G11 ChairVisE 4.0 Project Report

CS3219, Software Engineering Principles & Patterns

National University of Singapore, School of Computing

Muhammad Ahmed bin Anwar Bahajjaj
A0140051A

Johnson Ho Chong Xiang
A0168313H

Ni Tianzhen
A0164726Y

Yip Seng Yeun
A0173178X

Deployed Application URL
https://cs3219-madanalogy.appspot.com

Code Repository Link
CS3219-SE-Principles-and-Patterns/cs3219-ay2021-s1-project-2020-s1-g11

Individual Contributions

Student	Feature Contributions	Other Contributions
Ahmed Bahajjaj	- Test Infrastructure and Test Cases - Persistence and Management of Data (API Design and Frontend)	- Documentation - Deployment
Johnson Ho Chong Xiang	- Persistence and Management of Data (Backend)	- Documentation - Deployment
Ni Tianzhen	- Improvements to UI/UX - Persistence and Management of Data (Frontend)	- Documentation - Deployment
Yip Seng Yeun	- Flexible Data Schema	- Documentation - Deployment

1. Introduction

ChairVise 3.0 is a web application that enables conference organizers to easily visualize and share conference submission statistics through intuitive charts and presentations from data dumps from the Conference Management Software (CMS) they use. It currently supports the EasyChair data format and comes with a set of powerful and insightful premade queries that allow users to very quickly visualise their data.

ChairVise 4.0 improves upon the existing system by adding support for another popular CMS, SoftConf, as well as a custom data format option that allows users to define their own processing logic to support files from any other CMS. It also features an updated responsive UI that improves the usability of the application, especially for mobile users. The inability to manage already uploaded data files has been addressed with a new UI that allows users to rename, remove or replace previously uploaded data files without having to create a new version. Lastly, the codebase is more robust with a comprehensive test suite that spans both front and back-ends.

2. System Architecture

The system architecture for ChairVisE 4.0 does not differ largely from ChairVisE 3.0 in terms of high level component composition. As such, the following descriptions will describe the common system architecture for both ChairVisE 3.0 and 4.0.

2.1 Overall Architecture

The overall system architecture of ChairVisE is a layered design pattern. Specifically, it consists of 2 layers: the presentation layer (frontend) and the application layer (backend).

2.2 Frontend Architecture

The frontend is responsible for handling the graphical design and user interaction of the web page that users interact with. Figure 2.2A shows the frontend package overview.

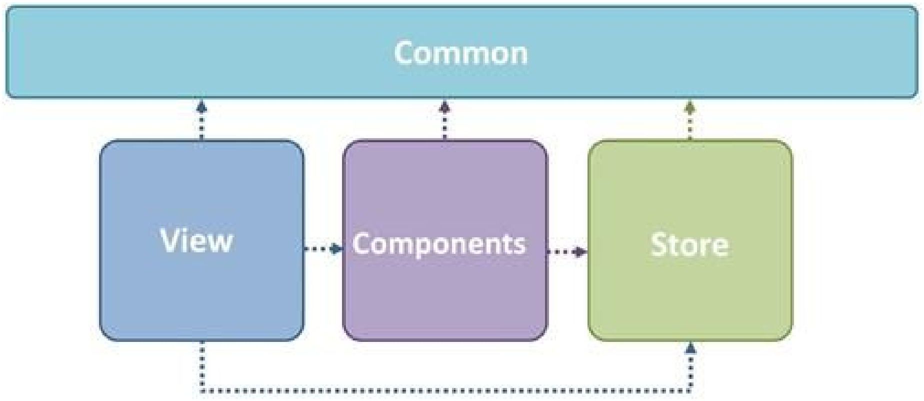


Figure 2.2A

There are 4 components in Frontend:

1. Common: Contains utility functions and constants.
2. View: Responsible for displaying each page of the application. Component: Responsible for UI and display logic. These are reusable and called by various pages across the entire application.
3. Store: Responsible for application logic. It maintains the state of the application. It also handles mutation and actions to the state committed by components.

Figure 2.2B shows the frontend architecture diagram.

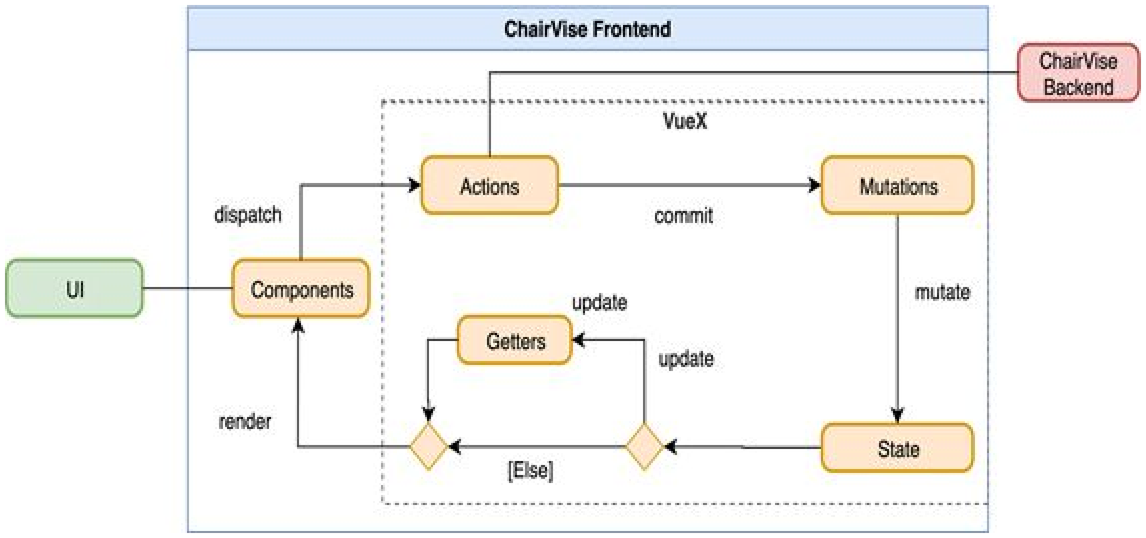


Figure 2.2B

In Vuex, UI components dispatch actions and send HTTP requests to the backend API based on events invoked by the user. When the HTTP request returns a result, the action commits a mutation which updates the states. This update is rendered to the components.

The frontend of ChairVisE 3.0 employs the shared repository architecture. Data is maintained in a centralised database in the backend accessed via Vuex, shared by all functional components. Components work directly on the data maintained in the shared repository so other components react if the data changes. Availability, quality and state of data triggers and coordinates the flow of application logic.

2.3 Backend Architecture

The backend is responsible for applying the required business logic and returning it to the frontend. In the case of ChairVisE, this involves logic related to visualization creation and data management.

Figure 2.3.A shows the backend package overview. It is also designed with a layered design pattern, made up of 4 layers:

1. Common
2. UI
3. Logic
4. Storage

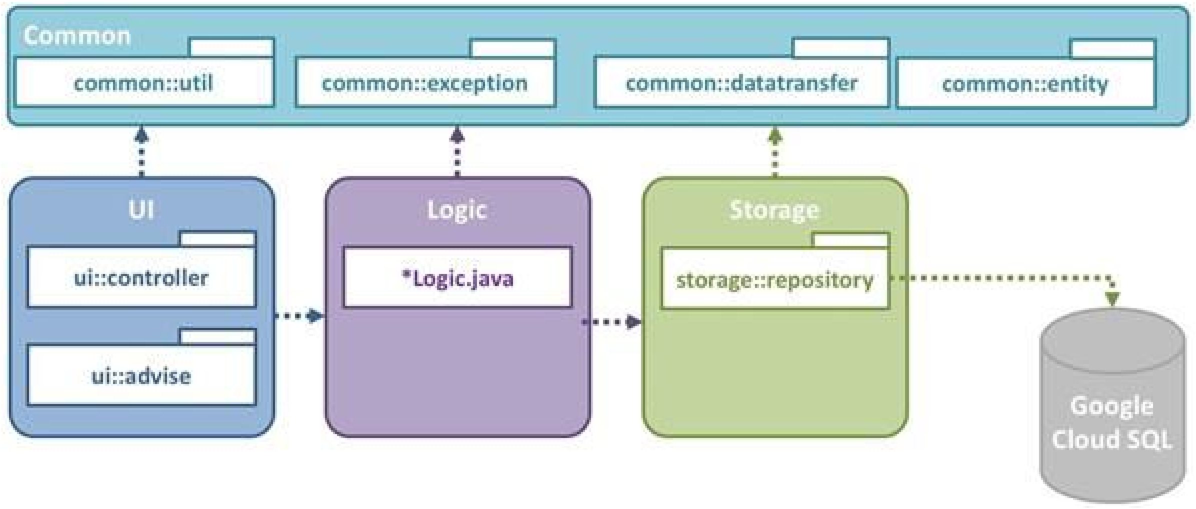


Figure 2.3A

The backend takes inspiration from the Model View Controller architecture. This is notable in its use of Controllers in the UI component, which accept user input as events and translate these events into service requests for the Logic component.

2.3.1 Common Layer

The Common layer contains utility functions and constants used by all other layers.

2.3.2 UI Layer

The UI layer directly handles HTTP requests from the frontend and calls the necessary Logic components to handle the requests. It contains a controller package that provides Representational State Transfer (RESTful) APIs via the use of controllers. Several controllers are implemented:

1. **AnalysisController**
   Handles analysis requests sent by frontend and issues SQL aggregation queries to the database.
2. **AuthInfoController**
   Checks the current authentication status of the user.
3. **ConferenceController**
   Handles CRUD operations on conference related information.

4. **DBMetaDataController**
   Exposes the metadata, including name and type of the standard data template used in the application.
5. **PresentationController, PresentationSectionController and PresentationAccessControlController**
   Handles users created CRUD operations of presentations, presentation sections in presentations and access checking respectively.
6. **RecordController**
   Stores CSV data imported by users into the database.
7. **VersionController**
   Persists version entries in the backend and allows the frontend to obtain version entries.
8. **WebPageController**
   Serves static production files built by Vue.

Implementing several controllers that each have a single and distinct responsibility is an application of the Single Responsibility Principle. This increases cohesion and reduces coupling.

2.3.3 Logic Layer

The Logic layer handles the backend logic and coordinates Authentication, Presentation, Analysis etc using Spring Framework.

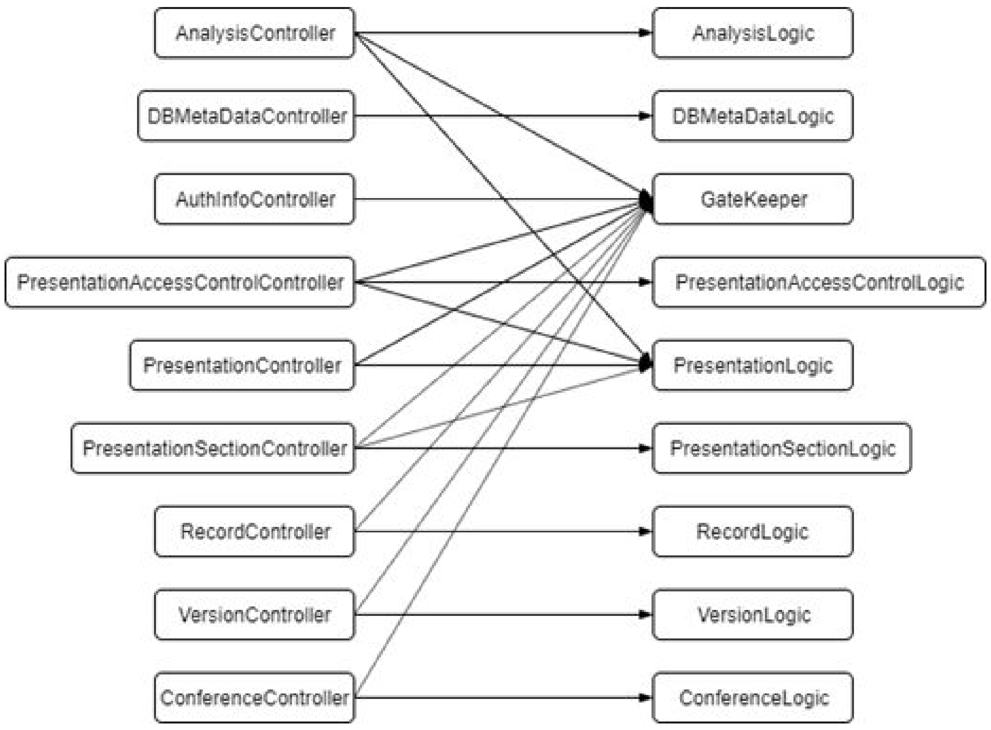


Figure 2.3.3A

Figure 2.3.3A shows a list of how the controllers are associated with the logics. As an extension of the Single Responsibility Principle, each controller has a corresponding logic. Coupling between controller and logic is kept to a minimum.

2.3.4. Storage Layer

The Storage layer handles changes to the database.

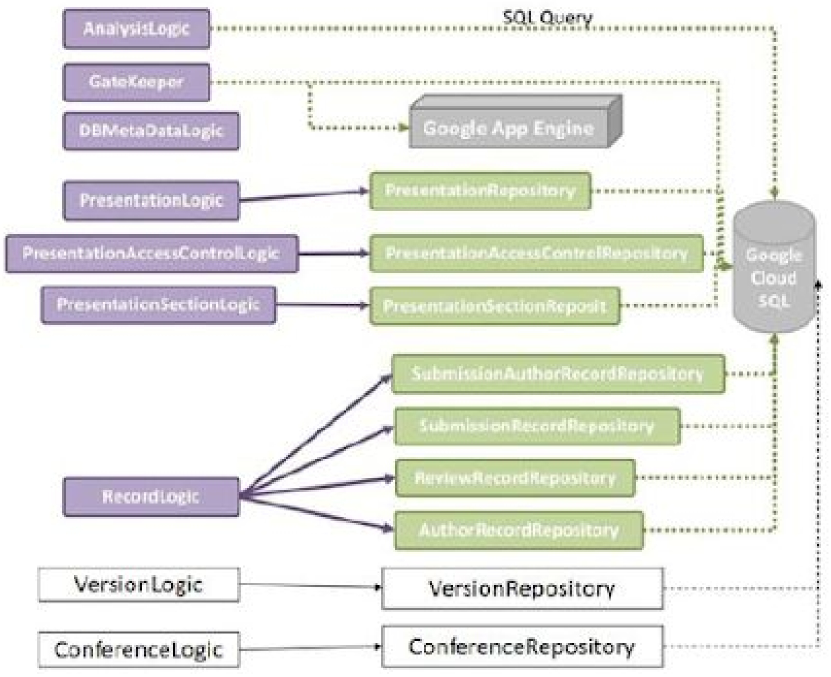


Figure 2.3.4A

Figure 2.3.4A shows a list of how the logics are associated with the repositories. Once again, the Single Responsibility Principle continues to be applied as each logic communicates with its corresponding repository (or repositories).

3. Software Requirements Specification

3.1 SRS Introduction

This Software Requirements Specification serves to outline the requirements of ChairVise 4.0, a web application that enables conference organizers to easily visualize and share conference submission statistics and presentations from data dumps from the Conference Management Software (CMS). In particular, we will focus on the improvements needed compared to its previous version, ChairVise 3.0. The scope of our improvements is usability, experience, and maintainability; apart from finishing up features not yet achieved in ChairVise 3.0, we have enhanced the overall usability of the software by adding the data management feature and the capability of accepting customised data format. User experience wise, we have done various improvements to the user interface and logic which aim to deliver a more intuitive and convenient user experience. Lastly, we have enhanced the maintainability of the software by providing both front and back-end test suites.

3.2 SRS Overall Description

In this section we will outline the background and rationale for us to come with the software requirements. Given that ChairVise is a software that enables conference organizers to visualize and share conference submission statistics and presentations, ChairVise 3.0 has already accomplished that basic target well. However, from our experience of using ChairVise 3.0 we did notice many areas that need improvement.

3.2.1 Product Perspective

ChairVise fills the gap in the conference management ecosystem by interfacing with other Conference Management Software (CMS) to create meta-analyses and visualisations of conference data from data dumps. ChairVise 3.0 supports only the EasyChair CMS, which limits its usefulness as there are more than a dozen CMS on the market.

3.2.2 User Classes and Characteristics

The users of ChairVise 4.0 are primarily the conference organisers who want to visualise and analyse conference data. As conference organisers, they should be somewhat familiar with the various data formats that CMS can export and what each data column in these files entails. Moreover, as conference organisers, they should be familiar with what each type of visualisation means (co-authorship, inter-organisational collaboration, etc). Additionally, we can also assume some degree of technical proficiency and familiarity with the data columns of the CMS, which allows them to take advantage of advanced features such as writing custom data transformation functions.

Viewers constitute another significant user class of ChairVise, but their interaction with the app is limited to viewing presentations created on the platform. Given the convenience afforded by mobile devices, we should ensure that ChairVise caters well to a mobile audience, especially when viewing presentations.

3.2.3 Product Features

This section outlines the product features, or user requirements in the form of user stories that ChairVise 4.0 should fulfill in excess of existing capabilities.

As a...	I should be able to...	So that...
Conference organiser	Import conference data in various data file format	I can use the CMS that I am most familiar with, without switching to another one

Conference organiser	Upload relevant data files for a conference in a batch	I can conveniently import all data files for a conference with only one single upload
Conference organiser	Label imported dataset with a conference name and year	I can easily identify the conference and year that the dataset belongs to
Conference organiser	Change the label of the dataset	I can easily correct my mistake after making a typo in the conference name/year
Conference organiser	Upload additional data files for an existing dataset	I can create other types of visualisations using part of the existing data
Conference organiser	Delete a data file from an existing dataset	I can remove data that are no longer relevant
Conference organiser	Read the user guide without logging in	I can understand the app’s features before registering
Conference organiser	Access my presentations from smartphones	I can show the visualisation to others conveniently, without using a laptop
Presentation audience	Access shared visualisations from smartphones	I can take a look at the visualisations conveniently

3.3 Specific Requirements

Based upon the user requirements and extrapolated product features elaborated upon earlier, we defined specific functional and non functional requirements. These requirements are specified in the following sub sections.

3.3.1 Functional Requirements

Label	Feature Legend
F-1	Flexible Data Schema
F-2	Persistence and Management of Conference Data
F-3	Improvements to UI/UX
F-4	Add Test Infrastructure and Test Cases

Label	Description	Priority	Feature
FR-1	The application should provide a toggle option during data import for users to import data in various data formats (EasyChair, SoftConf, Custom)	High	F-1
FR-2	The application should provide a component for users to delete, rename and update imported data records	High	F-2
FR-3	The application should provide a data import component to allow program chairs to upload Comma Separated Values (CSV) conference data records by conference ID and year	High	F-3
FR-4	The application should prevent integration of changes that fail integration test cases or fail to build	High	F-4
FR-5	The application should provide a component to generate presentations using any imported conference data format	Med	F-1
FR-6	The application should provide a component for users to choose the set of data records to use when creating a new presentation	Med	F-3
FR-7	The application should provide new unregistered users with access to the user guide	Med	F-3
FR-8	The application should show all user interface components (without blocking) in varying screen sizes and widths	Med	F-3
FR-9	The application should show a warning message to prompt users on the affected presentation(s) during the deletion of a record	Low	F-2
FR-10	The application should fail gracefully and show users any error messages when any data Create-Read-Update-Delete (CRUD) transaction fails	Low	F-2
FR-11	The application should provide a component to display the type of data records (author, review, and submission record) available for a presentation	Low	F-3
FR-12	The application should provide a component to display presentations’ sharing status (display the link sharing status and the accounts that the presentation is shared with)	Low	F-3
FR-13	The application should ensure that only the types of visualisations supported by the existing types of data records can be added to a presentation	Low	F-3
FR-14	The application should provide a set of test cases for each implemented component in the backend	Low	F-4

FR-15	The application should provide a set of test cases for each implemented component in the frontend	Low	F-4

3.3.2 Non-Functional Requirements

Label	Attribute Legend
A-1	Security and Robustness
A-2	Performance and Availability
A-3	Scalability and Reliability

Label	Description	Priority	Attribute
NFR-1	The application should prevent Structured Query Language (SQL) injection attacks carried out on the system database	High	A-1
NFR-2	Users with a stable internet connection should be able to load the website under 10 seconds	High	A-2
NFR-3	The app should respond within 5 seconds for any user input	High	A-2
NFR-4	The application should be accessible by at least 100 concurrent users utilising the application at any one time	Med	A-3
NFR-5	The application should be able to persist and manage at least 1000 data versions and associated records	Med	A-3
NFR-6	The application should prevent Cross-Site Request Forgery (CSRF) attacks carried out on application users uploading data	Low	A-1
NFR-7	The application should be able to rollback to an earlier version within 1 minute of detecting a breaking change	Low	A-3

4. Enhancements

The various subsections in this segment covers the features delivered in order to fulfil the Functional Requirements (FRs) as outlined in the SRS.

4.1 Flexible Data Schema

4.1.1 Existing Implementation

4.1.1.1 Importing Data

The data import, parsing and persistence pipeline present in ChairVise 3.0 is as follows:

1. Import raw CSV
2. Parse CSV into rows and columns
3. Map data columns into database columns using a column to column mapping
4. Upload to server
5. Store in database

4.1.1.2 Data Schema

The current database schema closely follows the data columns present in the EasyChair format. We shall classify these fields as metadata fields - fields used for identification and relational purposes (e.g. submission ID, author ID, track ID), and content fields - fields that are of interest to the user and will be used to create visualisations.

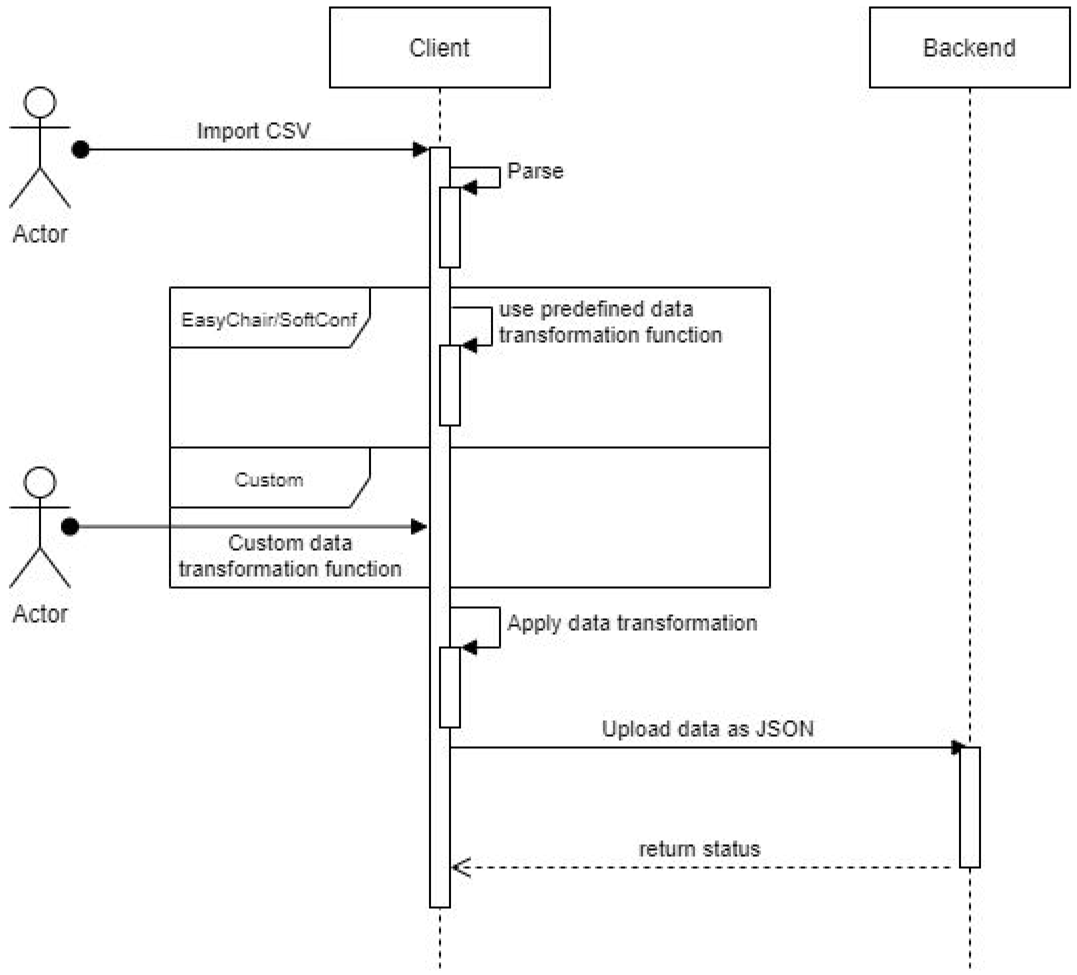
4.1.1.3 Review Existing Implementation

Due to new requirements to expand support for CMS beyond EasyChair and SoftConf, the existing data schema should be augmented to be more flexible, while still maintaining backwards compatibility with the current formats.

4.1.2 Proposed Implementation

4.1.2.1 Overview

The proposed enhancement will replace the mapping step from the import process with a customisable data transformation. The user will specify a data transformation that accepts a row of data from the uploaded file as input, and returns a data object based on the database schema. The data transformation follows the pipe and filter architecture.



Client
Backend
Actor
Import CSV
Parse
EasyChair/SoftConf
use predefined data transformation function
Custom
Actor
Custom data transformation function
Apply data transformation
Upload data as JSON
return status

Fig 4.1.2.1A

4.1.2.2 Design Goals

The proposed enhancement is designed with the following goals in mind:

- Flexible - Support any valid CSV file and make minimal assumptions about the data formats
- Backwards compatible - Existing users with existing data should not experience a disruption to the service or be required to reupload their data files

4.1.2.3 Design Considerations

Data Schema

Our options were:

- keep the existing schema
- insert additional rows to existing schema
- replace the database with a document based solution (NoSQL)

We made the decision to keep the existing schema, and provide a solution for users to map any arbitrary set of columns to an object based on the existing schema. The primary factor in our decision making was to maintain backwards compatibility - while a NoSQL data solution would be a perfect fit here, it would require a large refactor of the codebase and more importantly break existing users' data and presentations.

Extending the existing schema with more columns to accommodate new CMS formats is a good, albeit short term alternative. This is because the addition of SoftConf support will require the addition of over 80 columns across all 3 tables. The limitation of this solution is that the database will grow very quickly with support for new formats. It also makes it less resilient to changes from the CMS data format, i.e. every change in a CMS data format will also mean a change in ChairVise data schema.

The existing schema already contains the vital information that captures the basic relationships between the author, submissions and review entities (the metadata fields). We can expect that any new CMS will have a similar set of columns for each file. In which case, they could be mapped to the existing schema, and any mismatched columns could be either discarded or mapped into data fields of the existing schema.

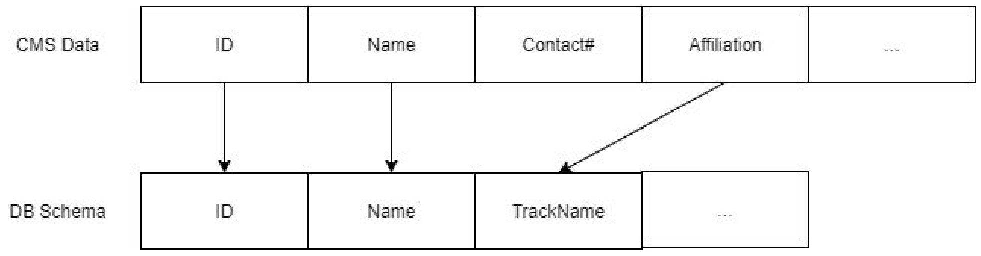


Fig 4.1.2.3A

Fig 4.1.2.3A shows a potential transformation, by choosing to discard the Contact# column and using the TrackName column to store affiliation data instead.

JavaScript Data Transformation

The data transformation is performed with a user supplied transformation function that accepts a row of data from the uploaded file as input, and returns a data object based on the database schema.

JavaScript is a natural choice for a web application. It is loosely typed and hence very forgiving, and comes with a powerful standard library for string, date and numerical data manipulations.

The transformations are performed on the client since the transformation function is potentially unsafe code.

4.1.3 Examples

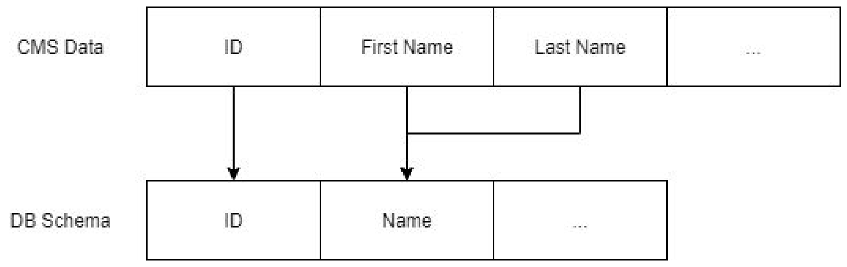
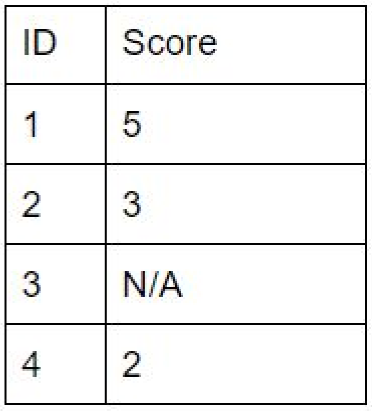


Fig 4.1.3A

Fig 4.1.3A shows a scenario where column to column mapping may be insufficient, and a simple transformation function may be used to concatenate data from two different columns to be stored together.



ID	Score
1	5
2	3
3	N/A
4	2

Fig 4.1.3B

Fig 4.1.3B shows an excerpt of a data file that contains data that users might want to clean. A simple transformation may be applied that replaces any non-integer values with the integer value 0.

4.2 Persistence and Management of Multiple Conference Data

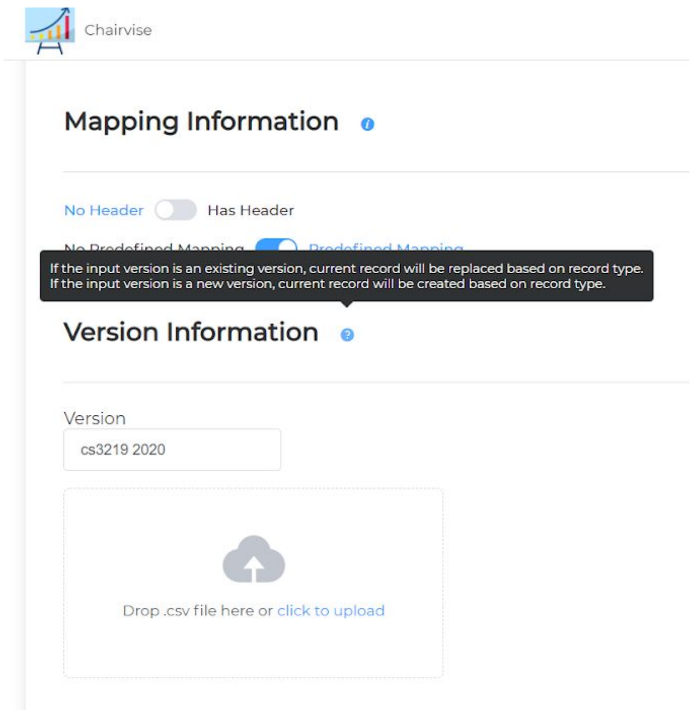
4.2.1 Existing Implementation

ChairVisE 3.0 supports the uploading of conference data. When uploading, the user specifies the “version” of the set of conference data files. For each version, one file of each type (author, submission, review) is allowed. If the user inputs a new version, the new version along with the records are stored into the database.

On the other hand, if the user inputs a version that already exists in the database, the newly uploaded records replace the existing records. On its own, this is not necessarily a bad feature as it is a potential use case for the user to wish to replace records when uploading to an existing version.

However, ChairVisE 3.0 does not present a way for the user to know what version and records already exist in the database. Importing to an existing version and replacing the existing files does not result in a different prompt to the user. As a result, users could mistakenly replace and lose records in the database that could still be important and irretrievable, without even being aware of it. This is aggravated by the fact that this crucial piece of information is very easily missed by the user, as it is conveyed via a pop-up that only shows if the user hovers over a minute icon. Figure 4.2.1A illustrates the icon and pop-up message when the user imports data in ChairVisE 3.0.

In addition, neither the deletion of versions and records, nor the renaming of versions are supported by ChairVisE 3.0. This represents a severe lack of CRUD features in the management of conference data.



If the input version is an existing version, current record will be replaced based on record type.
If the input version is a new version, current record will be created based on record type.

Figure 4.2.1A

4.2.2 Proposed Implementation

4.2.2.1 Overview

ChairVisE 4.0 introduces 2 enhancements related to management of data. Firstly, a “My Data” page was added to provide the user with a clear picture of existing data in the database and an interface to easily interact with these data. Secondly, instead of being limited to only adding data, user interactions with data is expanded to include full CRUD features, allowing the user to manage their conference data by adding, deleting and editing records and versions.

The combination of these 2 enhancements, explained in the following sections, directly fulfils FR-2 required of this feature.

4.2.2.2 My Data Page

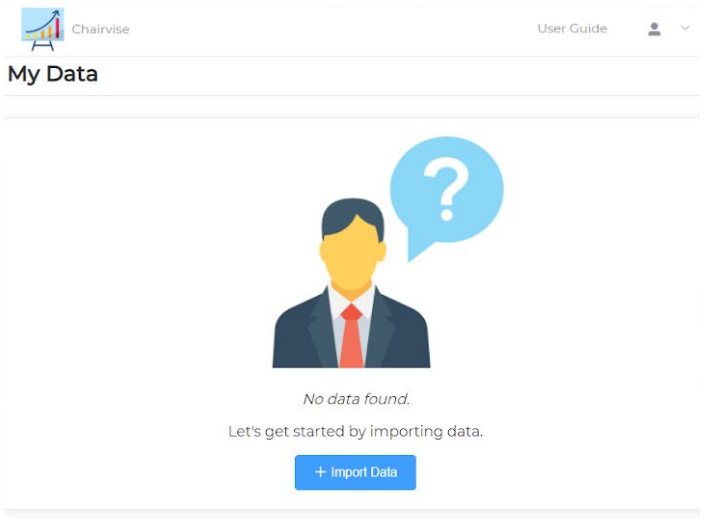
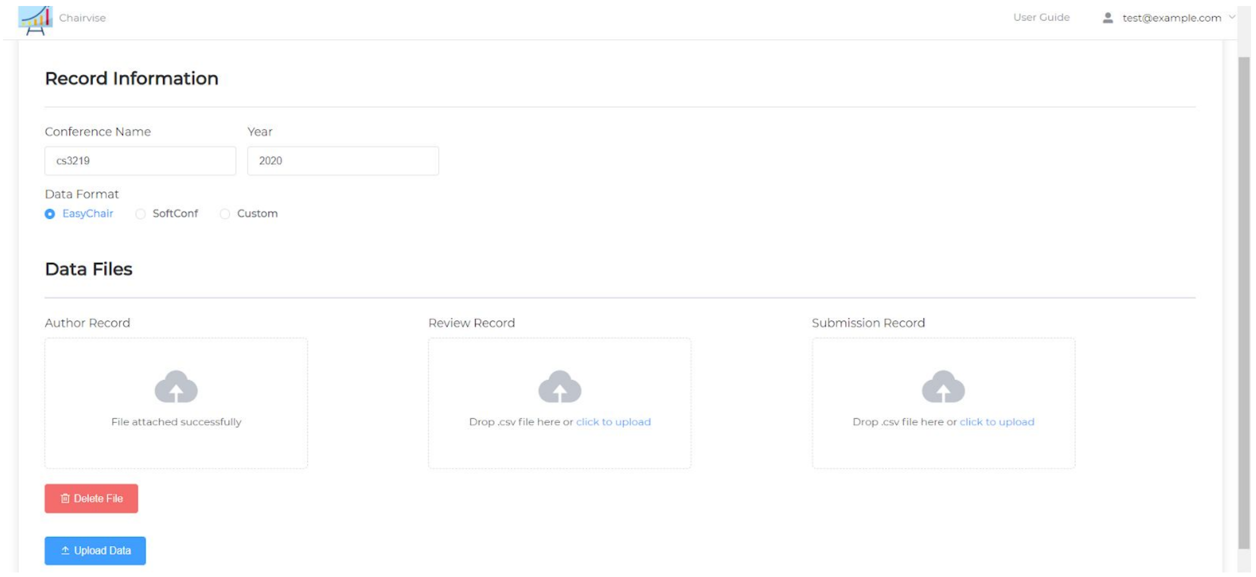


Figure 4.2.2.2A

Figure 4.2.2.2A shows the overview of the My Data page when the existing database is empty. The My Data page can be accessed via a drop-down menu in the top right. As illustrated in Figure 4.2.2.2A, the empty database is clearly made known to the user, who is in turn prompted to import data in order to begin using the full features of ChairVisE 4.0.



Record Information

Conference Name: cs3219
Year: 2020
Data Format: [✓ EasyChair] SoftConf Custom

Data Files

Author Record: File attached successfully
Review Record: Drop .csv file here or click to upload
Submission Record: Drop .csv file here or click to upload

Figure 4.2.2.2B

When the user chooses to import data, they are brought to the “Upload Data” page. Figure 4.2.2.2B shows the enhanced “Upload Data” page in ChairVisE 4.0. This is in contrast to that of ChairVisE 3.0 illustrated in Figure 4.2.1A.

Enhancements added to this page include:

- Simplifying “Version” into “Conference Name” and “Year”.
- Allowing the upload of all 3 types of conference files on the same page.

Simplifying “Version” into “Conference Name” and “Year” provides the user with an intuitive representation of a version. The underlying representation of a version remains, it is now simply the provided “Conference Name” and “Year” concatenated with an underscore. For example, in Figure 4.2.2.2B, the user creates a version with the name “cs3219_2020”.

Allowing the upload of all 3 types of conference files on the same page also provides the user a more intuitive data importing experience. Furthermore, if the wrong file were to be chosen mistakenly during import, the user can easily delete the file and switch to the correct file, all before upload is done. In ChairVisE 3.0, a mistake like this would result in the wrong file uploaded into the database.

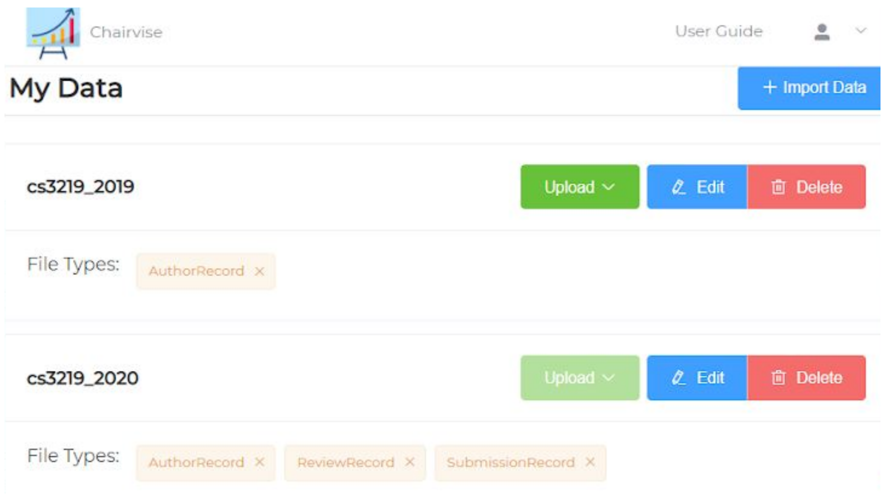


Figure 4.2.2.2C

Figure 4.2.2.2C once again shows the “My Data” page, this time with data in the database.

4.2.2.3 Expanded Data Management

Through several intuitive buttons in the “My Data” page, the user can interact and manage the records and versions uploaded to the database. CRUD operations previously supported by ChairVisE 3.0 are still supported, and made more intuitive by the “My Data” page.

1. Upload of new records to a new version.
   This is illustrated in the earlier section.

2. Upload of new records to an existing version.
   By clicking on the "Upload" button of the version, users can upload a record of types that the selected version does not have. For example, for version "cs3219_2019" in Figure 4.2.2.2C, users can upload a ReviewRecord, SubmissionRecord or both, as illustrated in Figure 4.2.2.3A.

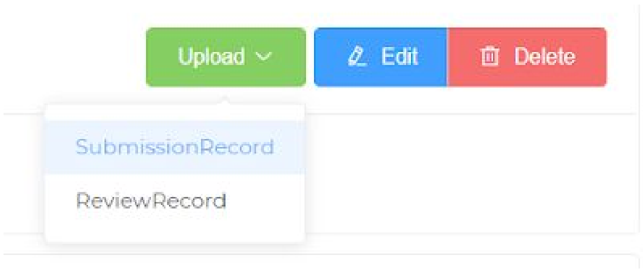


Figure 4.2.2.3A

3. Edit of the records in a version.
   This can be achieved by deleting the record from the version (illustrated in point 1 below), then uploading a new record to the version (illustrated in point 2 above).

Additionally, ChairVisE 4.0 supports the following CRUD operations previously impossible in ChairVisE 3.0:

1. Deletion of individual records from existing versions.
   By clicking on the "x" button of a record, it is deleted from the version, leaving other records in the version unscathed. This is illustrated in Figure 4.2.2.3B.

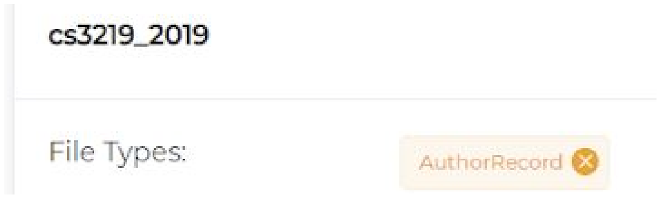


Figure 4.2.2.3B

2. Deletion of entire versions, along with all its records.
   By clicking on the "Delete" button of the version, the entire version and its records are deleted.

3. Edit of a version's name.
   By clicking the "Edit" button of the version, the version name can be changed. This is illustrated in Figure 4.2.2.3C.

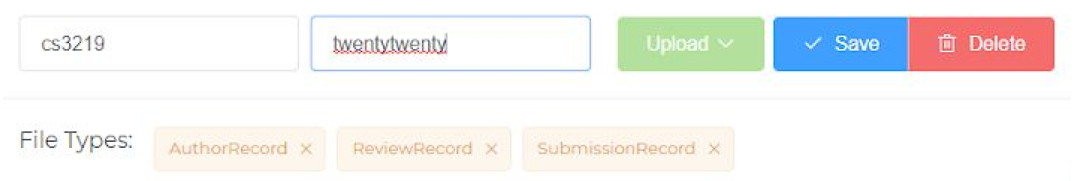
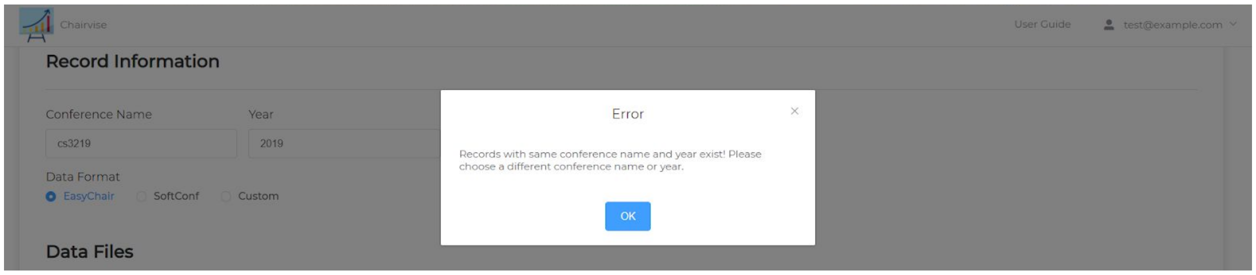


Figure 4.2.2.3C

In order to also fulfil FR-10 (failing CRUD transactions gracefully and showing the user appropriate error messages), error messages are also shown to the user if any CRUD transaction fails. An example of such a scenario is when the user imports records of a version that already exists in the database. Figure 4.2.2.3D shows the error message clearly explaining the scenario to the user. If the user truly wishes to replace existing records, he can do so via steps explained in 4.2.2.3 above.



Records with same conference name and year exist! Please choose a different conference name or year.

Figure 4.2.2.3D

4.2.2.4 Design Considerations

One design consideration was whether to continue applying the Single Responsibility Principle in the backend Logic component, described in 2.3.3 and 2.3.4.

In adding the additional CRUD operations supported in ChairVisE 4.0, the RecordController, VersionController and their corresponding logic RecordLogic and VersionLogic had to be edited. Specifically, API endpoints for “delete” and “put” were added. However, a complication arises for certain operations that require a change to both Version and Record. An example of such an operation is editing a version’s name. In order for RecordLogic to edit the version name of a record, the new version has to first exist in the VersionRepository, which is accessed only by VersionLogic due to Single Responsibility Principle. This presents a complication as the trivial solution of allowing RecordLogic to add a new version directly into the VersionRepository introduces undesirable coupling and violates SRP.

To solve the above conundrum, a design decision was made to continue to apply the Single Responsibility Principle and thus deny RecordLogic access to VersionRepository. An alternative solution was thus for frontend to make a series of API calls to the appropriate Controllers in the correct order in order to achieve the same effect. The API calls required to edit a version’s name are in the order as follows:

1. “Post” API calls to Version to create the new version.
2. “Put” API calls to Record to change the version name of each record to the new version.
3. “Delete” API call to Version to delete the old version.

This flow is illustrated in Figure 4.2.2.4A below.

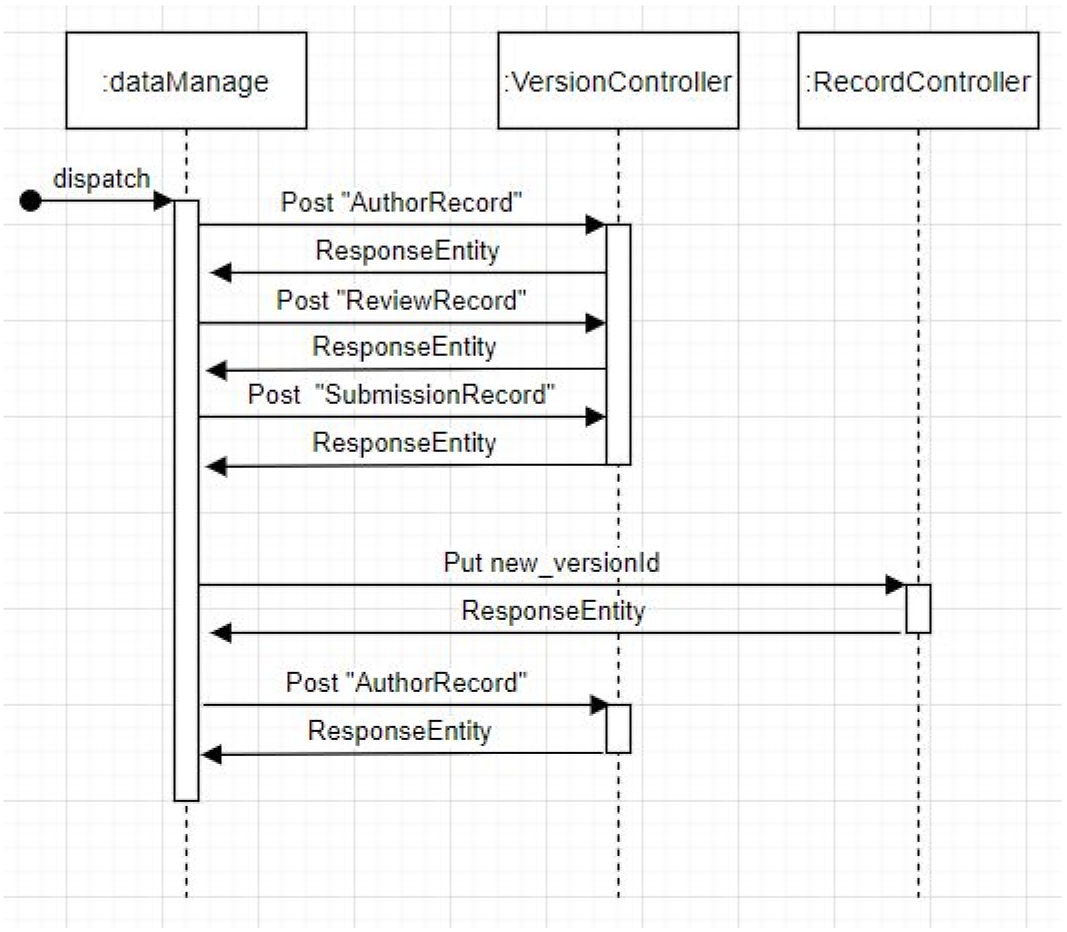


Figure 4.2.2.4A

4.3 Improvements to the User Interface and User Experience

This section explains several improvements made to UI/UX, their rationale and implementation details. Analyses of existing implementations and details of proposed improvements are discussed within subsections respectively.

4.3.1 Improved User Flow of Presentation Creation

Using uploaded conference data sets, users of ChairVisE can create meaningful visualisations that can be shared with the audience. The collection which contains a set of visualisations is called a presentation. According to the existing use case in ChairVisE 3.0, all visualisations from a presentation should be generated from only one data set, as it makes sense for the conference organisers to focus on presenting insights from one conference in one presentation.

However, there are some design issues in ChairVisE 3.0 which prevented this point from being conveyed to the users, leading to a confusing user experience. Specifically, we would like to highlight the following 2 design flaws: i) users can create a presentation without specifying a data set, and ii) the option of selecting a data set appears in a not-so-prominent position only after adding a chart of any type.

The user flow of creating a new presentation in ChairVisE 3.0 is illustrated in Figure 4.3.1A:

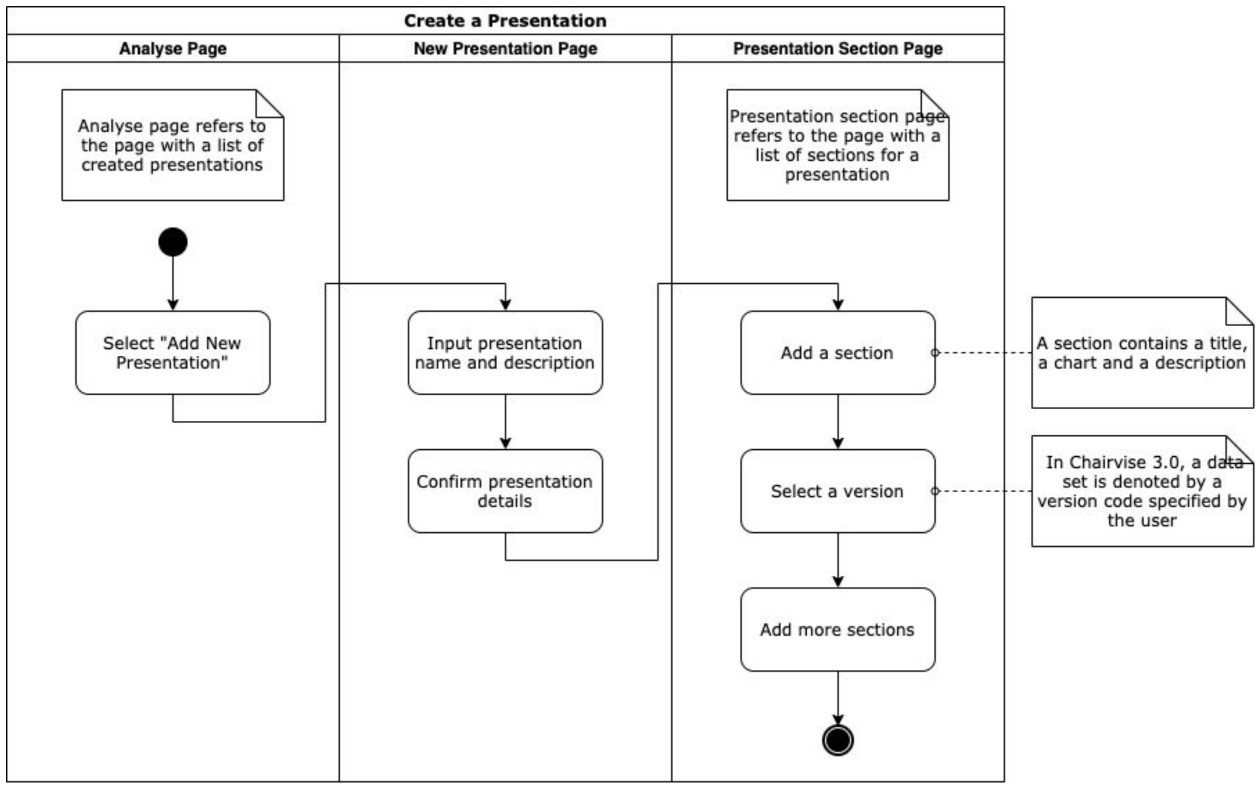

Create a Presentation
Analyse Page
New Presentation Page
Presentation Section Page
Analyse page refers to the page with a list of created presentations
Select "Add New Presentation"
Input presentation name and description
Confirm presentation details
Presentation section page refers to the page with a list of sections for a presentation
Add a section
Select a version
Add more sections
A section contains a title, a chart and a description
In Chairvise 3.0, a data set is denoted by a version code specified by the user

Figure 4.3.1A

As observed from the user flow, although selecting a data set is important for creating a presentation, in ChairVisE 3.0 it is left at the end of the presentation creation process. Its importance is not correctly signalled to the user. In order to address the design flaws, in ChairVisE 4.0, we have refined the user flow as illustrated in Figure 4.3.1B:

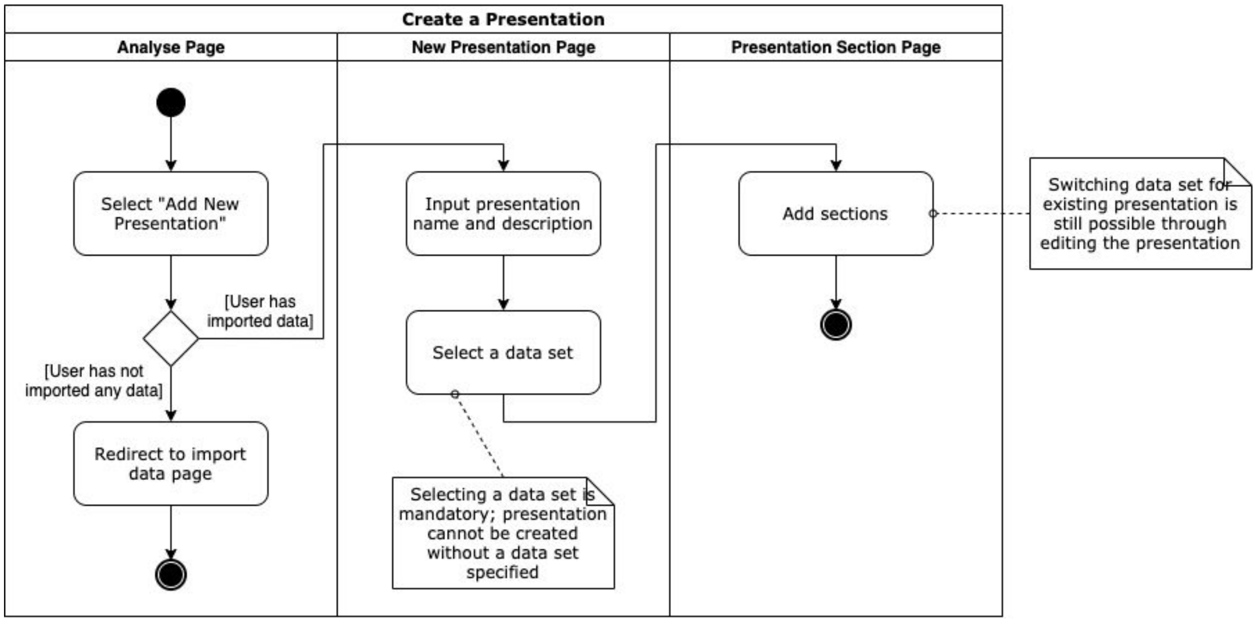
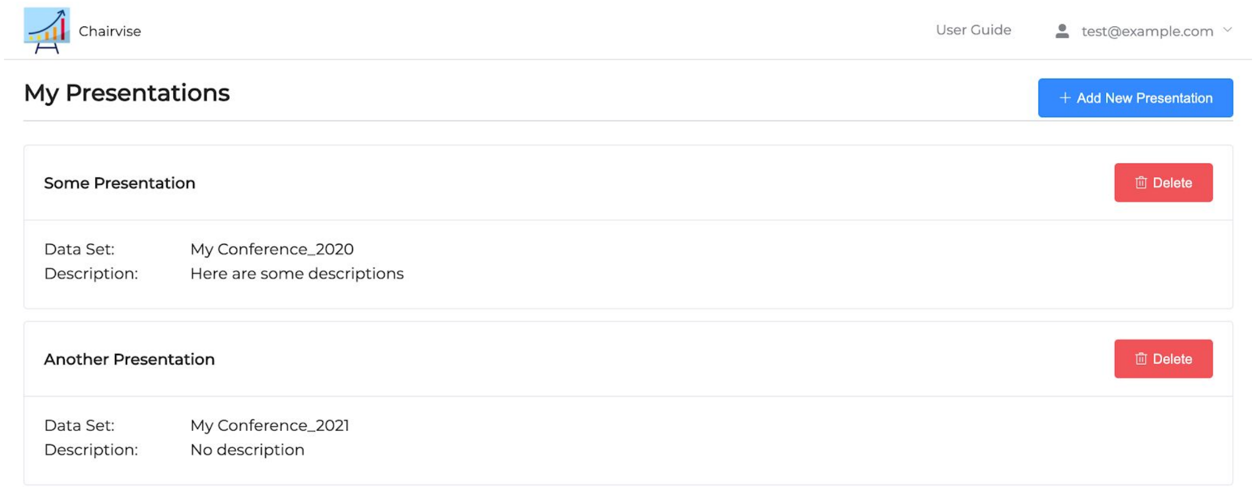

Create a Presentation
Analyse Page
New Presentation Page
Presentation Section Page
Select "Add New Presentation"
[User has imported data]
[User has not imported any data]
Redirect to import data page
Input presentation name and description
Select a data set
Selecting a data set is mandatory; presentation cannot be created without a data set specified
Add sections
Switching data set for existing presentation is still possible through editing the presentation

Figure 4.3.1B

In the improved user flow of ChairVisE 4.0, to highlight the importance of the data set as the source of information for the visualisations, we make it compulsory for a newly created presentation to be associated with a data set (FR-6), although the data set can be changed later. As a result, users are not allowed to create presentations if they have not imported any data. Furthermore, we have reduced the steps required by removing the confirmation dialog before presentation creation, since all the information is modifiable and it is redundant to have the confirmation for an easily reversible action.

As part of the improvements to emphasise the role of the data set in presentations, we have also included the data set label in the presentations list page, as shown in Figure 4.3.1C. As a side improvement, the delete operation is also made available from the presentation list.



My Presentations

Some Presentation

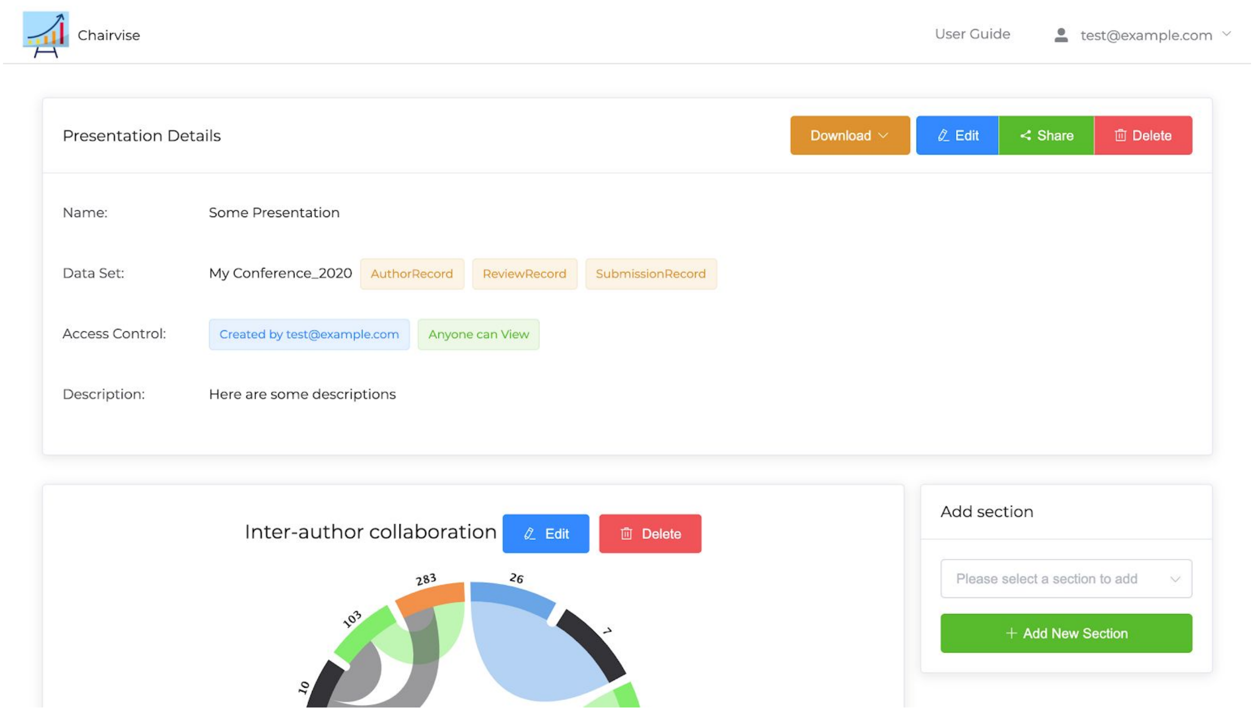
Data Set:	My Conference_2020
Description:	Here are some descriptions

Another Presentation

Data Set:	My Conference_2021
Description:	No description

Figure 4.3.1C

The user interface of the presentation detail page has also been improved, as shown in Figure 4.3.1D. Most notably, the data set is now moved to the presentation brief component, and is modifiable once the edit button is clicked. The label also displays which file format(s) of the three (author, review, submission record) is/are available in that particular data set (FR-11). Users can only add relevant types of charts that are supported by the current file formats in the data set to a presentation. Some other improvements in this page include making the “Add section” panel always floating at the side, adding the access control tags to the presentation brief component (FR-12), limiting the types of visualisations according to the available file types in the data set (FR-13), and making the overall layout more succinct and organised.



Name: Some Presentation

Data Set: My Conference_2020 AuthorRecord ReviewRecord SubmissionRecord

Access Control: Created by test@example.com Anyone can View

Description: Here are some descriptions

Figure 4.3.1D

Implementation wise, we made use of Vuex’s getter to reduce code redundancy and coupling. Vuex keeps a single state tree for an application, and this state tree can be splitted into subtrees called modules. In a module, there is a state (with many state variables), several mutations (to update the state), actions (to make asynchronous calls), and getters (to compute data based on state, similar to computed variables in Vue components), In ChairVisE 3.0, getters are not used.

However, we do notice some code redundancy without the use of getters. This is illustrated in Figure 4.3.1E. From the sequence diagram, we notice that the method getVersionIdList is implemented multiple times in different Vue components despite serving the same functionality. Also, this makes Vue components highly coupled to the Vuex store. If the format versionList changes (such as renaming the field name versionId to datasetId), it is necessary to change the method getVersionList in many components. This coupling between components and data source is undesirable. This is one of the examples and there are other instances where getters can be used (such as getFileTypes).

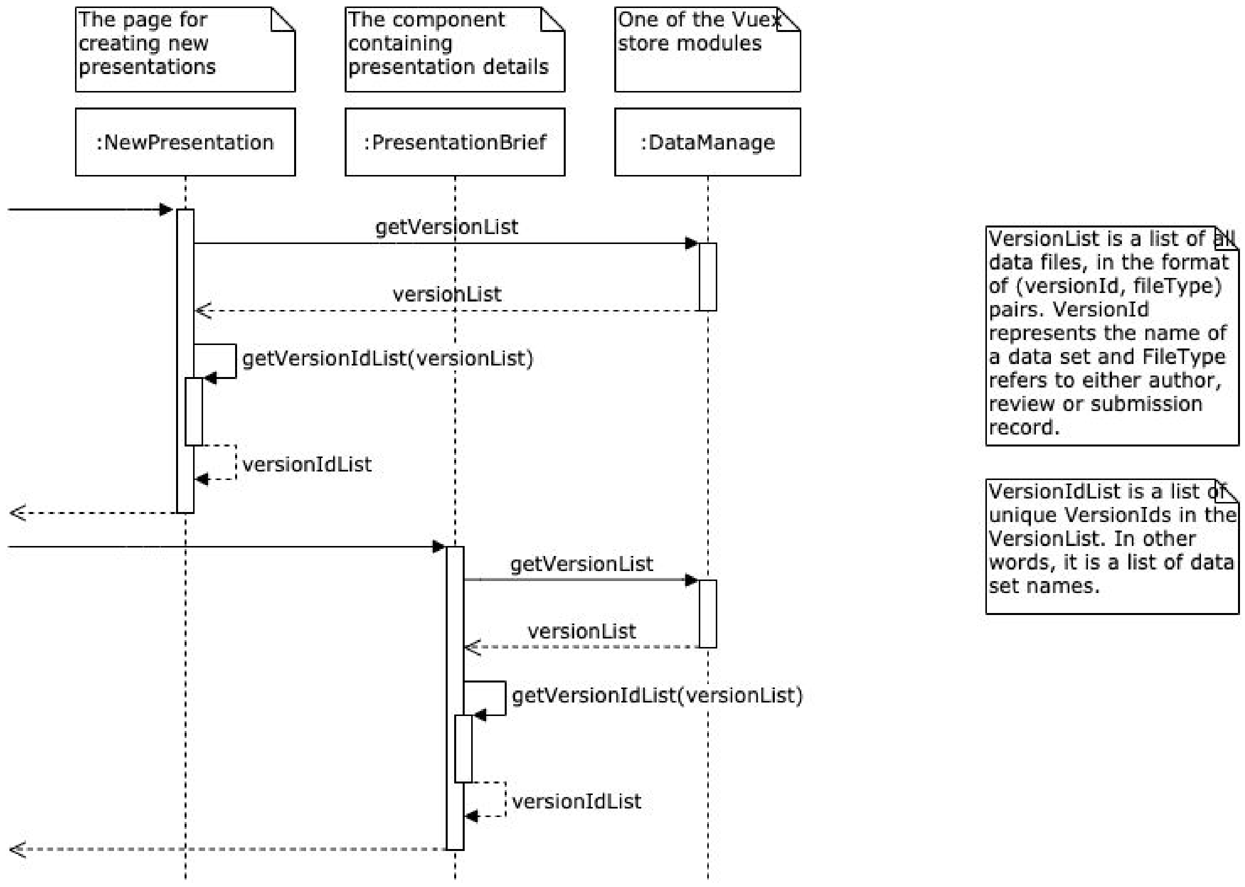

The page for creating new presentations
The component containing presentation details
One of the Vuex store modules
:NewPresentation
:PresentationBrief
:DataManage
getVersionList
versionList
getVersionIdList(versionList)
versionIdList
VersionList is a list of all data files, in the format of (versionId, fileType) pairs. VersionId represents the name of a data set and FileType refers to either author, review or submission record.
VersionIdList is a list of unique VersionIds in the VersionList. In other words, it is a list of data set names.

Figure 4.3.1E

As shown in Figure 4.3.1F, using the Vuex getters reduces code redundancy and coupling. The getter, getVersionIdList, is only implemented once within the Vuex store module and can be called by other Vue components. Moreover, it hides the actual implementation of VersionList from UI components. This reduces the coupling between Vue components and the Vuex store. The use of getters also conforms to the Separation of Concerns (SoC) Principle where the Vuex store is responsible for storing, computing, manipulating data, and other UI components should be responsible for data presentation only, and should not concern how to compute the data.

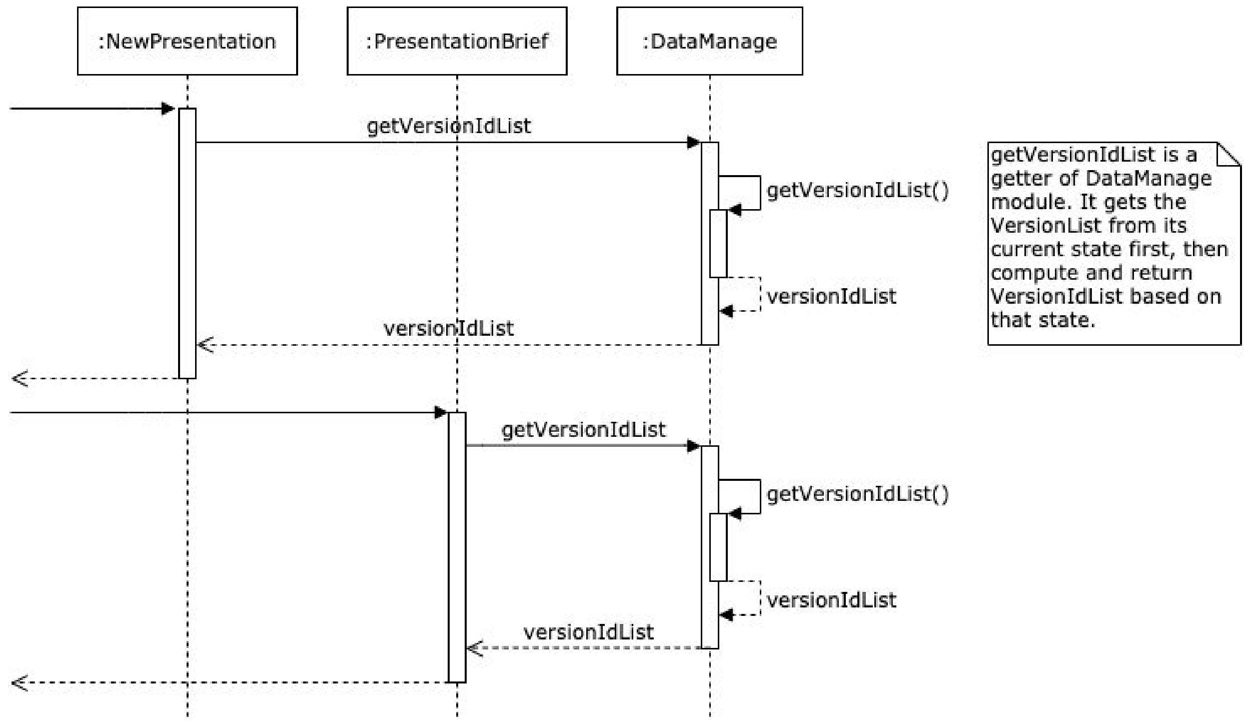


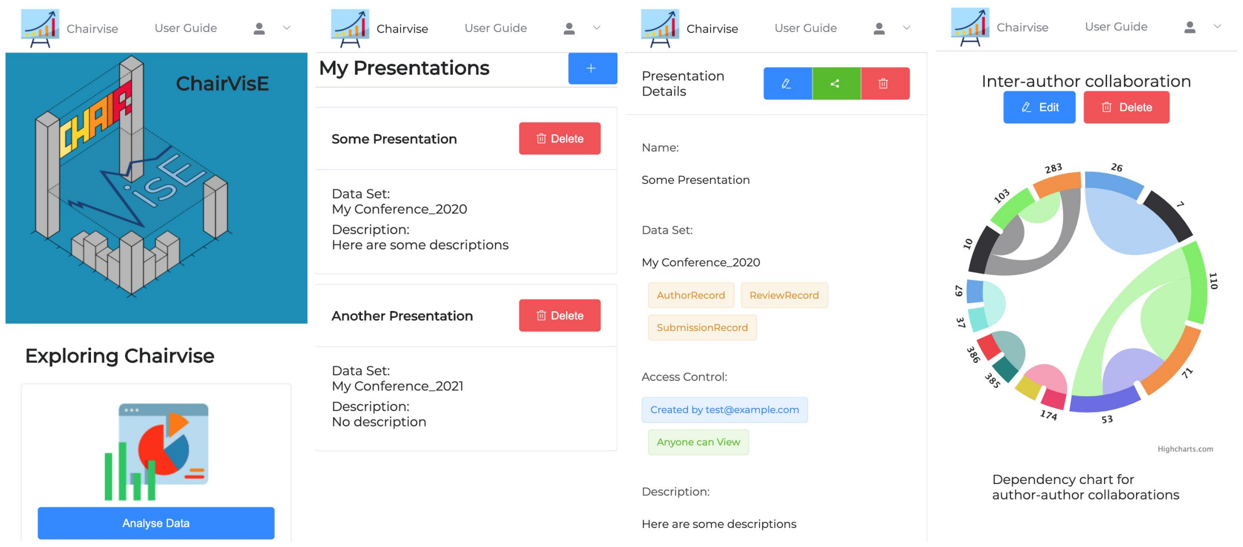
Figure 4.3.1F

4.3.2 Responsive User Interface

ChairVisE 3.0 is primarily a web application for desktop users. Although it can also be accessed using mobile devices, there is no responsive design in the user interface and components are often displayed incorrectly, making the application difficult to use, and sometimes unusable from mobile devices. ChairVisE 4.0 addresses this issue by introducing responsive UI design to the application (FR-8). Instead of integrating responsive UI design to the entire app, we focus more on the functionalities that are most likely to be accessed by users via mobile devices.

As per use cases of ChairVisE 4.0, we would like to allow the presentation audience to access the shared visualisations via mobile devices. Furthermore, in the case where the conference organiser would like to show a visualisation but does not have a desktop, it would be convenient if the conference organiser can also access the presentations using a mobile phone. Therefore, we decided to introduce responsive UI primarily for the presentation page.

Figure 4.3.2A shows the relevant pages with responsive UI design integrated.



Some Presentation

Data Set:
My Conference_2020
Description:
Here are some descriptions

Another Presentation

Data Set:
My Conference_2021
Description:
No description

Presentation Details

Name:
Some Presentation

Data Set:
My Conference_2020

Access Control:

Description:
Here are some descriptions

Inter-author collaboration

Dependency chart for author-author collaborations

Figure 4.3.2A

Regarding the implementation, the components are made responsive with various means. First of all, Element UI, the UI library used in ChairVisE, has support for responsive grids, allowing grid columns to have different width depending on screen size. Secondly, responsive design is achieved with the use of css, either through predefined css classes provided by Element UI or @media rule. Lastly, for some Vue components, a computed variable isMobile is added so that components may exhibit different behaviour depending on screen size.

4.3.3 File Importing in a Batch

The frontend UI, model and logic of the data import page are redesigned to allow importing different record types in a batch to form a complete dataset (FR-3). The UI is already shown in part 4.2.2.2. Hence, changes to the model and logic will be discussed here.

Regarding the model and logic, in ChairVisE 3.0, data is the object that holds relevant information associated with a record in the Vuex state of dataMapping module. Specific fields such as tableType, uploadedData, mappingResult are members of the data variable. In ChairVisE 4.0, we have encapsulated these relevant fields into a new object called record, and records, a member of the data object, is an array holding multiple instances of record. The class diagram is shown in Figure 4.3.3B.

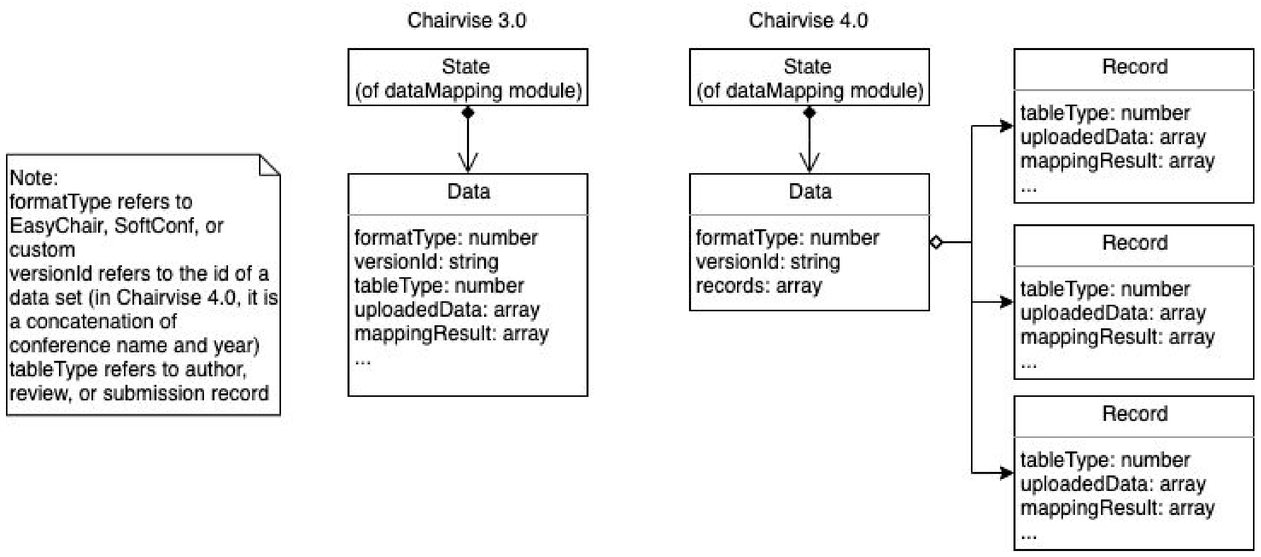

Chairvise 3.0
Chairvise 4.0
State (of dataMapping module)
Data
formatType: number
versionId: string
tableType: number
uploadedData: array
mappingResult: array
...
records: array
Record
tableType: number
uploadedData: array
mappingResult: array
...
Note:
formatType refers to EasyChair, SoftConf, or custom
versionId refers to the id of a data set (in Chairvise 4.0, it is a concatenation of conference name and year)
tableType refers to author, review, or submission record

Figure 4.3.3B

After each record is processed and stored in the Vuex state, once the user clicks the upload button, asynchronous post requests will be made to the server in parallel, uploading all records in a batch. Furthermore, we fixed a bug in ChairVisE 3.0 where record uploading can be unsuccessful if record endpoint is reached before version endpoint. In ChairVisE 4.0, the post record requests are only made after successful responses are received from the version endpoint. In future versions of the application, we envision that only post requests to records endpoints would be necessary and versions should be created at the backend. This will ensure that the process of uploading records is atomic.

4.3.4 Making User Guide Accessible without Login

In ChairVisE 3.0, the user guide is not accessible from the home page without login. However, as first time users, they are likely to want to find out more about what ChairVisE can do before login or registering an account. Therefore, we have made the user guide accessible from the home page without login (FR-7). Moreover, recognising the importance of the user guide, we made the link to the user guide persistent on the navigation bar, next to the user account.

4.4 Add Test Infrastructure and Test Cases

The goal of this enhancement is two-fold. First was to provide the infrastructure necessary to fulfill FR-4 and second was to try and fulfill FR-14 & FR-15 where possible. The two objectives were defined as such due to the relative priorities of the associated functional requirements. A breakdown of the functional components for these objectives are provided below.

4.4.1 Testing Infrastructure

The objective in this segment is to address FR-4. Essentially, we wanted to enforce testing as part of the Continuous Integration (CI) pipeline. In order to do so, we first had to fix the CI build as we inherited a failing build. This was done by adding the missing dependency in the frontend test file used as part of the CI check.

After fixing the CI build, we configured GitHub to add branch protection rules on our master branch. This enforced two requirements: (1) All commits to master must be merged from a PR, and (2) All PRs must pass the CI check to be allowed to merge. A summary of the eventual CI process resulting from these configurations is shown in Figure 4.4.1 below.

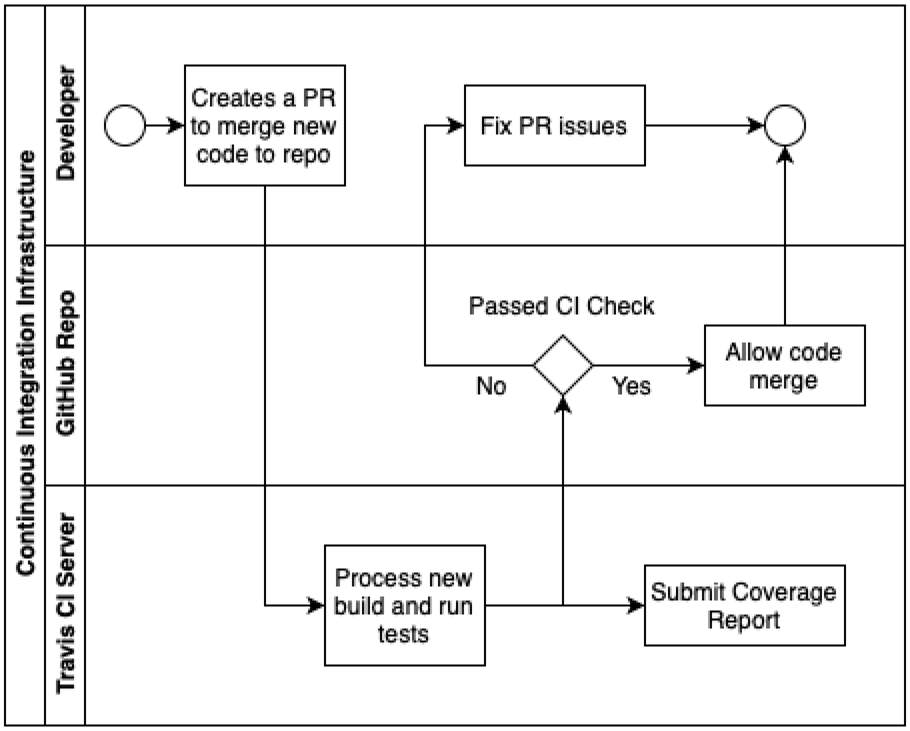


Figure 4.4.1 - DevOps Infrastructure Swimlane

An important component of this process is feedback to the developer on changes in code coverage. This was intended to motivate developers to write tests alongside any code contributed to the main repository. In order to achieve this, we needed to configure the production of coverage reports. This was done via JaCoCo in the backend and Jest in the frontend.

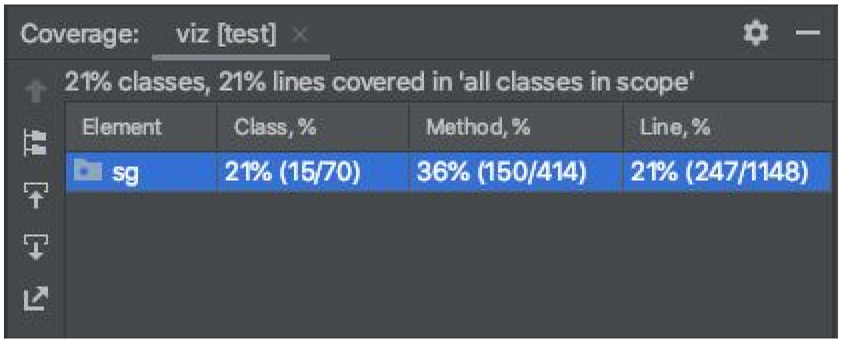
After running all the tests, the CI server collects the coverage reports and sends the data to coveralls. Coveralls then collates the different (frontend and backend) reports to provide a holistic view of changes in coverage resulting from each PR build. This is an accurate reflection of changes to the main repository coverage as all PRs are required to be updated with the master branch.

4.4.2 Backend Test Cases

Backend test cases were designed to maximise method coverage. The challenge however was that even though the system is a SpringBoot application, user services (e.g. authentication and user management) were managed by Google App Engine (GAE). This meant that we had to first set up a GAE Simulation class to mock the dependency on GAE services.

The other challenge came in trying to mock both the web MVC as well as data JPA services. SpringBoot however provided annotations which helped provide autowired dependencies based on the class tree of the main SpringBoot application. This class tree is built via annotations in the main class application (e.g. @Component).

We managed to achieve 36% method coverage of the entire backend system. Evidence of this achievement is shown in Figure 4.4.2 below. Though substantially less than our original aim of 60%, this coverage is deemed satisfactory due to the relatively low priority given to FR-14.



21% classes, 21% lines covered in 'all classes in scope'

Element	Class, %	Method, %	Line, %
sg	21% (15/70)	36% (150/414)	21% (247/1148)

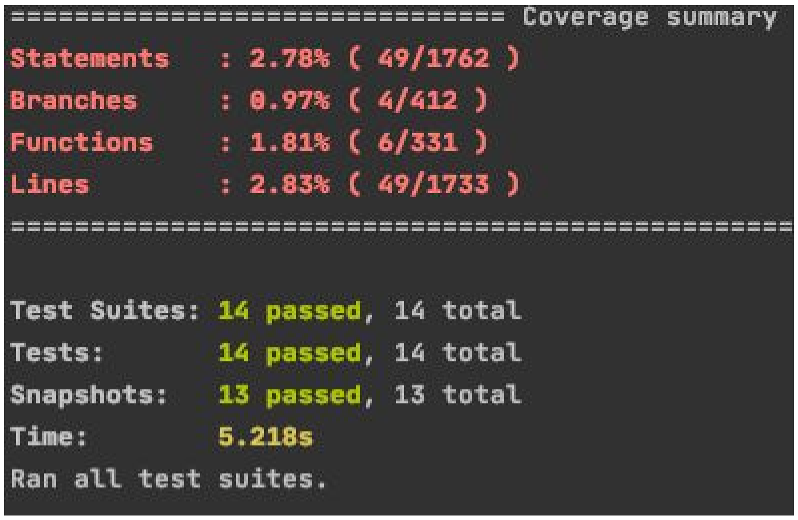
Figure 4.4.2 - Backend test coverage from IntelliJ

4.4.3 Frontend Test Cases

Backend tests were designed primarily as snapshot tests. This was to ensure that any changes to the frontend UI was a deliberate consequence of development and not an unintended one. Since the frontend was undergoing an extensive overhaul, this approach was deemed the most appropriate tradeoff between efficiency and effectiveness. That is to say, though the tests might not be as effective at preventing regression in terms of functionality, it is relatively much more efficient to setup and run as opposed to unit testing individual functions.

The primary alternative that was used to compare efficiency vs effectiveness was unit testing with Jest. We would have had to mock the various dependencies with spies and test individual functions. The problem was that since we were making quite a lot of changes to the frontend, we would then have to rewrite these tests, and so snapshot testing won out.

Since we primarily utilised snapshot testing however, our eventual coverage for the frontend as reported by Jest is quite low. The frontend functionality coverage is shown in Figure 4.4.3 below. This result however is acceptable considering that snapshot testing does not cover function methods and that the FR fulfilled by this enhancement was given a low priority.



```
=============================== Coverage summary
Statements   : 2.78% ( 49/1762 )
Branches     : 0.97% ( 4/412 )
Functions    : 1.81% ( 6/331 )
Lines        : 2.83% ( 49/1733 )
================================================

Test Suites: 14 passed, 14 total
Tests:       14 passed, 14 total
Snapshots:   13 passed, 13 total
Time:        5.218s
Ran all test suites.
```

Figure 4.4.3 - Frontend test coverage from vue-test-cli

5. Suggested Improvements

5.1 Add User Authentication, Migrate to Java 11

As mentioned before in 4.4.2, ChairVisE though being a SpringBoot application is very reliant on services provided by GAE. One of the biggest reliance being authentication. Migrating to OAuth will decouple that reliance on GAE and allow for more developer controlled user management. Additionally from a DevOps perspective, decoupling these services will allow for migration to Java 11 (instead of the current Java 8) which provides much better performance and support from the GAE environment[1].

5.2 Security Enhancements

The current ChairVisE implementation has no safeguards to protect against well known security vulnerabilities. Two of the most prominent vulnerabilities that ChairVisE is susceptible to are SQL Injection (SQLI) attacks and Cross Site Request Forgery (CSRF) attacks. SQLI attacks can be carried out via the custom SQL feature which additionally has no input validation while CSRF attacks can be carried out due to the lack of anti-CSRF token present in http communications between the frontend and backend.

5.3 Add End-To-End Testing

The test suites added as part of our enhancements are all implementations of white-box testing. A good addition to have is some form of black-box testing that assumes no knowledge of the internal workings of the system. This helps with quality assurance and ensuring that user facing

[1] https://cloud.google.com/appengine/docs/standard/java11/java-differences#user_authentication

features and functions are not compromised during development. One possible framework that can be leveraged for E2E testing with the current ChairVisE architecture is Cypress.

6. Development Process

Our team adopted the breadth first iterative software development life cycle. Additionally, the Scrum framework was employed, revolving around **weekly sprints and stand up meetings** where we review our progress and discuss goals for the coming week.

For the project codebase, contributions were done via **pull requests** to the master branch where members had autonomy to choose between branching or forking models. Each pull request had to **pass the Travis CI pipeline** to be merged and ideally should have been reviewed by at least one other team member.

In terms of organisation, the team had no fixed roles for development. Instead, each feature is assigned a manager who oversees the direction of development. We also focused on parallel development of enhancements where possible, since most of the enhancements were relatively decoupled from each other.