as if committed by the victim. While this does not leak the session itself (see XSS), it utilizes the fact that the web application trusts the victim (session).

Using a URL-based method to transfer the session id

Jovanovic et al. found that there are multiple ways to mitigate a CSRF attack. One could design an application to not use the session id from a Cookie but to expect it in the query as a parameter (`?session_id=23817361...`). This however has the disadvantage that it is captured in bookmarks the user creates and is overall quite easy to obtain by an attacker (e.g. in a public space). Additionally, submitting a large form via an HTTP `GET` request is sometimes not possible due to the length limitation.

Relying on the HTTP `Referer` header

Another approach to this problem is by using the HTTP `Referer` header. However, this would induce privacy concerns as the application would always leak the previous website and overall not solve the issue to the fullest, as modern browsers can be configured to omit the HTTP `Referer` header.

Hidden CSRF-tokens to mitigate the attack

As Jovanovic et al. points out, a good solution to this problem is the use of special CSRF-tokens. The idea behind this approach is that the application will always append a hidden input field to any form and pre-fill this field with a random string that is hard to guess. When the user is then validly submitting the form the hidden CSRF-token field will be transmitted along with the form data. The application will then only have to compare the expected CSRF-token value (which was pre-filled) with the one present in the request. This way only a user who has knowingly loaded the web page can submit it.

Jovanovic et al. argue that implementing such a drastic change for every request can be quite cumbersome to integrate into an existing application and also induces a lot of overhead for building new applications. Fortunately, Laravel provides support for this right out of the box for any Blade template (enabled by default). Any submitted form is expected to contain a hidden CSRF-token field. To include it into a Blade template's form we would use the macro `@csrf` within the forms body. Consider this simple example for editing a user's first name:

CSRF protection is already built into Laravel

```
<form action="/user/{{ user.id }}" method="POST">
    @csrf
    <input type="text" id="firstname" value="{{ user.firstname }}">
    <button type="submit" value="Save">
</form>
```

This will make Laravel generate a hidden input form and pre-fill it with a CSRF-token only known to the application. On submit (HTTP `POST`) the token will be checked again by the back-end application and the request will be processed or rejected.

Hidden field generated by `@csrf`

As we explained earlier, we started CHISV's development with Blade templates (and its CSRF mitigation strategies). However, after the point where we moved away from Blade templates, we could no longer make use of this concept. Fortunately, Laravel integrates with Vue.js and the HTTP client Axios, which we used for all calls to our API (see Documentation).

Also applicable to our SPA approach

To protect our API endpoints from CSRF attacks, Laravel issues three Cookies when the user logs in. Each of those Cookies will be sent along with every request automatically. The first one (`chisv_session`) is the classical authentication (session) Cookie that cannot be read by JavaScript. It is used for authenticating the session. The other two Cookies are used for CSRF protection. One of them (`laravel_token`) holds an encrypted JWT token that can only be decrypted by the back end (Laravel) and is not readable by JavaScript. This JWT token includes a "csrf" field in its payload with a 40 characters long random string. This is the string that will later be compared

Cookie's structure explained

against the headers and Cookies. The other Cookie, the third one in our list, is called `XSRF-TOKEN` and does also carry the **same** JWT token that can only be decrypted by Laravel. This Cookie, however, can be read by JavaScript.

Axios will automatically use the `XSRF-TOKEN` Cookie

When Axios boots it will automatically search for the `XSRF-TOKEN` Cookie and if found add a new header to the "common headers" list that contains the Cookie's content. This header is called `X-XSRF-TOKEN`. All these steps until this point would suffice to fight CSRF. However, Laravel will also add an HTML meta tag to the web page that holds the "csrf" string from the JWT token. As it's best practice, Axios will also pick up this token from the DOM and also add it to its "common headers" list. However, this entry will be called `X-CSRF-TOKEN`. Any request to the back end will from then on contain the encrypted `laravel_token` Cookie, which cannot be read by JavaScript, a bunch of XSRF/CSRF tokens (Axios' `X-XSRF-TOKEN` and `X-CSRF-TOKEN`, and the `XSRF-TOKEN` Cookie).

One needs to be present and valid, none may fail when existing

The back end (Laravel) will then decrypt the `laravel_token` Cookie and validate that it's the same one it issued earlier. If the Cookie is intact Laravel will require at least one of the following to be present and true:

1. The `X-CSRF-TOKEN` header exists and its value is the same as the decrypted `laravel_token`'s "csrf" string
2. The `X-XSRF-TOKEN` header and the `XSRF-TOKEN` Cookie exist, both can be decrypted and the resulting JWT's "csrf" string is the same as the decrypted `laravel_token`'s "csrf" string

Using both ways follows best practice

If only one of these two checks fail Laravel will reject the request and return an HTTP response of `401 Unauthorized`. While it is completely sufficient to only use either the `X-CSRF-TOKEN` or the `XSRF-TOKEN/XSRF-TOKEN` approach to fight CSRF, both ways are actually the default design for protecting API endpoints with Laravel Passport and Axios. Since we would like to deviate as little a possible from the official documentation, we chose to use both approaches in parallel. With this approach of Cookie-based authentication combined with CSRF mitigation, we think that our application can rely on a strong foundation for authentication.

4.4.2 Job Extension

Laravel jobs can be queued

Laravel comes with a very capable Job scheduling and queueing system (see Documentation). It supports queueing a Job for later async executing, allows for prioritization, and provides multiple different queue drivers right from the start like Beanstalk, Amazon SQS, Redis, or even a relational database. We have chosen to use a relational database as our queue driver, as this is also where we are storing our resources.

Lock jobs before execution

Typically multiple queue workers are running on the webserver where the Laravel application is hosted. The Laravel documentation recommends launching them with `supervisor` – a Linux tool for automatically restarting crashed or ended processes. We have set up our configuration to launch 8 concurrent queue workers. Due to our relational database and the fact that Laravel's queue workers lock a job before processing it, we can ensure that a job will only be processed by one queue worker at a time. This even applies to multiple web hosting instances when configured to use the same relational database.

One queue for everything

While Laravel supports having multiple queues for different priorities, we chose to use only one queue for CHISV. This made development easier and brought no tangible downside, as we have multiple queue workers running in parallel, which ensure that queued jobs are processed relatively quickly after insertion. These many queue workers become really important when CHISV sends e-mails. E-mails are also queued. Since for large conferences many e-mails (e.g. 300) need to be sent quickly, we adjusted the number of workers.

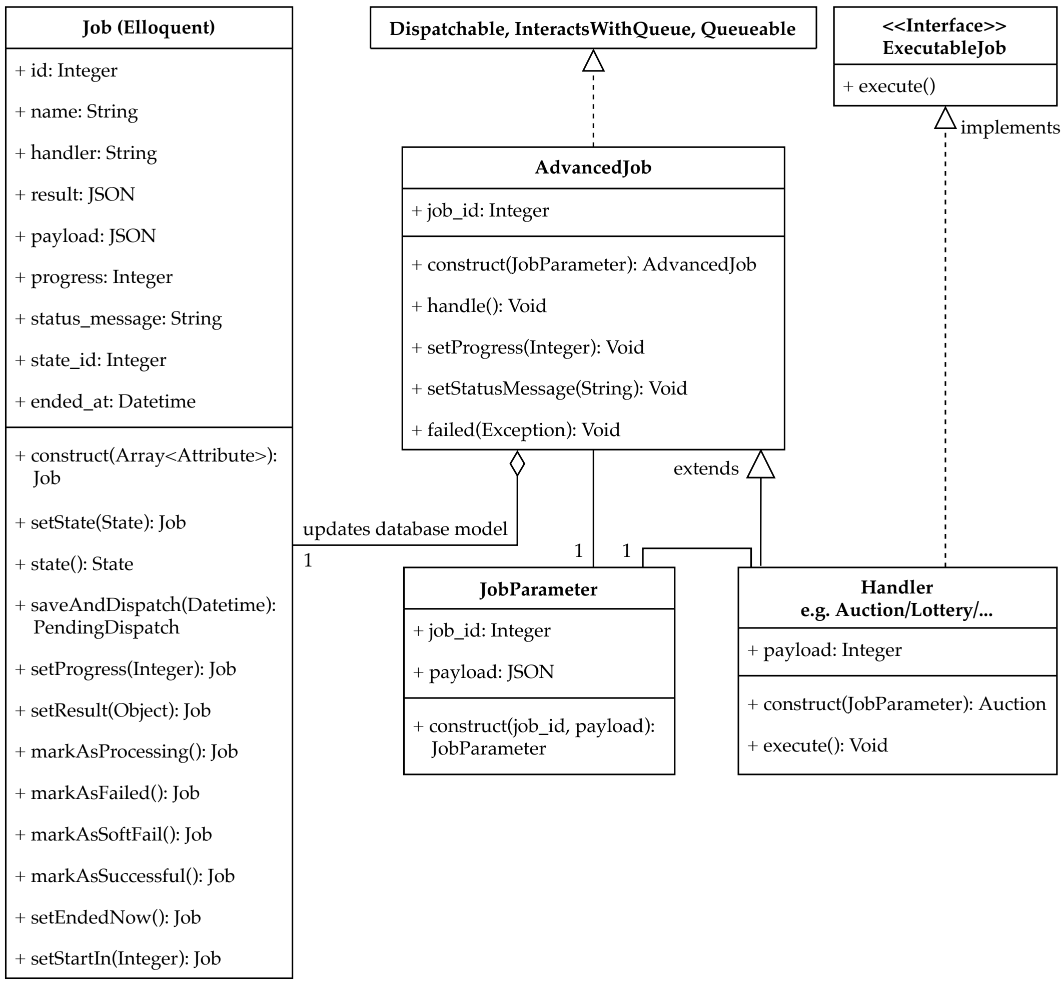

Job (Elloquent)
+ id: Integer
+ name: String
+ handler: String
+ result: JSON
+ payload: JSON
+ progress: Integer
+ status_message: String
+ state_id: Integer
+ ended_at: Datetime
+ construct(Array<Attribute>): Job
+ setState(State): Job
+ state(): State
+ saveAndDispatch(Datetime): PendingDispatch
+ setProgress(Integer): Job
+ setResult(Object): Job
+ markAsProcessing(): Job
+ markAsFailed(): Job
+ markAsSoftFail(): Job
+ markAsSuccessful(): Job
+ setEndedNow(): Job
+ setStartIn(Integer): Job
Dispatchable, InteractsWithQueue, Queueable
<<Interface>> ExecutableJob
+ execute()
implements
AdvancedJob
+ job_id: Integer
+ construct(JobParameter): AdvancedJob
+ handle(): Void
+ setProgress(Integer): Void
+ setStatusMessage(String): Void
+ failed(Exception): Void
extends
updates database model
1
1
1
JobParameter
+ job_id: Integer
+ payload: JSON
+ construct(job_id, payload): JobParameter
Handler e.g. Auction/Lottery/...
+ payload: Integer
+ construct(JobParameter): Auction
+ execute(): Void

Figure 4.11: We extended Laravel's Job system to allow for additional feedback while the job is processing (`setStatusMessage`, `setProgress`). Each job/handler is referenced in a permanently stored Eloquent model before, while, and after completion. A handler, for example the auction, will implement an interface `ExecutableJob` and extend our `AdvancedJob` wrapper. By calling methods on `$this`, the handler implementation (e.g. Auction/Lottery) can provide additional feedback, which can then be show to the user.

Jobs vanish after successful execution

We noticed that whenever Laravel successfully finished processing a job, the job would vanish from the queue. While jobs that ran into an exception during execution are placed into a separate table until they can be successfully processed, jobs that end successfully can no longer be referenced. Furthermore, a job that is processing can provide no further information about its state. It's not possible to express an estimate for the remaining time, nor any information about what is currently happening within the job. However, we especially felt the need to express additional information about the job after its completion. For instance, for the auction, we would like to get additional information about any task that could not be filled after the auction ran successfully (#46). For the lottery, SV Chairs would like to get feedback on how many SVs have been accepted or waitlisted. Even while a job is running (especially for longer running jobs) feedback is important. When taking a look at the auction, for example, the job may run multiple minutes until completion for many SVs, tasks, and bids. Providing feedback during the auction can help SV Chairs to understand how well bids can be factored in, if there are many conflicts or if they are short on SVs such that certain tasks cannot be filled.

Thin layer on top of Laravel's Job class

All this feedback during and after the execution is important and needed to provide a good UX for CHISV. Unfortunately, Laravel's job system cannot provide this for us. This is why we chose to implement a thin layer on top. We will now see how this extension is composed of four simple entities (see figure 4.11).

AdvancedJob Class

`AdvancedJob` class connects native job system to an Eloquent representation

The `AdvancedJob` class (`App\Jobs\AdvancedJob.php`) is the central entity that holds all the logic to interface with the native job implementation from Laravel and to update the Eloquent model according to its state (see (a) in figure 4.12). The class can be seen as an abstract class, meaning one would never instantiate `AdvancedJob`. Its sole purpose is to act as a wrapper around the native Job API and the Eloquent Job model. For that, it provides a `handle` method for Laravel and a reference to the Eloquent Job model to persist its state and provide feedback. Since any actual job (e.g. auction) extends the `AdvancedJob` class (see figure 4.11), it also inherits the `handle` method but will never overwrite it. We call this

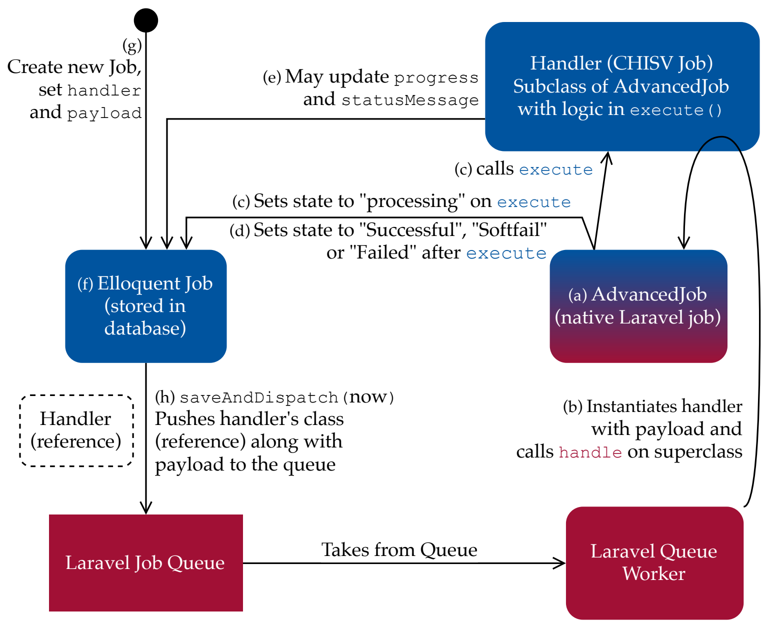

(g) Create new Job, set handler and payload
(e) May update progress and statusMessage
Handler (CHISV Job) Subclass of AdvancedJob with logic in execute()
(c) calls execute
(c) Sets state to "processing" on execute
(d) Sets state to "Successful", "Softfail" or "Failed" after execute
(f) Elloquent Job (stored in database)
(a) AdvancedJob (native Laravel job)
Handler (reference)
(h) saveAndDispatch(now) Pushes handler's class (reference) along with payload to the queue
(b) Instantiates handler with payload and calls handle on superclass
Laravel Job Queue
Takes from Queue
Laravel Queue Worker

Figure 4.12: Overview about CHISV's job extension program flow (blue) and its integration into the existing Laravel job system (bordeaux). To create a CHISV job one creates a new `Job`, sets the handler (e.g. Auction), and payload (input). The job object can then be persisted and the associated handler pushed to Laravel's job queue. Laravel will call the `handle` method on the handler's superclass (`AdvancedJob`), which will update the Eloquent Job model and also execute the code of the (subclass) handler. After execution, the job's status will be updated in the database.

actual implementation of a job, which makes use of our job extension, a "handler" – not to be confused with the `handle` method on the superclass (`AdvancedJob`).

`Handle` method (`AdvancedJob`) will call the `execute` method (handler)

To start a job, Laravel reaches into the job it wants to process and calls the `handle` method on it (see (b) in figure 4.12). Our `handle` method will then mark the Eloquent Job model as "processing" and calls the `execute` method on itself (see (c)). This method contains the actual job's code and resides in the inheriting class, the handler (e.g. `App\Jobs\Auction.php`).

The `execute` method was called by the `handle` method implementation in the `AdvancedJob` class. It will take care of updating the Eloquent model on completion or failure when the program flow returns from the `execute` method (see (d)).

Eloquent Job Model

The Eloquent model is the jobs representation in the database

The `Job` model (`App\Job.php`), represented as (f) in figure 4.12, is an ordinary Laravel Eloquent model. It represents an entry in the `jobs` database table, similar to any other Eloquent model (`Task`, `User`, `Conference`). A `Job` uses our state system we talked about earlier (see 4.2.4 "Conference, Users, and Permissions" or 4.2.4 "Tasks, Bids, and Assignments"). We express its current state with these five options:

1. **Planned** – The job is planned to be run in the future
2. **Processing** – The job is currently running
3. **Successful** – The job finished successfully
4. **Failed** – The job stopped and failed
5. **Softfail** – The job encountered an error and will restart shortly (e.g. for e-mails)

Additional fields for feedback

In addition, a job can also have a status message and progress (0%-100%). This additional information can be shown to the user in front-end applications. To persist the result of a completed job, we encode it as JSON and save it as the `result` in the job model. Each job has a field called `handler` which points to the job's class, which inherited from `AdvancedJob`. If the `Job` model is representing an auction, for example, its handler field would be set to the string `App\Jobs\Auction`. As many jobs require some input to work with (e.g. the auction needs a conference and date), we save the job's input as JSON in the `payload` field.

JobParameters Model

Using a PHP object to carry multiple parameters

Native Laravel jobs can only have one input object. As we have to provide two inputs to our extended jobs (the Eloquent Job id and the job's payload) we built the `JobParameters` object whose only purpose is to hold the job's id and payload. Laravel can serialize the object and later on pass it to the job when it is executed. This allows the job (which is a subclass of `AdvancedJob`) to set the progress, state, and status message on the associated Eloquent Job by id.

ExecutableJob Interface

We introduced this PHP interface to ensure that every job we try to execute contains a method called `execute`. This is important as it gets called by the `AdvancedJob` from the `handle` method.

Handler

The handler is the class a developer would have to create to implement a new job in CHISV's job extension. Since it implements the `ExecutableJob` interface, we will need to provide a `execute` method. We also want to extend the `AdvancedJob` class. This way we get access to the methods that we can use to persist the handler's current state to the database and integrate with CHISV's job extension. Without the inheritance, our `execute` method would never be called. To better understand how a handler can look like we will take a look at our lottery implementation in `App\Jobs\Lottery.php`:

```
class Lottery extends AdvancedJob implements ExecutableJob
{
    public $conference;

    public function __construct(JobParameters $params)
    {
        parent::__construct($params);
        $this->conference = Conference::find(
            $params->payload->conference_id
        );
    }
    public function execute()
    {
        // This is the place where we implemented the
        // Lottery algorithm. Anything returned from
        // this function will be saved in the Eloquent
        // Job by the AdvancedJob's handle code
    }

}
```

As we can see, all we need to implement for a handler is the constructor (in case we need to provide input) and the `execute` method (with the algorithm). The `execute` method will contain the handler's logic and algorithm. The constructor will get the `JobParameters` object, which contains our payload and associated Eloquent Job id. The job id will be used by the `AdvancedJob` to update the Eloquent model in the background. This is why it is important to call the super constructor, in the case we decide to overwrite it. Anything returned from the `execute` method will be persisted in the database through the associated Eloquent Job model.

Implement the execute method and optionally overwrite the constructor

To better understand the underlying logic, we take a look at the constructor and `handle` method:

```
class AdvancedJob implements ShouldQueue
{
    public $job_id; // points to the Eloquent Job
    public $tries = 3; // retry 2 times
    public $delayOnFail = 5; // 5 seconds

    public function __construct(JobParameters $params)
    {
        if (!$params->job_id) {
            throw new Exception(
                'JobParameters are invalid: Contains no job_id'
            );
        }
        $this->job_id = $params->job_id;
    }

    public function handle()
    {
        $job = Job::find($this->job_id);
        $job->markAsProcessing();

        try {
            $result = $this->execute();
            $job->markAsSuccessful($result);
        } catch (Throwable $e) {
            if ($this->attempts() <= $this->tries) {
                $job->markAsSoftFail();
                $job->setStartIn($this->delayOnFail);
                $this->release($this->delayOnFail);
            } else {
                $this->fail($e);
            }
        }
    }
}
```

A job may fail once or multiple times, which is reflected by its state

We can now see why it is so important that the subclass of `AdvancedJob` calls the super constructor when overwriting it: Without it, the association to the Eloquent Job is not set and the loop for providing feedback is cut (see (c)-(e) in figure 4.12). In the `handle` method we see how we first load the associated Eloquent Job by id and then set its state to "processing" (see c) in figure 4.12). We then continue and try executing the `execute` method. In case that this is successful we save its results to the associated job. If the execution fails we either "softfail" and reschedule the job to be executed later again or we set the Eloquent Job to the "failed" state since we tried all defined times (see `$this->tries`).

We now know about the `Lottery` handler and its superclass `AdvancedJob` and how these two components work together. To conclude the example, we will now look into how to create and start a job instance. For this, we take a look at this PHP code:

```php
public function runLottery(Conference $conference)
{
    $job = new Job([
        'handler' => 'App\Jobs\Lottery',
        'name' => "Lottery for " . $conference->key,
        'payload' => ["conference_id" => $conference->id]
    ]);
    $job->saveAndDispatch();
}
```

Saving and dispatching the handler

As we can see, we create a new Eloquent Job model, set its handler to point to our lottery handler implementation, and set the payload to carry the conference's id for which the lottery should run (see (g) in figure 4.12). Next, we call the `saveAndDispatch` method on the Eloquent Job itself (see (h)). This persists the job to the database and creates a new instance of `App\Jobs\Lottery` in Laravel's native job scheduling system. After this step Laravel queue workers will take care to start the job. The handler itself will then use the `job_id` from the payload to update its state on the Eloquent Job model in the database – closing the feedback loop (see (d),(e)).

Improves UX by a lot

With this extension to Laravel's job system, we enabled our jobs to provide rich feedback during and after the execution. We think it brings better user experience and makes the application feel overall more robust, as the user can now precisely see what the job is processing.

4.4.3 Lottery

Definition: *Lottery*

Lottery:
The lottery is an algorithm to accept student volunteers (SVs) for a conference. When a lottery is executed, lottery numbers greater than any existing number are randomly assigned to "enrolled" SVs. The algorithm will then accept SVs in ascending lottery number order until all available SV slots have been filled. Any SV who could not be accepted is "waitlisted" to get the chance to be accepted in a later run.

The lottery is accepting SVs for the conference

In this section, we will make heavy use of the state names we introduced in 4.2.4 “Conference, Users, and Permissions”. The lottery is one of CHISV’s most used features. It helps SV Chairs with accepting multiple student volunteers at random (#29). Nearly all conferences have to limit the amount of available SV slots for a conference. This means that not all volunteers who enroll[12] can be accepted. Some of them will have to be appended to the waitlist[13]. We make sure that any available SV spot is first filled with SVs from the waitlist before any newly enrolled SVs are considered. Any SV who could not be accepted, due to the limited spots, is appended to the waitlist.

Our algorithm runs in two phases:

1. We make sure that each SV in the state "enrolled" has a `lottery_number`. SVs on the waitlist already have a `lottery_number` from a previous run.

2. We loop through all SVs in the state "waitlisted" and "enrolled". We do this in ascending `lottery_number` order. For each SV we check if there are free SV slots available. If a slot is available we set the SV’s state to "accepted" or "waitlisted" otherwise.

[12]Enrolling places them in the initial "enrolled" state
[13]SVs on the waitlist are in the state "waitlisted"

As we have to iterate over all SVs in the worst case, our algorithm runs in $\mathcal{O}(2n)$. Since n (number of enrolled SVs) is considerably small (< 1500) the proposed algorithm runs in well under one minute in our production environment. We go into more detail about this in 5.2 “Scalability and Performance” and seed our application with very extreme values. As the lottery job is realized by implementing a CHISV job with our job extension (see 4.4.2 “Job Extension”), we can provide concurrent feedback to the user while the algorithm is running. We know this algorithm has the potential for improvement but at the cost of maintainability and transparency. We see no practical benefit of inducing more complexity than necessary.

Improving complexity at the cost of maintainability and transparency

Phase 1

We will now take a quick look at each of the two phases mentioned above. First, we need to set the `lottery_number` for every enrolled SV. For that, we get all SVs who are in the state "enrolled" and shuffle the list such that no preference exists when we iterate of this list. While the following code samples are extracted from our lottery handler (`App\Jobs\Lottery.php`), we renamed some variables for readability in this context. We also removed the additional method calls to provide rich feedback to the UI while the handler is running.

Get "enrolled" SVs and set `lottery_number`

```php
$newEnrollments = $this->conference->permissions
 ->where('role_id', $svRole->id)
 ->where('state_id', $enrolled->id);

$newEnrollments = $newEnrollments->shuffle();
```

Next, we get the highest `lottery_number` any SV of the conference has. We want to make sure that we assign only numbers that are larger than the existing ones to the new SVs. We start setting the new `lottery_number` to the SVs we hold in `$newEnrollments`. We increment the number before each assignment.

```
// max() returns null if none exists
$maxPosition = $this->conference
    ->permissions->max('lottery_position');

foreach ($newEnrollments as $enrollment) {
    // ++$maxPosition equals 1 if
    // $maxPosition is null
    $enrollment->lottery_position = ++$maxPosition;
    $enrollment->save();
}
```

End of phase one

After these lines, we know that all SVs in the state "enrolled" now have random numbers assigned to them. These numbers are larger than any number from the waitlist. It might also happen that the waitlist is empty (no "waitlisted" SV). In this case, the algorithm will generate a random distribution of numbers 1 to n (length of `$newEnrollments`) for all "enrolled" SVs.

Phase 2

To accept SVs, we check how many available slots there are. For this, we subtract the number of already "accepted" SVs from the number of defined slots of the conference. We continue by creating a list of all SVs we need to process, which are in the state "enrolled" or "waitlisted". We sort the list ascending by `lottery_number`.

```
$availableSlots =
    $this->conference->volunteer_max -
    $this->conference->permissions
       ->where('role_id', $svRole->id)
       ->where('state_id', $accepted->id)
       ->count();

$enrollmentsToCheck = $this->conference->permissions
    ->where('role_id', $svRole->id)
    ->where(function ($query) {
       $query->where('state_id', $enrolled->id);
       $query->orWhere('state_id', $waitlisted->id);
    })
    ->sortBy('lottery_position', 'asc');
```

The last step remaining is to finally accept SVs or append them to the waitlist. Accepting is done by associating the state "accepted". To append an SV to the waitlist we only need to associate the state "waitlisted". The position on the waitlist is already given through the `lottery_number`:

End of phase two: Accepting or waitlisting SVs

```
foreach ($enrollmentsToCheck as $enrollment) {
    if ($totalAccepted < $availableSlots) {
       // Still slots available for SVs,
       // make the current SV 'accepted'
       $enrollment->state()->associate($accepted);
       $totalAccepted++;
    } else if ($enrollment->state != $waitlisted) {
       // No more slots, put the SV who is not on
       // the waitlist yet on the waitlist
       $enrollment->state()->associate($waitlisted);
    }
    $enrollment->save();
}
```

This concludes the lottery algorithm and the excerpt from the handler. The lottery can be started multiple times. In each run, it will accept as many SVs as there are slots available and append remaining to the waitlist. Volunteers from the waitlist are accepted first. The lottery is not limited to any conference state. The lottery can always be run to fill available SV slots.

Lottery can be run multiple times

4.4.4 Auction

Auction:
The auction is an algorithm to assign student volunteers (SVs) to tasks. Its result is constrained by task priorities, SVs' preferences, their aggregated hours and time conflicts with other tasks.

Definition: *Auction*

The auction will iterate over all tasks of a day and try to fill any free task slots with SVs. SV Chairs and SVs have some requirements for the auction algorithm. The SV Chairs want to make sure that tasks with high priority

Auction is heavily constrained by multiple variables get more likely filled than tasks with lower priority. They also want that as many tasks are filled as possible. Student volunteers, on the other hand, want the auction to respect their bid preference and also consider the number of hours they have already fulfilled. Our algorithm tries to find a good tradeoff between both sides. Besides, we also need to account for other constraints before we create a task assignment. We aim for an average workload among all SVs and we have to ensure that we never assign an SV multiple tasks at times that overlap.

To summarize, our auction has to:

1. Fill tasks in descending priority: "High", "medium", "low" (#44)
2. Assign a task to SVs descending by preference: "High", "medium", "low" (#36)
3. Never assign a task to SVs who bid "Unavailable"
4. Create an average distribution of worked hours among all SVs (#45)
5. Never assign more than one task for a specific timeframe (task time conflict)
6. Handle SVs who did not bid as if they bid with "low" preference
7. Mark any processed bid according to the algorithm's decision (#42)
8. Report any task that could not be filled (#46)

Decision against using a linear optimization

CHISV's auction algorithm is, similar to the lottery algorithm, implemented in our job extension to provide additional feedback during execution. As the amount of information processed is drastically more compared to the lottery, the auction algorithm runs considerably longer. It's not uncommon for it to take multiple minutes. We ran extensive benchmarks (see 5.2 "Scalability and Performance") and heavily improved the algorithm over time to provide a reasonable quality and performance. While a linear optimization approach seems like an ideal fit for this application, we turned away from this solution as it requires administrators specialized in Linear programming (LP) to maintain the auction.

Two-phased algorithm ensured to first assign SVs with less than expected hours

We opted for an iterative approach with two phases. In the first phase, we will only handle SVs who have less than the expected hours completed. In the second phase, we will then focus exclusively on SVs having more than the expected hours. This approach makes sure to first only process SVs who should still work more hours. The second phase is then used to fill any tasks that could not be filled before. While we always respect the "Unavailable" bid (excluding an SV from the task), this design ensures that SVs who have worked less than the expected hours will always be assigned before SVs who completed all expected hours. For example, SVs who bid "low" (or did not bid at all) on a task will be assigned before any SV who bid "high" and completed more than the expected hours.

Generate a list with tasks that have free slots

Through figure 4.13 we can get a good overview of the two phases the auction algorithm runs in. We start by preparing a list of tasks that have free slots that need to be filled with SVs (see (a) in figure 4.13). This list is ordered by the task's priority in descending order. Next, we iterate through the list and process each task one by one (see (b)). Due to the list's order, we process tasks with "high" priority first. The block that processes one task is colored in blue, while the outer loop is set in bordeaux.

Creating a list of all SVs and setting the preference to "low" if no bid was placed

For each task, we generate a list of all accepted student volunteers (see (c)). This includes students who might already have an assignment conflicting with the task's timeframe we are currently processing. This list does also contain the SVs who bid "Unavailable" and the ones who did not place a bid a all. If the SV did not place a bid we set the SV's preference for this specific task to "low".

Removing some invalid candidates

In our next step (see (d)) we are removing those SVs from the list who bid "Unavailable" or have a task time conflict with another task/assignment. We also remove them if we are in phase 1 and the SV has already worked

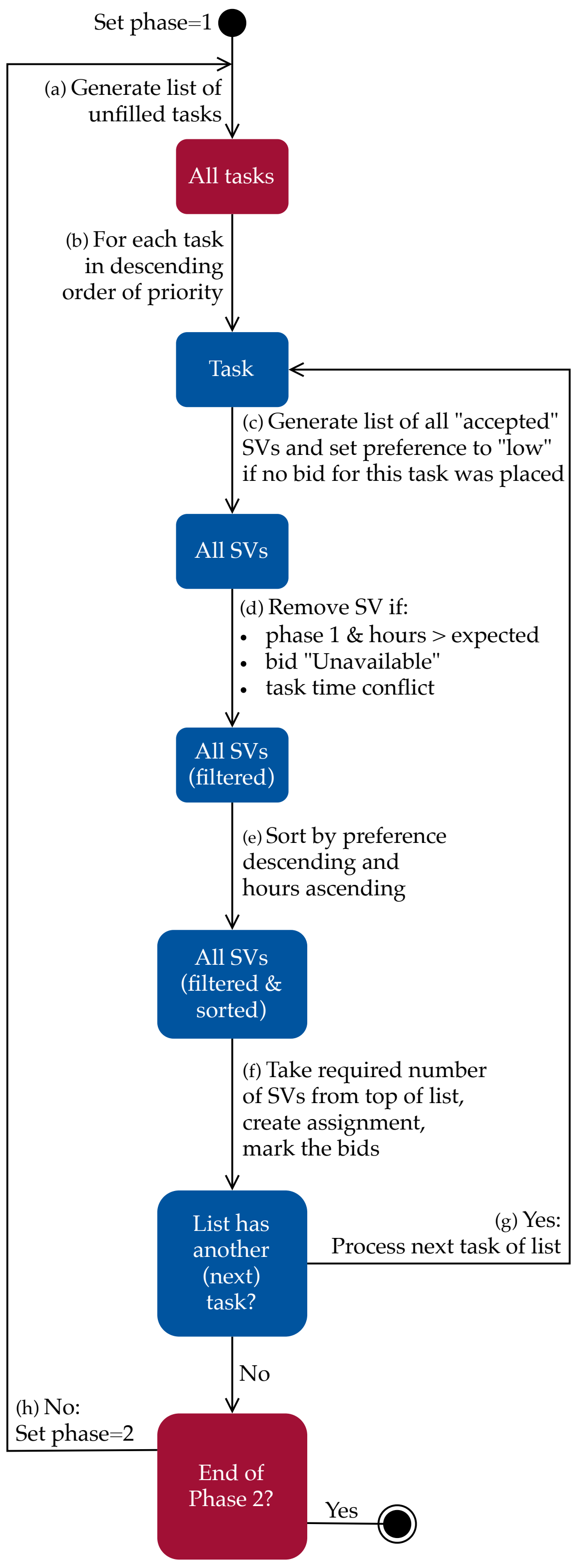

Set phase=1
(a) Generate list of unfilled tasks
All tasks
(b) For each task in descending order of priority
Task
(c) Generate list of all "accepted" SVs and set preference to "low" if no bid for this task was placed
All SVs
(d) Remove SV if:
• phase 1 & hours > expected
• bid "Unavailable"
• task time conflict
All SVs (filtered)
(e) Sort by preference descending and hours ascending
All SVs (filtered & sorted)
(f) Take required number of SVs from top of list, create assignment, mark the bids
List has another (next) task?
(g) Yes: Process next task of list
No
(h) No: Set phase=2
End of Phase 2?
Yes

Figure 4.13: CHISV's auction algorithm runs in two phases. In the first phase, we process tasks and SVs who worked less than the expected hours. In the second phase, we process tasks and SVs who completed all expected hours hoping to fill all remaining tasks.

more hours than expected. This is only true for phase 1. In phase 2 we would only remove SVs if they match the other two criteria.

Sort in Preference Group List

This gives us a list of possible candidates for the task. Next up, we need to sort the list to account for the SVs preference and also make sure we take SVs with a few hours first. For this, we introduced the concept of Preference Groups and Preference Group Lists.

Definition: *Preference Group*

Preference Group:
A Preference Group is a list of SVs (usually candidates for task assignments) where all SVs have the same preference and are sorted descending by the hours they have completed.

Definition: *Preference Group List*

Preference Group List:
A Preference Group List is a list of volunteers for one specific task that is sorted in a certain way. Since each volunteer can only bid once per task each volunteer will only exist once in the list. The list consists of up to four Preference Groups, one for each preference ("High", "medium", "low", "Unavailable"). The Preference Groups are sorted in descending preference order starting with "high".

Consider this example of a Preference Group List with the format [SV, Preference, Hours completed]:

- [Jacob, High, 3]
- [Alice, High, 5]
- [Bob, Medium, 0]
- [Jannis, Low, 10]
- [Kimia, Low, 12]

Not all Preference Groups have to be present

We see that Jacob and Alice form a Preference Group, as does Bob or Jannis and Kimia. Note how each Preference Group is sorted by the hours the SVs have completed and that not every preference has to be present in a Preference Group List. This valid example is missing the "Unavailable" Preference Group as no SV placed such a bid.

After filtering SVs we create the Preference Group List

Back to our auction algorithm: We left off after generating a list of all SVs for one specific task. We have already filtered the SV list to only include valid candidates for the currently processed task. We now create a Preference Group List based on our SV list (see (e)). This gives us a list where we have all candidates for the current task sorted in descending preference order. We can now safely start picking the first n SVs from this list to fill n available slots of the current task (see (f)). For each of the n volunteers, we create a new assignment in the state "scheduled" which associates the student to the task. We also mark the corresponding bid (should the SV has placed one) as "successful".

Either continue with the next task or drop into phase 2

We proceed with the next task in the list of tasks (see (g)) if there are more. In case there are none we have to check if we are in phase 1. If we are in phase 1 we enter phase 2 and start over again with creating a list of tasks (see (h)). The difference between phase 1 and phase 2 is the filter in step (d). In phase 2 we will no longer remove SVs who have completed more than the expected hours and if all other constraints are fine also create an assignment for them.

Phase 2 tries to aggressively fill tasks

Phase 2 can be seen as the extra emergency loop in case the SV Chairs set the expected hours too small. As our main goal is filling tasks, we continue to do so, even if this means that the SVs go over their expected hours. Nevertheless, the overall workload will still be distributed among all SVs, due to the Preference Group Lists.

After the auction the SV Chairs see a summary of the process

At the end of phase 2, when there are no more tasks in the task list, the auction ends. We will provide a list of all tasks that could not be filled and also some key values of how many assignments could be created and how many task time conflicts were encountered. This gives the SV

Chairs a better understanding of why a task could not be filled and how good the overall outcome of the auction is (#46).

4.4.5 Custom Enrollment Forms

Urgent need for more customizable enrollment forms

While maintaining the previous version of CHISV, we noticed how many SV Chairs reached out to us with the desire to include some special question in the (static) enrollment form, to update the wording or to disable a question. As the previous version did not allow for such adjustments, we focused even more on this topic with our reimplementation. Our first interviews and surveys in 2019 showed us the urgent need for a more flexible enrollment form system. SV Chairs liked the idea that an enrollment form could be fully customized on a per conference level (#23).

Keeping enrollment forms and the lottery detached

As we anyhow aimed to treat the enrollment form and the lottery as two separate entities (#32), we would not have to include any enrollment forms into the logic for the lottery. The previous version of CHISV used so-called tickets on a per question level to make it more or less likely that a student volunteer gets accepted. Many users of the previous version of CHISV saw themselves confronted with problems in understanding the algorithm behind the lottery. This was mostly induced by the enrollment form questions and how they altered the result of the lottery. While SV Chairs had full control over the tickets for every question, the default values were used in all cases that we observed, as the implications of modifying them were unknown.

New solution was needed for enrollment forms

For us, this was a clear indication that we would need to improve the transparency of enrollment forms while also making each form fully customizable on a per conference level. While SV Chairs can search for SVs with special criteria (#12) in the "Reports" section of CHISV (see 4.4.7 “Notifications and Reports”), we also wanted to give SV Chairs a powerful tool to make accepting SVs easier. As the lottery used to make decisions based on the enrollment form questions in the previous version of CHISV, we liked the idea of scoring enrollment forms to achieve a similar result – yet more controllable. We will now take a look at how an enrollment form is structured, the process of how it can be customized, and finally how scoring forms can help with accepting SVs with a special skill.

Enrollment forms stay editable for the time of "Enrollment"

One should notice that a student volunteer can always modify a submitted enrollment form as long as the state of the conference allows for it (#27). This is very useful as the SV's skills or answers might change over time and enrollment forms are usually submitted months before the conference takes place.

Structure

An enrollment form has static attributes and a dynamic body

To realize our custom enrollment form, we introduced a new class – The `EnrollmentForm` (see figure 4.1). An enrollment form has multiple attributes like an `id`, a reference to a parent, a `name` to identify it, a flag denoting whether it is a template, a `total_weight` (we'll cover later) and its actual `body`. The body is the part that makes the difference between forms. It can be dynamically adapted to the conference's need and is stored as a JSON string.

One model for enrollment forms

Only forms that are set to be a template can be used for conferences. A form that is no template is an enrollment form filled by a student volunteer. We decided to keep both (unfilled and filled) enrollment forms in one model (and database table), such that we don't introduce too much complexity. Having different models and tables would require us to duplicate a lot of our logic or use higher-level Laravel concepts, which again would make it harder to maintain.

Body is a dynamic JSON that is adapted per conference

We now want to focus on the `body` attribute of an enrollment form, as it's the part that makes all the difference and enables conferences to have customized enrollment forms.

An enrollment form's body has the following structure and can contain multiple "fields":

```
{
    "header": "Please answer the following questions:"
    "fields": {
        "know_city": {
            "description":"Are you local to the conference?",
            "hint": "Helps us in picking you for certain tasks",
            "type": "boolean",
            "value": false,
        },
    },
    "agreement": "SVs will have to work during the conference.",
}
```

While the above is only a small example, actual enrollment forms contain many more questions of different types. We implemented the enrollment forms in a way such that the "header" and "agreement" variables are optional and will not be shown if the SV Chairs decide to. An enrollment form can contain many questions (#24). Each question has an entry in the "field" dictionary. We called it "fields", as this allows for future expansion of different types of fields – not only questions.

Some variables are optional – "fields" have to be provided

We defined three types of questions, which will make use of special interface components on the front end: "Boolean", "Integer" and "String". These types will later also help us to score an enrollment form.

Let’s look at the structure of a question in detail:

```
"how_many_times_sv": {
    "description": "How many times have you been an SV?",
    "hint": "This will make selection more fair",
    "required": true,
    "type": "integer",
    "range": [
        0,
        5
    ],
    "value": 0,
},
```

To define questions in the "fields" dictionary, each question has to be defined with a unique name. We will now go over all the keys that are available for all types of questions.

Questions may provide more information through a hint

Available for all types The "description" will be the question that is rendered on the SV’s web application. If the SV Chairs feel the need to clarify some terms or provide additional information to the question they can do so by setting a "hint" (#25). This field is optional and will only make the front end show a question mark next to the question when set. Next, we have the "required" field. It denotes if specifying an answer to the question is required for submitting the form. This is especially useful if SV Chairs want to force an answer to a question. Another variable that is common on all question types is the "value". It expresses the default value that the UI component will be in.

Additional options for different types

Additional options based on the type Each type can make use of some additional variables. While the "Boolean" type does not provide any additional variables, the "Integer" type does. If a question is set to be of type "Integer" the SV Chairs can limit the range of possible answers. This is possible by setting a key "range" with an array of the form [min value, max value]. With this key set, the back end and front end will limit the available numbers (see above for an example). This also applies to the type "String". SV Chairs can set the "maxlength" key and by doing so limit the number of available characters for the answer. Furthermore, the UI component will adapt to the number of allowed characters and render either a small or a large text field.

A visual example of an enrollment form that uses multiple different types of questions can be seen in figure 4.7. This figure also shows the hint a question may have. It will show the message when hovering or clicking on the question. As we know how users interact with CHISV in these early steps, we put special emphasis on the experience with a mobile device.

Individual Forms per Conference

Situation-related request for custom enrollment forms

We have adapted the enrollment form of the previous CHISV. It is set as the default enrollment form for any new conference created on CHISV. While we give new SV Chairs the introduction into the system, they are also told that it is possible to customize the enrollment form if they feel the need to. In our short production phase, this has happened a couple of times but not for every conference. We think this is mainly due to the current situation with COVID-19, as many conferences got canceled, and attributable to the fact that the entire process of customized enrollment forms is rather new for the SV Chairs. How well the ability to customize the form is adopted, will become clearer in the following months.

Process of creating a custom enrollment form

Should an SV Chair decide to adapt an enrollment form to their needs, they can create an enrollment form themselves by utilizing the template from our GitHub repository. As each enrollment form is just plain JSON, it can be easily sent to the administrators of CHISV. To make the new template available for all conferences, the new form is inserted into the database while giving it a unique name. SV Chairs can then again select the newly inserted enrollment form in the conference settings. Any volunteer enrolling after this step will be greeted with the new enrollment form.

Scoring with Weights

Another important part of the new custom enrollment form system is the ability to score submitted enrollment forms. As we described earlier, we decoupled the lottery from the enrollment form questions. However, we saw great potential in using the knowledge available in these forms in a semi-automatic way. It can help SV Chairs with their SV selection process. While the lottery will only accept SVs at pure random, they need a way to precisely filter for SVs with special criteria before the lottery is run. For this, we made it possible to weight enrollment forms.

SV Chairs can open a new menu in the CHISV web application on the "SVs" page where they can define weights per question. These questions are the ones from the enrollment form the SV Chairs selected earlier for their conference. A weight is a positive or negative integer number. Currently, CHISV will evaluate forms and take questions with the type "Boolean" and "Integer" into consideration. The answer to the question will be multiplied by the weight the SV Chair set for that specific question. Let's take a look at this example:

1. How many times have you been an SV?
   Type: Integer
   Answer: 2
   SV Chair set weight: -10
2. Are you local to where the conference will be?
   Type: Boolean
   Answer: Yes
   SV Chair set weight: 50
3. Please explain why you want to be an SV:
   Type: String
   Answer: The SV gives specious arguments
   String questions cannot be weighted

When "Boolean" questions get evaluated `false` is equivalent to 0 and `true` to 1. In the example above the `total_weight` of the enrollment form is 30, which is the result of the term $2 \cdot (-10) + 50 \cdot 1 = 30$.

The `total_weight` is the sum of all questions' products

Each question's value is multiplied by the weight the SV Chair defined. The `total_weight` is then simply the sum of all these products. This example will yield a larger number for any SV who has not been an SV but is local to where the conference will be. To give an even more high-level definition: We are looking for SVs who are local but have not been volunteering before.

The weights of all enrollment forms are then available to the SV Chairs in the web application. They can sort all SVs by their enrollment form weight and then accept as many of them as they need. After that, they might adjust the weights and pick some additional SVs who match different criteria. Only then, when they know they have a solid set of SVs available and accepted, they would run the lottery to fill all other available SV slots of the conference automatically.

SV Chairs can pick as many SVs with certain criteria as they need

As we can see, scoring the SV's enrollment forms provides a very powerful tool for SV Chairs to pre-accept (#31) certain SVs to account for special needs. How beneficial this feature can be for accepting or rejecting SVs solely depends on the questions the SV Chairs decide to put in the enrollment form. We think that through the use of enrollment form weights we compensated the loss of having a lottery that takes enrollment form questions into consideration by far. Furthermore, we now also have a versatile tool with which SV Chairs can pick volunteers based on many different criteria until they feel confident about the selection – all in a semi-automatic but very transparent process.

No hidden and hard-to-track algorithm

4.4.6 Calendar

Calendar is the most requested feature

One of the features we added to CHISV, which was missing in the previous version, is the calendar. As 21 of all 69 interviewed student volunteers requested it (#38), we first considered the environment where SVs would use the calendar in (#40, #41). As we saw a twofold area of application we decided to provide multiple views to better adapt to the changing requirements. This is why we equipped the calendar with three different views: A month, week, and day view (see figure 4.14). While the day view is mostly suited for a device with a mobile form factor, the month and week view help getting a broader overview and are usually seen with devices from the desktop form factor category.

Month, Week, and Day View

Dedicated view in CHISV – not bound to conference

We noticed that to give users a good experience with the calendar, we would first have to reduce distraction while we show the calendar. One could argue that placing the calendar view into the conference view, as we did with "SVs" or "Tasks", would be a great fit since this would allow us to only show tasks associated with the conference. We thought it would be a better fit to extract the calendar view and give it its own place in CHISV (just like the FAQs, as explained in 4.3.2 “FAQs”).

One calendar per conference might confuse users

On the one hand, we made this decision because we think that a user would expect to see all tasks or assignments that are present – not only the ones of the active conference. To clarify this, we would have had to put additional visual hints in place. This would clutter the interface even more. We would have to build a mental model where the user expects to see one calendar per conference. This could be remotely compared to as if one would use a different calendar for each occasion. While this might also have certain benefits, we saw no point in having dedicated calendars on a per conference basis.

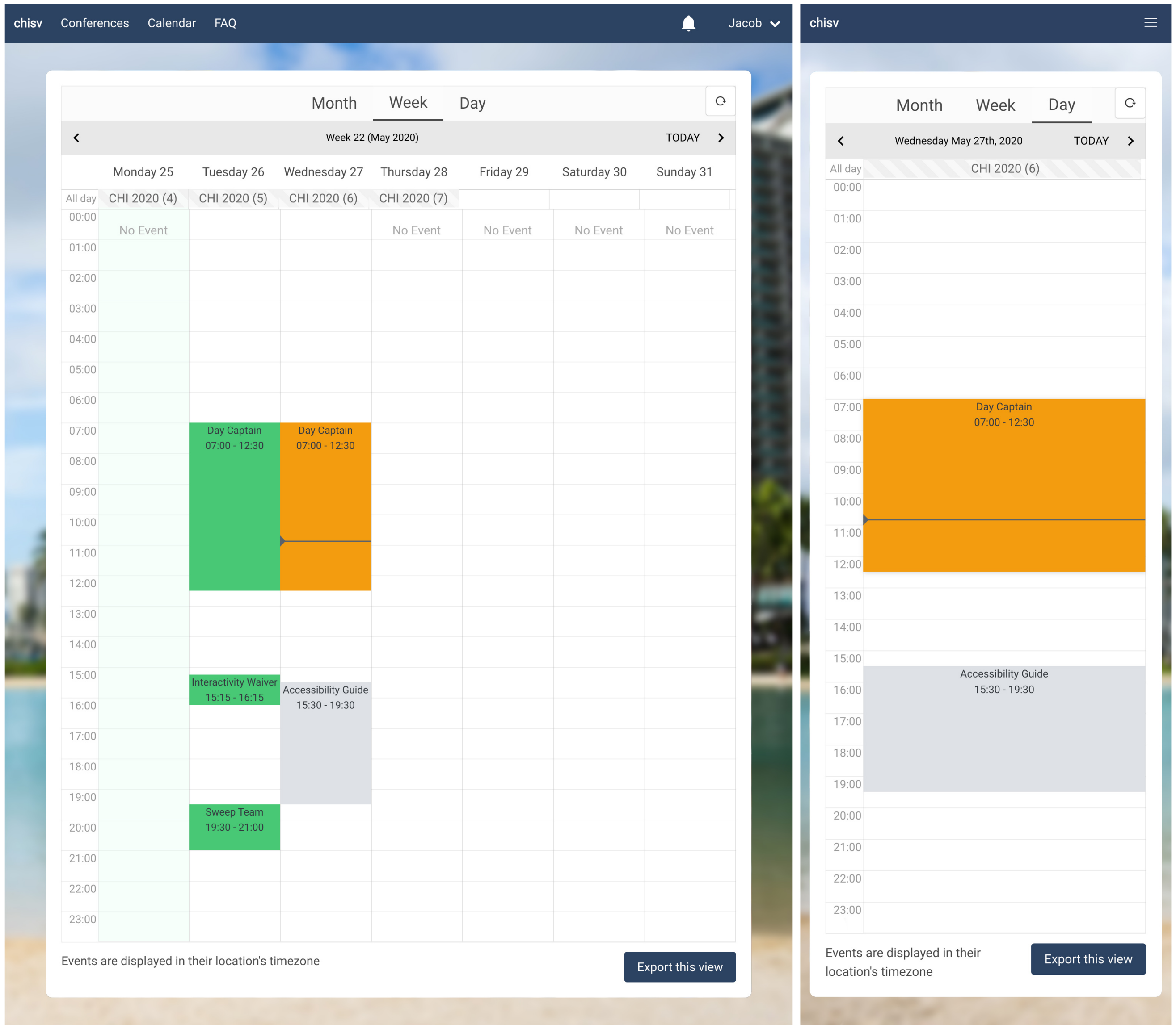


Figure 4.14: CHISV calendar shows the volunteer all assigned tasks in a month, week, and day view. The calendar adapts to the desktop (left) and mobile (right) form factor. Tasks in the color grey ⬜ represent tasks that are assigned, but the SV has not started working on them. Orange 🟧 indicates that the SV started working on the task but has not yet finished. As soon as a task has been completed it will appear green 🟩. The current time is shown with a grey horizontal line. Users can export the active view by clicking the button at the bottom. Individual tasks can be exported by clicking on the task and triggering the export there. Each day's header will list all ongoing conferences and the day.

A dedicated view gives the calendar more space

On the other hand, breaking free from the per conference navigation gave us a lot of more visual space to better render events and present the calendar in a full-screen setting. While in theory, our calendar also features a year view, we found this rather unsuitable to use, as events were merged and becoming indistinguishable. Every view will have a small grey horizontal line showing the current day and time (see figure 4.14). This is giving SVs on-site using the application in-between tasks a quicker entry to the calendar.

Prepare all conference days beforehand

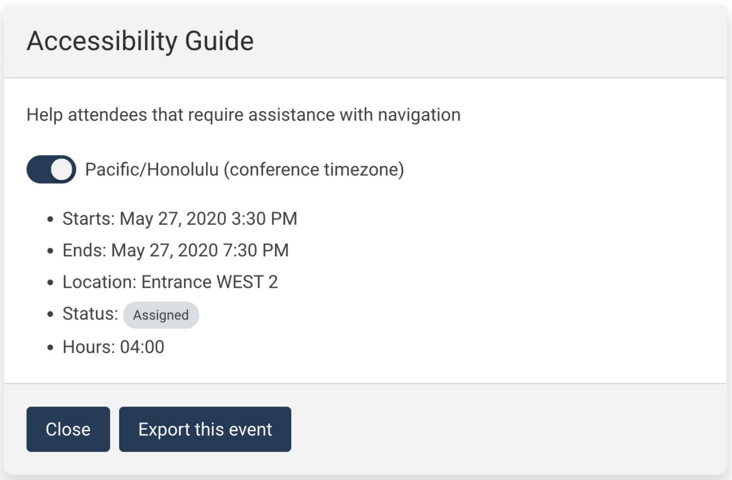
However, showing all conferences in the day's header also forced us to have the timeframes of them available at all times when the user scrolls through the calendar. The timeframe of all conferences is available from our Vuex store as soon as the web application finished loading. This enables us to only fetch events for the days the user currently views. As the days that a conference takes place on rarely change after they've been set, caching the conferences timeframes is a feasible approach.

Month and week view for desktop form factor

As some SVs would like to use the calendar to create their daily schedule, they prefer a view that gives them the ability to look at all tasks at once. This is usually done on a device that would be represented by the desktop form factor (including tablets), how we learned through our user surveys in 2019. For this, we integrated the month and week views. This is great for getting a broad overview of all tasks and estimating how long they take. Usually, SVs would also use the month or week view to export the calendar. We will take a deeper look at the calendar export in 4.4.6 "Universal Event Export".

The day view is mostly used on-site

SVs requested a way to quickly check on their assigned tasks while in-between tasks or at the end of the day. For this, we integrated the day view (see figure 4.14). It will show all assigned tasks of one specific day, ranging from and to midnight. The view is designed to give as most space as possible to the calendar and the events in it. Again, just like in any other view, we will show a grey horizontal line to represent the current time.


Accessibility Guide

Help attendees that require assistance with navigation

Pacific/Honolulu (conference timezone)

- Starts: May 27, 2020 3:30 PM
- Ends: May 27, 2020 7:30 PM
- Location: Entrance WEST 2
- Status: Assigned
- Hours: 04:00

Figure 4.15: Event detail view: A user can see the description, location, current state, and the precise timeframe. The timezone used to show the time can be toggled between the conference's and the user's zone. Besides, the user can export the event.

Any event can be clicked or touched to get more details

Currently, any event in CHISV's calendar represents an assignment for a task. We thought it's beneficial to be able to get more details about a task while in the calendar without having to go back to the conference's "Tasks" view. To get more information, a user can click or touch the desired event and get a short overview of the event (see figure 4.15). As SVs use this in-between tasks, we show the location and accurate time. To account for SVs who work on tasks from remote, we also included a way to display the task's time in the user's local timezone. SVs can also use this event detail view to export a single event.

As we can see in figure 4.15, assignments in the calendar can have different colors. Those represent the assignment's current state.

Colors help to reflect the assignment's state

How we explained earlier (see 4.2.4 "Tasks, Bids, and Assignments" for a state description), an assignment can be in one of three states:

- Assigned ⬜
- Checked-In 🟧
- Done 🟩

We attached three colors to the states (grey, orange, green) to make the current state instantly visible in the calendar. This enables SVs to check on their assignments' state with a simple page reload – no further interaction is necessary.

Universal Event Export

Conference's schedule is important

We found through our survey that integration of CHISV's calendar with other external calendars would help even more to plan the own daily or weekly schedule. Often times SV's want to attend certain sessions at a conference and want to make sure that no task is interfering at that time.

Different conferences use different tools for scheduling

Integrating conference schedules into CHISV is rather hard, if not impossible to maintain. There are many different formats and applications with which different conferences manage their schedule. Sometimes it is even not possible for the SV Chairs to get hold of the conference schedule in a structured digital format. All this makes it very hard to integrate conference schedules into CHISV. We think that providing adapters to read all these various formats will be impossible to maintain in the long run.

Focus on export rather than import

This is why we put great emphasis on providing a universal export of CHISV calendar events. This is also true for other resources like we will see in 4.4.7 "Notifications and Reports". For calendar events, we decided to use the universal Internet Calendaring and Scheduling

Core Object Specification (iCalendar, see Desruisseaux [2009]) which is more commonly known by its file extension ".ics".

Internet Calendaring and Scheduling Core Object Specification (iCalendar):
The iCalendar specification defines an interchangeable format to present components in a calendar. One of these components is the event component. Most calendar applications can process event component definitions, including Google Calendar, Apple Calendar (formerly iCal), IBM Notes (formerly Lotus Notes), Evolution, Yahoo! Calendar, Mozilla Thunderbird, and partially Microsoft Outlook.

Definition: *Internet Calendaring and Scheduling Core Object Specification (iCalendar)*

Exports preserve the timezone

Each of our exported iCalendar files may include one or multiple events. Whenever an event has a location set, we will provide it in the native `LOCATION` attribute in the event component. This means if SV Chairs decide to use locations known to (e.g.) Google or Apple Maps, users will be able to navigate to the location after importing it into their calendar. Each event that we export has also a time. We set it in the components start and end time (`DTSTART`/`DTEND`). As we also know the event's timezone, we append it as `TZID`. This is especially important when SVs import tasks while being in a different timezone. The task's description gets set into the `DESCRIPTION` attribute such that volunteers can quickly check on those in their private calendar as well.

Static export

Any export from CHISV's calendar is of course static, meaning that any tasks that have been imported into a private calendar will not update when it changes in CHISV. This is also the main reason why we do not export the task's current state. To overcome this issue we propose a solution for further development in 7.2 "Realtime Calendar Integration". This approach will still preserve all the interoperability with various calendars but will also enable some to receive updates in semi-realtime.

4.4.7 Notifications and Reports

Notification System

CHISV needs to send out e-mails

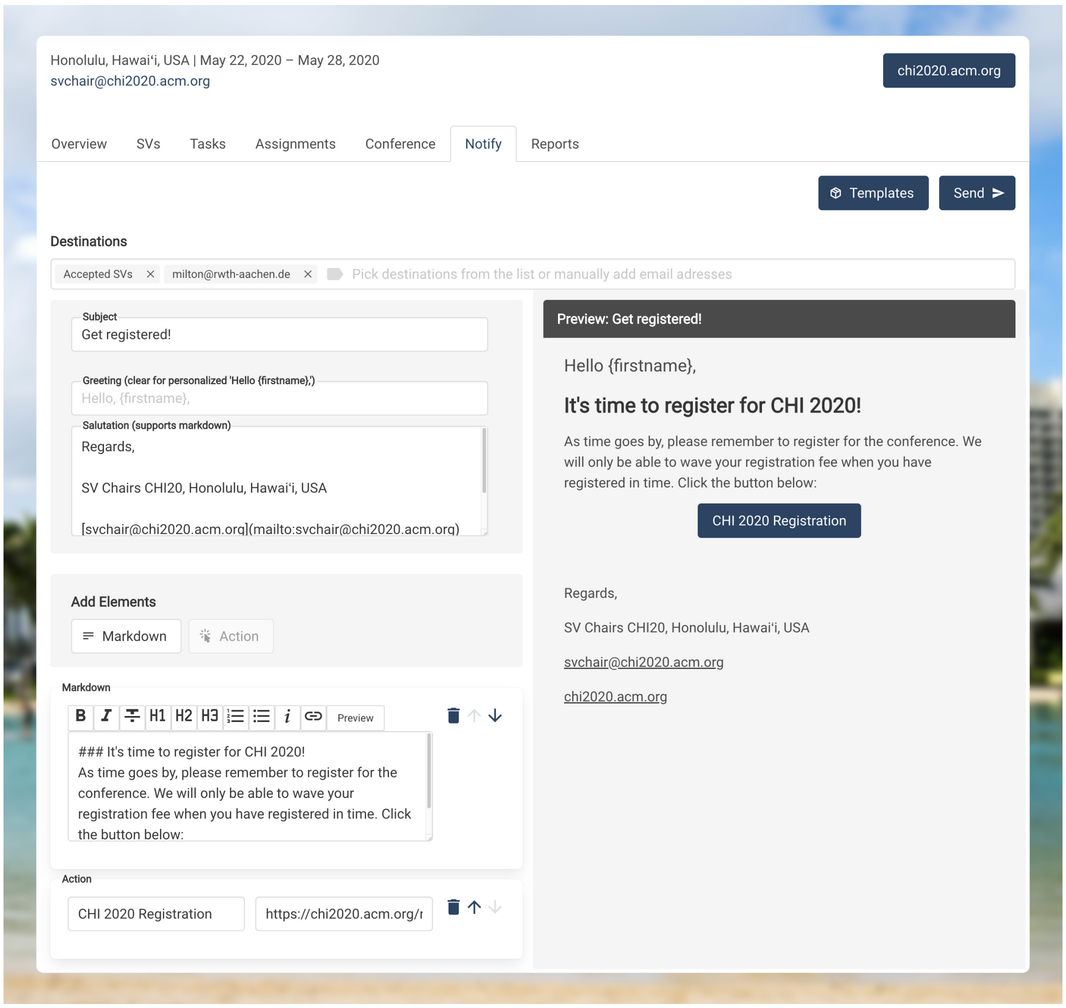
The previous version of CHISV was also used by the SV Chairs to send announcements and other SV related information to the volunteers. These messages were delivered as an e-mail to the volunteer's e-mail address. As we replicated many of CHISV's earlier functionality, the notification system was no exception (#49). Laravel provides native abilities to send notifications to users what enabled us to stick closely to the reference implementation from the Documentation. As we explained earlier (see 4.2.4 "Conference, Users, and Permissions", page 69), notifications in CHISV are not simply stored as text but in a special internal structure, which allows us to handle it more efficient on different destination channels.

Notifications get dispatched to Laravel queues

Each notification is dispatched to a Laravel queue and later picked up by a queue worker (see 4.2.2 "Job Queue") to deliver the message to the different channels. We incorporated a template system (#50) where every SV Chair can add, edit, and delete templates. These act as small building blocks of messages and announcements, which get regularly sent for each conference.

To make it easier to direct a message to multiple users, we added predefined groups (#53) to the available destinations:

- **All SVs** – All SVs currently associated to the conference regardless of the state
- **Accepted SVs** – Only SVs who are currently in the state "accepted"
- **Waitlisted SVs** – Only SVs who are currently on the waitlist
- **Captains** – Only users who currently carry the "Day Captain" role


Honolulu, Hawai'i, USA | May 22, 2020 – May 28, 2020
svchair@chi2020.acm.org

chi2020.acm.org

Overview SVs Tasks Assignments Conference Notify Reports

Templates Send

Destinations

Accepted SVs × milton@rwth-aachen.de × Pick destinations from the list or manually add email adresses

Subject: Get registered!

Greeting (clear for personalized 'Hello {firstname},'): Hello, {firstname},

Salutation (supports markdown):
Regards,

SV Chairs CHI20, Honolulu, Hawai'i, USA

svchair@chi2020.acm.org

Add Elements

Markdown Action

Markdown

```
### It's time to register for CHI 2020!
As time goes by, please remember to register for the conference. We will only be able to wave your registration fee when you have registered in time. Click the button below:
```

Action

CHI 2020 Registration | https://chi2020.acm.org/r

Preview: Get registered!

Hello {firstname},

It's time to register for CHI 2020!

As time goes by, please remember to register for the conference. We will only be able to wave your registration fee when you have registered in time. Click the button below:

CHI 2020 Registration

Regards,

SV Chairs CHI20, Honolulu, Hawai'i, USA

svchair@chi2020.acm.org

chi2020.acm.org

Figure 4.16: CHISV's notify view used for sending out announcements and other messages to users. The interface is structured into a destination field on top, an edit panel (left), and a preview (right). Any changes in the edit panel are instantly reflected in the preview. Templates can be used to store and retrieve a complete view with all settings. A message's text is Markdown compliant and will render correctly on all devices.

These get resolved to a list of user models in the back end and do not represent any role model. Due to the design of the notification system with Laravel, any Eloquent model may become a `notifiable` model (see page 69). However, we opted to resolve these predefined groups our self, rather than introducing a more complex structure by making role and state models `notifiable`.

Groups are resolved to users models

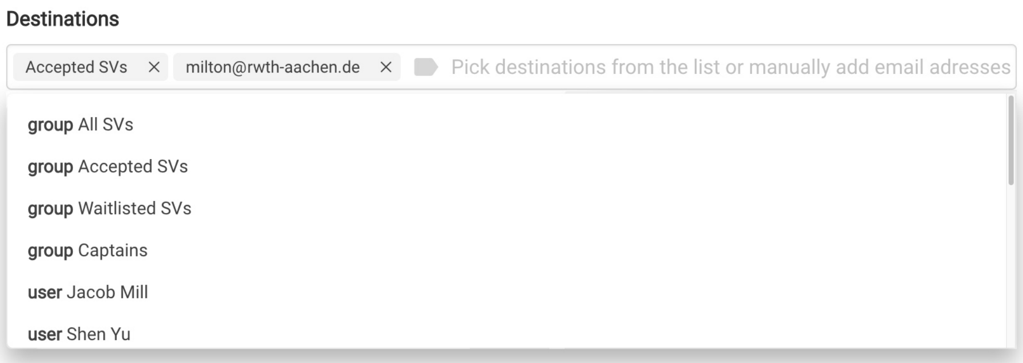


Figure 4.17: Users of the Notification system can pick multiple recipients from a dropdown field. While predefined groups can be selected, every associated SV may also be included as well as any manually typed e-mail address.

Available destinations include all SVs, predefined groups, and manually added e-mail addresses

SV Chairs can freely pick from the available destination (see figure 4.17) in the "Notify" view (see figure 4.16). These include the aforementioned groups, an entry for every single user associated with the conference, and any manually typed e-mail address. Any group or user who is picked from the destinations will receive the notification over all configured channels. CHISV currently offers e-mail and internal messaging (#51) and is easily extendable to other channels like SMS, Slack, or other instant messengers. Nevertheless, we will always deliver the message as an e-mail (#54, #8) as all conferences could agree on this common channel. For manually typed in e-mail addresses, this is, of course, also the only available channel as we have no reference to an actual user object.

Realtime updates of assignments

As CHISV may get access to more channels in the future, this could help to push realtime updates (#21) to SVs. While providing realtime updates is already possible via e-mail, we decided to postpone the feature further as to us e-mail is not the ideal channel for delivering realtime updates. We don't want to fill SV's inboxes just to let them know that their assignment was marked as "done". In our eyes, channels like Slack or other instant messengers are more appropriate for this type of message.

Marking read messages appropriately

Some channels also can keep track of if the message was already read or not. As this is not possible in a reliable manner for e-mail, we focused on our internal notification system. Every notification that is sent through CHISV will also be stored in the database should it be able to match it to a registered user. This is true for any message that is not manually sent to an e-mail address. After logging in, users of CHISV can read notifications through the internal notification center. When the user opens the message we set it as "read". This allows SV Chairs to keep track of delivered messages more easily (#52).

Appearance similar on different channels

Earlier we talked about how we abstracted the content of the notification such that we can dynamically render different parts of the message based on the channel (see 4.2.4 “Conference, Users, and Permissions”). Figure 4.18 gives an example of a message that was sent through the notification system. The message on the top is the received e-mail, while the message on the bottom represents the same message viewed from the internal notification center.

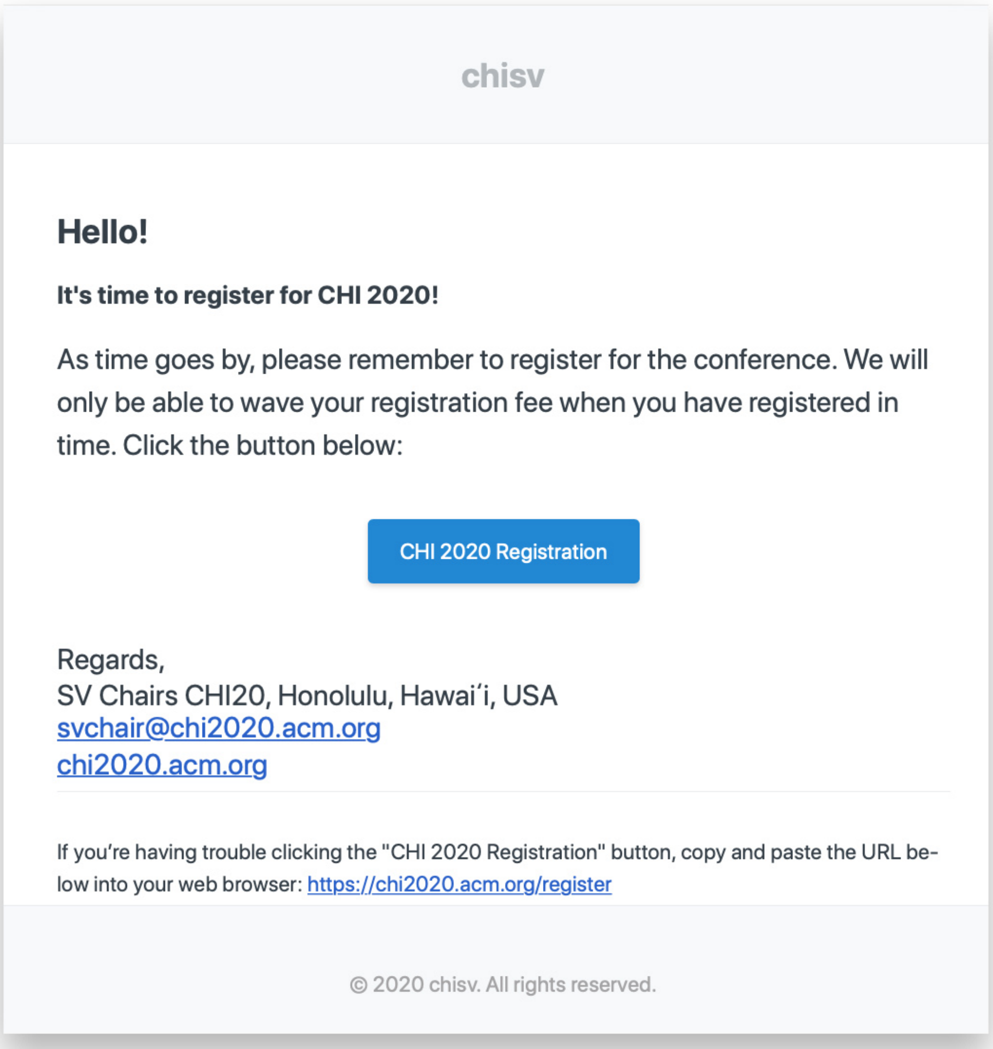
Message adapts to the channel

One may notice that the message at the bottom of the figure does only include the messages content and the call to action button. We will not show all the other parts of the message, which one would usually see in an e-mail. Not only does this save some transferred data but also does it let the user focus more on the action that has to be carried out.

Reports and Export

Reports help other conference chairs

SV Chairs require the ability to export statistics from CHISV. This is used by other conference chairs to generate reports based upon this data. This is also used by the SV Chairs of the next year, as it helps them to accept a fair amount of every minority group. We build the report view (see figure 4.19) to be as versatile as possible. That means that users with access can sort and filter reported data directly in CHISV, and also export it as a CSV file.



chisv

Hello!

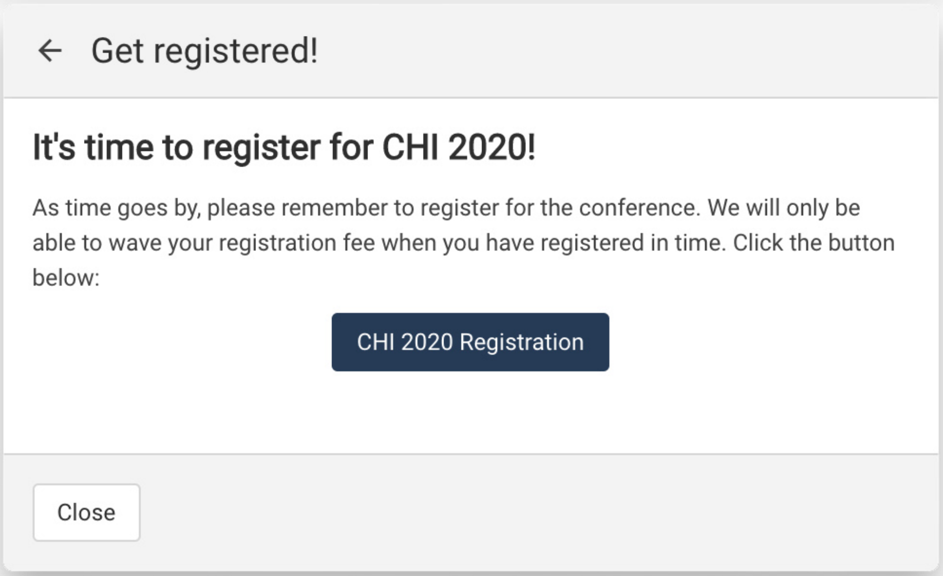
It's time to register for CHI 2020!

As time goes by, please remember to register for the conference. We will only be able to wave your registration fee when you have registered in time. Click the button below:

Regards,
SV Chairs CHI20, Honolulu, Hawai'i, USA
svchair@chi2020.acm.org
chi2020.acm.org

If you're having trouble clicking the "CHI 2020 Registration" button, copy and paste the URL below into your web browser: https://chi2020.acm.org/register

© 2020 chisv. All rights reserved.

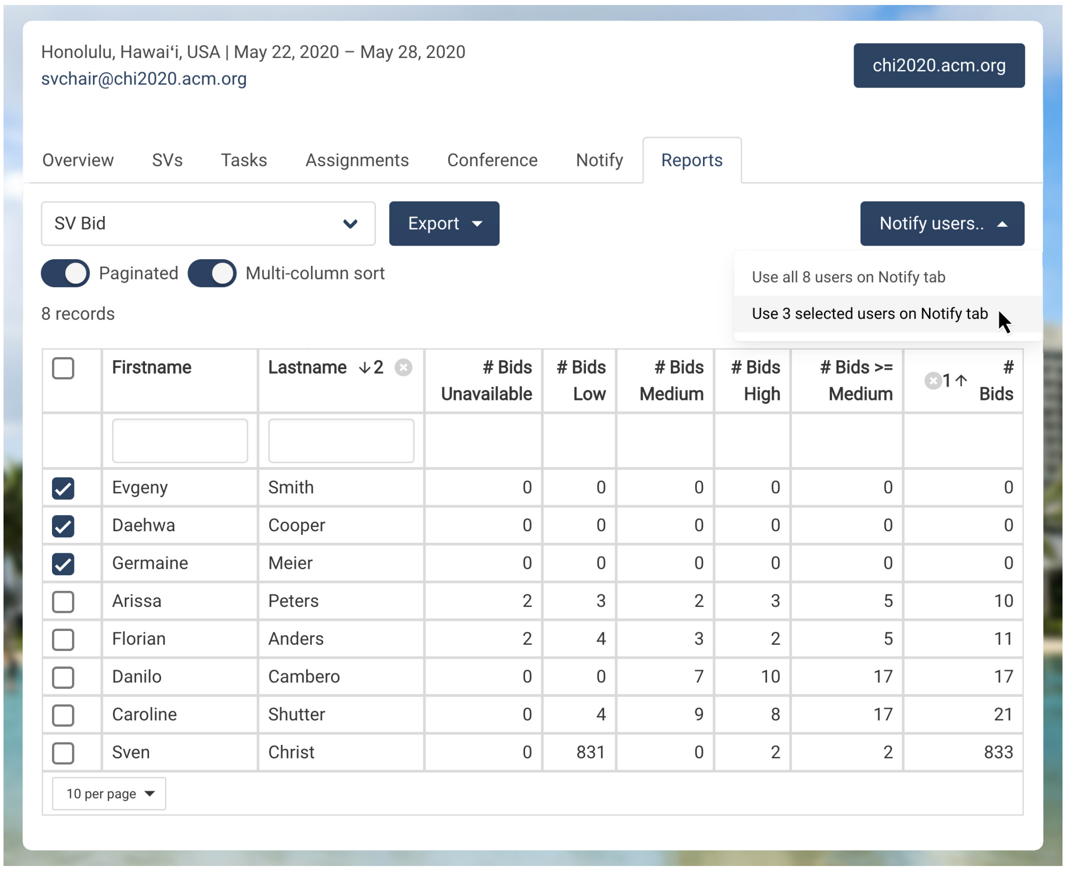


Get registered!

It's time to register for CHI 2020!

As time goes by, please remember to register for the conference. We will only be able to wave your registration fee when you have registered in time. Click the button below:

Figure 4.18: CHISV Notification System: The image on the top represents a notification received by e-mail, while the image on the bottom is the same notification viewed through the internal notification center.



☐	Firstname	Lastname ↓2	# Bids Unavailable	# Bids Low	# Bids Medium	# Bids High	# Bids >= Medium	1↑ # Bids
✓	Evgeny	Smith	0	0	0	0	0	0
✓	Daehwa	Cooper	0	0	0	0	0	0
✓	Germaine	Meier	0	0	0	0	0	0
☐	Arissa	Peters	2	3	2	3	5	10
☐	Florian	Anders	2	4	3	2	5	11
☐	Danilo	Cambero	0	0	7	10	17	17
☐	Caroline	Shutter	0	4	9	8	17	21
☐	Sven	Christ	0	831	0	2	2	833

Figure 4.19: CHISV's Reports: Example for filtering for SVs who have no bids placed. We can select them (checkboxes on the left) and transfer them to the "Notify" view to send them a message (see button and dropdown on the right).

Universal CSV export

We give the users many options for the CSV export, such as which delimiter to use or whether or not Excel support should be provided. The CSV export can be done for an entire report or only for a filtered and sorted subset that is currently visible to the user.

Multi-column sort

Furthermore, as reports get presented in a table, we not only support sorting a single column but also to use multiple columns to sort based on multiple criteria. This is especially asked for by SV Chairs as they use the report view to filter for certain volunteers. This helps them to find SVs with certain language skills or few bids (#12). For some reports, the user can also adjust query parameters that affect the outcome of the report (e.g. adjust minutes for the "SVs accepted in the last X minutes" report).

CHISV features 11 reports:

1. **SV T-Shirt** – This will create a report with all required T-Shirt sizes that the SVs selected
2. **SV Hours** – Shows how many hours every SV has completed
3. **SV Bids** – Shows the bidding behavior for each SV
4. **SV Detail** – Shows all personal data and enrollment form answers of each SV and is often used for picking SVs for tasks and preliminary acceptance
5. **SV Doubles by Name** – Lists volunteers with same name
6. **SVs accepted in the last X minutes** – The minutes (X) on how far to look back can be adjusted, resulting in a list of SVs who have been accepted in the last X minutes
7. **SV Demographics (Country)** – Will list all known countries and the associated volunteer count
8. **SV Demographics (Language)** – Will show all languages and how often they are being spoken
9. **Task Overview** – Shows all conference days and how many hours have been completed in total or how many slots have been filled
10. **Tasks with free Slots** – Will show all tasks that have not been filled to the fullest
11. **Tasks Table Dump for later Import** – Dumps all tasks associated with the current conference such that they can later be used in the "Tasks" view for reimport (#6)

The reports 2-6 contain uniquely identifiable users. Any report that returns uniquely identifiable users will show checkboxes on the left of each table's row. SV Chairs can then select multiple users and send them to the "Notify" view (see figure 4.19). This will push the users to the "Destinations" field (see figure 4.17) such that they can receive the notification via all available channels. In figure 4.19 we see how we can either send all or only the selected users to the "Notify" view. As with any report, we can sort by multiple columns. In the example figure from above, we have sorted the table by the number of all bids ascending first, followed by the last name descending. This is especially useful when preparing a report for export.

Sending SVs from report to the "Notify" view

As we have seen CHISV reports are very versatile. Not only do they allow for sorting and filtering the data but also integrate with other features of CHISV perfectly. Exported tasks can be imported again and will update existing tasks (#13). Reports that contain users can be used to send those users notifications. We think that with the versatility and expressiveness CHISV reports provide, SV Chairs can much more easily find volunteers with certain criteria and organize them more efficiently. Task-related reports give the SV Chairs a single pane to check for unfilled tasks or evaluate the tasks with uncompleted hours. CHISV's reports provide them with a way to quickly check on all important conference topics – SVs, tasks, bids, and the completed hours.

Reports integrate with Notifications perfectly

Chapter 5

Evaluation

5.1 Requirements Coverage

We have defined our requirements in detail in 3 “Requirements Analysis”. Since then we have always appended a reference to each requirement in the form #$Number$ when we explained the functionality we introduced to fulfill the requirement. From all of our 54 requirements, we covered 45 of them so far, leaving us with 9 to cover.

9 requirement left to cover

For 8 of these requirements, we will present our solution, and explain why we had to back away from one requested feature (#48).

Authentication and Profiles

We introduced CHISV’s available authentication methods earlier. CHISV uses Cookie-based authentication for it’s Vue front-end application (see 4.3.2 “Login”), as this helps us to protect the application against multiple attack vectors (see 4.4.1 “Cross-Site Scripting (XSS) and Cross-Site-Request-Forgery (CSRF) Mitigation”). While it might be obvious that we would need authentication to secure our application and to give users different lev-

We require authentication for accessing CHISV els of permission, we also formulated it in a requirement (#5). To authenticate users we need them to sign up (see 4.3.2 “Register”) before they can use the application.

Additional information about the SV are collected

During the registration and after, users can submit additional details to let the SV Chairs know a bit more about their abilities. These include the languages a user can speak (#9), the university currently associated with (#10), and also the volunteer’s current degree program (#11). These details become part of the user’s profile.

Profile details should be available for every SV Chair

Collecting these additional metrics (in addition to the enrollment form questions) was requested by SV Chairs (#26). The information should be kept on a global access layer such that these personal details are bound to a user rather than a user’s enrollment. This is important as it enables other SV Chairs to look into the details as well.

Extending the session timeout

Every user who logs in to CHISV on our Vue application will be using Cookie-based authentication to access CHISV’s API endpoints under the hood. From the previous version of CHISV, we’ve learned that the session timeout was set to a too low number. This forced volunteers to log in again multiple times while being at the conference. Thus, it was of high demand to extend the session timeout (#19).

OAuth tokens are valid for one year

CHISV will issue Cookies and tokens that are valid for multiple months. Tokens for third-party applications will be valid for one year. This is especially important for native applications (like for iOS and Android), as the general expectation for a native app is to see the login only once.

Conference Settings

Enable bidding for only certain days

Another requirement we covered is a change to how a conference is opened for task bidding (#18). In the previous version of CHISV, the SV Chairs would switch the conference’s state from "Running" to "Bidding" – and back again to disable bidding. We thought that having a simple flag to signal whether or not volunteers can bid on tasks is modeling the reality better. Furthermore, this allows us to set a range of days for which volunteers can bid on tasks. Not only will this avoid the SV Chairs forgetting to switch the conference state to disable bidding, but also give the volunteers a better understanding of the days for which they can place bids.

CHISV has no maintenance mode for conferences

In the previous version of CHISV, an SV Chair member would usually turn on the so-called "Maintenance Mode" for a conference while the auction is being run. This prevented SVs from adding or altering bids while the algorithm was running. Unfortunately, the maintenance mode was poorly explained to the SVs. Thus, many did not understand why the system is periodically inaccessible and in maintenance mode. While many only wanted to check on their assignments, the entire web front end blocked access for volunteers. We think that CHISV should always be accessible and only limit access in the front end and back end partially. We thus did not implement a maintenance mode (#14) but focused on reflecting and explaining certain limitations within the web interface where they applied.

Extending SV's Abilities

Desire to alter the own assignment's state

During our survey and interviews, many student volunteers expressed their desire to be able to alter their state of an assignment themselves (#48). For example, they desire to set their assignment's state to "Checked-in" or "Done". While this is not a complicated thing to include for SVs in the calendar or "Tasks" view, we first wanted to discuss this rather important change with the SV Chairs. During this discussion, we noticed how adding this functionality could negatively impact certain conferences.

Develop a solution with every group first

We think that this feature has great potential for the overall UX with CHISV when implemented right. We have postponed the integration for now. However, we think that future development wants to focus on this topic again, to integrate a solution with which every party is satisfied.

Up to now, we have seen how we could integrate all 53 requirements, which we collected from users in multiple iterations. We went through each of the solutions for the requirements with our users after we have integrated them, and got valuable feedback on how well they see the requirement covered. We used this in our next iteration, such that in the end we were able to deliver a solution that covers our requirements and also satisfies the users by the way how we integrated them.

5.2 Scalability and Performance

CHISV works in a multi-instance cluster

In 4.2.1 "Database" we briefly explained how we designed CHISV's back-end application to be able to scale-out to withstand any number of requests. While we focused more on the shared session database between multiple instances, we always took special precautions to ensure that CHISV would always be able to be run by multiple instances that simultaneously process requests. For every extension we wrote to the Laravel core (e.g. 4.4.2 "Job Extension"), we made sure that our application would also perform while being spread over multiple instances.

Adding more instances for more performance with multiple clients

We are certain that by adding more CHISV instances to a cluster and then using all these instances in that cluster in parallel, we can withstand any realistic amount of demand. While we would expect some reduction in performance by the load-balancer, we could naively expect to be able to process n-times as many requests in a cluster with n instances. However, to evaluate the performance of our application we think it is important to look at the performance of a single instance, rather than on a cluster of instances.

User's latency is highly dependent on the geographical location

To measure the application's latency, we will run our testing from the same machine that hosts CHISV. This will give us nearly constant network latency. As these tests only model the application's response time, any ac-

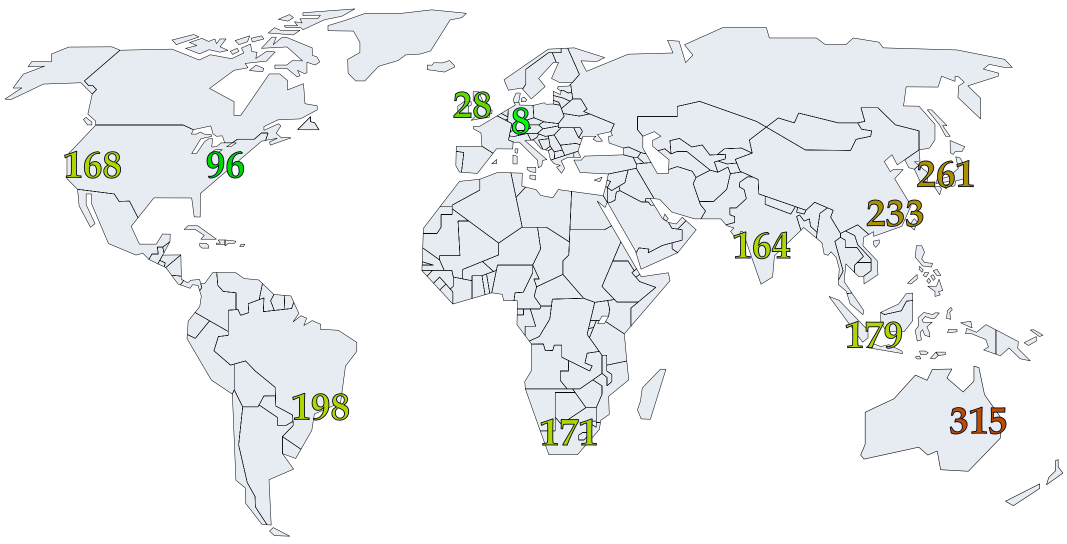


Figure 5.1: Map showing the additional latencies from different regions. All values are in milliseconds and taken from table 5.1.

tual request from active users would be traveling additional paths to reach our infrastructure. These additional latencies are highly different for every connection as we deployed our application in a centralized environment without any CDN.

5.2.1 Geographic Latencies

CHISV can be served from a CDN

While CHISV is currently deployed in a centralized location, it is also possible to host CHISV's API endpoints with any of the large CDN providers, like Amazon Web Services (AWS), Google Cloud Platform (GCP), or Microsoft Azure. Since our web application is a SPA, it only needs to be loaded once. After that, the application runs within the user's browser. That means it does not need to load any views from the server, as it already ships with all views compiled into the JavaScript bundle. The only interactions that need to be carried out are the calls to the API endpoint to get the raw data for the views.

Additional latency added by location

We can see in figure 5.1 and table 5.1 that with our exemplarily chosen 11 Points of Presence (PoPs) we ex-

	Frankfurt, Germany	London, UK	Cape Town, South Africa	New York City, USA	San Francisco, USA	
Latency (in ms)	8.21	28.24	171.05	96.90	168.54	

	Sydney, Australia	Tokyo, Japan	Bangalore, India	São Paulo, Brazil	Shanghai, China	Singapore
Latency (in ms)	315.29	261.58	164.54	198.04	233.93	179.10

Table 5.1: Table showing the different latencies CHISV users will encounter while using the application from different regions. It is important to notice that these numbers can fluctuate depending on the distance to our infrastructure and how utilized a link is.

pect CHISV to have relatively low latency. In multiple tests with 50 PoPs (CDNPerf) we encountered that CHISV servers show the highest latency in Australia (around 350ms).

Europe and the USA is where CHISV is mostly used from

CHISV is most commonly used from locations in Europe or the United States, as that is where the conferences are. When we look at Europe we see that we have latencies way below 100ms. The east and west coast of the United States connect to CHISV's servers with around 100ms to 150ms. While this is notably higher than in Europe, it will not interfere dramatically with CHISV's user experience.

5.2.2 Test Setup

Performance measured with one instance

As we now know about the additional latency CHISV users will be confronted with when accessing the application from different regions, we can now focus again on the performance of a single instance. This will help us understand how well CHISV scales with different conference sizes and users.

Configuration details

For our tests, we will be using a clone of our production instance with the same hardware and the same amount of provisioned resources. CHISV runs in a Linux container where we deployed Ubuntu Server 18.04 and gave it 12 cores of the installed dual 10-Core Intel Xeon E5-2670v2 CPUs. We run 8 queue workers, and the Linux container (kernel 5.3.18-1-pve) has access to 4 GB of system memory. We use the Linux TCP congestion algorithm CUBIC (see RFC8312[1]). Laravel is run by Nginx version 1.14.0 and interfacing with PHP-FPM 7.2.24 for running the PHP code of CHISV v1.0.7.

Limited to 100 PHP-FPM workers

PHP-FPM will start answering requests with workers from a pool of 5. We set the pool size adjustment to `dynamic` and limited it to 100 workers. PHP-FPM will always try to keep 5 workers idle and increase the pool size appropriately, as any request we make will be handled by one worker.

There are four variables that affect CHISV's response time. As the connected models are bound to a conference, increasing or decreasing the number of objects will only affect the associated conference. Thus, we will look at these variables for one conference at a time only:

- Number of users
- Number of tasks
- Number of bids
- Number of assignments

[1]https://tools.ietf.org/html/rfc8312

We will test different conference sizes (scenario), namely these four:

1. **Small conference** – 20 SVs and 50 tasks per day
2. **Medium conference** – 50 SVs and 100 tasks per day
3. **Large conference** – 100 SVs and 200 tasks per day
4. **Huge conference** – 300 SVs and 400 tasks per day

One of the biggest conferences that uses CHISV is the CHI. It's reflected by the "Large conference" scenario. We have added the "Huge conference" scenario to see where CHISV's limits are when run by only a single instance.

Test setup and database seeding

For each scenario, the conference will run for 7 days and we generate bids for SVs as if they were bidding on 30% of all tasks for a day. In case we generate a bid its preference will be "Unavailable" with a probability of 20%. When a bid's preference is not "Unavailable" each other preference ("low", "medium", "high") has the same probability and is randomly chosen. We generate tasks randomly between 8:00 AM and 6:00 PM once for each scenario. A task has a random duration between 15 and 135 minutes.

We will be measuring the response time SVs encounter while accessing the "SVs" and "Tasks" views (see figure 4.3) and the time it takes for the lottery and auction to complete. As both views support pagination we will always request the recommended default amount of items but also as much as the UI allows for. For the "SVs" view, this means that we will first load 10, then 40 items at once. Our tests for the "Tasks" view will first request 10, then 300 items at once. We will always request the first page, and for the "Tasks" view, we will not limit the selection by days or time.

Auction gets more complex over time

As the number of assignments SVs are associated with will increase over the course of the conference, we will be measuring the response time of the auction for each day separately. The auction will run longer for each day as the algorithm also evaluates all completed hours of all SVs while searching for a good fit. Thus, we will mark all assignments as "Done" before running the auction for the next day.

Simulate access of 30% of all SVs simultaneously

To simulate multiple requests we are using the tool Wrk by Will Glozer. Each test will run for 60 seconds. We are running at most 6 threads and will be opening c connections to account for 30% of all SVs continuously refreshing the view.

Test is set up to show CHISV's limits

From the insight we have with hosting the previous version of CHISV, we know that this assumption is (most of the time) highly exaggerated. Even during the highest peaks of access, we would rarely encounter so many requests. Thus, the response times we will measure represent the worst expected experience for each scenario respectively. Each test we run will give us:

- Latency per thread
- Requests per second per thread
- Actual requests per second of all threads combined
- Total number of all requests
- Detailed percentile spectrum

Test procedure

We will take a look at each conference scenario separately and run all our tests for one scenario before proceeding to the next one. After each test, we clear all Laravel caches and wait 10 seconds before starting the next run. We will start with the "small" conference scenario with 20 SVs and 50 tasks per day.

5.2.3 Results

Latency measurements are noisy by design

We tested the raw response time in milliseconds (ms). Due to the nature of process scheduling, paging, and the fact that the test (virtual) machine was running on a shared server along with other services, we expected somewhat noisy data. We expected that a measurement can be wrong by about 100ms or more. Thus we tested for 60 seconds to average out some of these outliers. Nevertheless, we expect that our results are affected by these factors. Depending on the hardware and amount of CHISV instances, these response times can be decreased further. As we discovered some unexpected results during our testing, we verified our setup and run the same tests multiple times. The outcome was similar. Nevertheless, we think that our results are representative of how good CHISV scales with different conference scenarios.

SVs View

Similiar results for 10 items per request

In our tests for the response time of the "SVs" view (see figure 5.2), we found that most of the values we measured are below 500ms for any realistic conference scenario ("small", "medium", "large"). We also see that the results for the tests where we requested 10 items are very similar. However, we also notice that in these cases our data contains more random spikes, due to the increased number of overall requests.

For 40 items per requests Laravel's caching changes the results

As we said earlier our test run for 60 seconds. While we also cleared all Laravel caches before we started each test, during a 60 seconds test Laravel will make use of a newly created cache. This can (and did) influence the response times. We can see, for instance, that the "large" conference scenario for 40 items is actually faster ($Q_1 - Q_3$ quartile) then the "medium" sized conference. For the unrealistically "huge" conference, we see the expected trend again. The increased latencies can be easily improved by load balancing all requests across multiple CHISV instances.

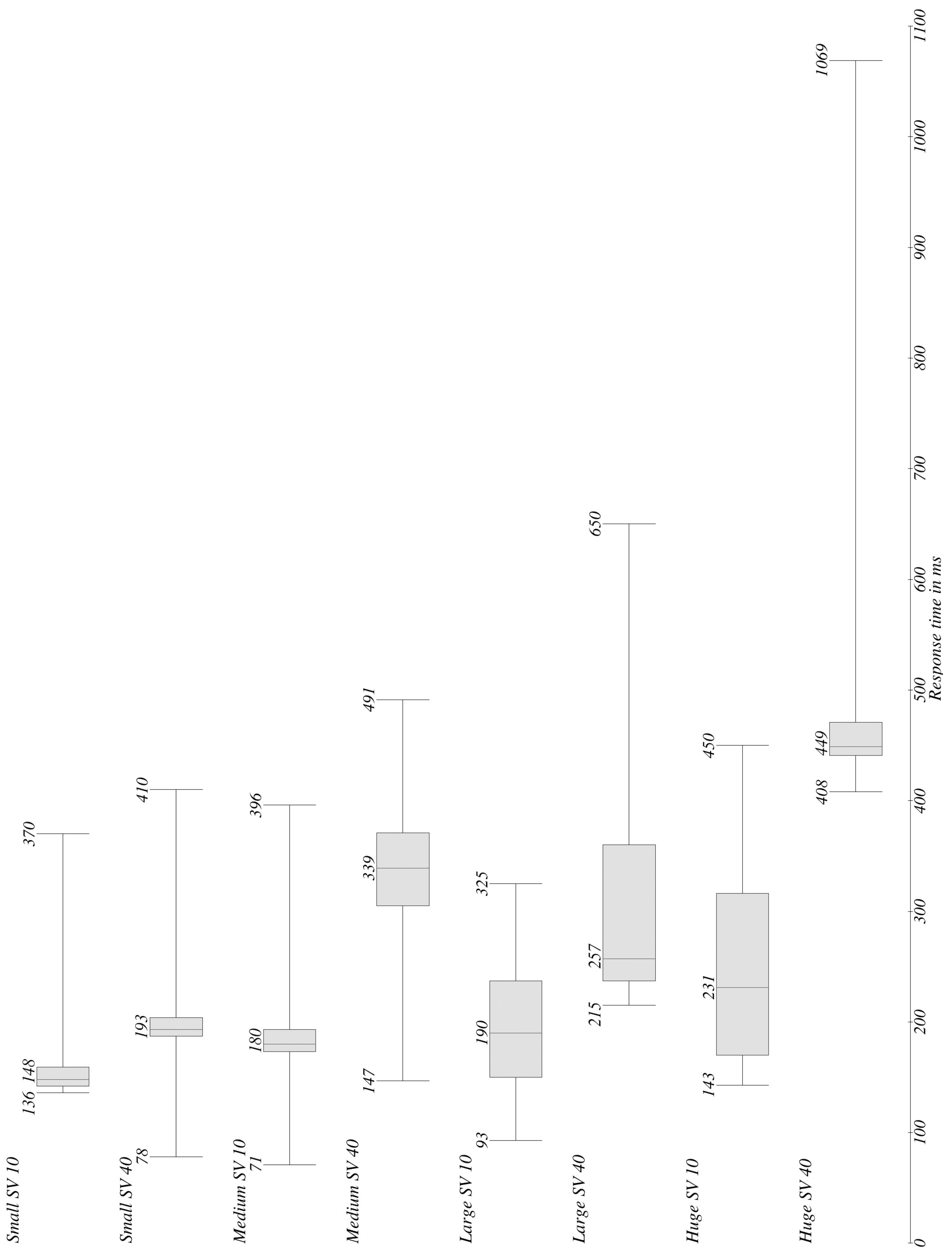


Figure 5.2: "SVs" view response time for various conference scenarios as defined in our test. As the requested data is paginated (chunked) we tested with the minimum and maximum items per request (10/40). For realistic conference scenarios, the response time is mostly under half a second. For each plot, we have appended the min, median, and max values.

Tasks View

Requesting 10 items for the "Tasks" view yields equal response time

Figure 5.3 shows our results for testing latencies with the "Tasks" views. We found that for 10 items per request we see no difference (with respect to measurement inaccuracies) between all conference scenarios. For our tests with 300 tasks per request, we see similar unexpected results as we did for the "SVs" view. We think this is mostly because of Laravel's query caches and measurement inaccuracies. However, while requests for realistically sized conferences are below 1 second, response times for the "small" scenario are higher than the ones for the "medium" or even the "large" conferences. For the "huge" scenario we see quite high results with around 1 second of response time. When confronted with such high demand, we recommend scaling-out CHISV by adding more instances to a load-balanced cluster.

Lottery and Auction

Lottery runtime always very short

Apart from the response time of the "SVs" and "Tasks" views we did also measure the runtime of the lottery and the auction. We found that the lottery took more time to complete as the number of SVs rose. The largest runtime we measured was the one for the "huge" scenario with 300 SVs. For this test, it took 3 seconds to complete.

Day	1	2	3	4	5	6	7
Small	0:05	0:05	0:06	0:09	0:09	0:13	0:15
Medium	0:12	0:14	0:17	0:31	0:38	0:44	0:52
Large	0:40	0:49	1:01	1:51	2:22	3:06	8:56
Huge	1:47	2:32	3:36	5:15	6:16	7:43	9:12

Table 5.2: Table showing the auction runtime for different conference scenarios in m:ss format.

Auction takes longer over time

For the auction, we measure the completion time per scenario and per day, as described in the test setup. We found that the auction takes longer to complete as the

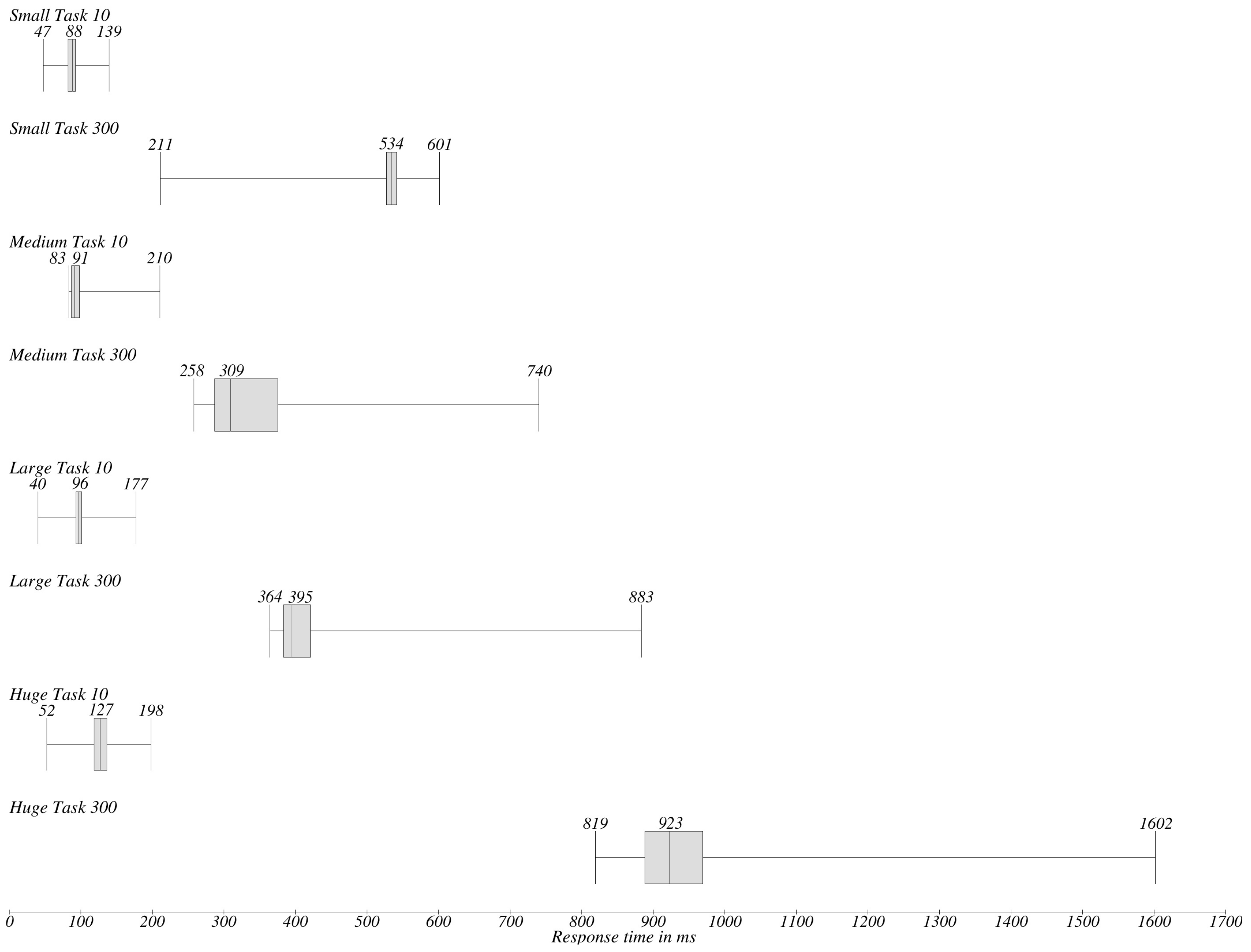


Figure 5.3: "Tasks" view response time for various conference scenarios as defined in our test. As the requested data is paginated (chunked), we tested with the minimum and maximum items per request (10/300). For realistic conference scenarios, the response time is mostly under half a second. The "huge" scenario’s response time can be improved by adding CHISV instances. For each plot, we have appended the min, median, and max values.

days go by (see table 5.2). This is also true for the scenarios: The more SVs and tasks, the longer the runtime.

Auction runtime is below 10 minutes

We noticed that for a "large" conference, like the CHI, the auction is going to take less than 10 minutes on the 7th day – an acceptable value.

5.3 User Experience

Comparing surveys from 2019 and 2020

In chapter 3 “Requirements Analysis” we already explained how we implemented CHISV with our users and how we used concurrent feedback to iterate over our features. As all these interviews, opinions, and evaluations won’t give us any precise numbers to compare the user’s experience to the previous version of CHISV, we also asked our users to express their opinion by filling out a survey.

Different participation numbers

The first survey we prepared was filled by SVs, Day Captains, and SV Chairs in 2019 for CHI 2019 while still working with the previous version of CHISV. In 2020 we presented a very similar survey to the first users of the new version of CHISV. Unfortunately, we could not get hold of as many SVs in 2020 due to CHI 2020 and many other conferences being canceled. The survey in 2019 was filled by 67, the one in 2020 by 7 users.

Questions targeting reported issues

Some of the questions we posed were targeting reported issues with the previous version of CHISV. An example can be seen in figure 5.4. Here, where we asked about the experience with the task bidding process to see if the new interface and it’s structure does equally or better suit the users’ needs. We found that users like the new interface and how it expresses the available options. The instant feedback, for example, on placing a bid makes it a lot easier for users to understand when the desired action has been fulfilled and if the system will preserve the changes.

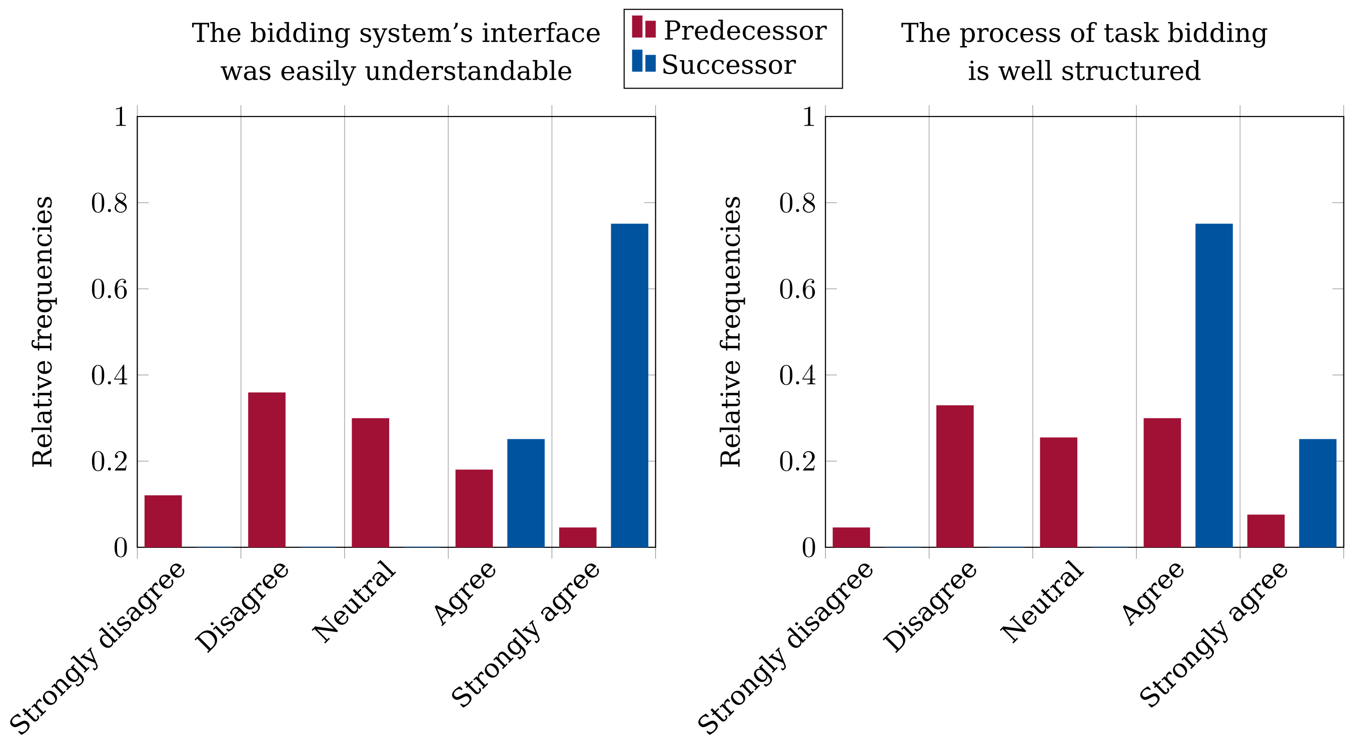


Figure 5.4: The survey showed us that users of the new CHISV found the user interface easy to understand and the steps required for bidding well structured.

Enhanced UI's appeal and responsiveness

Since we are using advanced front-end frameworks, like Bulma, Buefy, and Vue, it was very simple to provide a responsive and visually appealing interface. This is also what our users found, as can be seen in figure 5.5. Another big improvement is that the interface adapts to mobile devices' form factors. As this is something the previous version of CHISV was not built for, it is apparent that our rebuilt with modern web frameworks would greatly improve in this field.

Visual appeal improved due to Bulma and Buefy

We also see a huge improvement in how appealing users find the web interface. Again, when CHISV's predecessor was built the field of web frameworks was not nearly as popular as nowadays. This can also be seen in how polished the existing web frameworks looked. As a consequence, it is drastically easier to create a visually appealing interface nowadays. As the frameworks we used (Bulma and Buefy) embody well-known interface design

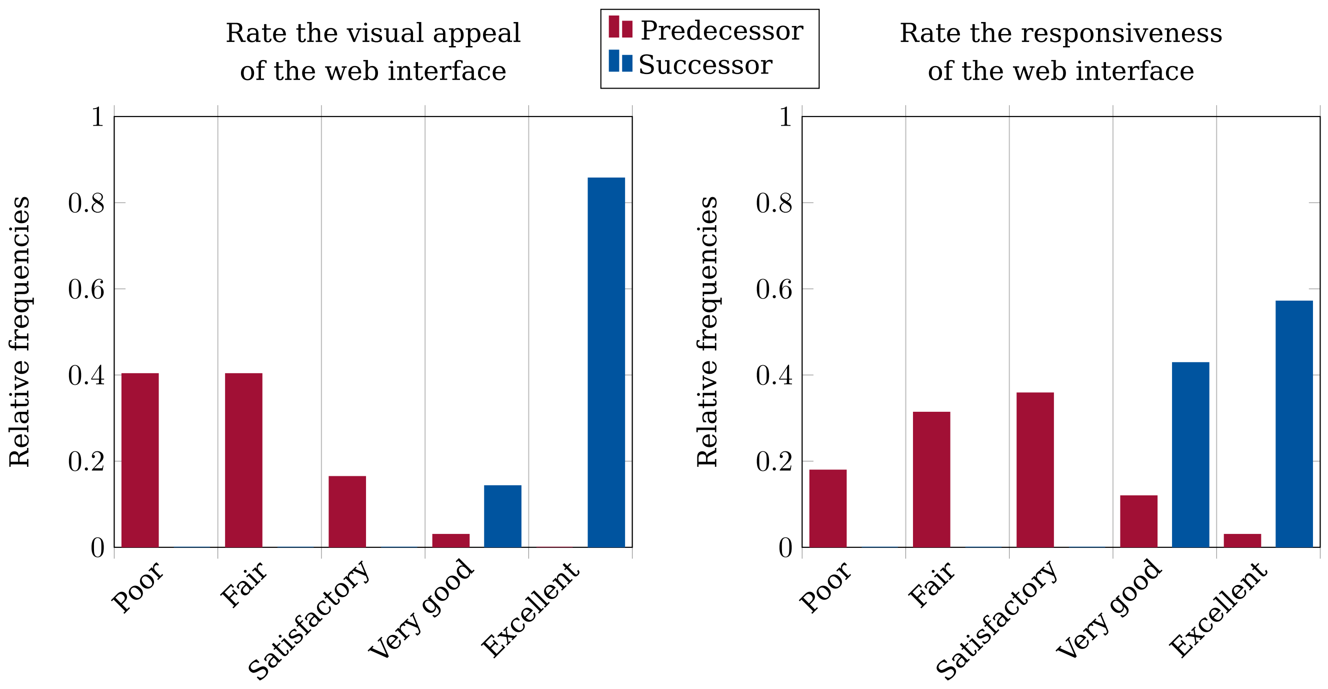


Figure 5.5: Overall, users stated that they like the visual appeal of the new version of CHISV more. We also see an increase of responsiveness when compared to the previous version.

guidelines, interfaces look better and are also easier to work with.

Similiar trend across all questions

These four graphs only represent a small subset of our questions. However, we mostly see the same trend. Due to the incorporated modern web technologies and new features, CHISV improved in many areas in contrast to its predecessor. We think another important area, which users typically not tend to see or notice this often, is the back end. We saw that many of our presented structures not only work great today but also leave enough room for future changes, increased demand, or new requirements.

Summarizing evaluation

Since we have precisely chosen each part of CHISV to be extendable, maintainable, and scalable, we think that this paves the way for others to extend, maintain, or develop applications on top of it.

This is true for the back end how we saw earlier in 5.2 “Scalability and Performance” but also for the front end, as we saw how a front-end framework can make it easier to create a responsive and appealing interface.

Decoupling from quickly changing front-end frameworks

However, as our front-end application incorporates all of CHISV’s functionality, we thought it is also important to loosely couple our back end and front end. Thus, we are also giving direct access to CHISV’s API for other applications. These, developed by third-party developers, can then make use of other usability concepts and, for example, also only focus on a certain area of application of CHISV. We will go into more detail about the API endpoints in 7.1.2 “Endpoints”.

Native mobile applications have great potential

Apart from the back-end and front-end applications, we see the API as CHISV’s third major contribution. It will enable future developers to create personalized, fast, and responsive native applications to run, for instance, on smartphones. Unfortunately, we can yet not evaluate the experience for these applications. However, we are certain that through CHISV’s API endpoints we paved the way for future applications that can adapt to recent technology trends and the users’ devices to provide an even better personalized user experience.

Chapter 6

Summary and Contributions

Various solutions

In this thesis, we have seen how we built the multi-conference student volunteers application CHISV by focusing on our users and closely evaluating all their needs and requirements. In the beginning, we started with an overview of the broad field of solutions various conferences use to manage student volunteers. While some conferences use simple online forms to gather the volunteers' data, others use more sophisticated software, as we have exemplarily seen with CHISV, SIGCSE, and SIGGRAPH.

Evaluation of CHISV, SIGCSE, and SIGGRAPH

These systems, which we looked at as part of the "Related Work", have been custom-built for the conferences they host. We compared the ways how volunteers sign up and enroll and took a look at the different features that the application provides. As this thesis focuses on the reimplementation of CHISV, we took a deeper look at how the previous version of CHISV handled the lottery and auction.

Collecting requirements from the users

In the next chapter, we identified our user groups that are using the new CHISV. For each group, we explained how we collected requirements and ideas in multiple iterations via interviews and surveys. Before we continued with presenting all 54 requirements, we sorted, filtered, grouped, and enumerated them. In the following of the thesis, we used the requirement's number whenever referring to it.

Back-end application

With all this knowledge about the users and their needs, we started with the introduction of our reimplementation of CHISV. First, we looked at the structures we developed for our back end, which is the part of CHISV that provides the requested data to front-end applications. We explained how different models, like `Users` and `Assignments`, are related and how we specifically always kept maintainability and scalability in mind for every structure that we put in.

Front-end stack

For the front-end application, we then focused more on the user experience and ease of use. This was possible, as we chose to go with modern web frameworks that made it easier to focus on the experience and compatibility with various devices. We explained the structure of our web application and how we tried to approach each of the important UI requirements.

Selected features in-depth

After describing our back-end and front-end solutions, we drew attention to selected features of both. We learned how the lottery and auction work and how we try to provide a fair and transparent algorithm. We introduced the new enrollment forms, which can be weighted, saw how the new calendar can ease schedule creation for SVs, and how our enhanced reports can assist SV Chairs with volunteer selection and statistics generation. Furthermore, we explained how our extension on Laravel jobs can improve the feedback for users on long-running jobs.

Scalability and performance

To get an idea of how well our solution scales, performs, covers the requirements, and is experienced by the user, we took CHISV to the test. First, we looked at all 54 requirements and evaluated the ones we haven't mentioned before. After that, we focused on the latency we measured from Points of Presence from all around the world to get an estimate of how responsive our application is. We found that the latency is relatively low in all regions where CHISV is used.

Response and user experience

As we now knew the induced latency of the users' connection to our servers, we concentrated on the application's response time for the most used features and views, which turned out to be reasonably low. In the end, we evaluated and compared the results from our two surveys in terms of user interface efficiency and appeal. This gave us an idea of how users experienced the new CHISV in comparison to the previous version. As expected due to the use of more modern web frameworks, the new version of CHISV scored better.

Contributing to the community

CHISV turned out to be of great value for the HCI community in all the years that the previous version was in use. We think that with the new version of CHISV we were able to tackle some of its shortcomings and obscurities. While our main aim was always to build an application for the community, we never lost sight of the essential need for maintainability and extendability. As we have designed the entire application to be publicly accessible, well-documented, and easily extendable by third-party applications, we hope that the community can easily pick up where we left off and create even better experiences for SV Chairs and student volunteers.

Chapter 7

Future Work

7.1 Third-Party API Access

We built CHISV to consist of a back-end and front-end application. We chose to expose all of the back end’s features directly via a RESTful API to provide a standardized interface that not only we can use. Our idea was to build a front-end application that could as well be written by a third-party developer. This means that our front-end application does not get more or less access to CHISV’s back end than any other application would.

Designed to provide a RESTful API from the beginning

7.1.1 Authentication

To authorize requests we made CHISV OAuth 2.0 compliant (RFC 6749, see Hardt [2012]). OAuth offers different grant types that not only allow for authorizing users but also machines. However, with CHISV our scope lies in the user authentication. This is why we only enabled (as of now) "Password Grant" requests, as these are required for authorizing a user with credentials to access the API.

Authentication is OAuth 2.0 compliant

In the context of OAuth we have the following roles:

- **Resource Owner** – The user
- **Resource Server** – CHISV back end
- **Authorization Server** – CHISV back end
- **Client** – Any third-party application

True OAuth authentication requires `client_id` and `client_secret`

To obtain an access token from the Resource Server the Client will have to send a request to the Authorization Server. In our implementation those two roles are hold by the back end. Such a request will have to contain five parameters:

- **`grant_type`** – Set to "password"
- **`client_id`** and **`client_secret`** – Have to be requested from CHISV's administrators
- **`username`** – The user's username, in our case the e-mail
- **`password`** – The user's password

A POST request with valid client and user credentials to `/oauth/token` will yield a response similar to this:

```
{
    "token_type": "Bearer",
    "expires_in": 31535999,
    "access_token": "eyJ0eX...cN0",
    "refresh_token": "def...2d4"
}
```

Client credentials have to be protected

The Client can then send requests to the Resource Server while appending the `access_token` to fetch the desired information. While this procedure is OAuth compliant, it poses a great security risk at some types of client applications as the `client_id` and `client_secret` can get easily exposed. An example of this would be a single-page web application written in JavaScript, which exposes the credentials in the source code.

Getting access to the API has to be possible without the interaction of the administrators

Furthermore, it also requires the developer to contact CHISV's administrators before being even able to talk to the API. We think that this can dramatically slow down, or even prohibit, development of third-party applications. For any native application (think of iOS and Android), using the OAuth interface has clear benefits, as it allows for revocation of client credentials or custom API limits. On the other hand, the OAuth interface makes it hard for any non-native application to access CHISV. To tackle this issue, we introduced a second token authentication endpoint.

Second token endpoint for SPAs and alike

This second endpoint enables authentication with only the user's credentials (e-mail and password). When we think of SPAs or Progressive Web Apps (PWAs), we require this sort of endpoint. To request a token, an application would send a POST request to `/api/v1/login` and supply the user's `email` and `password`. We provided an example of this process in our API Reference (see Reference). Internally, we proxy the request to the OAuth interface while supplying an internally used `client_id` and `client_secret`. As a result, the Client will receive the same response as in the example above. The only difference is that the Client did not need to know the `client_id` and `client_secret`. On the downside, revocation of the internally used client credentials would render all tokens issued via this endpoint invalid.

Two different options for authentication with equal request processing

As we have seen, both endpoints issue a token via the OAuth interface. This makes it easy for the back end to process token-based requests since each token is a valid OAuth `access_token` and can be handled by the Laravel OAuth core package.

7.1.2 Endpoints

CHISV's resources and actions are available via 77 endpoints

To interface with CHISV's resources and actions we have created 77 API endpoints, which are all covered in our API Reference. For each endpoint, we provide an example request that shows not only what the available parameters could exemplarily be but also their type and if they are optional or required. Each endpoint that requires authentication (either by token or Cookie) is marked with "Requires authentication".

Content-Type is JSON

While all resources can be accessed by their internal `id`, for conferences we chose to reference them by their unique `key` (e.g. chi2019, uist2020). Each API endpoint will expect the data in a request's body (`Content-Type`) to be in `application/json` format. Every response will also be in the JSON format. Sometimes a request to a resource will not only return the requested resource but also connected models that are likely also required. With this approach, we can reduce the number of required requests drastically, and make a big difference in terms of response time and transmitted data.

Append associated models to the requested resource whenever reasonable

A good example of this is the conference preview API endpoint. It does not require authentication and returns all conferences that are currently open to be displayed in an image carousel above CHISV's login form (see figure 4.5). Whenever a request is made to `/api/v1/conference/preview`, the returned conferences will not only contain the requested conference model with keys pointing to the icon, artwork, and state objects. Instead, we eager-load the associated models and append them to the model such that the client can directly render the conference without having to fetch the icon, artwork, and state in separate requests (see figure 7.1).

Future clients can focus on user experience

We have taken all these steps to open up CHISV to a broad variety of end devices, developers, and users. With the publicly available and well-documented API, we are excited about the experiences others can now create to make volunteering even more fun and accessible.

```
[
    {
        "id": 5,
        "name": "UIST 2020",
        "key": "uist2020",
        "location": "Online",
        "state_id": 2,
        "icon": {
            "owner_id": 5,
            "web_path": "\/storage\/images\/1..H.jpeg"
        },
        "artwork": {
            "owner_id": 5,
            "web_path": "\/storage\/images\/V..1.jpeg"
        },
        "state": {
            "id": 2,
            "name": "enrollment",
            "description": "Students can enroll in the conference"
        }
    }
]
```

Figure 7.1: Example response of the conference preview endpoint with eager-loaded icon, artwork, and state models.

7.2 Realtime Calendar Integration

iCalendar exports are static

As we already explained in 4.4.6 “Universal Event Export”, we found out that users would like to see their assignments in a calendar view with options for the month, week, and day. Thus, CHISV offers a calendar view and additionally an export in iCalendar format. This exported file contains all the selected events and can be imported into any compatible calendar software. However, once generated and downloaded the content of the export is no longer updated and may not reflect any updated details of an assignment.

Personalized iCalendar URL

One approach to overcome this limitation could be the integration of new and personalized API endpoints. Whenever a request is sent to this URL, CHISV would answer with an update-to-date listing of all assignments in iCalendar format. The calendar solutions of Google and Apple both provide the option to subscribe to iCalendar URLs. These subscriptions are then also available on mobile devices. Google and Apple servers will periodically poll the given endpoint for updates and will always provide the latest polled version to the user. While this version might not be updated in realtime, it's more up-to-date than the static export CHISV produces at the current level of implementation.

Using a long random string in the URL

The URLs used for the personalized calendar subscriptions have to contain some long and random string that is hard to guess. One could extend the user model to also hold a `calendar_key`, a 64 characters long random ASCII string. This string could be regenerated in the case it gets compromised. We could envision a URL of the form `/api/v1/calendar_for/USERS_CALENDAR_KEY` which would then return the user's up-to-date iCalendar file.

Optional query parameters

To limit the selection further it would make sense to have optional query parameters. CHISV currently uses `start` and `end` for the existing calendar. One could also add a parameter for the conference's key, such that users can subscribe to only a certain conference, rather than having to give a time range, which might not precisely cover the conference.

Bibliography

AAMFT. Volunteers - aamft19, May 2020. URL `https://networks.aamft.org/conference/volunteers`.

ACL. Student scholarship applications and volunteers, May 2020. URL `http://www.acl2019.org/EN/student-scholarship-applications-volunteers.xhtml`.

AEA. Aea - american evaluation association : Evaluation 2019: Paths to the future of evaluation: Contribution, leadership, and renewal : Student volunteers, May 2020. URL `https://www.evaluationconference.org/p/cm/ld/fid=687`.

S. Ahmed and Q. Mahmood. An authentication based scheme for applications using json web token. In *2019 22nd International Multitopic Conference (INMIC)*, pages 1–6, 2019.

CakePHP. Cakephp, June 2020. URL `https://cakephp.org/`.

CDNPerf. Cdn benchmark, July 2020. URL `https://www.cdnperf.com/tools/cdn-latency-benchmark/?id=c7a6fe70632f061bfe72d4bb93dfa6c2`.

CITT. Annual conference - student volunteer program, May 2020. URL `https://www.citt.org/student_volunteer_program.html`.

CVPR. Submission site, May 2020. URL `http://cvpr2019.thecvf.com/attend/student_volunteer`.

Bernard Desruisseaux. Rfc 5545: Internet calendaring and scheduling core object specification (icalendar). Technical report, 2009.

Authentication Documentation. Authentication, June 2020a. URL `https://laravel.com/docs/7.x/authentication`.

Buefy Documentation. Documentation | buefy, June 2020b. URL `https://buefy.org/documentation/`.

Bulma Documentation. Bulma: Free, open source, and modern css framework based on flexbox, June 2020c. URL `https://bulma.io/documentation/`.

CSRF Protection Documentation. Csrf protection, June 2020d. URL `https://laravel.com/docs/7.x/csrf`.

Database Documentation. Database: Getting started, June 2020e. URL `https://laravel.com/docs/7.x/database`.

Laravel Blade Documentation. Authentication, June 2020f. URL `https://laravel.com/docs/7.x/blade`.

Notifications Documentation. Notifications, June 2020g. URL `https://laravel.com/docs/7.x/notifications`.

Queues Documentation. Queues and jobs, June 2020h. URL `https://laravel.com/docs/7.x/queues`.

Vuex Persisted State Documentation. robinvdvleuten/vuex-persistedstate: Persist and rehydrate your vuex state between page reloads., June 2020i. URL `https://github.com/robinvdvleuten/vuex-persistedstate`.

ECAI. Call for volunteers – ecai 2020, May 2020. URL `https://ecai2020.eu/call-for-volunteers/`.

Brendan Eich. Javascript at ten years. *Proceedings of the tenth ACM SIGPLAN international conference on Functional programming - ICFP '05*, 2005. doi: 10.1145/1086365.1086382. URL `http://dx.doi.org/10.1145/1086365.1086382`.

GitHub. Chisv: Conference student volunteer system with tasks, assignments, and statistics, June 2020. URL `https://github.com/dwhoop55/chisv`.

Will Glozer. Modern http benchmarking tool, July 2020. URL `https://github.com/wg/wrk`.

Ilya Grigorik. Making the web faster with http 2.0. *Communications of the ACM*, 56(12):42–49, Dec 2013. ISSN 1557-7317. doi: 10.1145/2534706.2534721. URL `http://dx.doi.org/10.1145/2534706.2534721`.

Jeremiah Grossman, Seth Fogie, Robert Hansen, Anton Rager, and Petko D Petkov. *XSS attacks: cross site scripting exploits and defense*. Syngress, 2007.

Dick Hardt. Rfc 6749: The oauth 2.0 authorization framework. Technical report, October 2012.

HCII. Student volunteers | hci international 2020, May 2020. URL `http://2020.hci.international/student-volunteers.html`.

HRI. Student volunteer program – hri 2020, May 2020. URL `https://humanrobotinteraction.org/2020/student-volunteer-program/`.

ICFP. Icfp 2019 - student volunteering - icfp 2019, May 2020. URL `https://icfp19.sigplan.org/track/icfp-2019-Student-Volunteering#Call-for-Participation-Student-Volunteers`.

ICSA. Student volunteers - icsa 2020, May 2020. URL `http://icsa-conferences.org/2020/attending/volunteers/index.html`.

ICSE. Student volunteers - international conference on software engineering 2019 in montreal, canada, May 2020. URL `https://2019.icse-conferences.org/track/icse-2019-Student-Volunteers#Call-for-Student-Volunteers-`.

IMS. Student volunteers, May 2020. URL `https://ims-ieee.org/students-main/student-volunteers`.

László Viktor Jánoky, János Levendovszky, and Péter Ekler. An analysis on the revoking mechanisms for json web tokens. *International Journal of Distributed Sensor Networks*, 14(9):1550147718801535, 2018. doi: 10.1177/1550147718801535. URL `https://doi.org/10.1177/1550147718801535`.

N. Jovanovic, E. Kirda, and C. Kruegel. Preventing cross site request forgery attacks. In *2006 Securecomm and Workshops*, pages 1–10, Aug 2006. doi: 10.1109/SECCOMW.2006.359531.

Laracasts. Laracasts, June 2020. URL `https://laracasts.com/`.

Laracon. Laracon online | the official laravel online conference, June 2020. URL `https://laracon.net/`.

Laravel. Laravel - the php framework for web artisans, June 2020. URL `https://laravel.com`.

laravel.io. Laravel community, June 2020. URL `https://laravel.io`.

K. Lei, Y. Ma, and Z. Tan. Performance comparison and evaluation of web development technologies in php, python, and node.js. In *2014 IEEE 17th International Conference on Computational Science and Engineering*, pages 661–668, 2014.

Mahith Madwesh and Sandeep Varma Nadimpalli. Survey on authentication techniques for web applications. *Available at SSRN 3510088*, 2019.

Medium.com. Top 3 best javascript frameworks for 2019, June 2020. URL `https://medium.com/cuelogic-technologies/top-3-best-javascript-frameworks-for-2019-3e6d21eff3d0`.

PASC. Student volunteer program – the pasc18 conference, May 2020. URL `https://pasc18.pasc-conference.org/about/student-volunteer-program/index.html`.

Passport. Laravel passport, June 2020. URL `https://laravel.com/docs/7.x/passport`.

POPL. Popl 2020 - student volunteers - popl 2020, May 2020. URL `https://popl20.sigplan.org/track/POPL-2020-student-volunteers`.

L. H. Pramono, R. C. Buwono, and Y. G. Waskito. Round-robin algorithm in haproxy and nginx load balancing performance evaluation: a review. In *2018 International Seminar on Research of Information Technology and Intelligent Systems (ISRITI)*, pages 367–372, 2018.

A Rahmatulloh, R Gunawan, and F M S Nursuwars. Performance comparison of signed algorithms on JSON web token. *IOP Conference Series: Materials Science and Engineering*, 550:012023, aug 2019. doi: 10.1088/1757-899x/550/1/012023. URL `https://doi.org/10.1088%2F1757-899x%2F550%2F1%2F012023`.

API Reference. Chisv api reference, July 2020. URL `https://chisv.org/doc`.

SC. Sc20 submission site, May 2020. URL `https://submissions.supercomputing.org`.

SIGCSE. Sigcse volunteer registration, May 2020. URL `https://www.cs.ubc.ca/~sig-cse/sigcse/user/index.php`.

SIGGRAPH. Siggraph student volunteer system, May 2020. URL `https://sv.siggraph.org/`.

SSWR. Society for social work and research - student volunteer opportunities, May 2020. URL `https://secure.sswr.org/2020-conference-home/student-volunteer-opportunities/`.

Daniel Stenberg. Http2 explained. *ACM SIGCOMM Computer Communication Review*, 44(3): 120–128, Jul 2014. ISSN 0146-4833. doi: 10.1145/2656877.2656896. URL `http://dx.doi.org/10.1145/2656877.2656896`.

SV-Portal. jdiehl/svportal: Student volunteers portal, May 2020. URL `https://github.com/jdiehl/svportal`.

Symfony. Symfony, June 2020a. URL `https://symfony.com/`.

Symfony. Laravel (projects using symfony), August 2020b. URL `https://symfony.com/projects/laravel`.

Kevin Tatroe and Peter MacIntyre. *Programming PHP: Creating Dynamic Web Pages*. O'Reilly Media, 2020.

S. Tilkov and S. Vinoski. Node.js: Using javascript to build high-performance network programs. *IEEE Internet Computing*, 14(6):80–83, 2010.

Udemy. Laravel | udemy, June 2020. URL `https://www.udemy.com/topic/laravel/`.

Allen Wirfs-Brock and Brendan Eich. Javascript: The first 20 years. *Proc. ACM Program. Lang.*, 4(HOPL), June 2020. doi: 10.1145/3386327. URL `https://doi.org/10.1145/3386327`.

N. Yadav, D. S. Rajpoot, and S. K. Dhakad. Laravel: A php framework for e-commerce website. In *2019 Fifth International Conference on Image Information Processing (ICIIP)*, pages 503–508, 2019.

Kazuhiko Yamamoto, Tatsuhiro Tsujikawa, and Kazuho Oku. Exploring http/2 header compression. *Proceedings of the 12th International Conference on Future Internet Technologies - CFI'17*, 2017. doi: 10.1145/3095786.3095787. URL `http://dx.doi.org/10.1145/3095786.3095787`.

Yii. Yii, June 2020. URL `https://www.yiiframework.com/`.

Raka Yusuf and Rizza Syah Pasha Ulin Nuha. Comparative analysis of haproxy& nginx in round robin algorithm to deal with multiple web request. *International Journal of Computer Techniques*, 5:90–94, 2018.

Index

Typeset August 26, 2020