School of Computing

CS3219 Software Engineering Principles & Patterns

ChairVisE 3.0

Final Report

Gan Ming Rui Jeff
A0167926N

Tham Si Mun
A0171711N

Lim Wai Lun
A0167102

Lim Zhi Yu, Thaddeus
A0176824W

Code Repository URL:
https://github.com/CS3219-SE-Principles-and-Patterns/chairvise3-0-ghost-5

Individual Contributions to Project

Name	Technical Contributions	Non-Technical Contributions
Jeff Gan	User Authentication (Backend) Persistent Data (Backend) Nearly everything else backend-related	Documentation
Lim Wai Lun	Data Mapping feature (Frontend + Backend) Templates feature for data mapping (Frontend + Backend)	Documentation
Tham Si Mun	Persistent Data (Frontend) Data Management (Frontend) API requests	Documentation
Thaddeus Lim	Improvement to User Interface User Authentication (Frontend) Data Mapping feature (Frontend)	Documentation

Contents

1. Introduction

Academic and industry conferences are frequently held worldwide to share and demonstrate their new products or research. Such conferences are essential platforms for researchers to facilitate their discussion. However, in these conferences, the chairs have to report some important statistics from all the research paper submitted in the conference. Due to the huge amount of research papers usually submitted during a conference, this application ChairVisE aims to help the conference program chairs to share and visualize the statistics from these research papers more easily.

While ChairVisE 2.0 has some essential features that allows the chairs to share and visualize the statistics from the research papers, ChairVisE 3.0 aims to enhance these features and introduce even better and more refined features.

2. Existing Version

2.1 Design

Overall Architecture

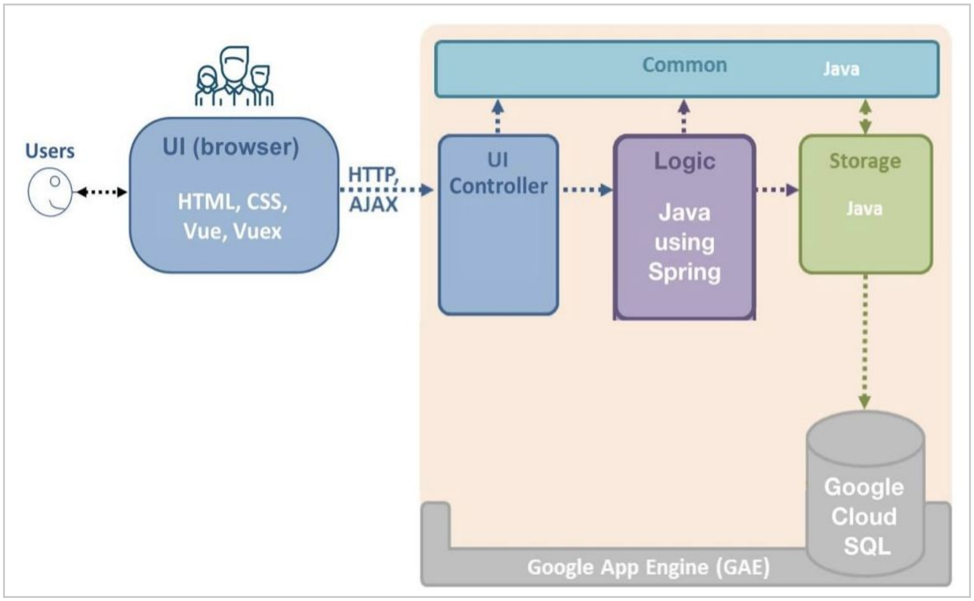


Figure 1: Overall architectural diagram of ChairVisE2.0

ChairVisE 2.0 is designed with a layered design pattern where each layer (e.g Logic, Storage etc) has its own responsibility. The layers consist of:

UI: Directly handles HTTP requests from the frontend/browser and calls the necessary Logic components to handle the requests.
Logic: Handles the backend Logic and coordinates Authentication, Presentation, Analysis etc using Spring Framework
Storage: Handles changes to the database which uses MySQL 5.6
Common: Contains utility code (e.g. helper classes) that is used by UI, Logic, and Storage

FrontEnd Architecture

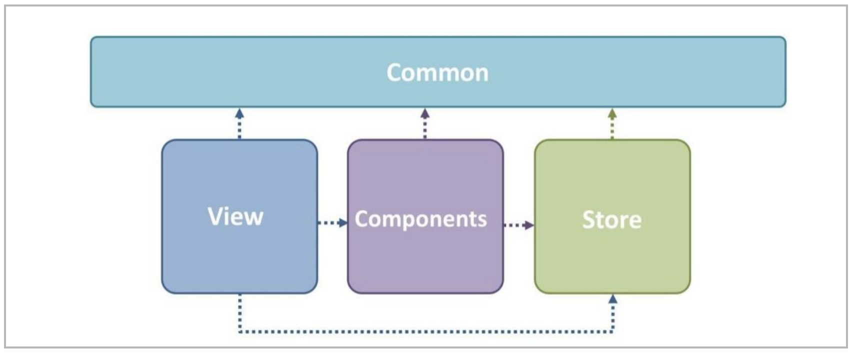


Figure 2: Frontend architectural diagram of ChairVisE2.0

View is in charge of displaying pages of the application, composed of components.
Components are reusable UI and display logic which then can be called by multiple pages.
Store contains core logic of the application. It maintains the state of the application and the associated data; it also handles mutations and actions to the state committed/dispatched by components using Vuex.
Common contains utility code and constants used.

Vue components are bound to the model such that when the state changes, the vue component is updated. ChairVisE2.0 uses the Vuex library, which is a flux-like pattern that implements a global store. With Vuex, a vue component can commit mutations and dispatch actions to the store to change its state or do asynchronous operations such as making HTTP requests, which can again cause vue components to be rerendered. The figure below succinctly sums up the unidirectional data flow in Vuex.

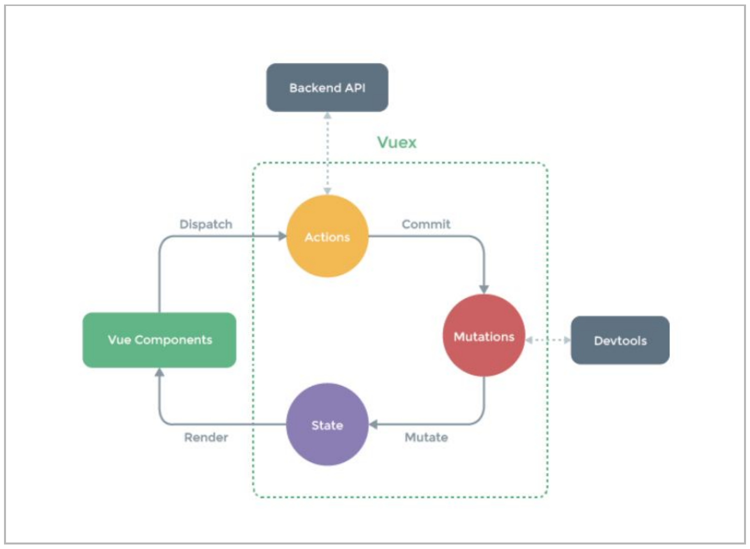
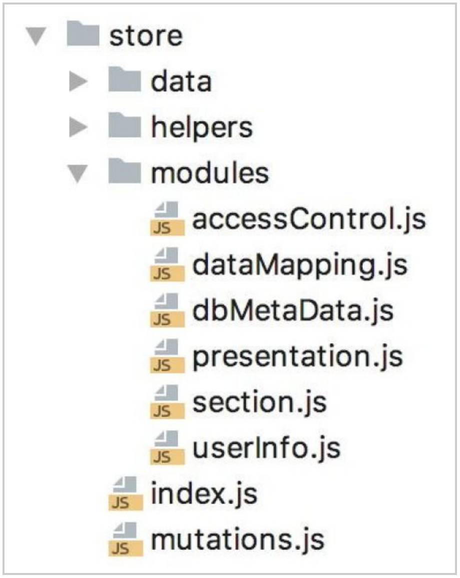


Figure 3: Unidirectional flow of data using Vuex

Because of the global state, the frontend of ChairVisE2.0 has a shared repository design pattern where data is maintained in a central repository which is the Vuex store. Data flows into the components from the store, and components can change its state through committing mutations or dispatching actions using the Vuex pattern. Changes in state will be propagated to the component and the component will render based on the state data. We see that the overall architecture of the front end uses Vuex to ensure that the state can only be mutated predictably. As such, the control flow of the application logic is then coordinated by the state of the data.



```
▼ store
    ▶ data
    ▶ helpers
    ▼ modules
        accessControl.js
        dataMapping.js
        dbMetaData.js
        presentation.js
        section.js
        userInfo.js
    index.js
    mutations.js
```

Figure 4: Store component of ChairVisE3.0

In ChairVisE2.0, the store component has the modules shown in the figure above. The Single Responsibility Principle is applied when designing the modules. The state is broken down into several modules to ensure that it is simpler to reason about the logic and state of the program.

Backend Architecture

The backend of ChairVisE2.0 uses Spring MVC and Spring Boot frameworks. This reduces the coupling between components as Spring automatically handles dependency injection for the components. Spring uses a Singleton pattern (Spring beans managed within a Spring container) as services should be stateless and hence would not require multiple components. This allows easy instance control and flexibility of instantiation.

Existing ChairVisE2.0 also uses the Google App Engine. In particular, the GAE's UserService follows a Factory pattern for the creation of **User** class objects.

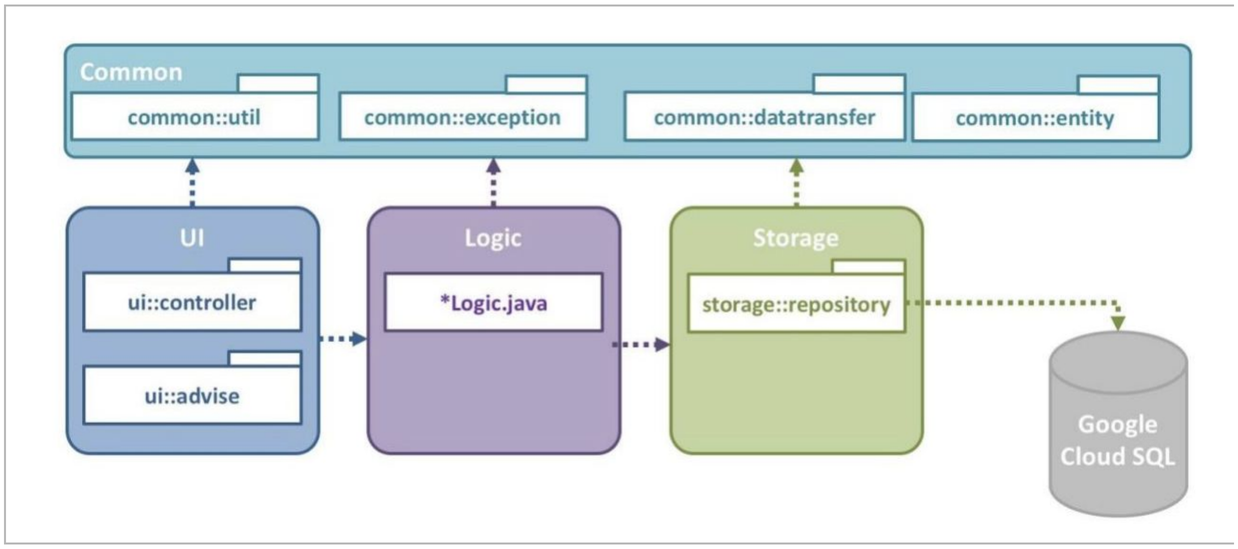


Figure 5: Backend architecture diagram

<u>**UI Component** - Provides backend Representational State Transfer (REST) through controllers</u>

AnalysisController: Handle analysis requests sent by the frontend and issue SQL query to the database
AuthInfoController: Check the current authentication status of user
DBMetaDataController: Exposes the metadata, including name and type of the standard data template used in the application.
PresentationController: Handles CRUD operations of presentations created by users
PresentationSectionController: Handles CRUD operations of presentation sections in presentations

PresentationAccessControlController: Handles CRUD operations that require checking of access (Example: to see shared presentation, user has to have at least read access)
RecordController: Store CSV data imported by users in the database
WebPageController: Serves the static production files built by Vue

Controllers	**API Requests**
Analysis Controller	POST /api/presentations/{id}/analysis
AuthInfoController	GET /api/auth
DBMetaDataController	GET /api/db/entity

PresentationAccessControlController	GET /api/presentations/{presentationId}/accesscontrol
	POST /api/presentations/{presentationId}/accesscontrol
	PUT /api/presentations/{presentationId}/accesscontrol/{accesscontrolId}
	DELETE/ /api/presentations/{presentationId}/accesscontrol/{accesscontrolId}

PresentationController	GET /api/presentations
	POST /api/presentations
	PUT /api/presentations/{presentationId}
	GET /api/presentations/{presentationId}
	DELETE /api/presentations/{presentationId}

PresentationSectionController	GET /api/presentations/{presentationId}/sections
	POST /api/presentations/{presentationId}/sections
	PUT /api/presentations/{presentationId}/sections/{sectionId}
	DELETE /api/presentations/{presentationId}/sections/{sectionId}

RecordController	POST /api/author POST /api/review POST /api/submission
WebPageController	GET /web/*

Table 1: API controllers and the relevant API calls

<u>Logic Component</u>

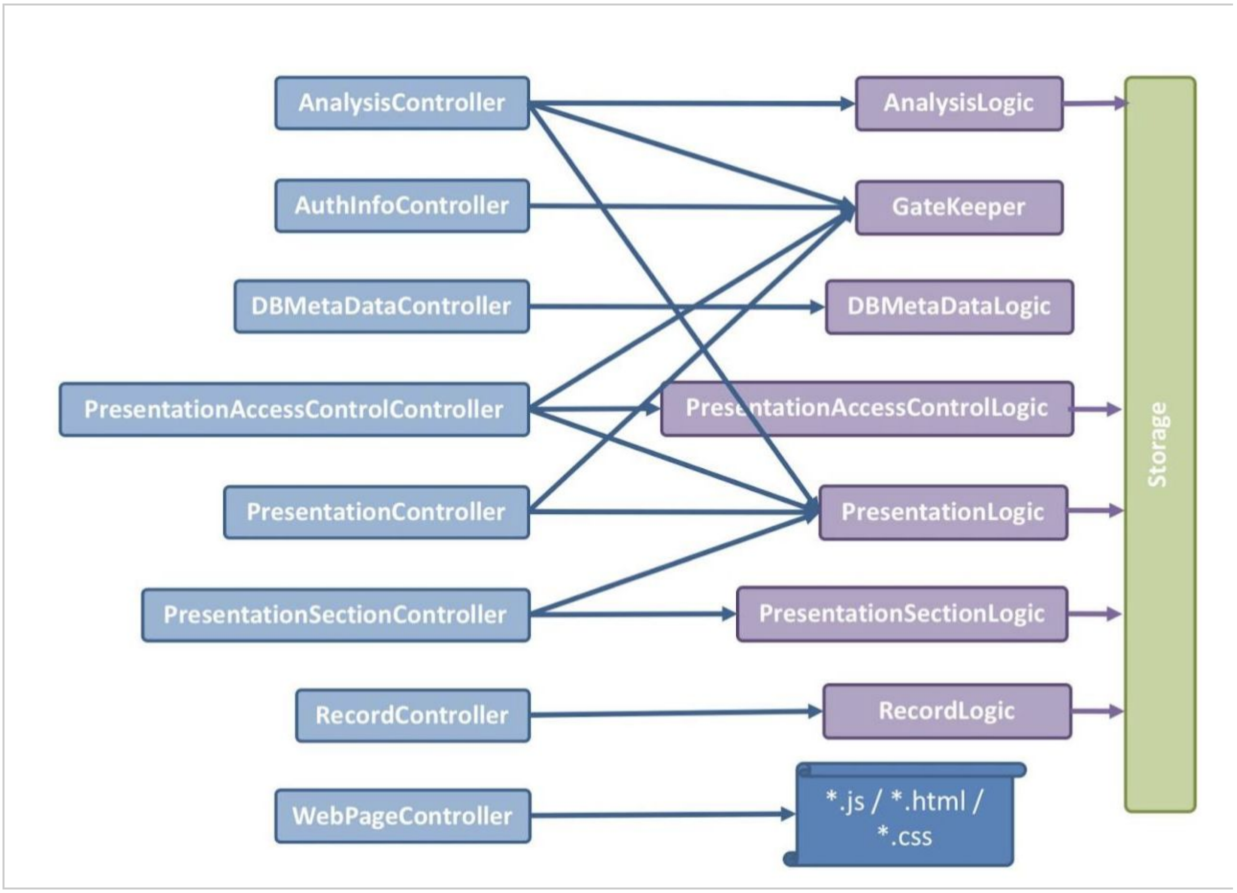


Figure 6

Figure 6 above is a representation of how each controller is using certain logic component and the dependencies here are injected directly by Spring Framework.

Storage Component - Perform CRUD operations on data entities in the database individually

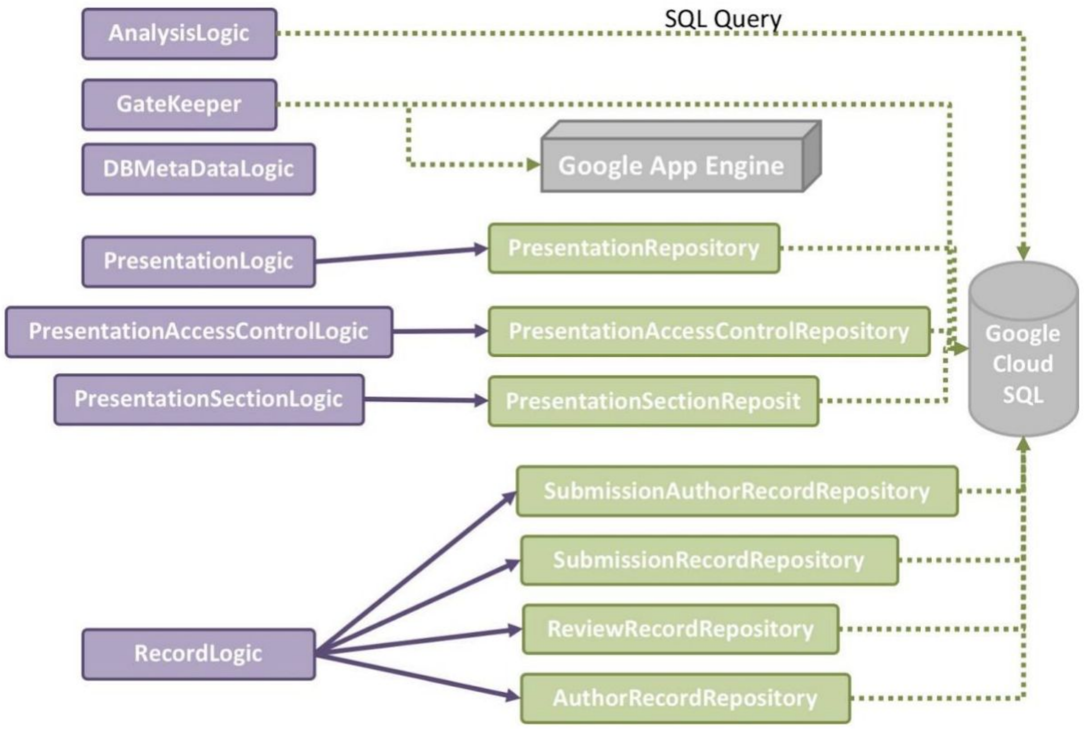


Figure 7

AnalysisLogic: Issue SQL query directly to the Database to perform analysis query
GateKeeper: Use GAE internal APIs to check logged in user and also issue queries to database to check the access rights.
Record Repositories: Persist user uploaded CSV data to the database
Presentation Repositories: Store user generated presentations, access control list and sections.

2.2 Existing features

In the sections below we will talk about the existing features of ChairVisE 2.0.

2.2.1 Login

Users are required to sign in using Google OAuth in order to use the application. During the development phase, only the user email is required.

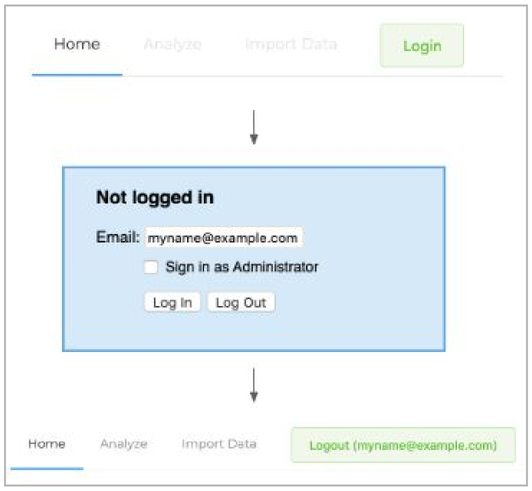


Figure 8

2.2.2 Import Data

The user is required to specify the Conference Management System (CMS) that the CSV is generated from (EasyChair / SoftConf) and the type of record (author, review or submission). Other information such as the presence of headers and what mapping to be used are also required. ChairVisE2.0 has default mappings for records from EasyChair and SoftConf, while there is no support for other CMSes due to the rigidity of the file-processing logic in ImportData.vue and processor.js.

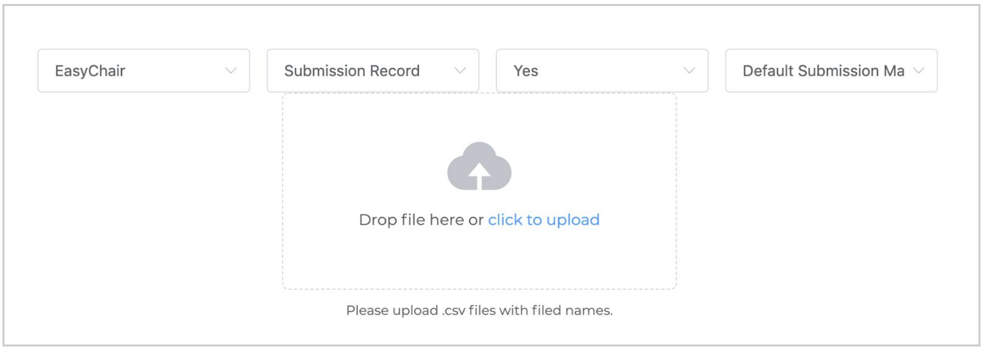

EasyChair
Submission Record
Yes
Default Submission Ma
Drop file here or click to upload
Please upload .csv files with filed names.

Figure 9

For EasyChair and SoftConf files, the predefined mapping logic make the upload process rather straightforward. However, there is no flexibility afforded for changing the mappings if required. For CMS-es outside of EasyChair and SoftConf, there is little support as the mapping tool only allows for a fixed order and number of columns in the CSV. Therefore, one of our primary goals was to improve on this inflexibility, and modularise out the logic for mapping of the data.

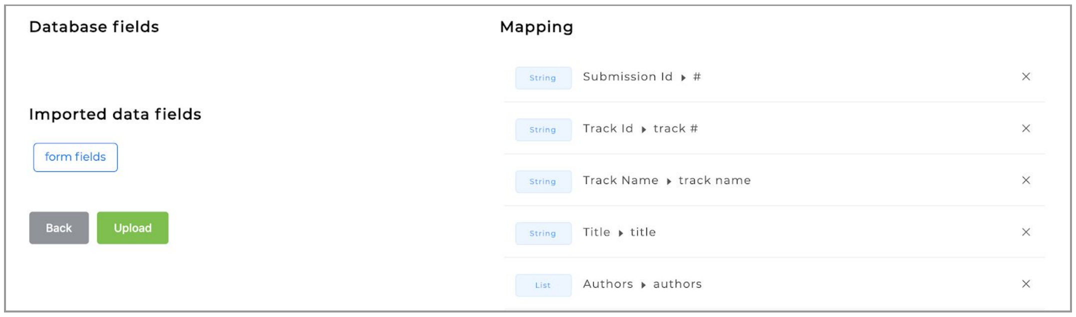
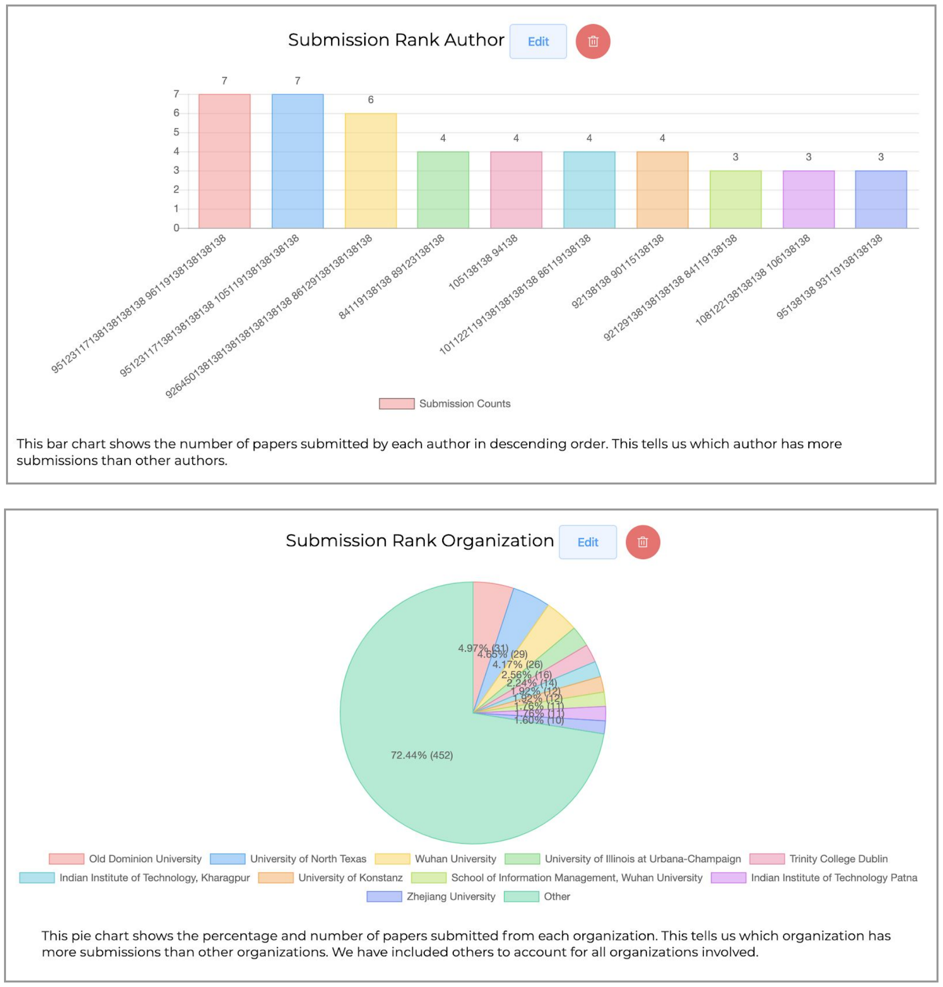


Figure 10

2.2.3 Analyse

A user can create multiple presentations using the data he/she previously uploaded. After creating a presentation, the user can add sections (visualisations) into the presentation. For example, some of the supported visualisations are Word Clouds, Bar Charts, Pie Charts and Line Charts.

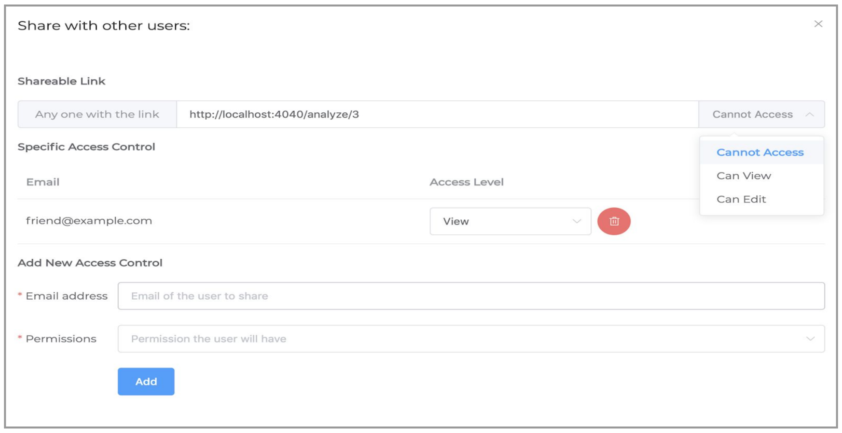


This bar chart shows the number of papers submitted by each author in descending order. This tells us which author has more submissions than other authors.

This pie chart shows the percentage and number of papers submitted from each organization. This tells us which organization has more submissions than other organizations. We have included others to account for all organizations involved.

Figure 11: Existing visualisations

A user can specify the access control rights of his/her presentations. By default, a user can only view their own presentations. However, a user can create a shareable link or grant permissions to other users to allow them to view the presentation.



Share with other users:

Shareable Link

Any one with the link	http://localhost:4040/analyze/3	Cannot Access

Specific Access Control

Email	Access Level
friend@example.com	View

Add New Access Control

* Email address	Email of the user to share
* Permissions	Permission the user will have

Figure 12: Sharing access control

3. ChairVisE 3.0

Although ChairVisE 2.0 was functional and had several essential features, our team felt that there were multiple improvements that could be made. In the following sections, we will elaborate about the requirements we had gathered as well as the enhancements we developed based on the requirements.

3.1 Cross-functional Requirements

The cross-functional requirements are detailed below, based on decreasing priority:

1	Users are prohibited from deleting data that are currently used by presentations
2	User must be authorised to view their presentations and files
3	Users cannot save more than 1 copy of the same template
4	Users are brought back to Landing Screen upon Logging out
5	Errors are informative to the users
6	Users should be logged out automatically after a session to prevent unauthorised access

3.2 Functional Requirements

The functional requirements are detailed below, based on decreasing priority:

1	User should be able to upload multiple files
2	User can create presentations using multiple files
3	User is able to remove files which are not in use
4	User should be able to upload different file formats
5	User should be able to create and save file templates

3.3 Design of ChairVisE 3.0

The design of ChairVisE 3.0 mainly follows the existing software architecture. The overall architecture we used is similar to the layered MVC shown in Section 2.1. Changes are incrementally added to the existing design of ChairVisE 2.0 to enhance and implement new features that will be discussed in the next section.

Front end:
The frontend of ChairVisE3.0 also uses the Vuex state-management pattern and a shared repository pattern like in ChairVisE2.0. In ChairVisE3.0, we implemented several new enhancements based on these patterns.

The below diagram shows what happens when the user imports data into the application. It is broken down to several vue components following Separation of Concerns. When the user interacts with the view, mutations are committed to the store to update its state. The updated state causes the relevant view components to rerender and the next view component is shown. The following components now have the data from the Vuex store to perform its function.

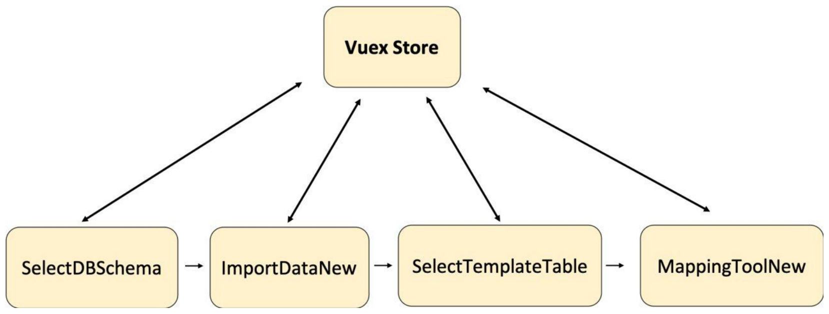


Figure 13: Diagram showing Vuex store

The list below shows the views and components used in ChairVisE 3.0.

Analyze.vue
FileData.vue
Guide.vue
ImportDataNew.vue
Landing.vue
PresentationCreated.vue
PresentationShared.vue
UserHome.vue

Figure 14

Back end:
In addition to the existing modules, we added several components in order to meet the requirements of ChairVisE 3.0.

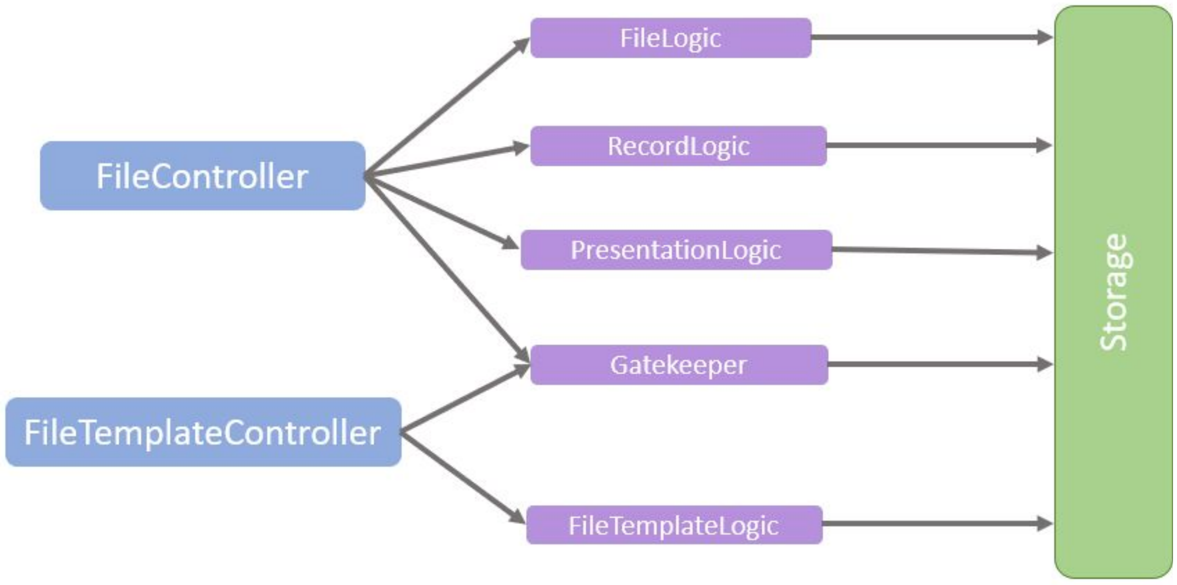


Figure 15: Diagram showing new components to ChairVisE3.0

Figure 15 shows the new controllers and logic components added to handle ChairVisE 3.0. The existing components from ChairVisE 2.0 are omitted for brevity and can be found in Section 2.1.

The **FileController** as well as its logic components deals with file management while the **FileTemplateController** and its components deals with handling file templates. Both utilise the gatekeeper to ensure access rights for the user.

3.4 Enhancements

In the sections below we will elaborate on each enhancement we developed. For each enhancement, we will cover the back-end and front-end changes as well as the reasons behind our design choices.

3.4.1 Enhancement 1: Flexible data schemes

ChairVisE2.0 is limited in the number of conferences that it supports; in ChairVisE2.0, the officially supported conferences were EasyChair and SoftConf. Users who use other CMSes will find ChairVisE2.0 to be very largely incompatible. We therefore sought to improve the compatibility of ChairVisE3.0 to be able to support even more CMSes.

Apart from solely expanding ChairVisE3.0 to be able to support more CMSes, another consideration was to improve ChairVisE2.0's design approach to handling different file formats. The new design should be easily expanded upon to support new CMSes and/or updates in format of previously supported CMSes. This is in contrast to ChairVisE2.0's rather inflexible design, which made it extremely difficult to add support for other CMSes.

In ChairVisE3.0, we still maintain the same database schemas for the Author, Review and Submission records as in ChairVisE2.0, where the fields are below:

Author Record	**Review Record**	**Submission Record**
Submission Id	Submission Id	Submission Id
First Name	Review Id	Track Id
Last Name	Num Review Assignment	Track Name
Email	Reviewer Name	Title
Country	Expertise Level	Authors
Organisation	Confidence Level	Submission Time
Web Page	Review Comment	Last Updated Time
Person Id	Overall Evaluation Score	Keywords
Is Corresponding	Review Submission Time	Is Accepted

	Has Recommended for Best Paper	Is Notified
		Is Reviews Sent
		Submission Abstract

Table 2: A table of the fields that ChairVisE3.0 (and ChairVisE2.0) supports.

In ChairVisE3.0, users have precise control over how their uploaded CSV file maps to the above template. While the users must conform to the above template, to improve compatibility with other CMSes, we allow what we call transformations on their data. Our transformations provide the user even greater flexibility to define their data in ChairVisE3.0, and can even work across multiple columns to, as an example, combine two columns.

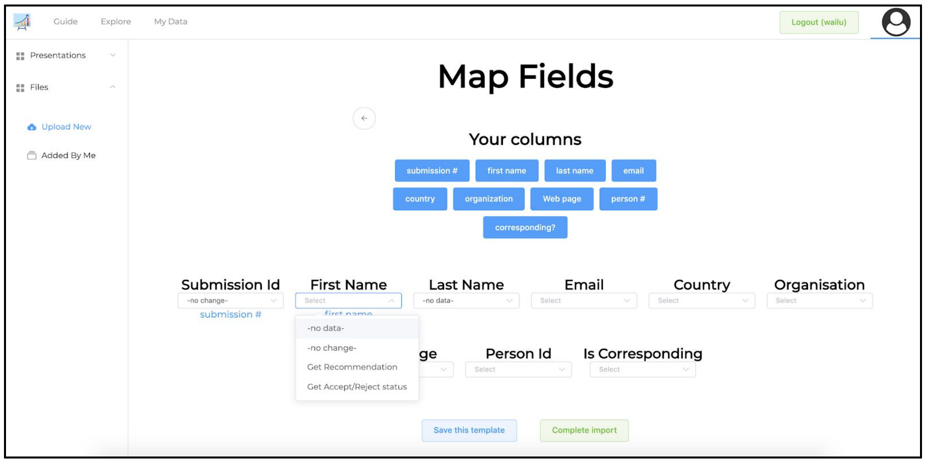
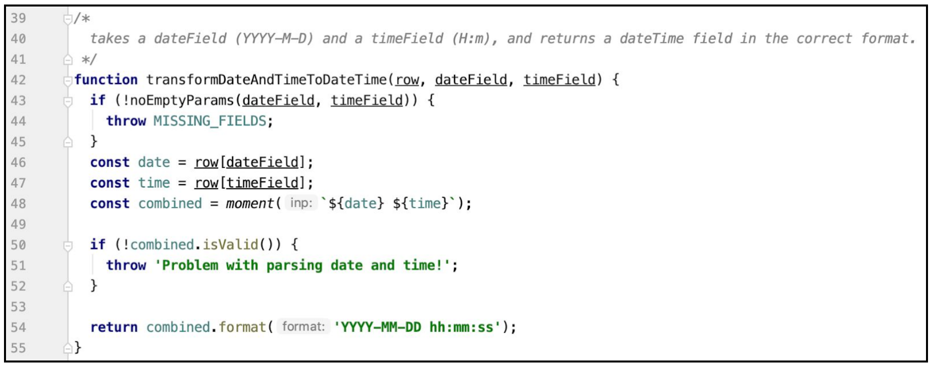


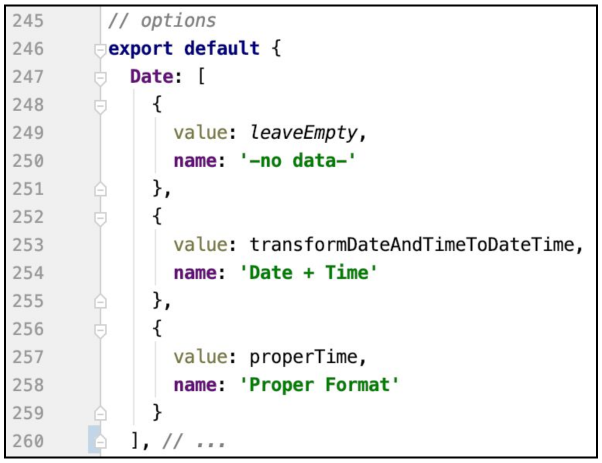
Fig 16: Example of a mapping experience. The user maps the columns to the fields and select transformations on the column data.

Our improved design makes it extremely easy to add a new transformation for the users. In particular, the transformations can be restricted to be for certain types of data only. To define a new transformation, it is sufficient to create a JavaScript function (like the below example). This modular design promotes extensibility, which translates to shorter development time when supporting a new CMS that requires specific transformation on data (i.e parsing the integer from “Confidence Level: 4”)



```
/*
  takes a dateField (YYYY-M-D) and a timeField (H:m), and returns a dateTime field in the correct format.
 */
function transformDateAndTimeToDateTime(row, dateField, timeField) {
  if (!noEmptyParams(dateField, timeField)) {
    throw MISSING_FIELDS;
  }
  const date = row[dateField];
  const time = row[timeField];
  const combined = moment( inp: `${date} ${time}`);

  if (!combined.isValid()) {
    throw 'Problem with parsing date and time!';
  }

  return combined.format( format: 'YYYY-MM-DD hh:mm:ss');
}
```

Figure 17: Example of a transformation that can act on two columns.



```
// options
export default {
  Date: [
    {
      value: leaveEmpty,
      name: '-no data-'
    },
    {
      value: transformDateAndTimeToDateTime,
      name: 'Date + Time'
    },
    {
      value: properTime,
      name: 'Proper Format'
    }
  ], // ...
```

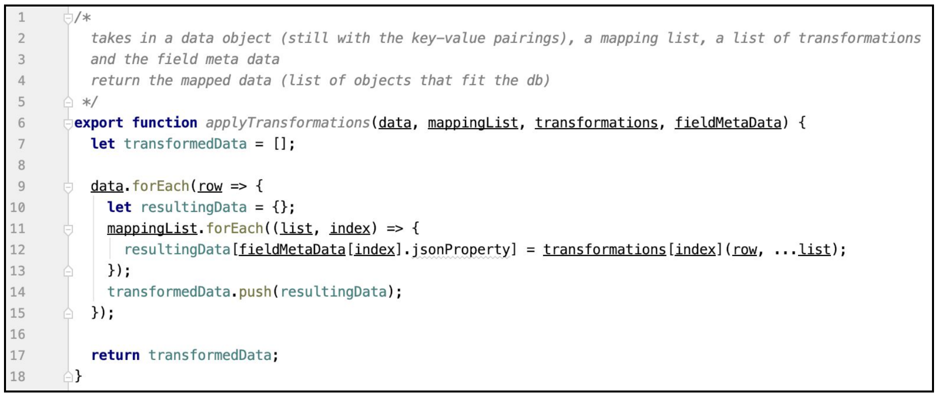
Fig 18: Defining which transformations are allowed for different types (in this case *Date*).

While this enhancement allows users great control in defining their data to be used in ChairVisE3.0's visualisations, we note that it is not practical for users to manually perform mappings for each file that they want to import. While in ChairVisE2.0, Mappings for EasyChair and SoftConf were predefined, we thought to take it one step further by allowing users to create their own such mappings. This would become our next enhancement, which will be gone into further detail below.

3.4.2 Enhancement 2: Creating and Storing Templates

Following from enhancement 4, it is unintuitive for users to manually map columns to database fields every time. ChairVisE2.0 has predefined mappings for EasyChair and SoftConf, this is under the assumption that majority of CSV files generated from a particular CMS would largely follow the same format. Therefore it is likely that for users to use the same mappings, and we decided to allow users to save such mappings. This enhancement allows the user to create 'templates' such that users can quickly import new data from the same CMS and have it mapped automatically, without the need to do it manually.

In our flexible data schemes enhancement, we introduced the concept of transformations that allow users to transform data during the mapping phase. Our templates also consist of these transformations on top of mappings. By isolating mappings and transformations from the initial raw data and the database fields, processing the data is reduced to a simple high level function as below:

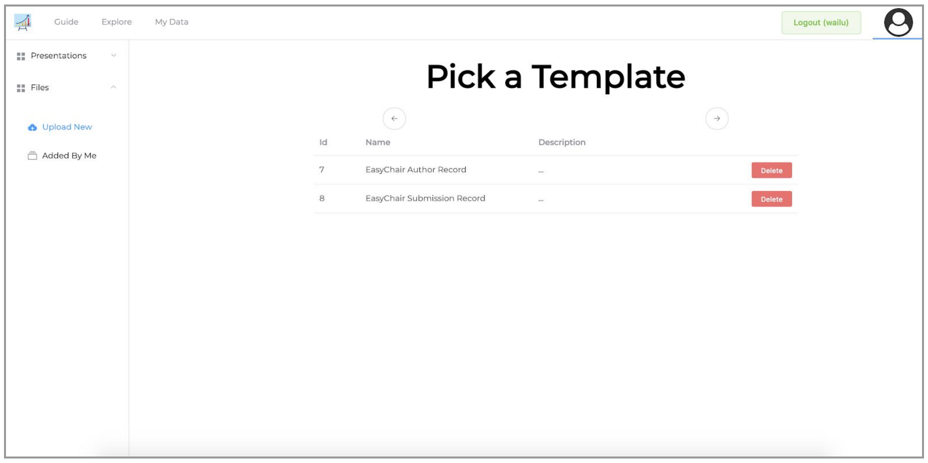


```
/*
  takes in a data object (still with the key-value pairings), a mapping list, a list of transformations
  and the field meta data
  return the mapped data (list of objects that fit the db)
 */
export function applyTransformations(data, mappingList, transformations, fieldMetaData) {
  let transformedData = [];

  data.forEach(row => {
    let resultingData = {};
    mappingList.forEach((list, index) => {
      resultingData[fieldMetaData[index].jsonProperty] = transformations[index](row, ...list);
    });
    transformedData.push(resultingData);
  });

  return transformedData;
}
```

Fig 19: High level view of how data is processed in ChairVisE3.0.

We can thus see that only the mappings and transformations are required as part of the template, as the user data and database field data are given by and chosen by the user respectively. Thus we only need to persist the mappings and transformations in our database. In particular, the Dependency Inversion Principle is applied here to avoid tightly coupled code. While JavaScript does not have the concept of interfaces, all our transformations are similar in form as if implementing an interface (an example in Fig 19). The higher level applyTransformations function only applies transformations to the correct parameters. This is the reason why it is easy to add transformations to ChairVisE3.0.

These templates are available for the user to select just before mapping, and by selecting a template, most if all of the mappings will be already completed for the user.



Pick a Template

Id	Name	Description	
7	EasyChair Author Record	...	Delete
8	EasyChair Submission Record	...	Delete

Fig 20: The user can opt to pick a template before proceeding.

The design of this enhancement also takes into account the need to make slight changes or edits to a template. In such a use case, the user can select the similar template, make changes to the template and save template as another. To prevent accidental saving of the same template, an additional check is implemented such that the user can only save a template that is different from the initially chosen template. Lastly, the user also has the option to delete a template if it is not required any more.

To create, delete and post template in the application, new API requests for the front end to call were added to **FileController** as shown in the table below. Method APIs (e.g. GET, POST etc) were added to allow users to retrieve the templates in the database and manage the templates they uploaded (for example, deleting).

Controller	API Requests
FileTemplateController	GET /api/file/mapping POST /api/file/mapping DELETE /api/file/mapping/{templateId}

The activity diagram below shows how a user would upload his file and select his file mapping under ChairVisE3.0:

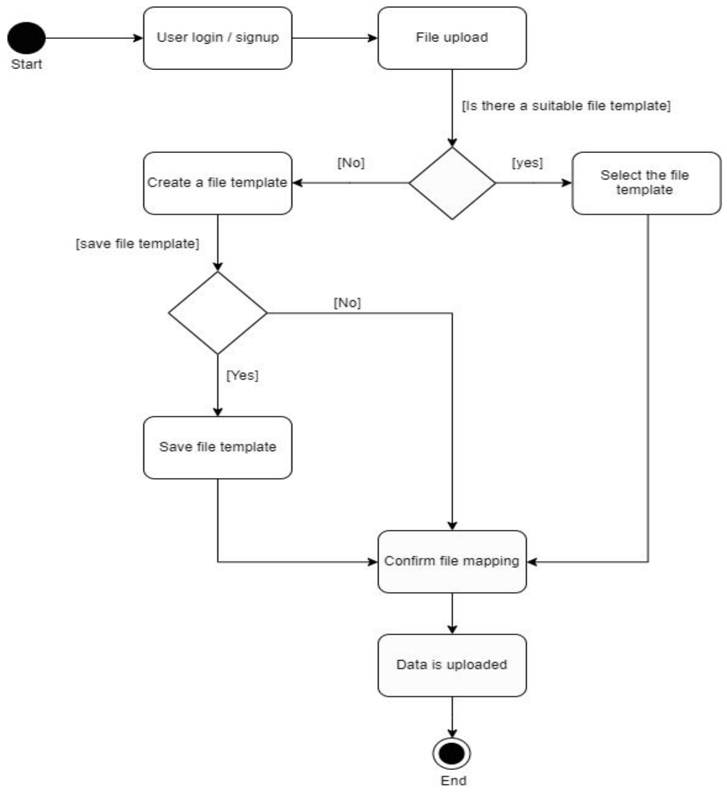


Figure 21: Activity diagram of user uploading a file

The sequence diagram below shows the interactions between the classes during the deletion of a template:

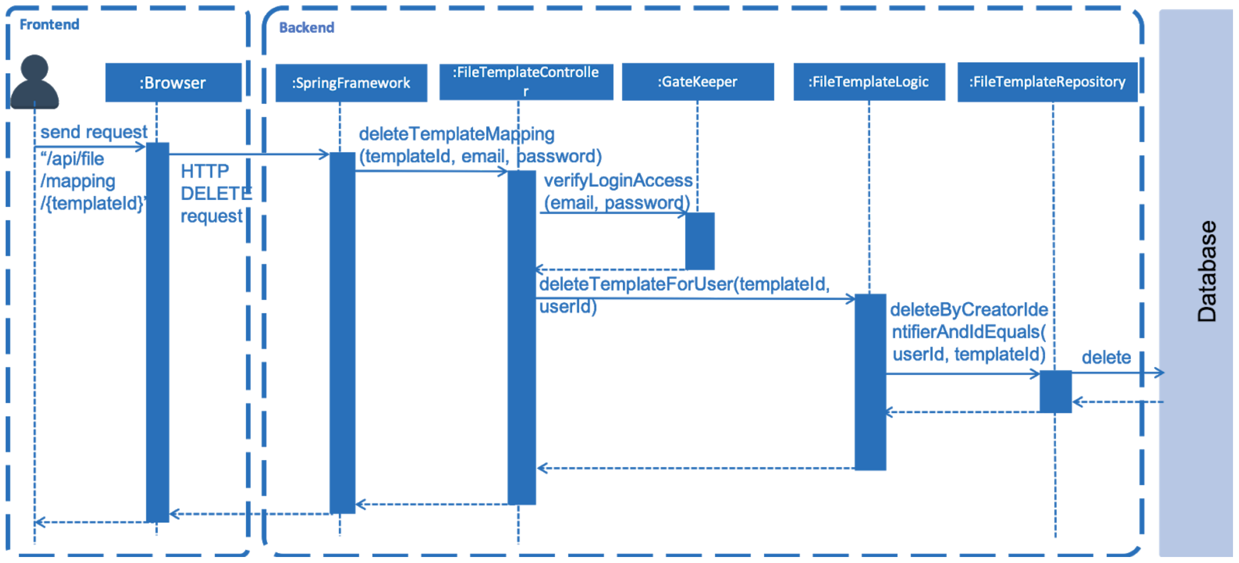

Frontend
Backend
:Browser
:SpringFramework
:FileTemplateController
:GateKeeper
:FileTemplateLogic
:FileTemplateRepository
Database
send request "/api/file/mapping/{templateId}"
HTTP DELETE request
deleteTemplateMapping (templateId, email, password)
verifyLoginAccess (email, password)
deleteTemplateForUser(templateId, userId)
deleteByCreatorIdentifierAndIdEquals(userId, templateId)
delete

Figure 22: Sequence diagram showing file deletion

Design Considerations?
We wanted modularity in they way we implemented our templates feature. The reason behind this was to allow for extensibility of our code to support different templates in the future without affecting other parts of the codebase that handles file upload and the application of transformations on data.

Another design consideration was how we were going to persist the templates in our database. While the front end sends both mappings and transformations to the database, both mappings and transformations have a variable length. Also, transformations is a list of JavaScript functions, but due to limitations of JSON we cannot encode functions. Our solution was first to convert transformations such that it can be encoded in JSON, and then to store templates as a JSON string in our database. This reduced the complexity of the backend and database as it is essentially storing exactly what the front end sends to the backend. When the frontend sends a request to retrieve the templates, the backend would be able to return the exact template without any reconstruction. This has the added benefit that if representation of templates were to be changed in the frontend, the backend would not be affected by this change. Because of this design however, this would also mean that old templates might be unusable and are to be identified by the frontend.

3.4.3 Enhancement 3: File management and data persistence

Based on ChairVisE2.0, users are limited to uploading only one file of each type (such as author and review) and they lose earlier visualisations based on previous data. To improve it, we enabled users to upload multiple data files of each type and be able to manage their files on the app. In addition, users can choose which files to use when making a presentation instead of using all the files they uploaded.

Back End:

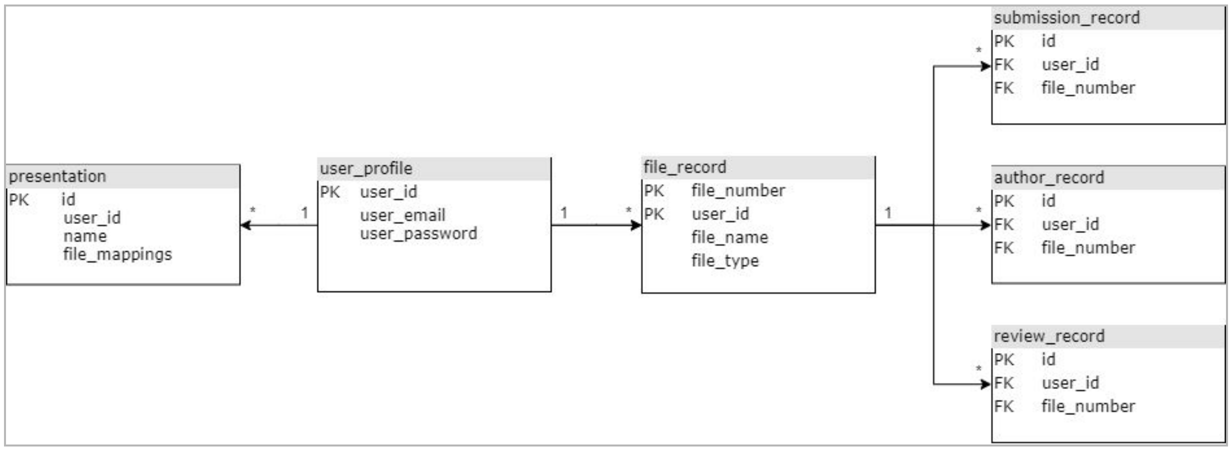
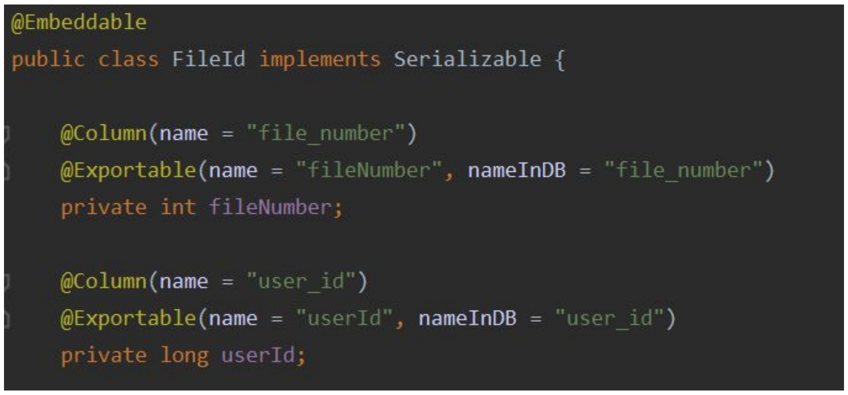


Fig 23: Entity-relationship diagram for file management

Figure 23 shows a portion of the entity-relationship (ER) diagram related to supporting file management. Each file record is created whenever a user uploads a new data file. The **user_id** will allow the system to uniquely identify the owner of the file. Subsequently when the information of the file is uploaded, it uses the unique composite key of **user_id** and **file_number** to identify which file it belongs to.

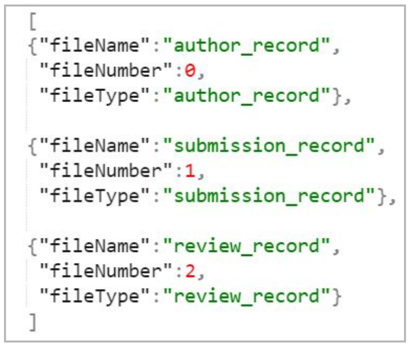


```
@Embeddable
public class FileId implements Serializable {

    @Column(name = "file_number")
    @Exportable(name = "fileNumber", nameInDB = "file_number")
    private int fileNumber;

    @Column(name = "user_id")
    @Exportable(name = "userId", nameInDB = "user_id")
    private long userId;
```

Fig 24: Example of implementation of composite key

The **@Embeddable** tag in Hibernate specifies that the value object is part of an owning entity. This encapsulation allows it to be reused easily in other modules and ensures **file_number** is used in conjunction with **user_id**.

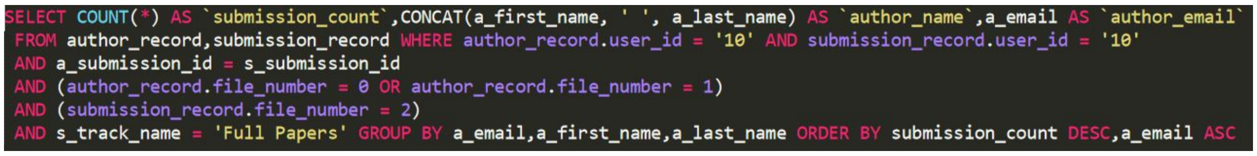
When a user creates a presentation, the user has the choice of which data files to use in that presentation. To support such a feature, a list of **file_mapping** is stored in the database as a JSON object. An example of a **file_mapping** is shown in Figure 25:



```
[
{"fileName":"author_record",
 "fileNumber":0,
 "fileType":"author_record"},

{"fileName":"submission_record",
 "fileNumber":1,
 "fileType":"submission_record"},

{"fileName":"review_record",
 "fileNumber":2,
 "fileType":"review_record"}
]
```

Fig 25: Example of file_mapping

Whenever a user wants to make an analysis using the data, an equivalent SQL query is generated to the backend database. An example is shown in Figure 26:

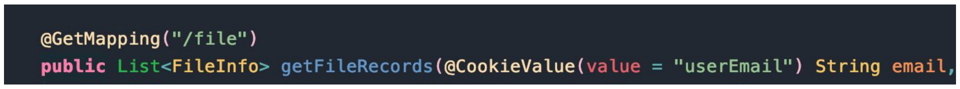


```
SELECT COUNT(*) AS `submission_count`,CONCAT(a_first_name, ' ', a_last_name) AS `author_name`,a_email AS `author_email`
 FROM author_record,submission_record WHERE author_record.user_id = '10' AND submission_record.user_id = '10'
 AND a_submission_id = s_submission_id
 AND (author_record.file_number = 0 OR author_record.file_number = 1)
 AND (submission_record.file_number = 2)
 AND s_track_name = 'Full Papers' GROUP BY a_email,a_first_name,a_last_name ORDER BY submission_count DESC,a_email ASC
```

Figure 26: Example of valid SQL Query

The SQL query specifies which files to query for in the database, based on their **file_number** and **user_id**. In particular for analysis requests with multiple of the same file type (such as multiple author files), it does a **OR** join to get data from both files. For requests with multiple file types (such as both author files and submission files), it does a **AND** join to handle it.

Also, new API requests for the front end were added to **FileController** as shown in the code screenshot and the table below. Method APIs (e.g. GET, POST etc) were added to allow users to retrieve the file records in the database and manage the files they uploaded (for example, deleting).



```
@GetMapping("/file")
public List<FileInfo> getFileRecords(@CookieValue(value = "userEmail") String email,
```

Figure 27

Controller	API Requests
FileController	GET /api/file POST /api/file

Design considerations:

1. Setting up a relational database for file management — Ensures data integrity of the data and makes it easier to support other features. For example, if a user were to delete their profile in future, then the deletion of their files and data can be done easily by a delete cascade using the foreign key in their entries.
2. Using a user_id instead of the user email as per in ChairVisE2.0 — Decoupling of the user email from the records and files. This is to support future enhancements such as allowing users to update their email address. (More in Enhancement y: user authentication)
3. Allowing multiple files of the same file type (such as multiple author files) — This allows users to combine data files from different conferences in the same presentation to create visualisations based on both conferences. Users can also combine data files such as those from previous years to create visualisations spanning multiple time periods.

Front end:

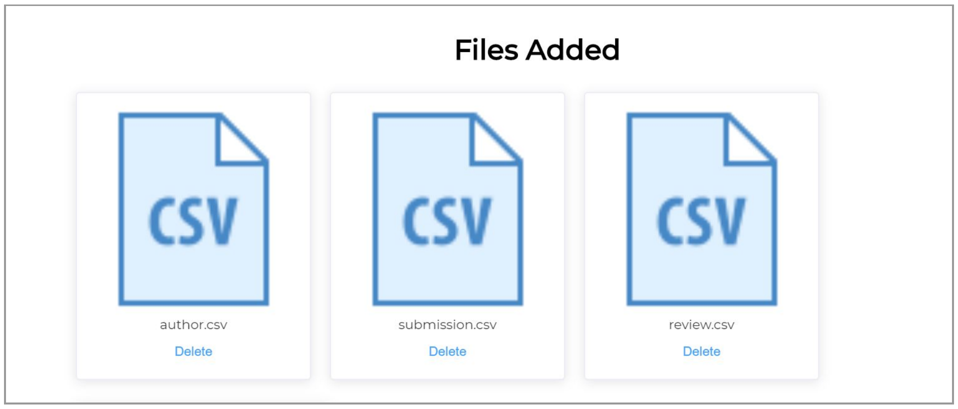
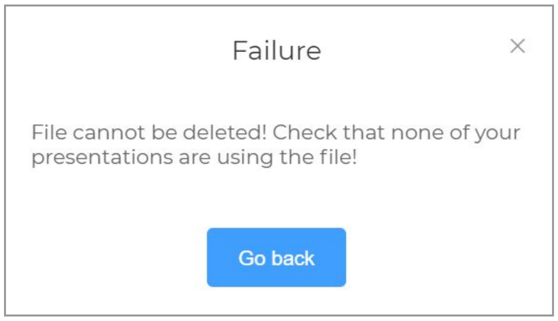


Fig 28: Uploaded files are displayed as tiles with icons and filename

Figure 28 shows all the files that the user owns. From this page, the user can delete files that are uploaded onto the app. To ensure the user does not accidentally delete files which are in use by his/her presentations, the user will be prompted by an error message shown below:


Failure

File cannot be deleted! Check that none of your presentations are using the file!

Fig 29: Error message when user deletes a file in use

To resolve this, the user would have to delete his/her presentations first before deleting the file.

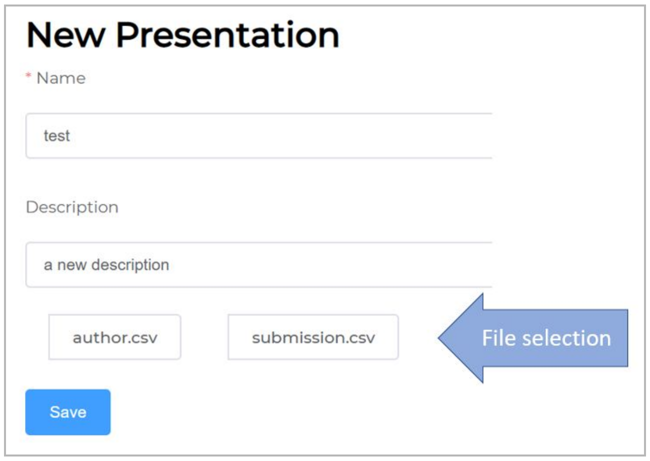


Fig 30: Files which the user has uploaded are displayed

When creating presentations, a user can also select which data files to use in the presentation. The user can select as many files as they would like to use and the presentation as well as all visualisations from that presentation will use that same data files.

The sequence diagram below shows the interactions between the classes during the retrieval of files added by the user:

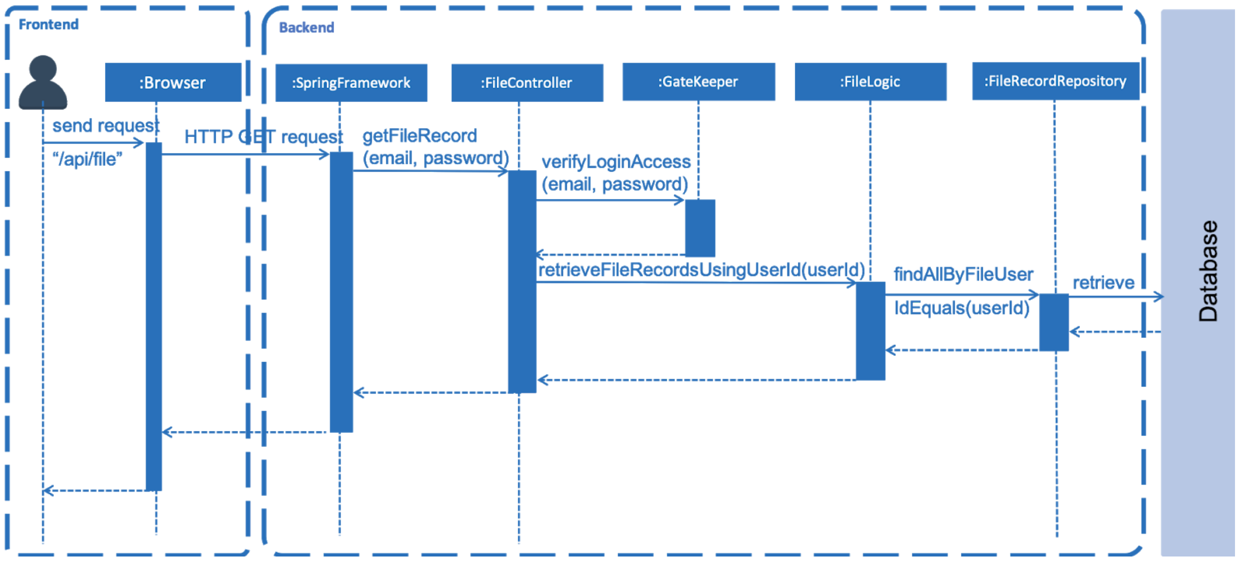


Figure 31: Sequence diagram showing retrieving files

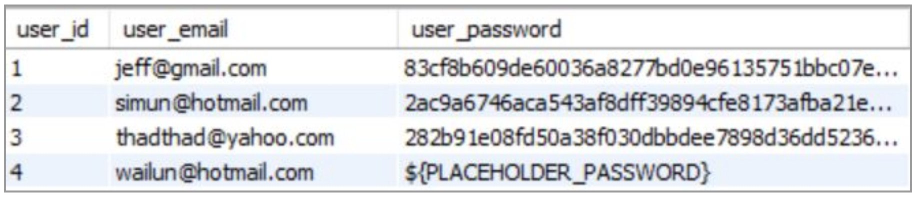
Design considerations:

1. Prompting the user to delete their presentations which use the to-be-deleted file instead of cascading the file deletion — This is to ensure users do not accidentally delete the files which they are using which would result in irreversible changes to their presentations.
2. Using a el-checkbox-button for the choosing of file for each new presentations instead of el-button — Allow user to unclick the files they might have accidentally chose

3.4.4 Enhancement 4: User login and authentication

Based on ChairVisE2.0, users have the option of logging in via Google OAuth. We felt that this was restrictive as it requires users to have a google account which might not work on all platforms. Therefore, we implemented our own user authentication and sign in system. This complements the existing Google OAuth, hence users can login with either one when using our app.

Back End:



user_id	user_email	user_password
1	jeff@gmail.com	83cf8b609de60036a8277bd0e96135751bbc07e...
2	simun@hotmail.com	2ac9a6746aca543af8dff39894cfe8173afba21e...
3	thadthad@yahoo.com	282b91e08fd50a38f030dbbdee7898d36dd5236...
4	wailun@hotmail.com	${PLACEHOLDER_PASSWORD}

Fig 32: User details under both login systems stored in the SQL database under **user_profile** table

Currently, our app supports 2 login methods, either via our implemented login/sign-up process, or via Google OAuth. However since Google OAuth only allows 3rd party applications to access the user email, we store a placeholder text to indicate it is a user of Google OAuth.

A limitation of such an implementation is its scalability. It becomes hard to manage future login systems such as using Facebook or other 3rd party apps based on this implementation. Instead, a better way would be to store an **account_type** field to indicate which 3rd party applications the user is from. In verifying whether the user is authorised for a 3rd party login system, we can then check its account type instead.

Front End:

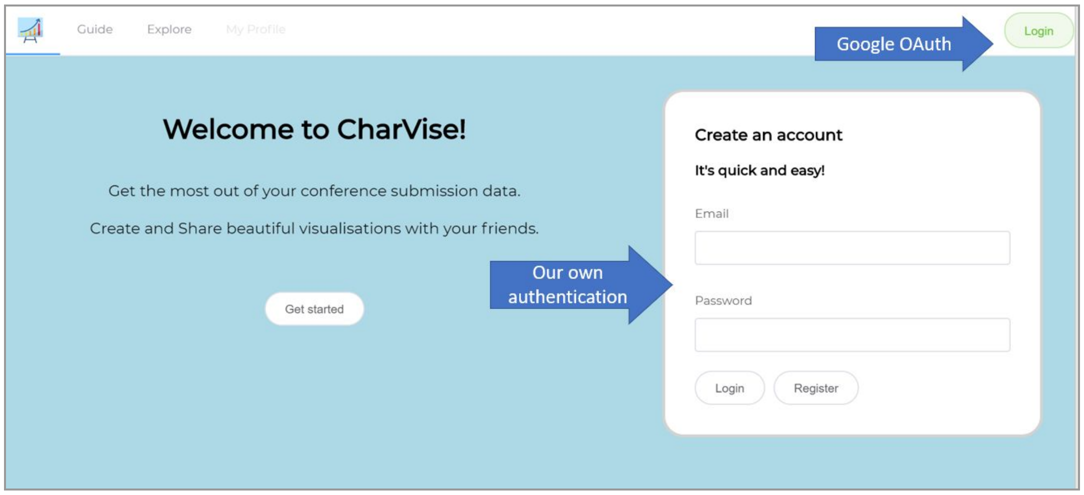


Fig 33: The UI for the User Sign-up/Login

The home page of ChairVisE 3.0 shows the different login systems available. If a user clicks on the Google OAuth button, they will be redirected to the Google Accounts login page. If they use our authentication system, they will be required to sign up if they have not done so. Else, they simply enter their user can click the login button.

In particular, we enabled handling of several common errors such as:

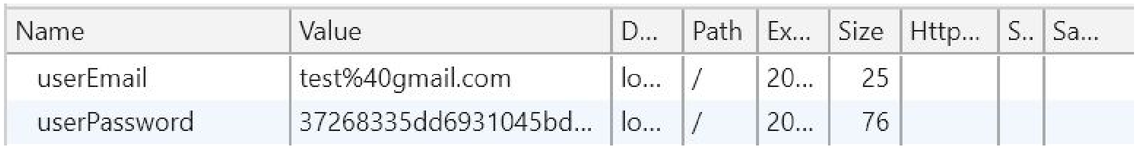
1. Invalid user email/password during login. User will be prompted to re-enter their credentials.
2. User email already used during sign up. User will be prompted to use a different email address.
3. Trying to log in/sign up using a different login system. Eg a user who used Google OAuth previously but try to log in using our authentication. User will be prompted that their credentials are invalid.

Hash Algorithm:

In order to prevent a Man-In-The-Middle attack where an attacker can eavesdrop on the connection to obtain the credentials, we used SHA-256 which is a hashing algorithm to hash the password before sending it to the backend. In verifying the validity of the passwords in the backend, we compare the user's hashed password with the one we have to ensure they are the same. Of course, this is still not the best implementation. Even though SHA-256 is more secure than SHA-1 and MD5, most actual implementations require salting of the password before hashing to prevent pre-computed hash attacks. Also, other algorithms such as PBKDF2, bcrypt, or scrypt are much better (but slower) than SHA-256. However, considering this is not a computer security module, we believe this should suffice.
The package we used is from: https://www.npmjs.com/package/js-sha256

3.4.5 Enhancement 5: Session Management

Previously, session management was handled using Google OAuth's User Service API. However, since we implemented our own authentication and not every user is a registered Google user, we implemented our own session management to handle these users. This is done using the browser cookies which are sent together along every API call made.

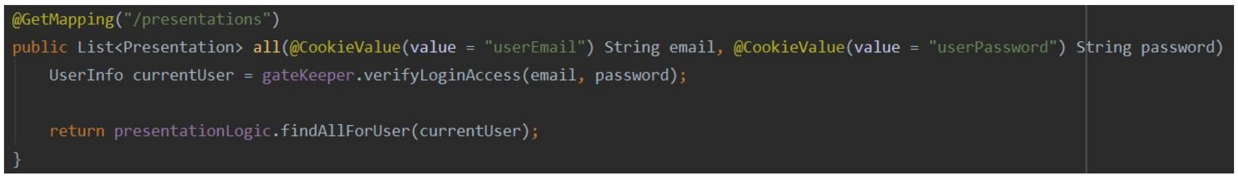


Name	Value	D...	Path	Ex...	Size	Http...	S..	Sa...
userEmail	test%40gmail.com	lo...	/	20...	25			
userPassword	37268335dd6931045bd...	lo...	/	20...	76			

Figure 34: Cookies used.

Figure 34 above shows the cookies used. This can be viewed in Chrome via:
Inspect -> Application -> Storage -> Cookies.

In order to receive the cookies from the user's API calls, we used Spring MVC's **@CookieValue** annotation to retrieve the cookies from the user. An example of how the new API controller works with the **@CookieValue** annotation is shown below:



```
@GetMapping("/presentations")
public List<Presentation> all(@CookieValue(value = "userEmail") String email, @CookieValue(value = "userPassword") String password)
    UserInfo currentUser = gateKeeper.verifyLoginAccess(email, password);

    return presentationLogic.findAllForUser(currentUser);
}
```

Figure 35: Figure showing cookie annotation

Design considerations:

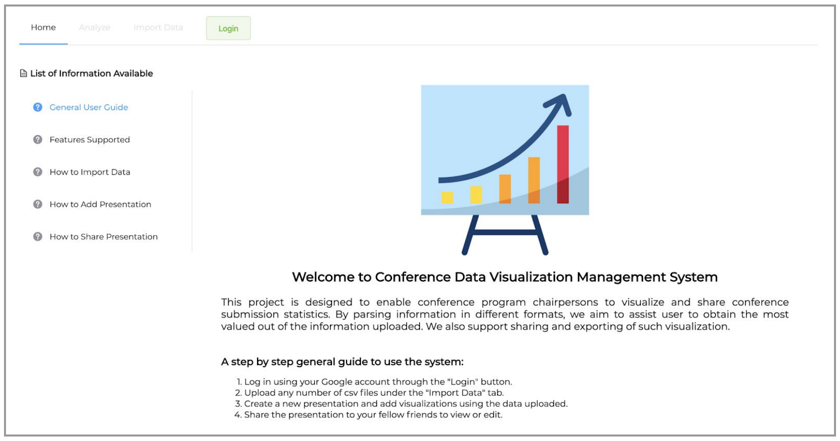
1. Cookie management to be done front-end instead of back-end — Ensures the RESTfulness of the application. All the information required to make the API call is contained within a call. Essentially decoupling of front-end from back-end to provide more flexibility such as load balancing.
2. Cookie time out — Prevents unauthorised users from logging in once the browser is closed. This might add some inconvenience for users if they accidentally close the app but we believe this is a necessary trade-off for better user protection.

3.4.6 Enhancement 6: Improving the user interface

We made major improvements the UI of ChairVisE 2.0 by reorganising the pages, as well as the features made available on each page to reduce cognitive load on the user. The Landing page now allows you to Login/Register directly, without having to click on the login button at the corner that requires extra steps, however, we have also kept that functionality for legacy users. We also created a new navigation bar with updated page routings and a “user icon” to indicate the login status of the user for recognition rather than recall of system status.

Home Page:

The home page in ChairVisE 2.0 shows the user guide right away which might not be intuitive and as welcoming to the user. As such, we have decided to make a separate home page altogether. The new home page in ChairVisE 3.0 allows the user to Login/Register right away which is similar to many web applications these days.



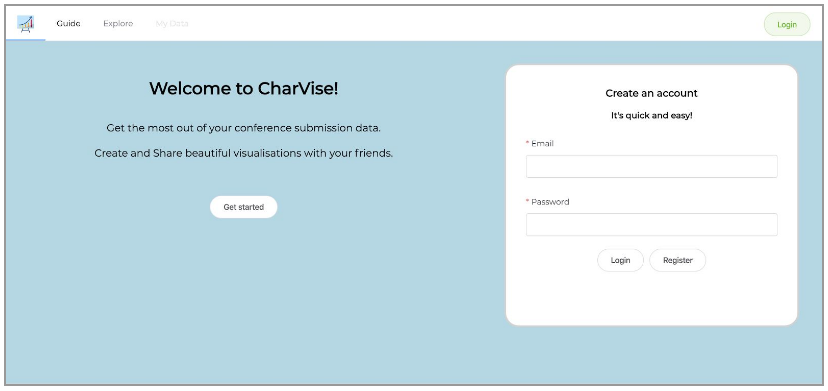
Welcome to Conference Data Visualization Management System

This project is designed to enable conference program chairpersons to visualize and share conference submission statistics. By parsing information in different formats, we aim to assist user to obtain the most valued out of the information uploaded. We also support sharing and exporting of such visualization.

A step by step general guide to use the system:

1. Log in using your Google account through the "Login" button.
2. Upload any number of csv files under the "Import Data" tab.
3. Create a new presentation and add visualizations using the data uploaded.
4. Share the presentation to your fellow friends to view or edit.

Figure 36: ChairVisE 2.0 Home Page With User Guide



Welcome to CharVise!

Get the most out of your conference submission data.

Create and Share beautiful visualisations with your friends.

Create an account

It's quick and easy!

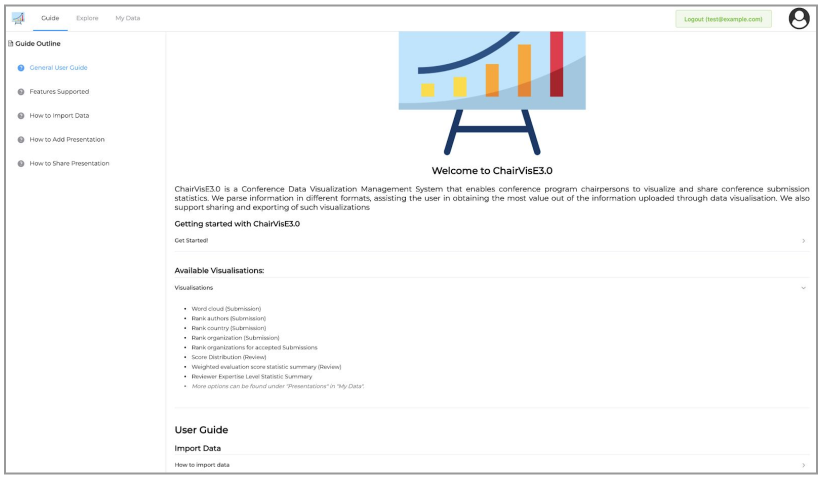
* Email

* Password

Figure 37:ChairVisE 3.0 Home Page

User Guide:

We have shifted the user guide to its own separate page to avoid cluttering the Landing Page, improving the way the information is displayed and allowing users to expand only sections they want to see.



Welcome to ChairVisE3.0

ChairVisE3.0 is a Conference Data Visualization Management System that enables conference program chairpersons to visualize and share conference submission statistics. We parse information in different formats, assisting the user in obtaining the most value out of the information uploaded through data visualisation. We also support sharing and exporting of such visualizations

Getting started with ChairVisE3.0

Get Started!

Available Visualisations:

Visualisations

- Word cloud (Submission)
- Rank authors (Submission)
- Rank country (Submission)
- Rank organization (Submission)
- Rank organizations for accepted Submissions
- Score Distribution (Review)
- Weighted evaluation score statistic summary (Review)
- Reviewer Expertise Level Statistic Summary
- *More options can be found under "Presentations" in "My Data".*

User Guide

Import Data

How to import data

Figure 38: ChairVisE 3.0 User Guide with collapsible segments

Data Section:

Majority of new features are located under “My Data”, therefore we have added a sidebar for users to navigate this section’s pages with ease. In this side bar, the user is able to view presentations created, presentations shared with him/her and files added.

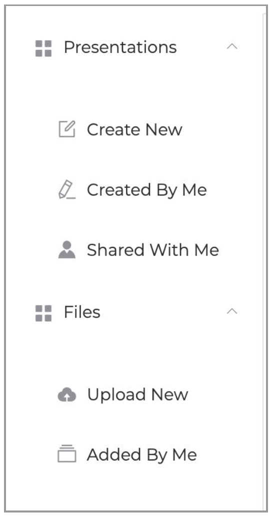


Figure 39: ChairVisE 3.0 “My Data” Sidebar

1. ***Presentations Created By Me***

   Previously, presentations created were displayed at the side which

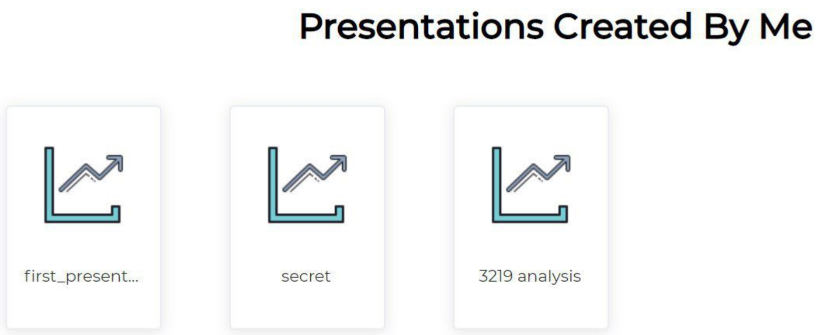


Figure 40: ChairVisE 3.0 Presentations Created By Me

2. ***Presentations Shared With Me***

   Previously, in ChairVisE 2.0, the user is not able to view all the shared presentations that were shared with him/her. However, in our enhanced ChairVisE 3.0, we created new API requests in the PresentationAccessControlController (in back end) for the front end to retrieve presentations that were shared. This is shown by the code screenshot and table below.

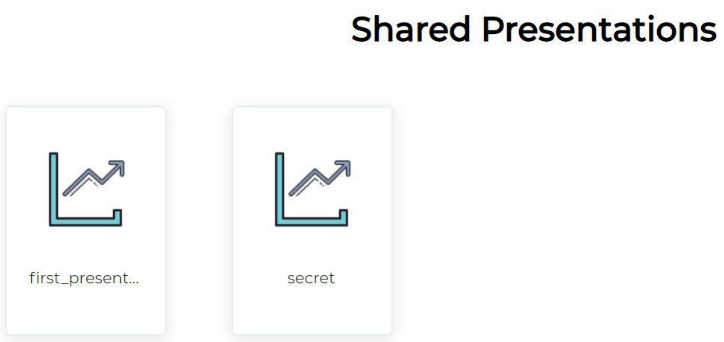
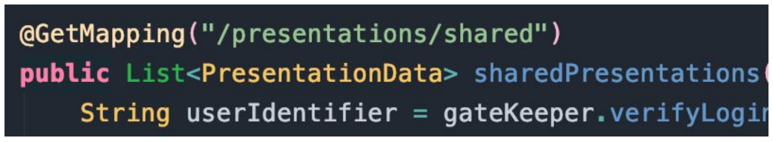


Figure 41: ChairVisE 3.0 Shared Presentations



```
@GetMapping("/presentations/shared")
public List<PresentationData> sharedPresentations(
    String userIdentifier = gateKeeper.verifyLogin
```

Figure 42: Example of shared presentations

Controller	API Requests
FileTemplateController	GET /api/file/mapping POST /api/file/mapping DELETE /api/file/mapping/{templateId}

3. Files Added By Me (UI support for Persistent Data and File Management)

As we have added data persistence, files that have been uploaded by the user are now displayed in tiles which allows for easier management of data.

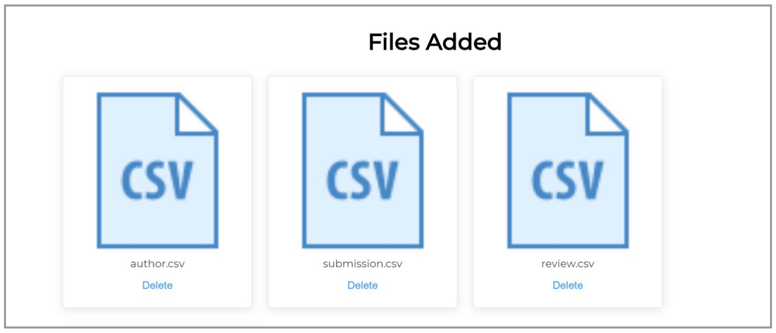


Figure 43: Figure showing files added

Mapping of Fields (UI support for Flexible Data Scheme):
We also support our Flexible data schemes by allowing the user to easily select columns in their uploaded files and map it to the database. By allowing for drag-and-drop of the fields, we increase the flexibility of UI.

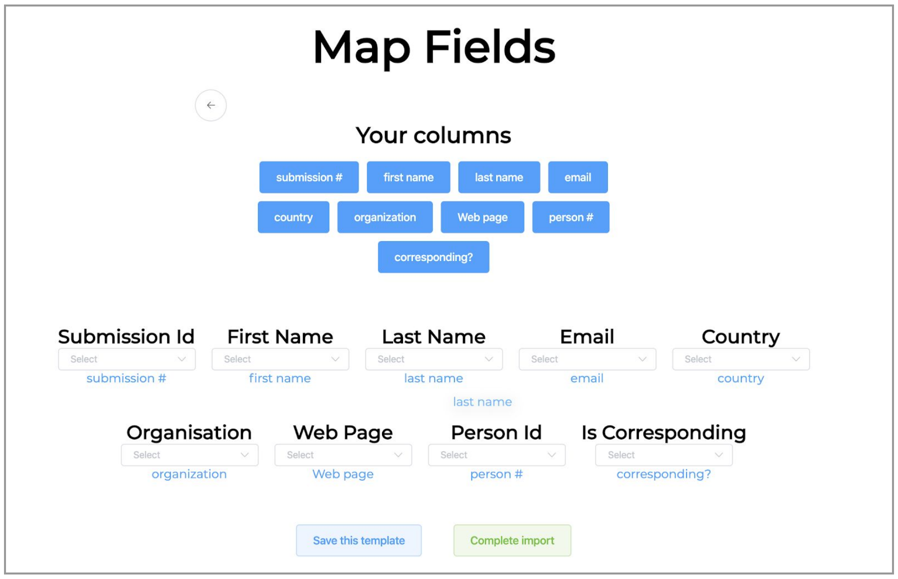


Figure 44: Diagram showing file template

Design Considerations
We overhauled the current User Interface as we believed that the User Interface for ChairVisE 2.0 was unintuitive and had a very cluttered HomePage that included the UserGuide.

We reorganised the pages such that the user is brought through an intuitive flow of the application, with the Login Section being the first page the users see, keeping the Landing Page simple and minimalistic.

We also created a separate page for the User Guide as it contains a lot of information that would be better off separated on a page of its own, we made each section of the guide collapsible as well.

User oriented pages are also located under the "My Data" tab, where the user can create presentations, upload data, and manage their data. As there are two main sections here, we have introduced a sidebar for the user to navigate these two sections with ease, without the sidebar taking up too much space on the screen.

Some of the more detailed UI considerations we had was making the process of uploading data more intuitive and efficient. This was done by allowing users to drag and drop the columns they wish to map to the database, instead of having to repeatedly click to select and unselect fields.

We also created “tiles” with a CSV logo to represent the csv files which users have uploaded to the database for a more visual representation.

4. Future Developments

4.1 Improving flexible data schemes

ChairVisE3.0 allows the user to map columns to fields and select transformations to transform their data. We wanted to give the users flexibility to do their own mappings, but also recognise that it might be tedious and thus we implemented file templates. However, when the user inputs a new file format, he/she would have to perform the tedious mapping at least once fully. A future improvement would be to have the application smartly map the columns to fields based on similarity in header names. Also, when mapping a column to a field, the application could pre-select a transformation that is likely correct. This can help reduce the effort required by the user to perform the mappings for a new CSV format.

4.2 Improving file templates

ChairVisE3.0 allows the user to create file templates for the user to reuse previous templates based on the observation that CSV files generated by a CMS would likely be similar. One immediate improvement that can be made is to enable sharing of templates from one user to another. Another improvement that can be made is to enable searching for templates, which because of sharing, there might be a great number of templates and it might be difficult to sieve through it.

4.3 Improving file management and data persistence

Although our current version supports sharing of presentations, we believe that an added improvement to this could be the sharing of files as well. For example, a user could share read access to his data so that other people can access them to make their own presentations.

4.4 Improving user login and authentication

As mentioned previously in Section 3.4.4, user login and authentication can be improved. Firstly, we can implement a better hashing algorithm to ensure that all the user’s information are properly protected. We can also integrate other 3rd party login systems such as using their Facebook accounts or Linkedin profiles.

4.5 Improving session management

We can explore other methods of session management apart from using client-side cookies. For example, we could have the server send a session cookie to the client that is only valid for a fixed time to prevent unauthorised personnel from tampering with the cookies.

4.6 Improving the user interface

One of our originally intended features was to share presentations with the public. This would be available under the “Explore” tab currently in our application. As we believe that insightful visualisations are a public good which can benefit the whole research community, the “Explore” tab would show visualisations shared to the public, perhaps sorted by popularity or date of submission.

5. Development process

We adopted an Agile Software Development Process that consisted of bi-weekly sprint cycles focused on iterative development of our features.

1. Each cycle began with setting goals with regard to features we plan to implement in the coming 2 weeks.
2. This was followed by work estimation on the amount of effort required to implement such features.
3. Finally, we delegated work according to the amount of time each member had available to work on implementing them.
4. Through the two weeks, we kept each other updated on each other’s progress, and at the end of each cycle, we conduct a review of our progress on our goals, and carry forward any backlog to the next sprint cycle.

We used Continuous Integration for our development pipeline, using Travis CI to automate the build, deploy and testing process for ChariVise 3.0.