CS3219 Project Report - Team 15

Code Repository URL:
https://github.com/CS3219-SE-Principles-and-Patterns/chairvise3-0-2019-team-15-1

Team Info:

Student Name	Jeremy Choo	Pratyay Jaidev	Yunjun	Tan Zhen Yong
Matriculation No.	A0168643W	A0166754U	A0169973H	A0156022U

Individual Contributions to Project:

Name	Contributions
Jeremy Choo	Technical • Utilized D3.js to build Force Directed Graphs. Built to be customizable with predefined queries. • Built co-authorship queries for authors, countries and organisations • Added more visualisations Non-technical: • Existing Features in ChairVisE2.0 report section • Developer documentation ○ Force Directed Graphs ○ Coauthorship Queries ○ Further visualisations
Pratyay Jaidev	Technical: • Persistence of Multiple Conference Data • Association of Conference with Presentation • Association of Conference with the different Record types Non-technical: • Existing Features in ChairVisE2.0 report section • Prioritized Requirements report section • Developer Documentation report sections: ○ Development Process ○ Persistence of Multiple Conference Data • Overall report formatting and organization
Yunjun	Technical: • E-mail generation • E-mail notification Non-technical: • Existing Features in ChairVisE2.0 report section

	• Developer Documentation (E-mail notification section) ○ Overall Architecture Diagram ○ Use Case Diagram ○ Sequential Diagram • Future Improvements and Enhancements
Tan Zhen Yong	Technical: • Landing Page • Dashboard • UI Improvements Non-technical: • Existing Features in ChairVisE2.0 report section • Developer Documentation ○ User Interface Enhancements • Future Improvements and Enhancements

Introduction

Many academic and industry conferences are held throughout the year. These conferences provide a platform for the presentation and discussion of researchers' work with others. Every conference has a Program Committee that processes the numerous submissions received, deciding which papers are accepted to the conference. An overview of statistics regarding the paper submissions for that conference in the form of data visualizations is then created by the Program Chair to provide a big picture take on the conference. Some examples of the visualizations that the current ChairVisE2.0 can produce include author(s)' ranking based on number of submissions and a word cloud illustrating which keywords are most prominent in the abstract section of submitted papers.

Our ChairVisE3.0 is an enhanced version that is more robust and provides greater utility to the Program Chair generating these visualizations. In addition, we have improved the user experience of the application. It is our hope that these enhancements provide the application's users the ability to glean more useful insights and improving the ease at which their work is done. In this report, we cover the existing features of ChairVisE2.0 and how it works using diagrams as visual aid. Following that, we will outline the requirements - both functional and non-functional - that we have prioritized for ChairVisE3.0 and proceed to detail the implementation of our new and/or enhanced features that enable us to achieve these outcomes.

Existing Features in ChairVisE2.0

In this section, we run through the existing features in the ChairVisE2.0 application. We will cover these from two perspectives. The first is requirements, which spell out the rationale for the creation of these features. This will be done through the medium of user stories. Having laid this foundation, we will then go in-depth to illustrate the design of these features, with the aid of sequence diagrams.

Broadly, ChairVisE2.0 has the following features:

1. **Basic authentication** to enter the web application using the user's Google account.
2. **Uploading and storing of conference data** (author, review and submission information) into database:
    a. Mappings of imported data that is to be retained can be selected.
    b. SoftConf and EasyChair management formats both are supported.
3. **Creation of presentations with visualizations** based on the data stored (as stated above):
    a. Presentations can have sections.
    b. Presentations can be modified after creation.
    c. Multiple types of statistics/visualizations available.
4. **Download presentations as PDF or make shareable** via a link.
5. **Basic user guide** on how to use the ChairVisE system.

ChairVisE2.0 Requirements

The aforementioned features were likely developed based on the following user stories.

User Role	Requirement	Benefit
As Program Chair…	I would like to be able to view visualizations detailing conference statistics…	As it is easier to understand and use compared to cold, hard numbers
	I would like to compile visualizations of conference statistics within presentations…	As it helps to keep related information together in one place
	I would like to import conference data	As it saves time instead of

	as CSV files generated from the two most popular conference management systems: EasyChair and SoftConf...	inputting values manually and makes the visualizations system versatile
	I would like to select which columns I in the CSV metadata files that I would like to retain...	As I might not need the entirety of information that is generated from the conference management systems
	I would like to be able to choose between conventional and custom mappings of data in the CSV files and database columns used to store the information...	As I might on occasion need the flexibility to map based on my own requirements
	I would like to be able to share and allow others to edit presentations that I have created...	As I might need to collaborate with other people on these presentations
	I would like to be able to download PDF versions of presentations I have created...	As I would like to print them and give them as handouts to other relevant personnel
	I would like to have a guide on how to use the platform...	As I want to get up and running using the system quickly
	I would like to be able to ensure that my data is safe from being used by other people...	As the data may be confidential and should be only be accessed by me
	I would like my data and presentations to persist even after I am done using the system for the day...	As I would like to be able to come back and work on it at another time

Table 1: User Stories Outlining ChairVisE2.0 Requirements

ChairVisE2.0 Design

Overall Architecture for ChariVisE2.0

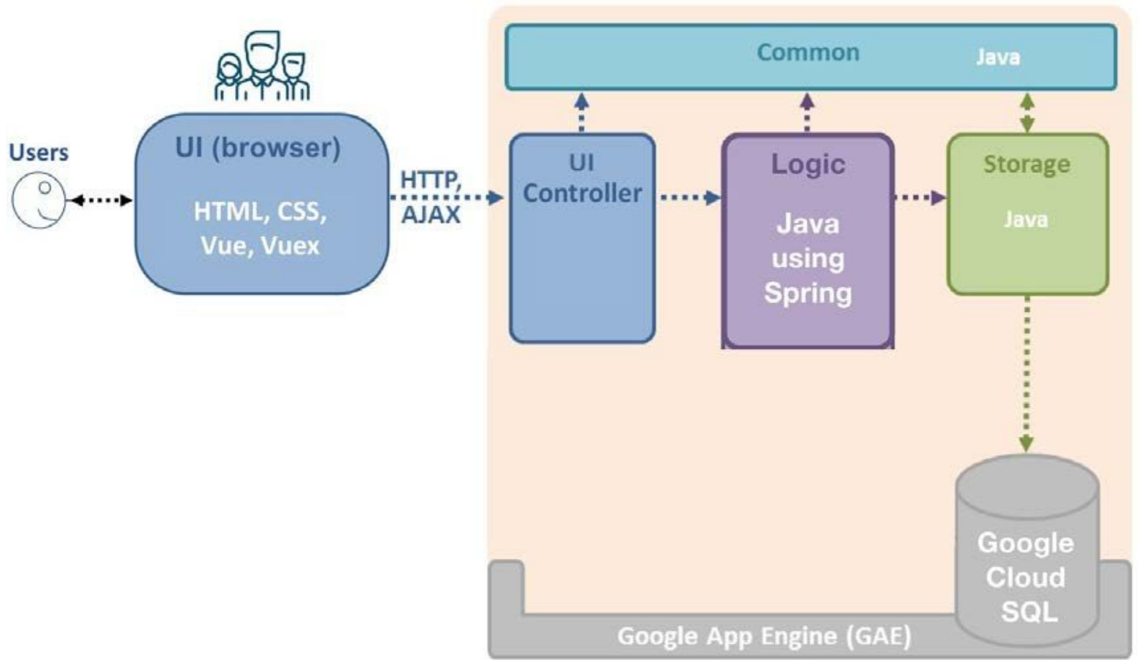


Figure 2: ChairVisE2.0 Overall Architecture

ChairVisE2.0 is architected around a layered design pattern, more specifically a 2-layer architecture that consists of the following:

- **Presentation layer**: handles the display of the UI that users see and interact with. It is built with HTML, CSS and Vue, a Model-View-ViewModel framework that provides two-way data bindings between the View and Model (in this case, plain JavaScript objects).
- **Application layer (i.e. server)**: handles the business logic. In this case it is the logic related to visualization creation and data importing. It is built with Java using the Spring framework.
    - **Resource Management layer**: handles the management and persistence of data. Hibernate is the framework used to perform the object-oriented domain model mapping in Java to the relational database that runs on MySQL. For simplicity, it can be considered that this resource management layer exists as a part of the application layer.
- Note: the Common component of the architecture provides additional utility code that is used across the application and resource management layers of the application.

Details of Frontend Architecture

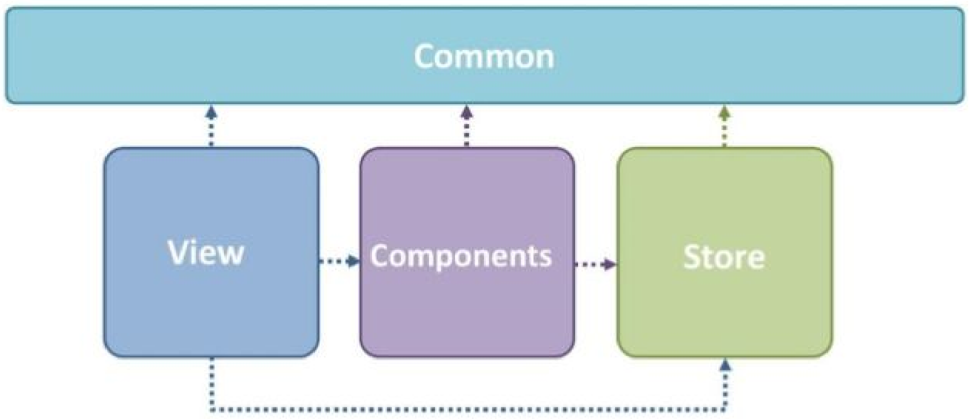


Figure 3: Detailed Frontend Architecture

The frontend of ChairVisE2.0 primarily uses the following four components to produce the displays that the user interacts with:

- **Common**: Contains utility functions and constants.
- **View**: Contains the Vue files that make up the base of each of the pages of the application.
- **Component**: Contains the UI and associated display logic that is reusable across different pages in the entire application.
- **Store**: Contains the application state and logic. It also handles the committing of mutations to state and dispatching of other actions (for example, an action which makes a call to the backend server) as a result of user interaction. The store is made up of the following modules that are closely related to the functionalities of the application:
    - **`accessControl.js`**: handles logic and state related to the access control of a presentation.
    - **`dataMapping.js`**: handles logic and state related to the uploading and mapping of CSV data with the database schema.
    - **`presentation.js`**: handles logic and state related to the creation and management of presentations.
    - **`section.js`**: handles logic and state related to the visualization sections of a presentation.
    - **`userInfo.js`**: handles logic and state related to authentication.

  All these modules are imported in the Store's `index.js` file which coordinates them.

In summary, an interaction with Vue components results in either a *committing* of mutation to the state or the *dispatching* of an action (which may possibly result in a committing of a mutation). These changes to the state cause the Vue components to be consequently updated. Crucially, what this allows for is the state inside the store being the single source of truth causing changes to the display seen by users. This state structure will be built upon when incorporating our enhancements, as detailed in later sections.

Details of Backend Architecture

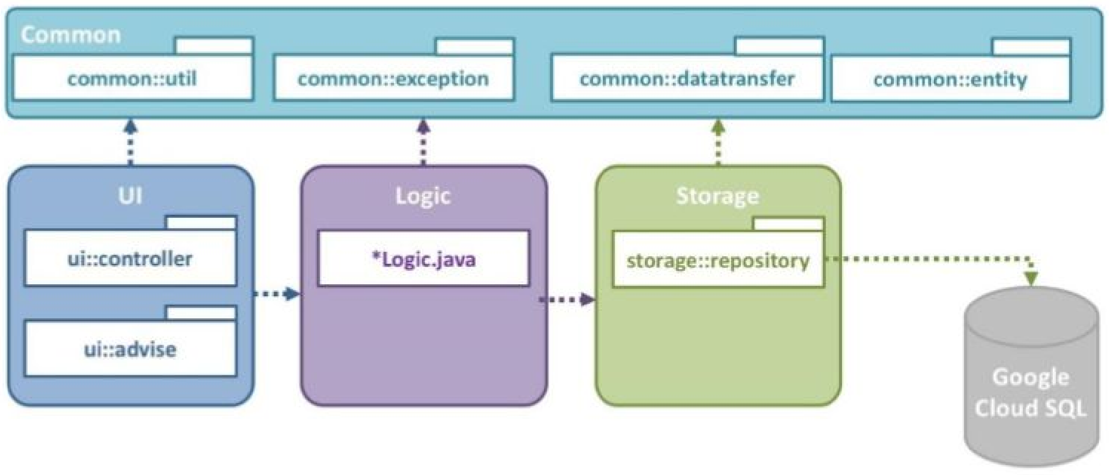


Figure 4: Detailed Backend Architecture

The backend can be broken down into the above five components which work together to apply the requisite business logic and return it to the client who is interacting with the frontend:

- **UI**: the UI component contains an `advise` package that handles exceptions thrown by the application, and a `controller` package that houses the Representational State Transfer (REST) API controller endpoints that return JSONs to the frontend.
    - Within the `controller` package there are 3 subpackages: `api`, `data` and `webpage`. The `data` subpackage contains helper objects to assist in the sending of JSON to the client while the `webpage` subpackage handles the serving of static production files built by Vue.
    - Within the `api` subpackage, there are 5 main categories of API controllers as outlined below (a full listing of the API endpoints of each controller can be seen in Figure 5):
        - **`AnalysisController`**: handles analysis requests that result from the user requesting a certain visualization.
        - **`AuthInfoController`**: handles authentication requests issued by the user (e.g. logging into the system).
        - **`DBMetaDataController`**: exposes the metadata including name and type of the standard data template used in the application.

- **`PresentationController`**, **`PresentationSectionController`** and **`PresentationAccessControlController`**: handle presentation related requests (CRUD presentation, presentation section or access control to a presentation).
- **`RecordController`**: handles data importing requests.

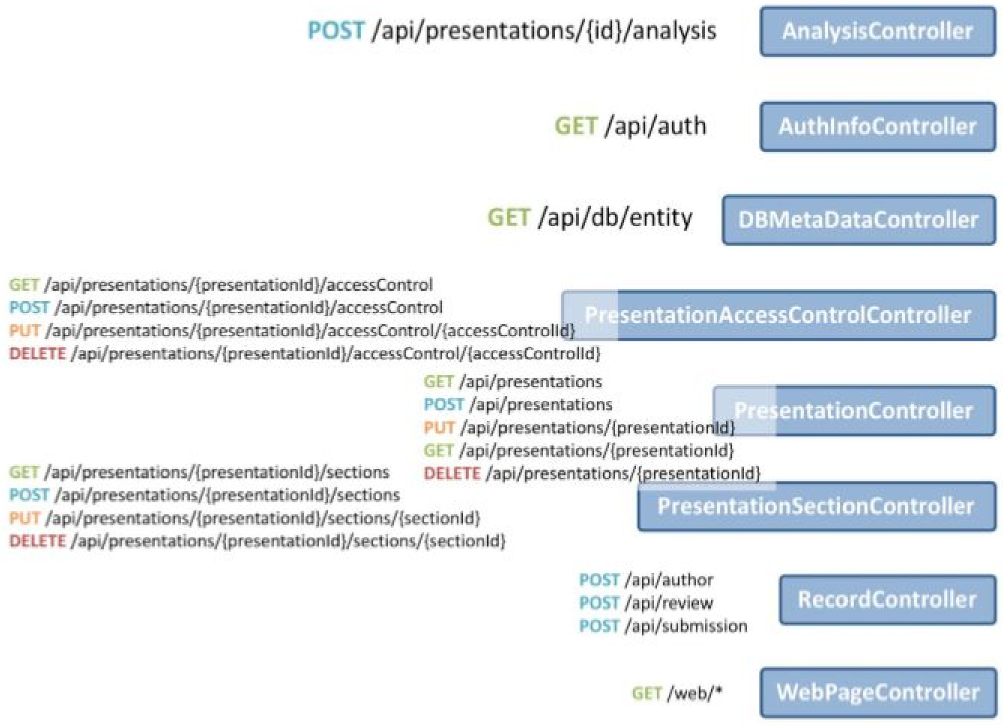

POST /api/presentations/{id}/analysis
AnalysisController
GET /api/auth
AuthInfoController
GET /api/db/entity
DBMetaDataController
GET /api/presentations/{presentationId}/accessControl
POST /api/presentations/{presentationId}/accessControl
PUT /api/presentations/{presentationId}/accessControl/{accessControlId}
DELETE /api/presentations/{presentationId}/accessControl/{accessControlId}
PresentationAccessControlController
GET /api/presentations
POST /api/presentations
PUT /api/presentations/{presentationId}
GET /api/presentations/{presentationId}
DELETE /api/presentations/{presentationId}
PresentationController
GET /api/presentations/{presentationId}/sections
POST /api/presentations/{presentationId}/sections
PUT /api/presentations/{presentationId}/sections/{sectionId}
DELETE /api/presentations/{presentationId}/sections/{sectionId}
PresentationSectionController
POST /api/author
POST /api/review
POST /api/submission
RecordController
GET /web/*
WebPageController

Figure 5: Full List of API Endpoints

- **Logic**: the logic component handles the entirety of the business logic. The main functionalities that the individual logic classes provide are:
  - CRUD operations
  - Checking of authentication and access control rights

  The above mentioned controllers are associated with the necessary logic classes to perform the required business logic. Using the Spring framework, the dependencies are automatically injected. Figure 6 illustrates these associations.

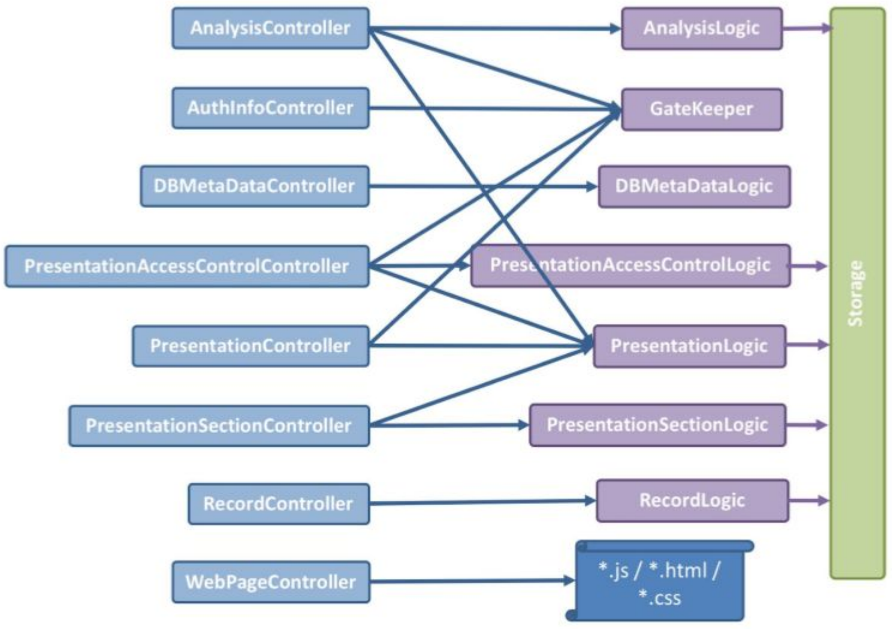


Figure 6: Association of Controllers with Corresponding Logic Classes

- **Storage**: the storage component performs individual CRUD operations on the database entities. The individual Repository interfaces in the `repository` subpackage all extend the `JpaRepository` and are associated with entities in the Common component's entity subpackage. Only by extending the `JpaRepository` is the CRUD functionality obtained. Figure 7 shows the association of the logic classes with the relevant repository interfaces to achieve the necessary business logic.

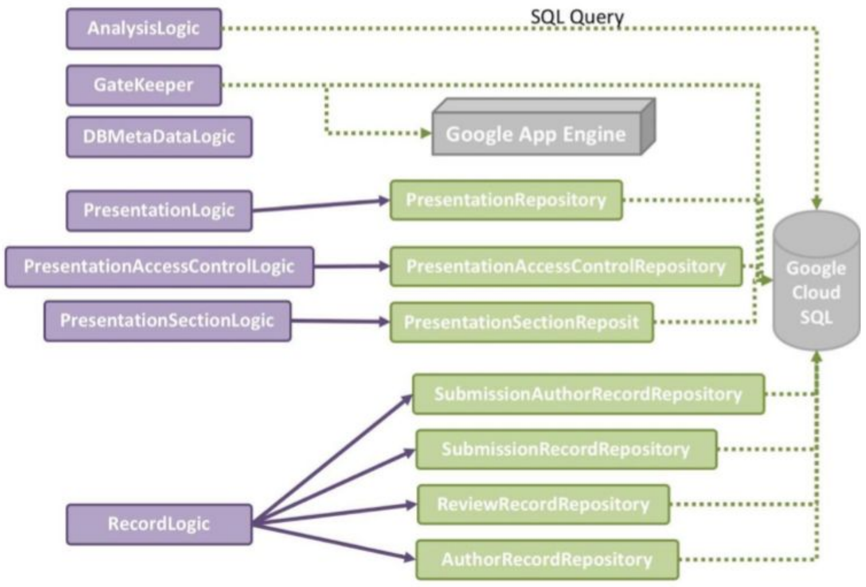


Figure 7: Association of Logic Classes with Repository Interfaces

  - `AnalysisLogic` issues SQL queries directly to the database to retrieve the data it needs for the visualization on the frontend.
  - `GateKeeper` uses Google App Engine's internal APIs to authenticate user and query access rights from the database.
  - `*RecordRepository` performs the requisite CRUD actions related to persistence of conference data that was uploaded in CSV format by the user.
  - `Presentation*Repository` performs the requisite CRUD actions related to presentations, presentation visualization sections and presentation access control.
- **Common**: the common component contains utilities that are used across the entire backend.
  - There are four main subpackages within this component that perform the following purposes:
    - **`util`**: contains utility classes.
    - **`exceptions`**: contains custom exceptions.
    - **`datatransfer`**: contains data transfer objects (to encapsulate and send data).
    - **`entity`**: contains the entities who have corresponding tables in the database with the attributes defined in the individual classes.
  - Figure 8 illustrates the relationship between the entities in the entity subpackage.

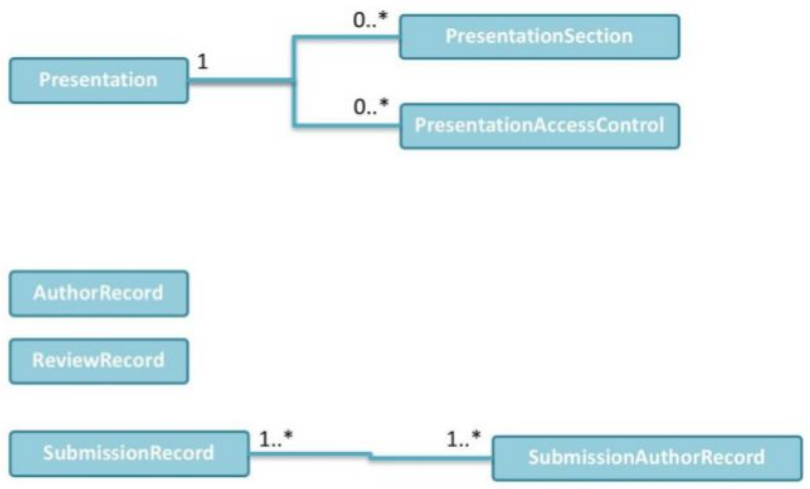


Figure 8: Relationship Between Entities

ChairVisE2.0 Features: Design Diagrams

Authentication with Google

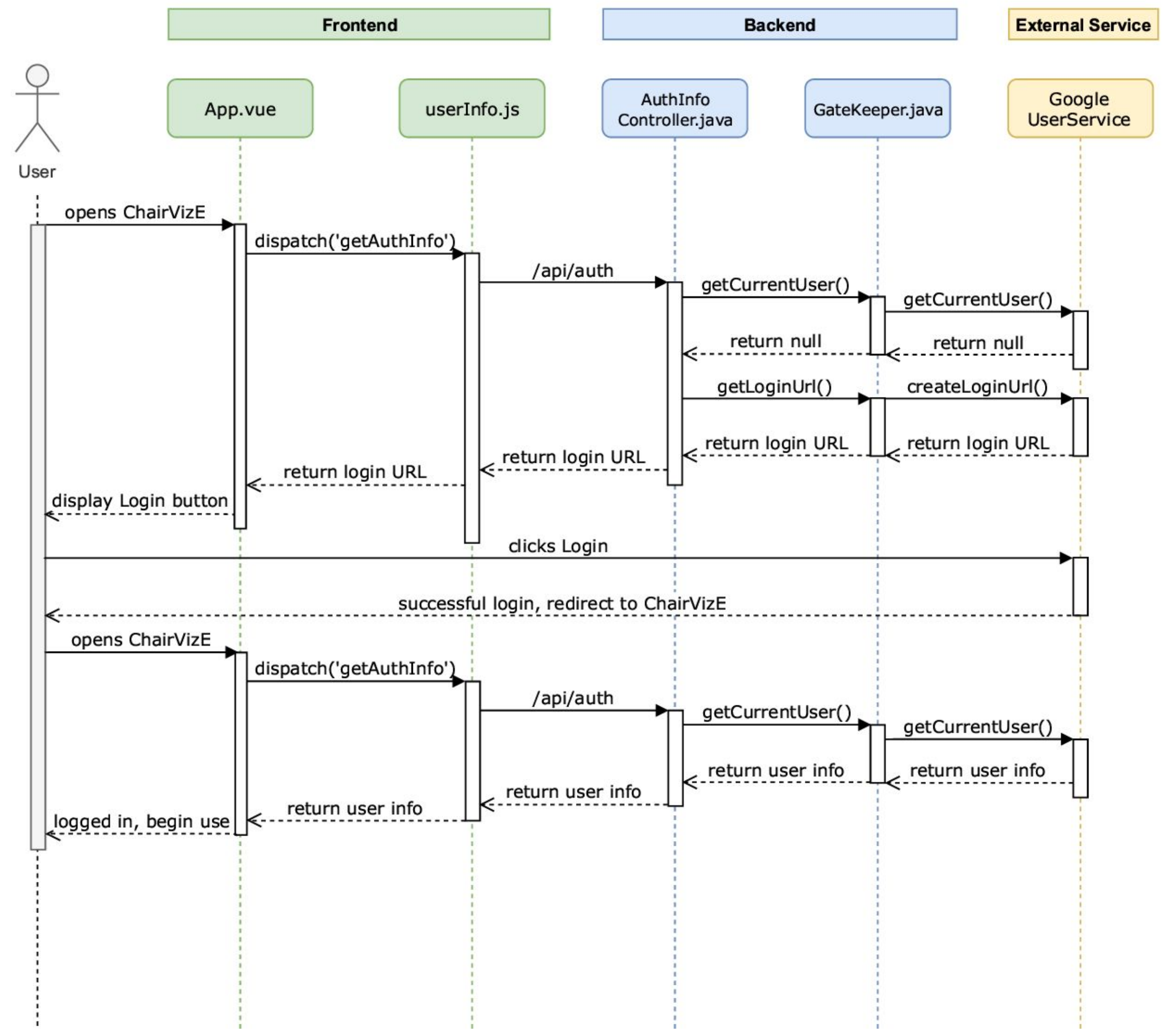


Figure 9: Sequence Diagram Illustrating Authentication with Google

The above Figure 9 illustrates a user logging into the ChairVisE system using their Google account. We assume that the user has already created a Google account for use with ChairVisE.

1. When the page loads, the Vue app will dispatch the Vuex action `getAuthInfo` to attempt to retrieve authentication information from the backend into the Vuex store. Among other data, this will include the Google login URL for the app.
2. The Vuex store retrieves the current authentication information from the backend using the endpoint `/api/auth`, without an authentication cookie. The current URL

is passed in the query parameter `redirectURL`, so that Google knows where to redirect the user to after login.

3. `AuthInfoController` attempts to get the currently logged in user from `Gatekeeper`, but does not find one. It then retrieves the login URL from `Gatekeeper`, which in turn gets it from Google, and returns the URL to Vuex.
4. Receiving the response from `/api/auth` that has a login URL and no user info, the Vue app is set to be unauthenticated and displays a login button.
5. When the user clicks on the login button, they are redirected to the Google login URL to login to the app via their Google account.
6. On successful login, Google redirects the user back to the app, setting an authentication cookie for the user.
7. When the page loads again, the Vue app will dispatch the Vuex action `getAuthInfo` to attempt to retrieve authentication information from the backend into the Vuex store.
8. The Vuex store retrieves the current authentication information from the backend using the endpoint `/api/auth`, this time passing along the authentication cookie form from step 6.
9. `AuthInfoController` attempts to get the currently logged in user from `Gatekeeper`, and returns the user info to Vuex.
10. Receiving the response from `/api/auth` that has user info, the Vue app is set to be authenticated and the user can begin using ChairVisE.

Uploading and Storing of Conference Data

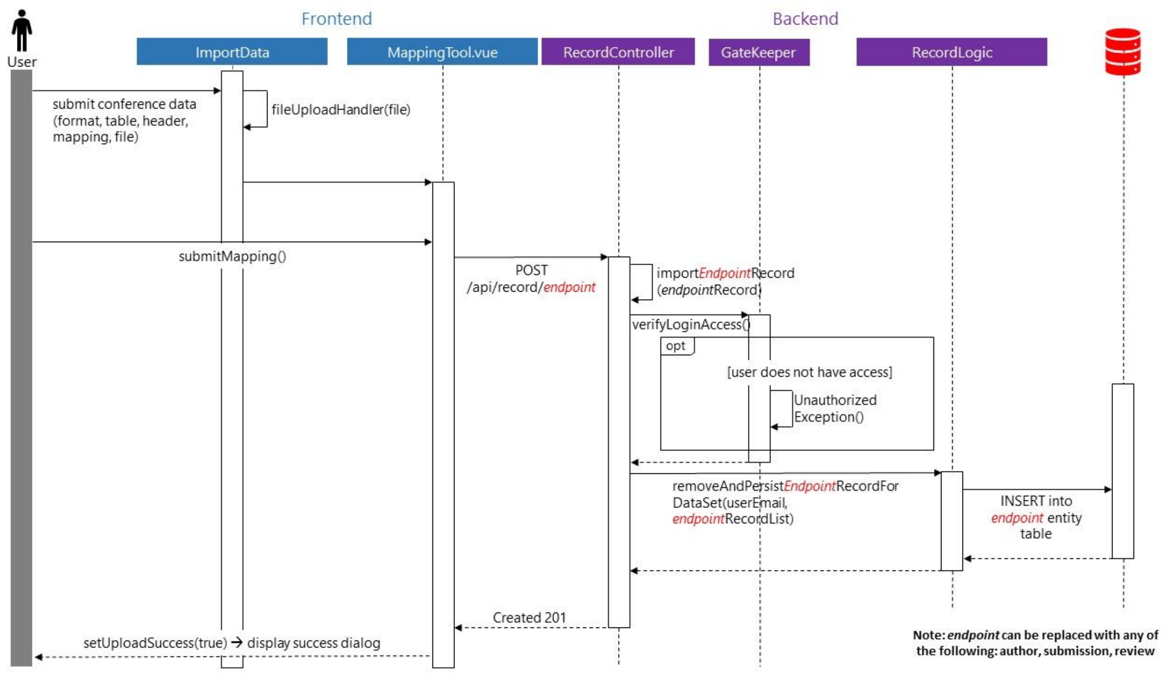

Frontend
Backend
User
ImportData
MappingTool.vue
RecordController
GateKeeper
RecordLogic
submit conference data (format, table, header, mapping, file)
fileUploadHandler(file)
submitMapping()
POST /api/record/endpoint
importEndpointRecord (endpointRecord)
verifyLoginAccess()
opt
[user does not have access]
Unauthorized Exception()
removeAndPersistEndpointRecordForDataSet(userEmail, endpointRecordList)
INSERT into endpoint entity table
Created 201
setUploadSuccess(true) → display success dialog
Note: endpoint can be replaced with any of the following: author, submission, review

Figure 10: Sequence Diagram Illustrating Conference Data Upload

The above Figure 10 briefly illustrates a user successfully uploading conference data into the ChairVisE2.0 system:

1. The user selects, on the interface, the relevant metadata options regarding the data he is uploading: the format (SoftConf or EasyChair), the table (author, submission or review) being uploaded, header (whether a header exists), and the type of default mapping. The file containing the data is then uploaded to complete this step.
2. The `fileUploadHandler` method handles the parsing of the input file following which the `MappingTool` Vue component is active. After the user selects the mapping(s) they want and confirms, a POST request with the parsed data in the request body is sent to the backend API using at the following URL: /api/record/*endpoint* where endpoint is 'author', 'submission' or 'review' depending on the type of data the user has uploaded.
3. The `RecordController` contains methods that map to the respective API endpoints (methods are named as import*Endpoint*Record) which receives the data and calls the relevant method defined in the `RecordLogic` class to save the records in the respective tables in the database.
4. On successful saving of records, a '201 Created' HTTP response is generated and returned to the `MappingTool` which then sets the state the isUploadSuccess state in Vue to true and generates a dialog indicating this that is visible to the user.

Creation of a Presentation Section

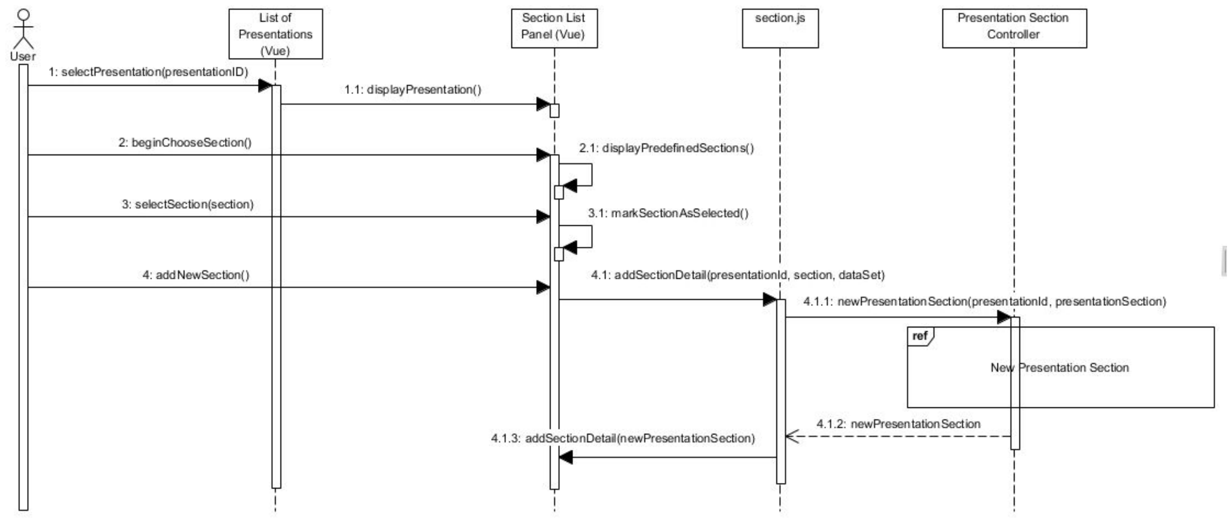

User
List of Presentations (Vue)
Section List Panel (Vue)
section.js
Presentation Section Controller
1: selectPresentation(presentationID)
1.1: displayPresentation()
2: beginChooseSection()
2.1: displayPredefinedSections()
3: selectSection(section)
3.1: markSectionAsSelected()
4: addNewSection()
4.1: addSectionDetail(presentationId, section, dataSet)
4.1.1: newPresentationSection(presentationId, presentationSection)
ref
New Presentation Section
4.1.2: newPresentationSection
4.1.3: addSectionDetail(newPresentationSection)

Figure 11: Sequence Diagram Illustrating Creation of a New Visualisation Under a Presentation

The above Figure 11 demonstrates the user successfully adding a new visualisation to a presentation that has already been created.

1. The user selects a presentation to display in the `ListofPresentation` Vue page. This the currently selected presentation to be updated, which is then displayed in the `SectionListPanel` Vue page.
2. The user begins by clicking in the section drop down list, which then display the list of all of the possible visualisations that the application can create. This list is generated by the computed `predefinedSections` variable.
3. The user then selects one of visualisations, which is then displayed in the drop down list.
4. The user clicks on the Add New Section button. This triggers the `AddSectionDetail` method within `section.js`, which performs the API request to the actual server to add the section, providing the presentation ID and the JSONified presentation section in the request. The request is then processed and the resulting created presentation section is returned, which is then added to the presentation and its visualisation shown. The details of what happens in the server when it receives the API call is shown in another sequence diagram, New Presentation Section, in Figure 12.

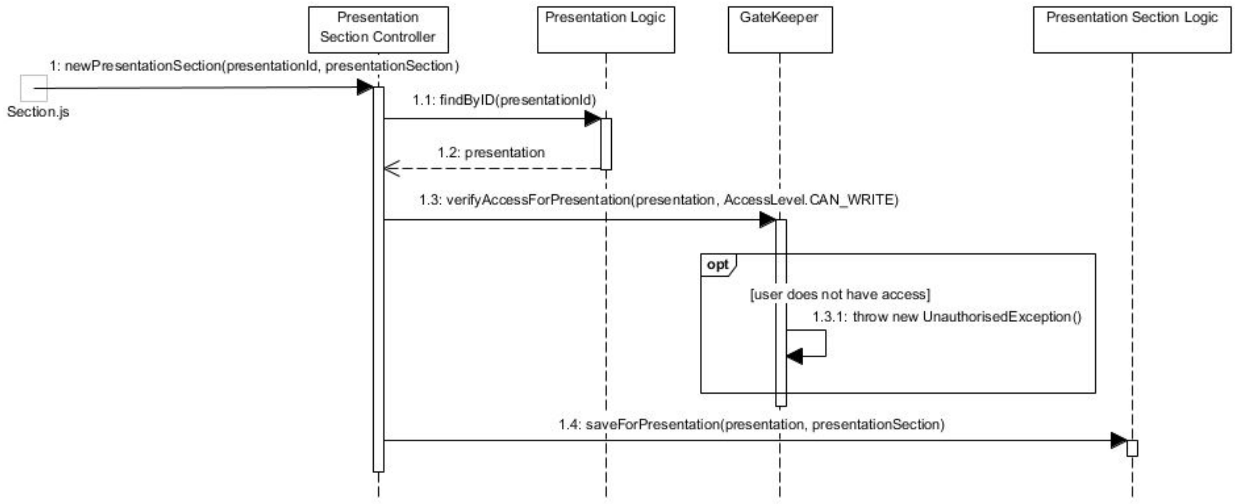

Presentation Section Controller
Presentation Logic
GateKeeper
Presentation Section Logic
1: newPresentationSection(presentationId, presentationSection)
Section.js
1.1: findByID(presentationId)
1.2: presentation
1.3: verifyAccessForPresentation(presentation, AccessLevel.CAN_WRITE)
opt
[user does not have access]
1.3.1: throw new UnauthorisedException()
1.4: saveForPresentation(presentation, presentationSection)

Figure 12: Sequence Diagram Illustrating What Happens When the Server Receives an API Call To Create a New Presentation Section.

The above Figure 12 demonstrates what the `PresentationSectionController` does after receiving an API call to add a specific visualisation to a presentation, which triggers the code within `newPresentationSection`.

1. Firstly, the `PresentationSectionController` attempts to retrieve the presentation from `PresentationLogic` by its ID.

2. Then, the `PresentationSectionController` sends a request to the `GateKeeper` to verify that the current user has access to modify the presentation. The `GateKeeper` will perform a series of checks, wherein if any of them fails, it will throw an `UnauthorisedException`, which is then propagated upwards and causes an unauthorised error to be sent back to the client.
3. If there is no exception, then the `PresentationSectionController` informs the `PresentationSectionLogic` to save the new presentation section into the presentation.

Download presentations as PDF

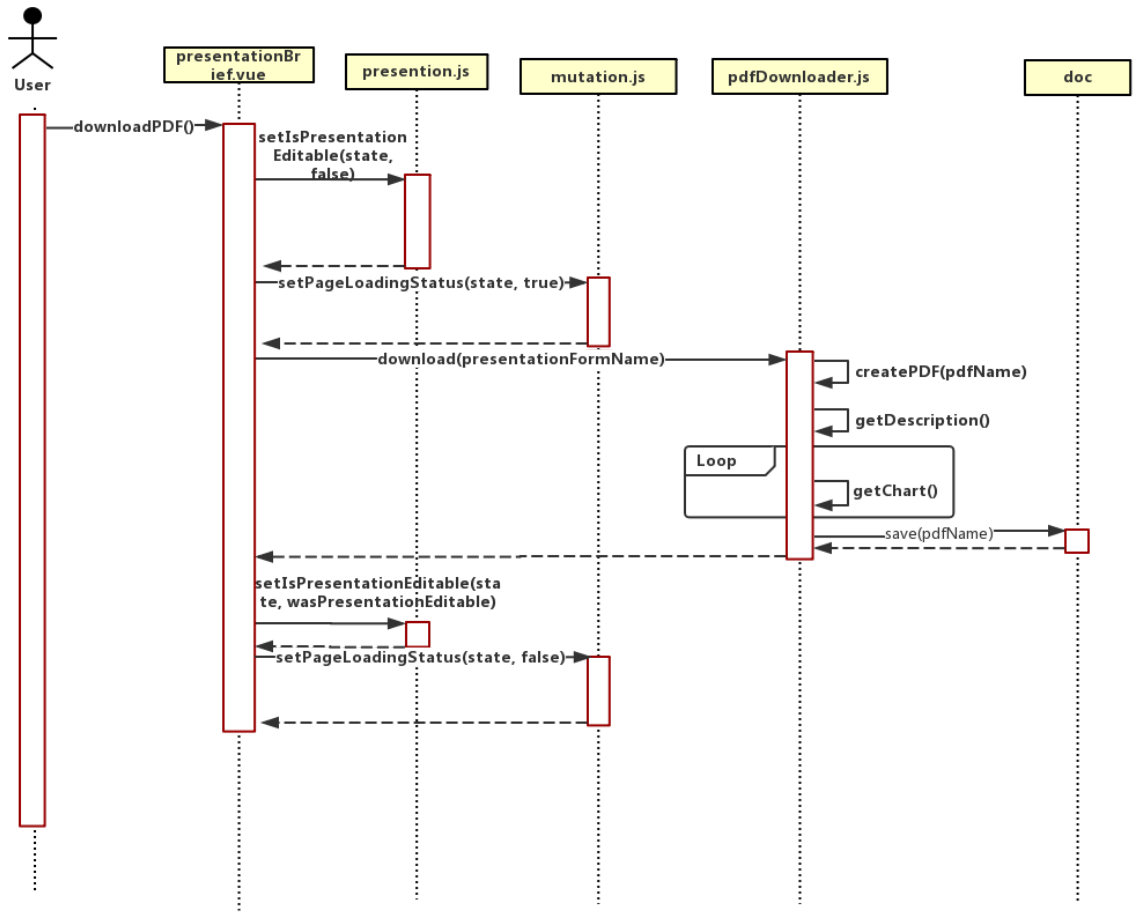

User
presentationBrief.vue
presention.js
mutation.js
pdfDownloader.js
doc
downloadPDF()
setIsPresentationEditable(state, false)
setPageLoadingStatus(state, true)
download(presentationFormName)
createPDF(pdfName)
getDescription()
Loop
getChart()
save(pdfName)
setIsPresentationEditable(state, wasPresentationEditable)
setPageLoadingStatus(state, false)

Figure 13: Download Presentation as PDF Sequence Diagram.

The above Figure 13 demonstrates what the Download Presentation as PDF does after a user clicking the download as PDF button at `PresentationBrief.vue` page.

1. `presentationBrief.vue` page calls the `setIsPresentationEditable` method from `presentation.js` and set the state to `false`.
2. `presentationBrief.vue` page calls the `setPageLoadingStatus` method from `mutation.js` and set the state to `true`.
3. `presentationBrief.vue` page calls the `download` method from `pdfDownloader.js`. The `pdfDownloader.js` calls `createPDF`, `getDescription`, and `getChart` methods to get the information of the presentation and then save all the information into a PDF document.
4. `presentationBrief.vue` page calls the `setIsPresentationEditable` method to set the state to the old state and calls the `setPageLoadingStatus` method to set the loading state to `false`.

Make shareable link

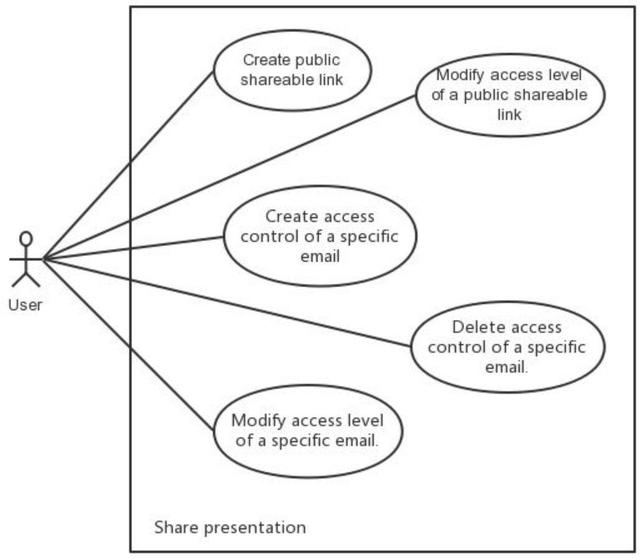

Create public shareable link
Modify access level of a public shareable link
Create access control of a specific email
User
Delete access control of a specific email.
Modify access level of a specific email.
Share presentation

Figure 14: Use Case Diagram for Sharing Presentation

The above Figure 14 shows the use case diagram of sharing presentation with other users. An organizer of the presentation can create a public shareable link to all users or can create a link only share to specific e-mails. The system allows the organizer to remove those access controls at any time.

Prioritized Requirements

Functional

Before listing the functional requirements that we have prioritized, we will detail the user stories that led us to them in the first place in Table 15 below.

No.	User Role	Requirement	Benefit
1.	As a Program Chair...	I would like to add co-authorship visualizations to my presentations	As it will help me glean insights into author, organizational, country or university collaborations in creating papers
2.		I would like to be able to store my data for different conferences without having to replace existing data	As it will save me time, making the system more convenient to use
3.		I would like to be able to create separate presentations, for many different conferences at one go	
4.		I would like to be notified when a presentation is shared with me or I would like the recipient to be notified when I share a presentation	As it makes me aware of the presentation resources that are available for me to collaborate/work on
5.	As a Program Chair who is a prospective new user...	I would like to know without creating an account what I can do with the system	As it gives me a clear indication of whether or not it will meet my requirements
6.	As a Program Chair...	I would like to see right upon logging in the presentations and conferences that I have created	As it improves the usability of the interface and gets me to where I need to go quicker

Table 15: User Stories Outlining Rationale For Functional Requirements

In Table 16 we define the functional requirements that we have focused on in the creation of ChairVisE3.0, and the user story that it is associated with.

User Story No.	Associated Prioritized Functional Requirements
1	Implementation of co-authorship queries, in order to visualize the collaboration between authors, organizations or countries based on the co-author data availability.
	Implementation of other visualisations so that the app provides more value to the user, such as average expertise for each evaluation score or confidence score, and visualising which papers are of similar quality based on the reviewer's evaluation score.
2	Creation of conference entities that allows the association of one author, submission, review record type each to a conference without the need of having to delete records of other conferences.
3	Association of presentations with conferences to enable many different presentations to be made for different conferences.
4	Sending an e-mail notification to inform that a presentation has been shared with the recipient and provide a hyperlink to quickly access that presentation on the system.
5	Creation of a landing page that clearly outlines the core functionalities of ChairVisE to prospective new users of the system.
6	Creation of a dashboard that clearly shows the user’s existing presentations and conferences with quick links to each of them.

Table 16: List of Prioritized Functional Requirements

Non-Functional

With respect to non-functional, quality requirements, our ChairVisE3.0 hopes to zero in on the following:

- **Authentication and Authorization**: To ensure that the same standards regarding data access is maintained, new entities such as Conferences can only be created or retrieved after having gone through a verification process that ensures the user is logged in and only the user’s own Conference data is retrieved.
- **Usability and User Experience**: We have further revamped the user interface in order to make it simpler and more efficient for the user to generate the visualizations and presentations that they require. The user guide has been updated with information about the additions to the data importing and presentation

creation processes. Furthermore, with the inclusion of Conference entities, it is hoped that the user finds it easier to be aware regarding the data they have uploaded it and manage it accordingly.

- **Extensibility**: With the addition of the new separate Conference entity as a means of persisting multiple conferences, we leave the door open for further functionality to be added as this entity can be used to tie in and coordinate/manage other new entities that may be created. We briefly outline one possibility under the section on Future Improvements and Enhancements.

Developer Documentation

Development Process

Agile Development

We developed ChairVisE3.0 in an agile manner, having sprints of 2 weeks. Before beginning the development process itself we discussed about the existing version, ChairVisE2.0, and broke down our features into more manageable components and elicited the requirements for our enhanced version. Following this, we analyzed and decomposed the requirements into more manageable parts, eventually writing down these functional requirements and illustrating them. In this time, we took charge of individual features and considered where there might be dependencies or closer need for coordination during development. Lastly, before coding, we reviewed and validated that our proposed enhanced features would lead to improvements that would meet the needs of the Program Chairs.

While we did not have daily Scrum meetings, we met once every two weeks to update one another about our incremental progress towards realizing the entire feature. Our sprints lasted 2 weeks which meant about 4 sprints in total, including one after a code freeze to iron out any bugs and to work on this report.

Design Process

Our team felt that the current layered architecture model of ChairVisE2.0 made it easy to modify and implement the features that we desired. As such, we did not require any major changes to the architecture. The only minor point of note with respect to architecture is the use of the D3 JavaScript library to create new visualizations for co-authorship.

To implement the persistence of multiple conference data, we had to create a couple of new entities: `Conference` and `ConferenceRecord`. While these new entities will be further explained in the subsequent section that covers the feature itself, it is important to note that this design choice was made so as to adhere to the **single responsibility principle**. While we could have incorporated the functionality given by these two new entities with the existing entities such as `Presentation` and the different `Records`, this would have resulted in high levels of coupling leading to messier code and lower ease of extension in the future, which the new `Conference` entity provides. As such, in the process we have also continued to ensure that there is **low coupling** and **separation of concerns**.

We also adhered to the principle of **information hiding** as we ensured that both existing and new internal data structures cannot be accessed by the client and implementation details of new classes were hidden from the other classes that used their methods or attributes (e.g. setter and getter methods and the use of appropriate access modifiers for methods).

Finally, we sought to adhere as close as possible to the **open-close principle**. While modifying certain existing modules was inevitable in some areas, in most others we merely extended the behaviour of the existing module. The most prominent example of this is in the generation of the co-authorship queries. We worked with the existing data aggregation framework to enable these new visualizations to be generated. Specifically, we used the existing `analyse` method to generate the SQL string and retrieve the necessary data. Further explication of this feature will be done in the section Co-authorship Queries.

Overall Architecture for ChairVisE3.0

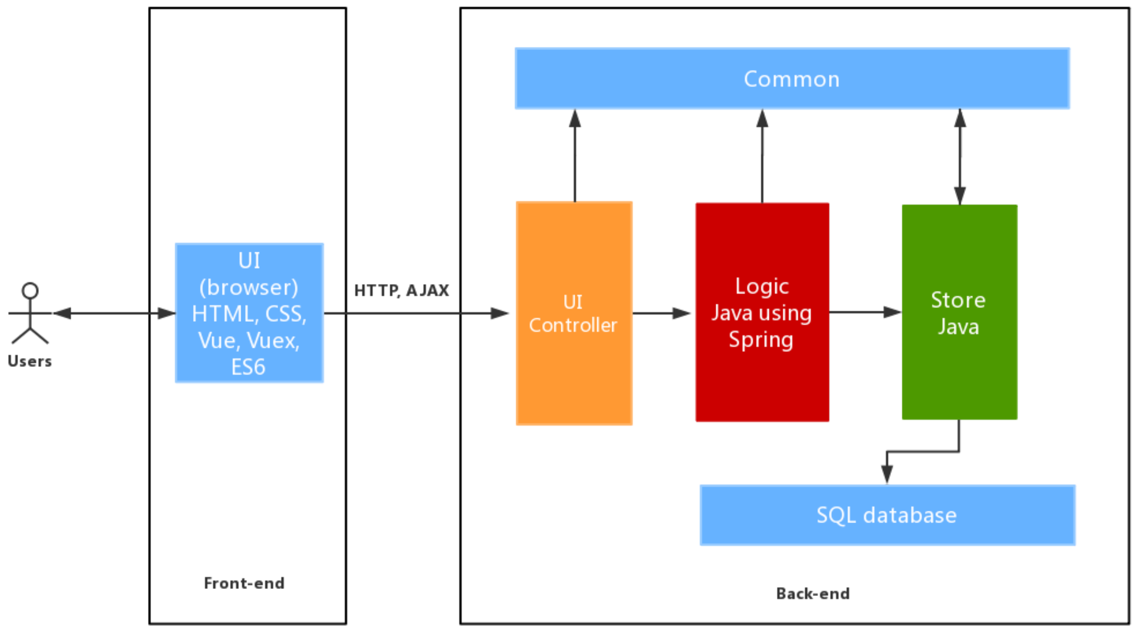


Figure 17: Overall Architecture of ChairVisE3.0

As mentioned earlier, our ChairVisE3.0 does not make any architectural changes to the previous 2-layer design used by ChairVisE2.0 but merely enhances it by building upon the existing version by including new entities, adding new visualizations, embedding an e-mail notification system and revamping the frontend interface. All these take place within the existing architecture and the changes made to system are outlined in the below sections detailing each feature.

Co-Authorship Queries

In order to implement co-authorship queries, D3.js was introduced in order to support creating force directed graphs in order to visualize the respective queries.

Force Directed Graphs

It is possible to view force directed graphs based on data sent from the server. In order to make it easier, 2 new API functions are available within `AnalysisController`: `coauthorshipdata` and `coauthorshipdatasimilar`. These API calls provide data in a set of nodes and links, which are then provided to D3 in order to create the mapping. In this way, the `AnalysisRequest` only needs to retrieve the list of links and the list of nodes are automatically generated based on the links retrieved.

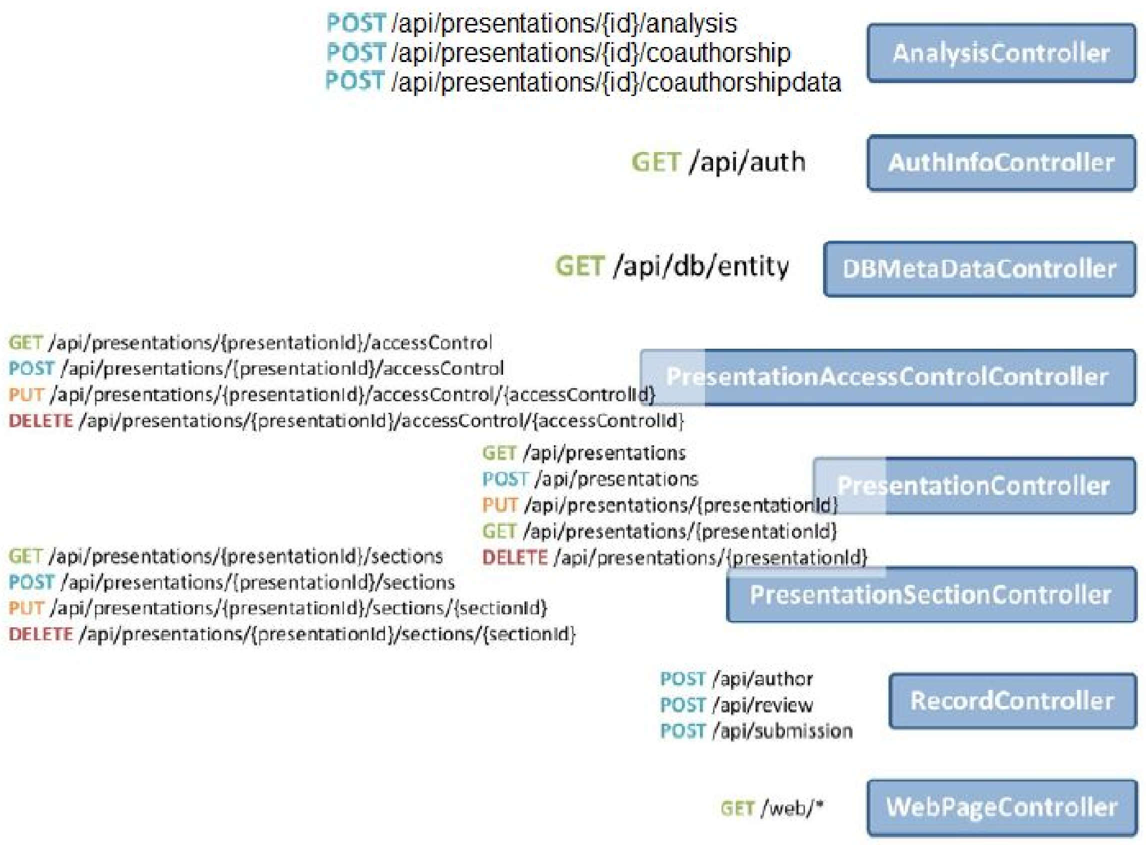

POST /api/presentations/{id}/analysis
POST /api/presentations/{id}/coauthorship
POST /api/presentations/{id}/coauthorshipdata
AnalysisController
GET /api/auth
AuthInfoController
GET /api/db/entity
DBMetaDataController
GET /api/presentations/{presentationId}/accessControl
POST /api/presentations/{presentationId}/accessControl
PUT /api/presentations/{presentationId}/accessControl/{accessControlId}
DELETE /api/presentations/{presentationId}/accessControl/{accessControlId}
PresentationAccessControlController
GET /api/presentations
POST /api/presentations
PUT /api/presentations/{presentationId}
GET /api/presentations/{presentationId}
DELETE /api/presentations/{presentationId}
PresentationController
GET /api/presentations/{presentationId}/sections
POST /api/presentations/{presentationId}/sections
PUT /api/presentations/{presentationId}/sections/{sectionId}
DELETE /api/presentations/{presentationId}/sections/{sectionId}
PresentationSectionController
POST /api/author
POST /api/review
POST /api/submission
RecordController
GET /web/*
WebPageController

Figure 18: Listing of New REST API Endpoints in AnalysisController

Due to need in difference in coloring for the the different coauthorship graphs depending on the difference in nodes, 2 API call is created in order to support both types of coloring for the nodes, whether they have the same color or not. However, the specific colors of the nodes and the links can be specified themselves in predefinedQueries, thus they can be changed depending on the graph.

New Visualisations are added using the force directed graph and the bar chart.

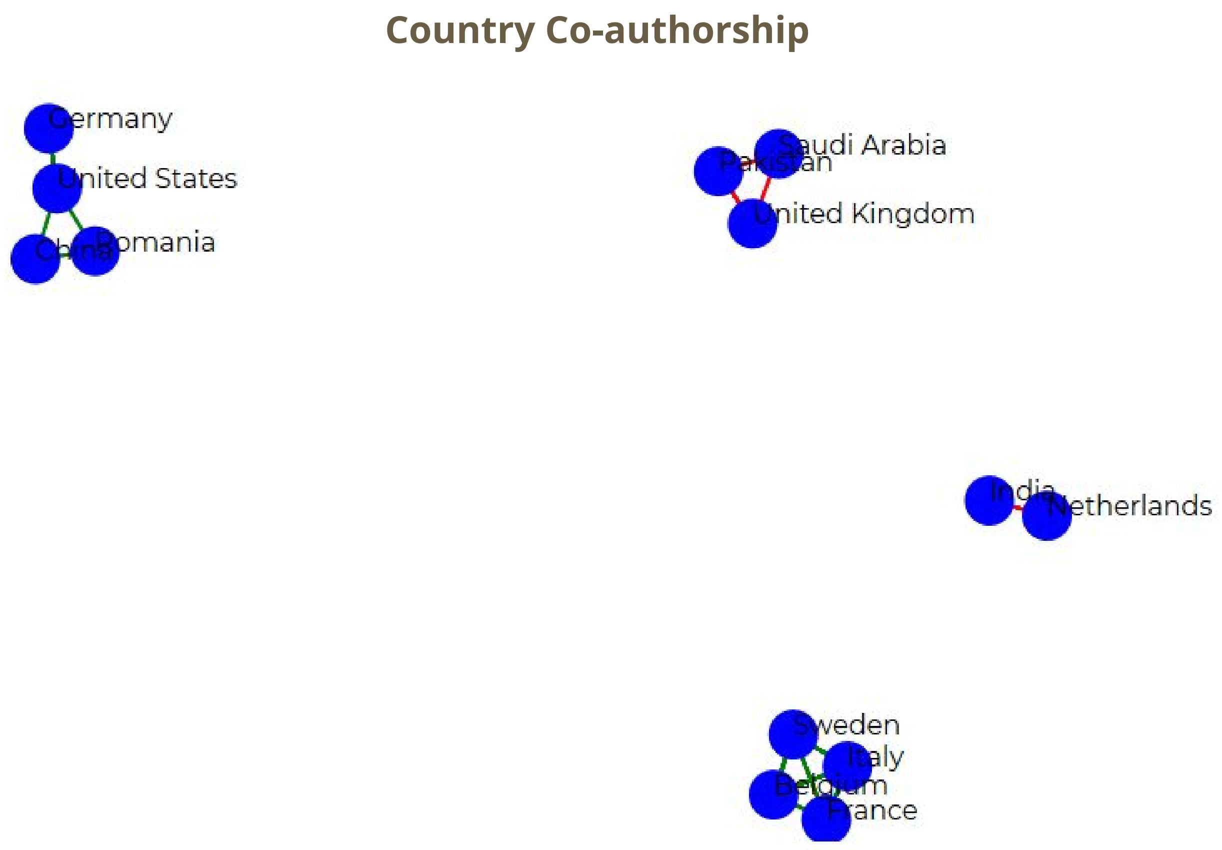


Figure 19: Country Co-authorship Visualization

By combining author and submission record, we can see which papers had collaborators from different countries and plot them. This could indicate organisations or teams within those countries working together. The line between the countries indicates whether the paper was accepted.

Organisation Co-authorship

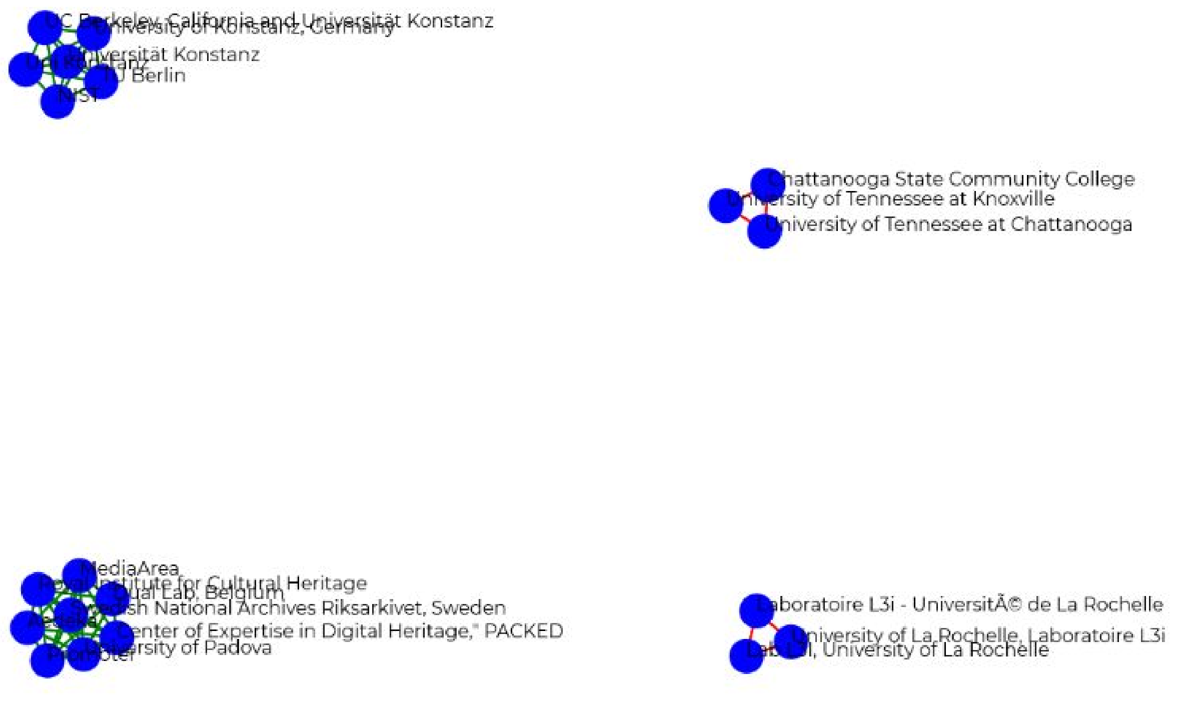

UC Berkeley, California and Universität Konstanz
University of Konstanz, Germany
Universität Konstanz
Uni Konstanz
TU Berlin
NIST
Chattanooga State Community College
University of Tennessee at Knoxville
University of Tennessee at Chattanooga
MediaArea
Royal Institute for Cultural Heritage
Visual Lab, Belgium
Swedish National Archives Riksarkivet, Sweden
Center of Expertise in Digital Heritage," PACKED
University of Padova
Laboratoire L3i - UniversitÃ© de La Rochelle
University of La Rochelle, Laboratoire L3i
Lab L3i, University of La Rochelle

Figure 20: Organization Co-authorship Visualization

By combining author and submission record, we can see which papers had collaborators from different organisations and plot them. This could indicate organisations that are working together. The line between the organisations indicates whether the paper was accepted.

Co-authorship Author Record

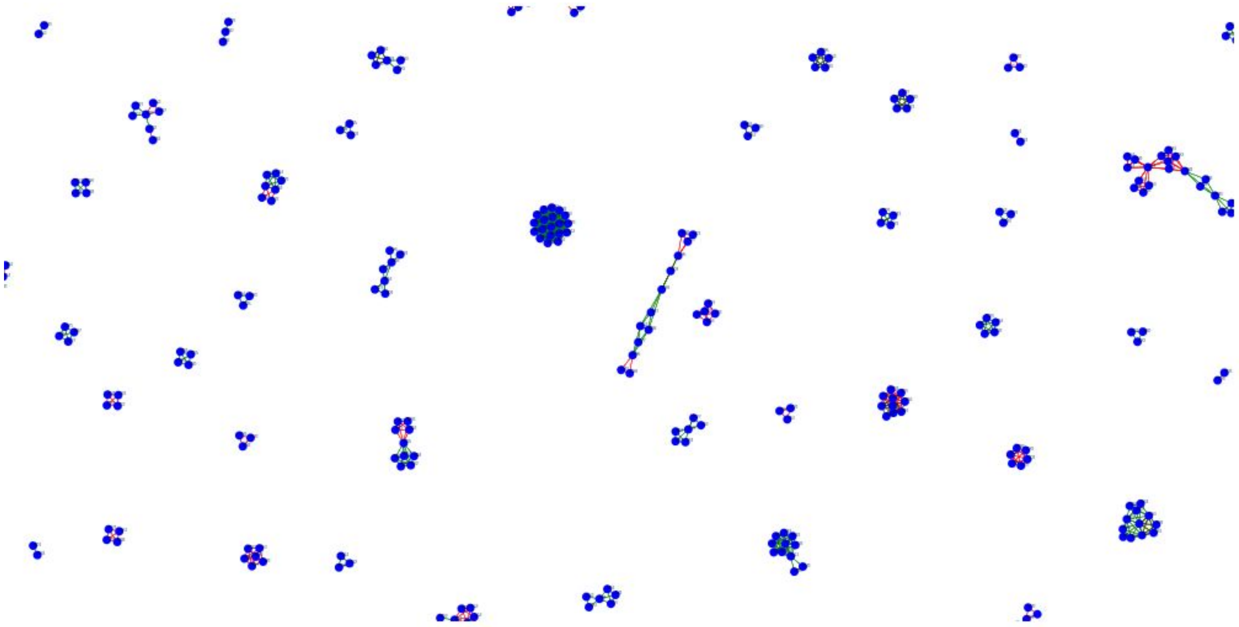


Figure 21: Co-authorship Visualization - Author Record

All authors that have worked with another author on any submission is depicted as connected. A green line indicates that their latest collaboration was accepted, a red line indicates otherwise. This graph could indicate certain authors constantly working together for research papers, and could also indicates authors which have worked with others in some papers but not others, which might imply that they aren’t as close as other authors.

Co-authorship Submission record

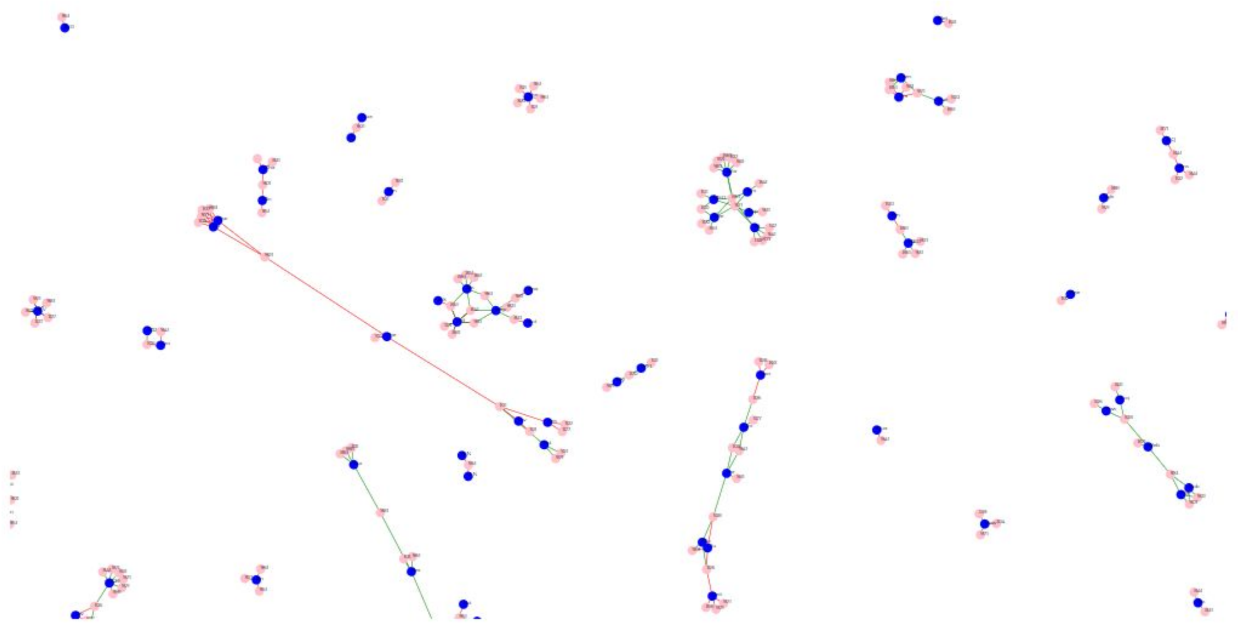


Figure 22: Co-authorship Visualization - Submission Record

This graph is a more indepth look from co-authorship submission record. It examines the papers that were submitted by those authors as well, which shows further detail on which papers the authors worked together on. Additionally, it indicates whether each paper was accepted or rejected

Average Expertise per Evaluation Score

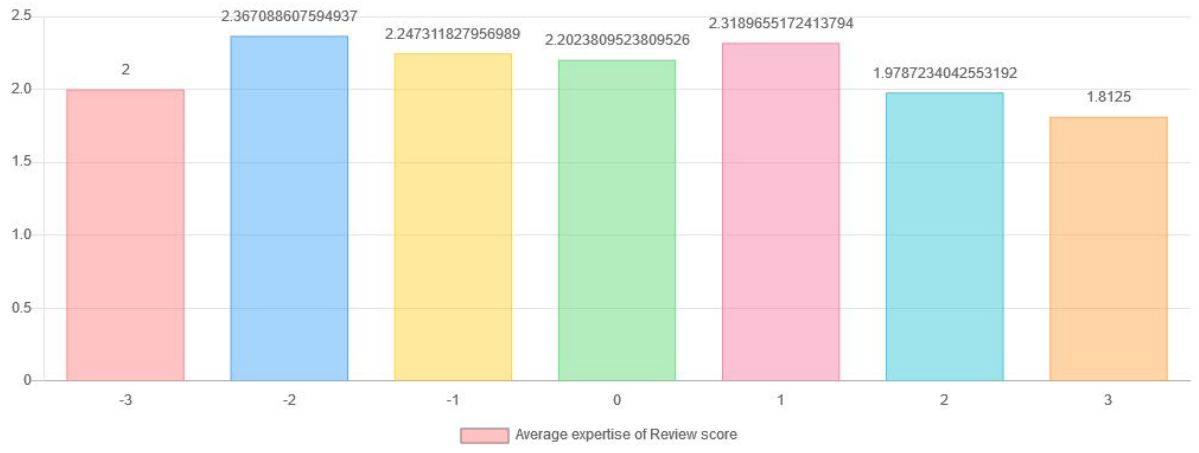


Figure 23: Average Expertise Per Evaluation Score Visualization

This bar chart shows the average expertise for each review that was given a certain evaluation score. This can show trends within reviewer' expertise that gives a certain score.

For example, a downwards slope would indicate that reviews that give higher scores are on average by reviewers without much expertise on the subject.

Reviewers Evaluation Expertise

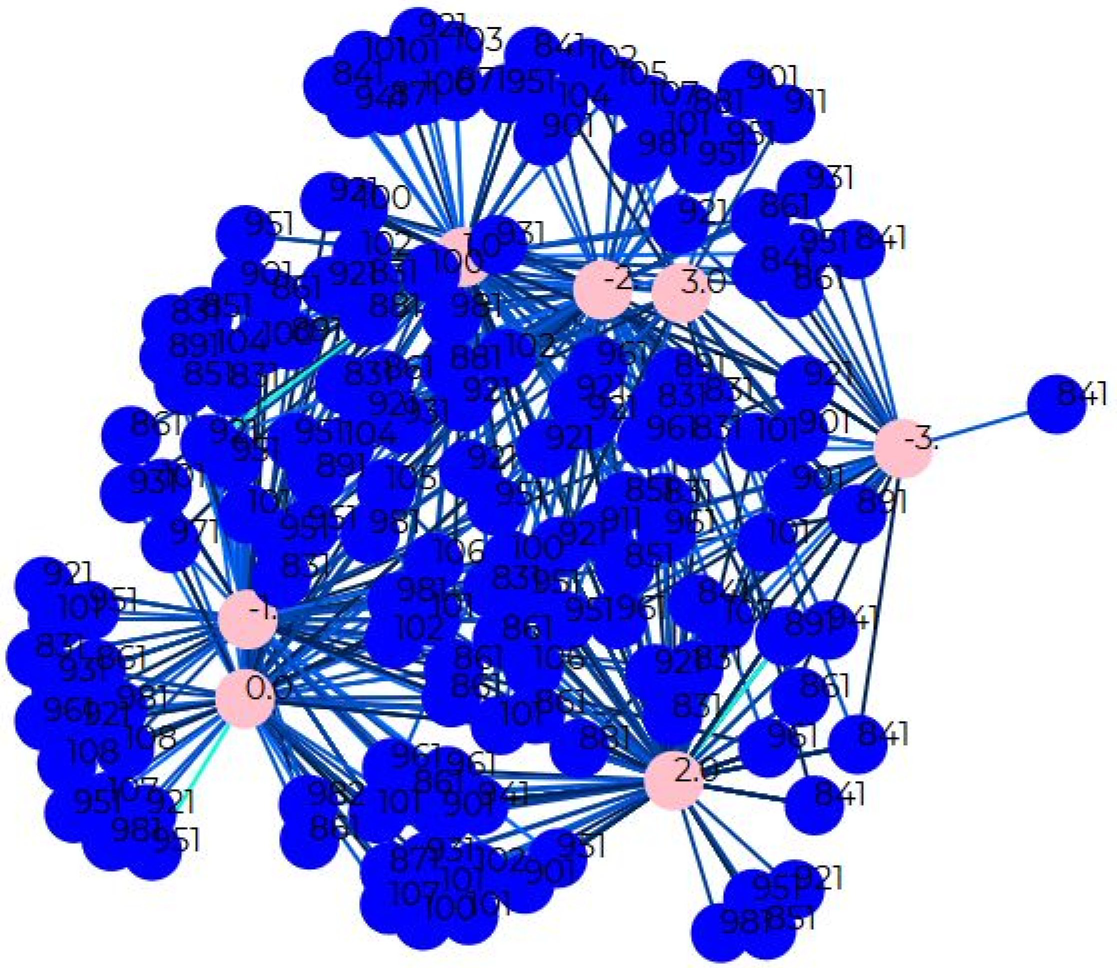


Figure 24: Reviewers Evaluation Expertise Visualization

This visualisation is expands on the previous bar chart to show the relationship between each reviewer, the scores they gave and their level of expertise. In this way, we can see based on each score, the general distribution of the level of expertise of the reviewers and which reviewers fall into the category by dragging the score out.

Average Expertise per Evaluation Confidence

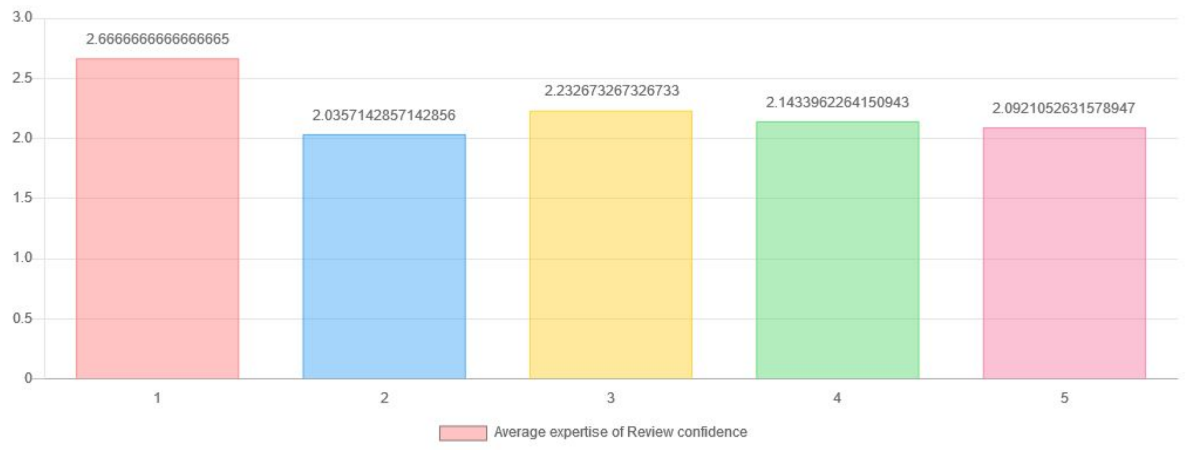


Figure 25: Average Expertise Per Evaluation Confidence Visualization

This bar chart shows the average expertise for each review that was given a certain evaluation confidence. This can show trends within reviewer' expertise that gives a certain confidence. For example, a downwards slope would indicate that reviewers that are not very confident of their score are on average have a lower level of expertise on the subject

Reviewers Evaluation Confidence

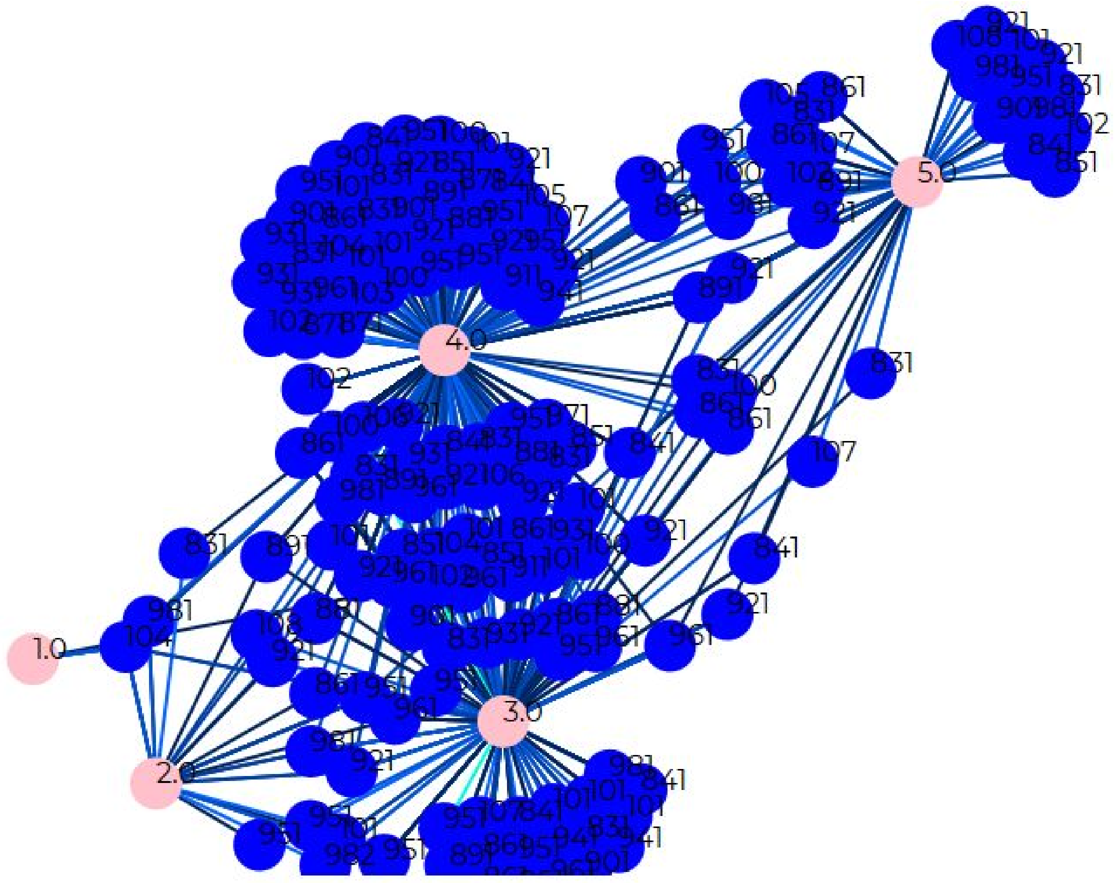


Figure 26: Reviewers Evaluation Confidence Visualization

This visualisation is expands on the previous bar chart to show the relationship between each reviewer, their level of confidence and their level of expertise. In this way, we can see based on each confidence level, the general distribution of the level of expertise of the reviewers and which reviewers fall into the category by dragging the confidence node out.

Authors Review Score

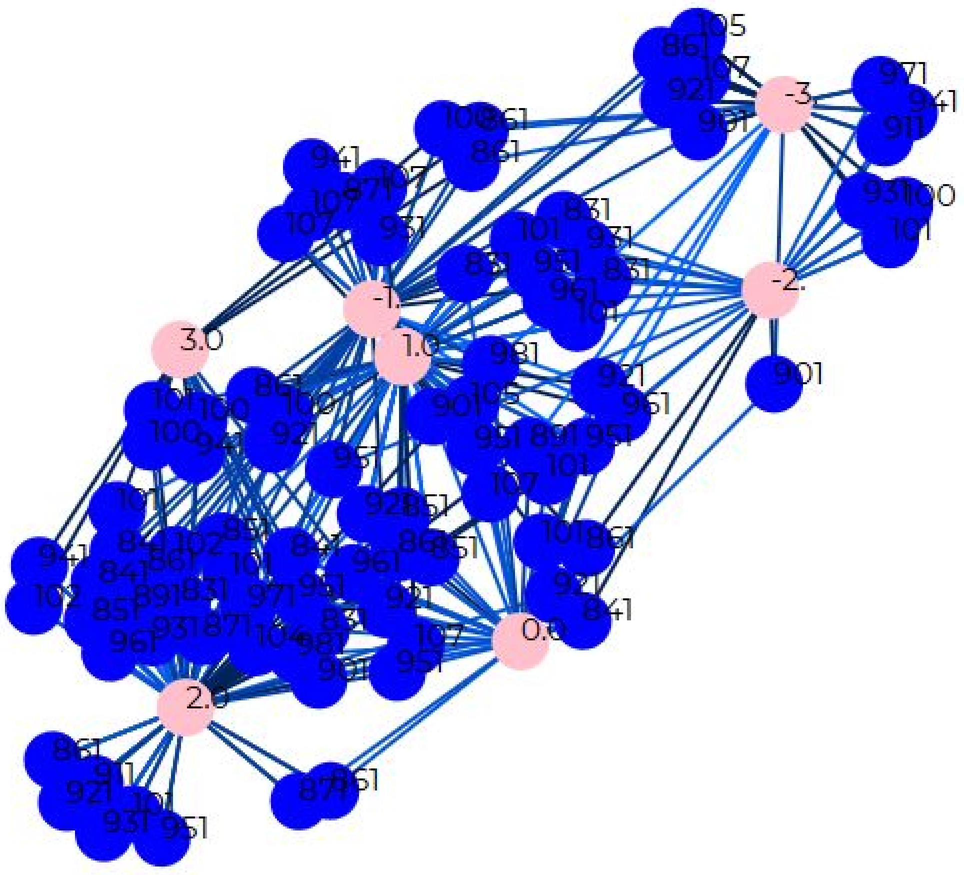


Figure 27: Authors Review Score Visualization

By combining Author and Review record, we can see a visualisation of how each author's review scores are, and the reviewer's level of expertise when performing the review. In this way, we can try to identify authors who received reviews from reviewers with low level of expertise and see their scores, or identify authors who received reviews from reviewers with high level of expertise and see their scores. We can also try to the general level of expertise of reviewers who gave a specific score.

Papers of Similar Quality

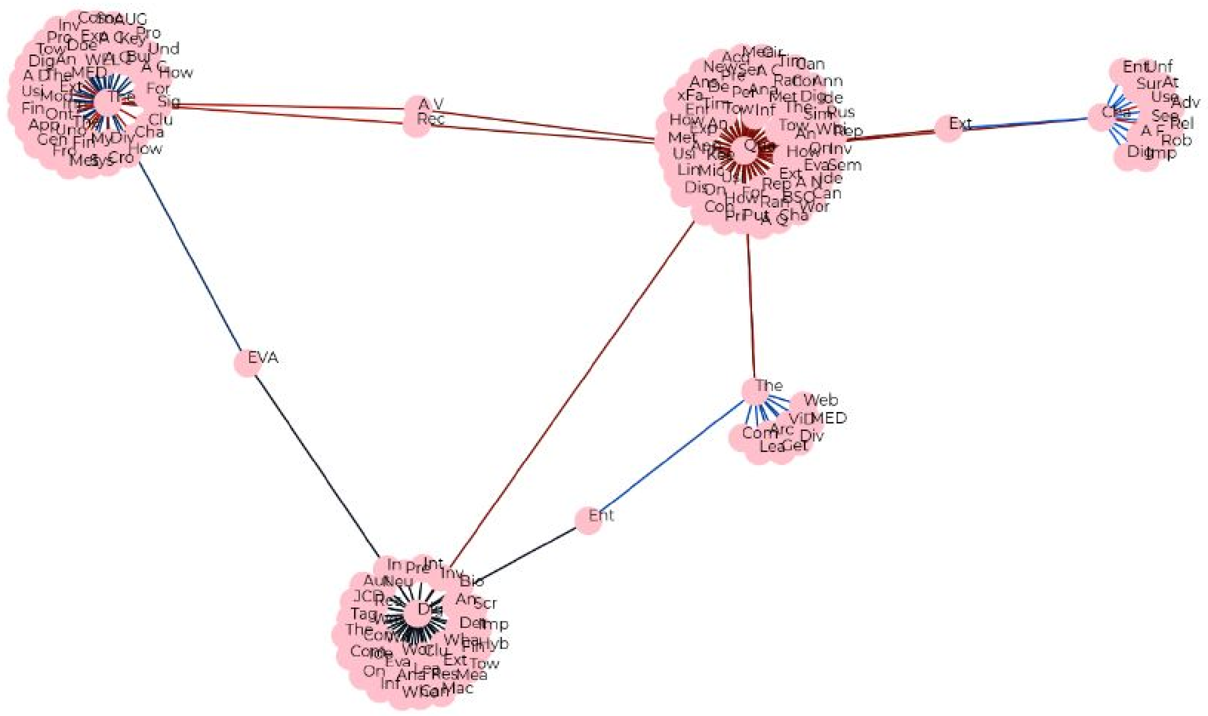


Figure 28: Papers of Similar Quality Visualization

By combining review and submission, papers which receive similar review scores can be matched to each other to see which papers are similar in quality to each other. Each pink node represents a paper. The color of the links between them represents the evaluation score given to those papers. A score in the large negatives is marked as red. A score of 0 is black. A score in the large positives is blue. A small positive number or small negative number interpolate colors between red, black and blue.

Persistence of Multiple Conference Data

To enable the persistence of multiple conference data and thus allow for the creation of multiple presentation that is based off different data fundamentally requires the creation of a Conference entity that acts as the “parent” entity associated with both presentations and the imported data records. Fortunately, Hibernate and Spring JPA allows such an entity to be created rather easily. To understand the relationship between Conference, Record and Presentation entities, the below figure builds upon the entity relationships from ChiarVisE2.0 shown in Figure 8. The new Conference entity is in dark blue.

Conference

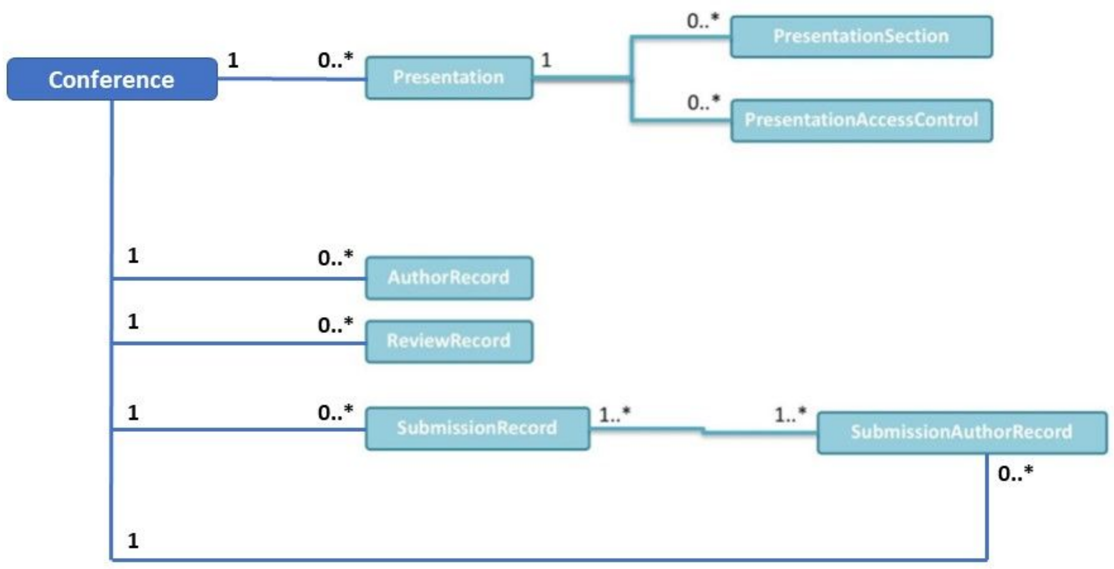


Figure 29: Conference Entity’s Relationships with Records and Presentation

A Conference entity is made up of just the conference’s name and the year of the conference. It can be associated with none or many of each of the record entity, Author, Review, Submission and SubmissionAuthor. However, of crucial importance is that once there exists this association, any future imports of data of that type will result in all current record entities of that type associated with the conference being deleted from the database before the new data is persisted. This was done so as to prevent any possible double-counting of data thus producing inaccurate visualizations. This is done through the means of a ConferenceRecord entity which will be detailed subsequently.

With this one to many association of the Conference entity with Presentation entity , it means that every presentation is associated with one conference from which it draws its data from. This relationship cannot be modified at this point in time in the current

ChairVisE3.0 but will be a future addition. More importantly, this means that every presentation can be associated with the same conference as another presentation or with another entirely different conference altogether. The selection of the conference with which the presentation or record being imported is associated with, is presented as a simple dropdown of all existing conferences during the process of presentation creation or importing of record data. Do note that without at least one conference in the system, the user *will not* be able to import any data or create any presentations.

ConferenceRecord

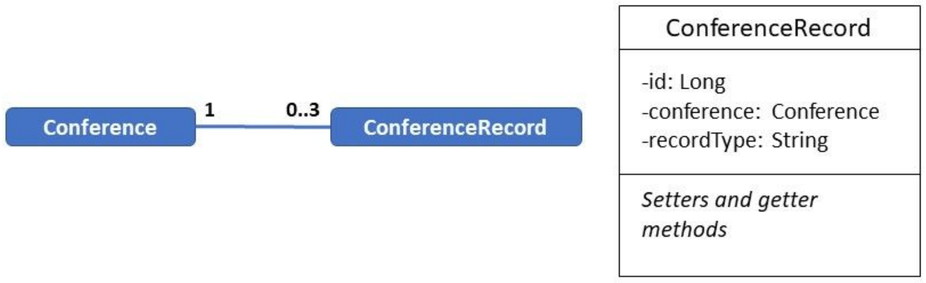


Figure 30: Conference and ConferenceRecord Entity Relation and ConferenceRecord Class Diagram

A ConferenceRecord entity helps to keep track of the record types that have already been uploaded to the database so that, should a new set of records of that same type be uploaded again then it will check if there exist and current ones, remove them if required, before saving the new imported data. Hence one Conference can have only 3 ConferenceRecord entities associated with it maximum (for each type of record).

If a Conference is deleted, the effect cascades, resulting in a deletion of all associated ConferenceRecords, Presentations (and thus PresentationSections and PresentationAccessControls), as well as any AuthorRecords, ReviewRecords and SubmissionRecords. Through this, it can be clearly seen that the Conference entity is now the “parent” entity without which other entities cannot exist.

ConferenceController, ConferenceLogic & ConferenceRecordController, ConferenceRecordLogic

Clearly, to use the newly added Conference and ConferenceRecord entities, the necessary API endpoints and logic had to be added to their respective controller and logic classes. Mainly, REST API endpoints had to be added for conference CRUD operations (thus requiring the POST, GET, PUT, DELETE HTTP methods respectively) and to retrieve the type of records associated with a certain conference.

Figure 31 shows how these new controller and logic classes fit in, building upon the existing association of controllers with logic classes seen in Figure 6 previously. In addition, Figure 32 shows how the new logic classes are associated with the repository classes, building upon the existing associations seen in Figure 7 previously. New classes and associations are indicated in gold.

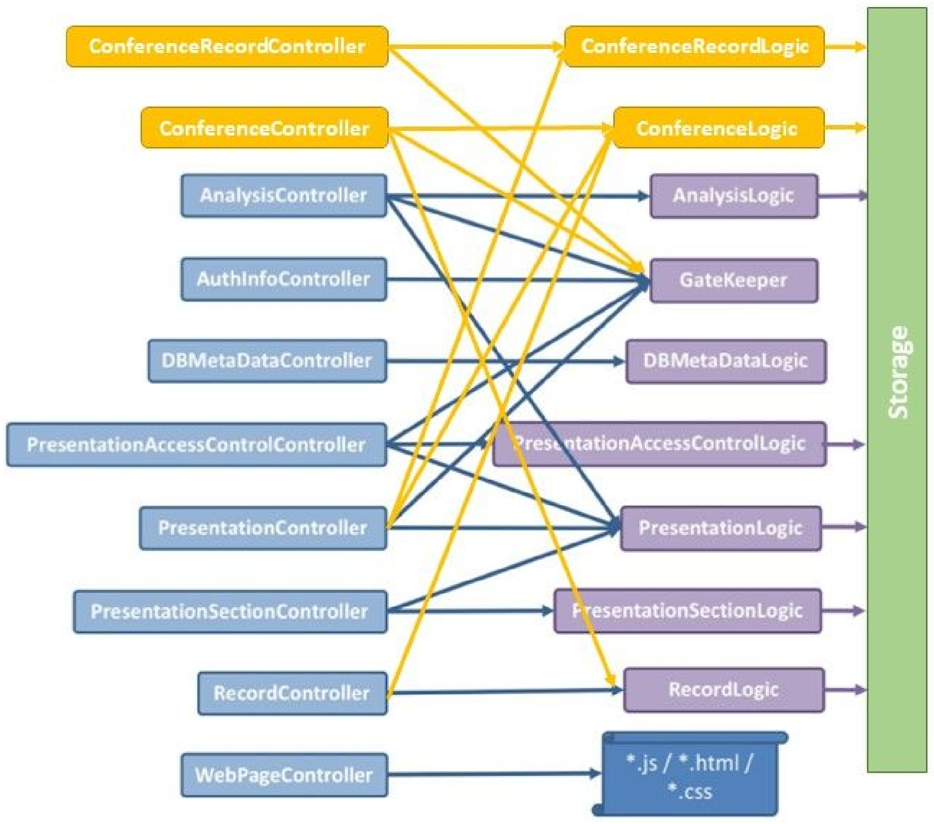


Figure 31: Association of new Conference and ConferenceRecord Logic and Controller Classes with Existing Classes from ChairVisE2.0

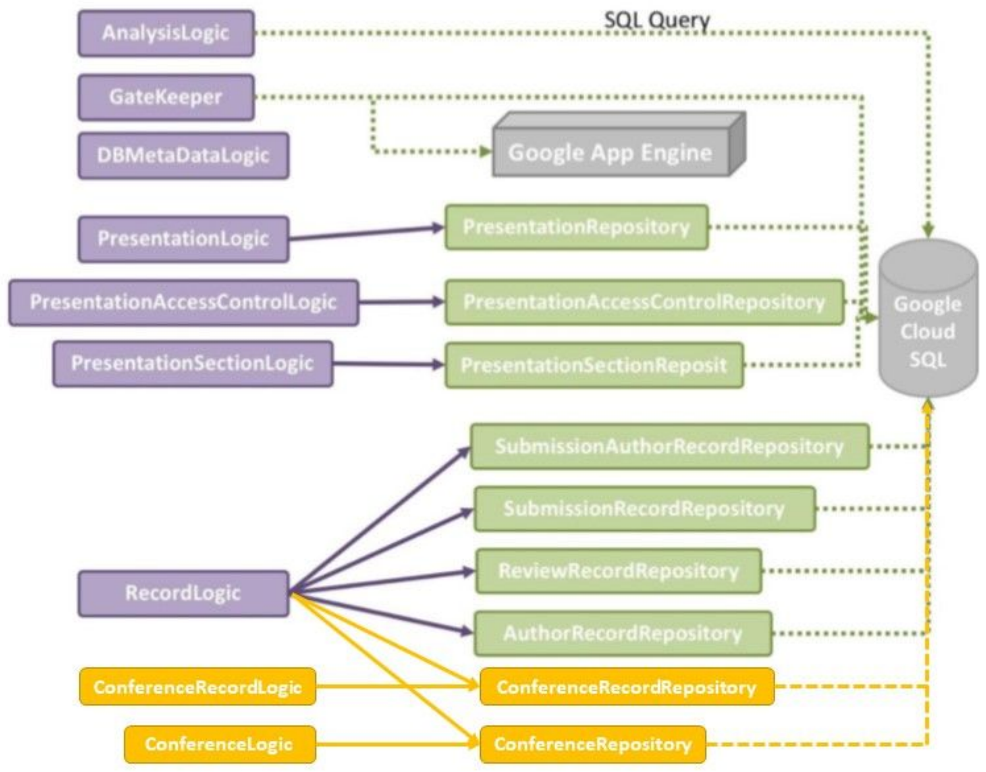


Figure 32: Association of new Conference and ConferenceRecord Logic Classes and Repository Interfaces with Existing Repository Interfaces from ChairVisE2.0

The newly added REST API endpoints of the `ConferenceController` and `ConferenceRecordController` classes are also shown in the figure below (in gold). It builds upon the existing endpoints previously seen in Figure 5.

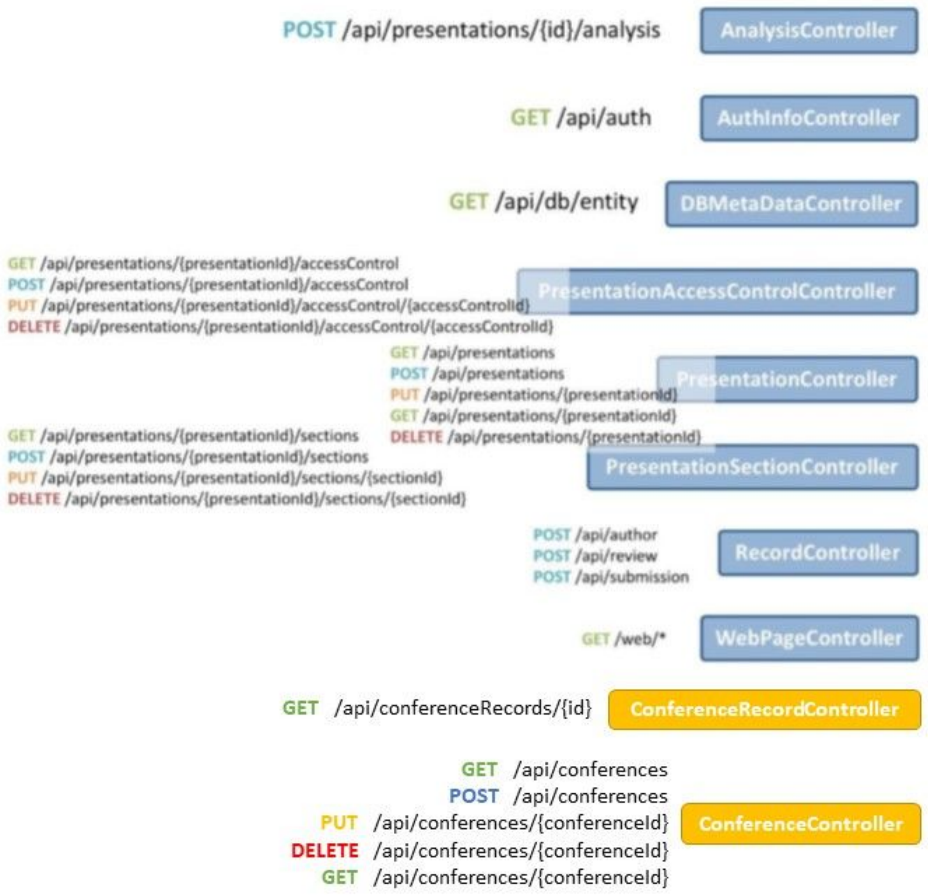

POST /api/presentations/{id}/analysis
AnalysisController
GET /api/auth
AuthInfoController
GET /api/db/entity
DBMetaDataController
GET /api/presentations/{presentationId}/accessControl
POST /api/presentations/{presentationId}/accessControl
PUT /api/presentations/{presentationId}/accessControl/{accessControlId}
DELETE /api/presentations/{presentationId}/accessControl/{accessControlId}
PresentationAccessControlController
GET /api/presentations
POST /api/presentations
PUT /api/presentations/{presentationId}
GET /api/presentations/{presentationId}
DELETE /api/presentations/{presentationId}
PresentationController
GET /api/presentations/{presentationId}/sections
POST /api/presentations/{presentationId}/sections
PUT /api/presentations/{presentationId}/sections/{sectionId}
DELETE /api/presentations/{presentationId}/sections/{sectionId}
PresentationSectionController
POST /api/author
POST /api/review
POST /api/submission
RecordController
GET /web/*
WebPageController
GET /api/conferenceRecords/{id}
ConferenceRecordController
GET /api/conferences
POST /api/conferences
PUT /api/conferences/{conferenceId}
DELETE /api/conferences/{conferenceId}
GET /api/conferences/{conferenceId}
ConferenceController

Figure 33: Listing of New REST API Endpoints in Conference and ConferenceRecord Controllers

To accommodate the addition of the conference ID in the request body of the HTTP request that contains a list of the records being imported, we composed the existing list of Records inside a wrapper class that also took an additional Long for the conference ID. This enabled us to not make any drastic changes to the existing structures in the Vue store on the frontend, or record controller and logic classes in the backend. A similar wrapper was created for Presentation.

For illustration purposes, Figure 38's sequence diagram shows the changes to the method signatures for importing of conference data. It is essentially a marginally modified version of the sequence diagram in Figure 10. Changes are highlighted.

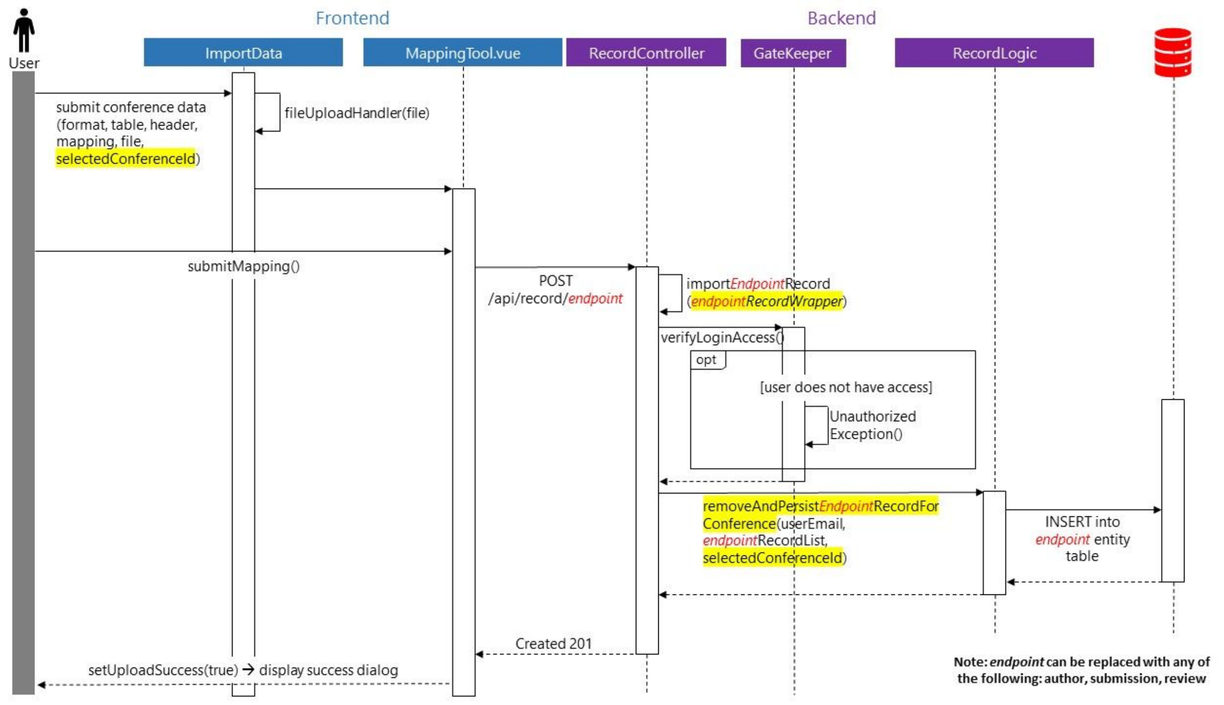

Frontend
Backend
User
ImportData
MappingTool.vue
RecordController
GateKeeper
RecordLogic
submit conference data (format, table, header, mapping, file, selectedConferenceId)
fileUploadHandler(file)
submitMapping()
POST /api/record/endpoint
importEndpointRecord (endpointRecordWrapper)
verifyLoginAccess()
opt
[user does not have access]
UnauthorizedException()
removeAndPersistEndpointRecordForConference(userEmail, endpointRecordList, selectedConferenceId)
INSERT into endpoint entity table
Created 201
setUploadSuccess(true) → display success dialog
Note: endpoint can be replaced with any of the following: author, submission, review

Figure 34: Sequence Diagram Illustrating Conference Data Upload in ChairVisE3.0

The method name was changed to `removeAndPresist`*`Endpoint`*`RecordFor`**`Conference`** to better represent what the method now does (note, *endpoint* refers to the type of record - Author, Review, Submission):

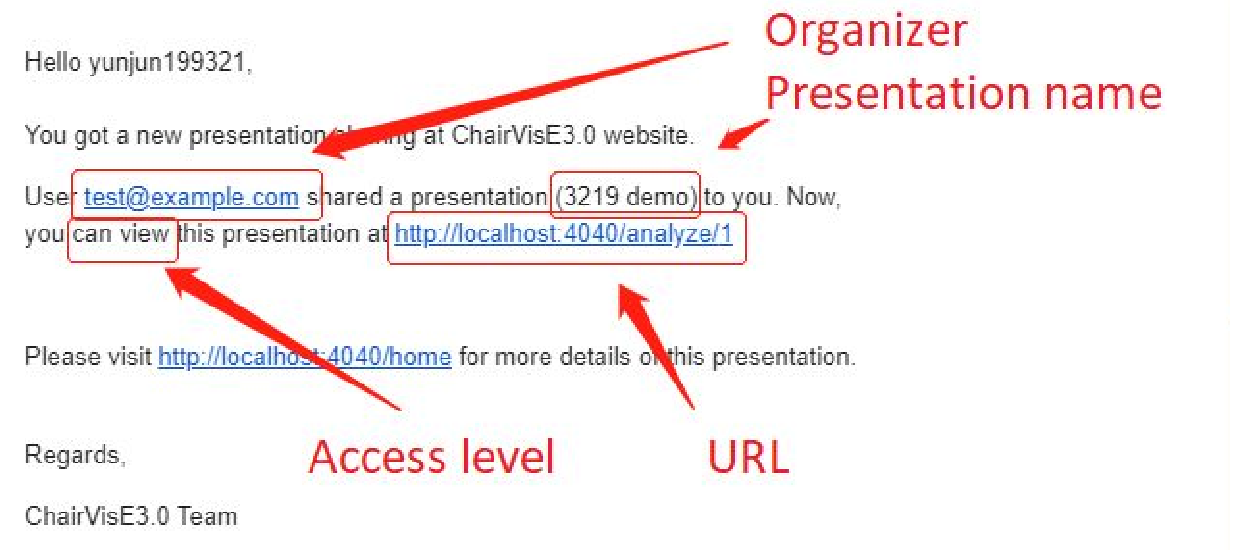
- It retrieves the selected Conference entity from the database table based on the conference ID.
- It deletes from the database table of that record type, any records of the type in question that are already associated with the selected conference, if any.
- It deletes ConferenceRecord entities from the database table based on the conference ID and type of record, if any.
- It performs the same saving of records in the respective table of that record type.
- It saves a new ConferenceRecord entity in the database table to indicate a record of the type in question is now associated with the selected conference.

Finally, with respect to the `analyse` method of the `AnalysisLogic` class, we added an additional parameter to the method signature, a `Long` for the conference ID. This would be utilised in the `dataSetFilter` string generated: instead of generating a `WHERE` clause that matched with the user's e-mail login, it now matches with the Conference entity that has the same ID.

E-mail Notification for Sharing a Presentation

The Program Chair who creates a presentation can also create a shareable link that allows them to share the presentation with another person. Access is given based on the e-mail of the other individual and there are two levels of access control: edit or view. The Program Chair can also modify or remove one's access control to the presentation.

CharVisE3.0 now automatically generates a notification e-mail and sends it to that specific e-mail account whenever the organizer creates, modifies or removes the access control given to that specific e-mail. The recipient of the e-mail can visit the shared presentation in the CharVisE3.0 through the link in the notification e-mail.


Hello yunjun199321,

You got a new presentation sharing at ChairVisE3.0 website.

User test@example.com shared a presentation (3219 demo) to you. Now, you can view this presentation at http://localhost:4040/analyze/1

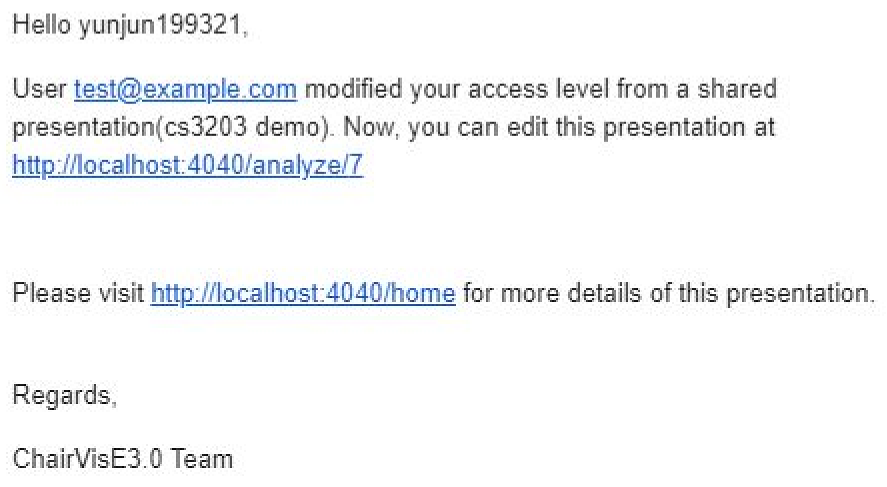
Please visit http://localhost:4040/home for more details of this presentation.

Regards,

ChairVisE3.0 Team

Figure 35: Sample of the Notification E-mail - Sharing of Presentation

Figure 35 shows that the Program Chair who created a presentation added a new access control to the e-mail yunjun199321@gmail.com. ChairVisE3.0 generated a notification e-mail with the name of the presentation, the e-mail of the Program Chair, access level given to the user, and the URL of the presentation.



Hello yunjun199321,

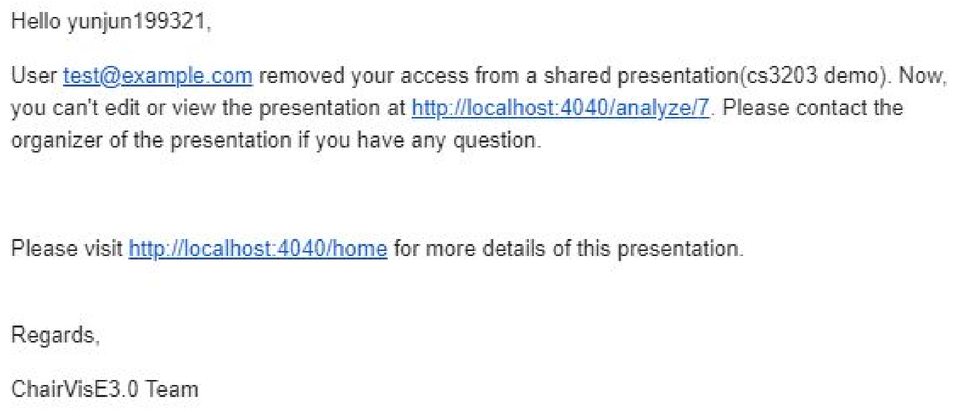
User test@example.com modified your access level from a shared presentation(cs3203 demo). Now, you can edit this presentation at http://localhost:4040/analyze/7

Please visit http://localhost:4040/home for more details of this presentation.

Regards,

ChairVisE3.0 Team

Figure 36: Sample of the Notification E-mail - Modification of Access Level



Hello yunjun199321,

User test@example.com removed your access from a shared presentation(cs3203 demo). Now, you can't edit or view the presentation at http://localhost:4040/analyze/7. Please contact the organizer of the presentation if you have any question.

Please visit http://localhost:4040/home for more details of this presentation.

Regards,

ChairVisE3.0 Team

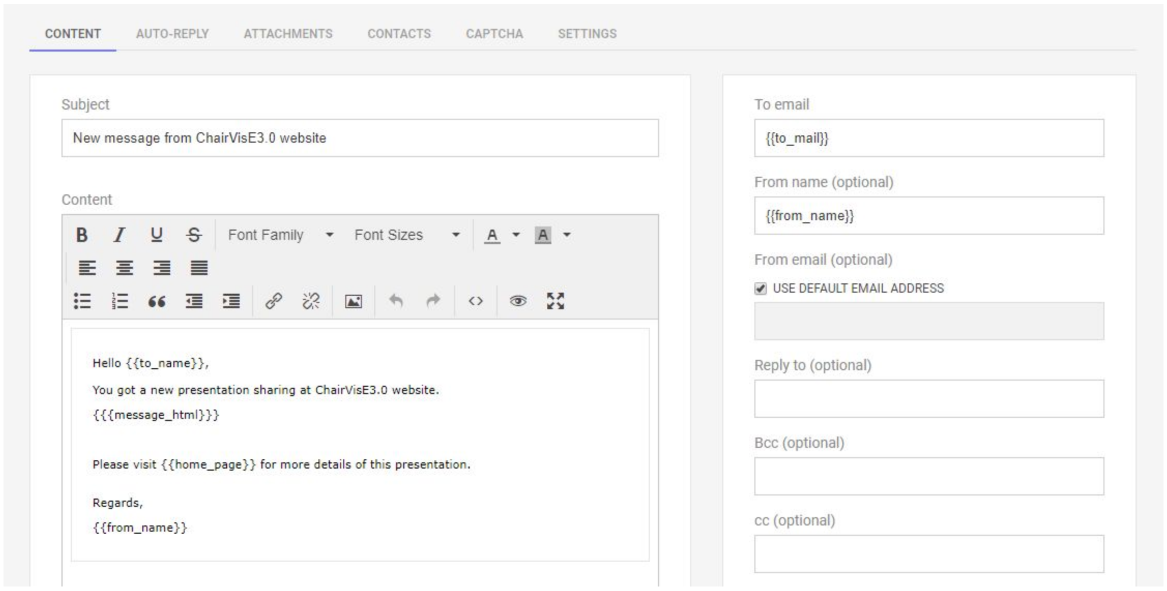
Figure 37: Sample of the Notification E-mail - Removal of Access

Figure 36 shows that the Program Chair modified the access control level of the user. Figure 37 shows that the Program Chair removed the user's access to the presentation.

The Program Chair now does not need to copy the shareable link and send it through third party applications, a security vulnerability in itself, because ChairVisE3.0 can generate a notification e-mail and send it directly to the specific e-mail account. It is a much more user friendly design feature compared to ChairVisE2.0. This way the user who is collaborating with the Program Chair will have the latest URL for the presentation.

Implementation Details of E-mail Notification

This feature used the third-party library, Emailjs. E-mails can be sent directly through the use of JavaScript. No server code is required needed, all that is needed is registration and the setting up of e-mail templates. For ChairVisE3.0, we created two templates under the 'Email Templates' tab on the control panel of the Emailjs website. Figure 38 shows the e-mail template when a new access control is added to a specific e-mail while Figure 39 shows the e-mail template for modifying and removing access control level of existing e-mails.



Subject: New message from ChairVisE3.0 website

Content:

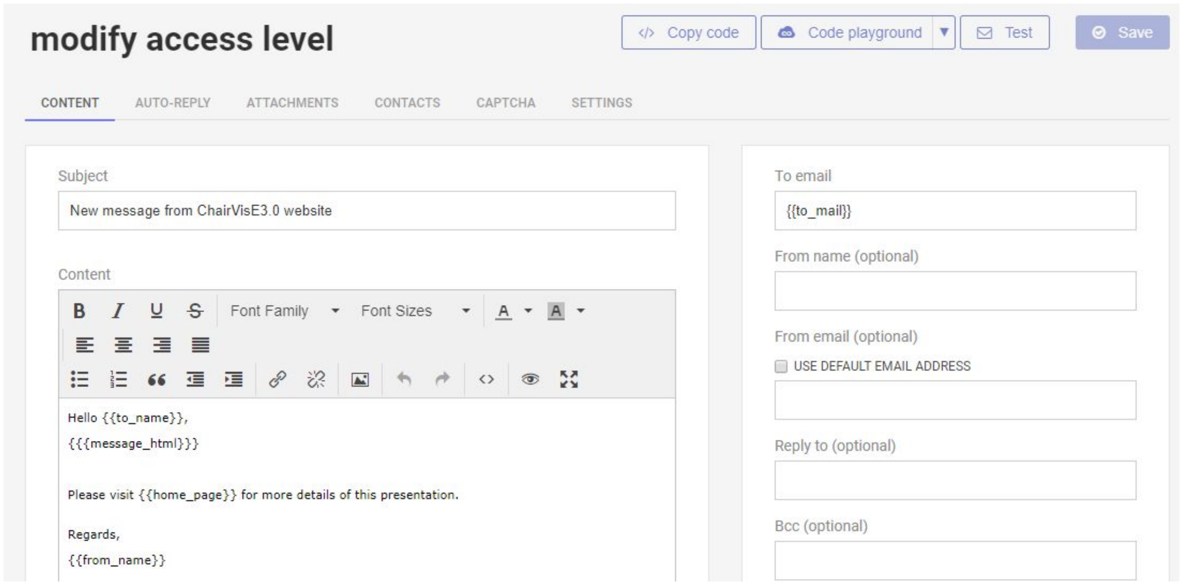
Hello {{to_name}},
You got a new presentation sharing at ChairVisE3.0 website.
{{{message_html}}}

Please visit {{home_page}} for more details of this presentation.

Regards,
{{from_name}}

To email: {{to_mail}}
From name (optional): {{from_name}}
From email (optional): ☑ USE DEFAULT EMAIL ADDRESS
Reply to (optional):
Bcc (optional):
cc (optional):

Figure 38: Create Access Control E-mail Template



Subject: New message from ChairVisE3.0 website

Content:

Hello {{to_name}},
{{{message_html}}}

Please visit {{home_page}} for more details of this presentation.

Regards,
{{from_name}}

To email: {{to_mail}}
From name (optional):
From email (optional): ☐ USE DEFAULT EMAIL ADDRESS
Reply to (optional):
Bcc (optional):

Figure 39: Modify and Remove Access Level E-mail Template

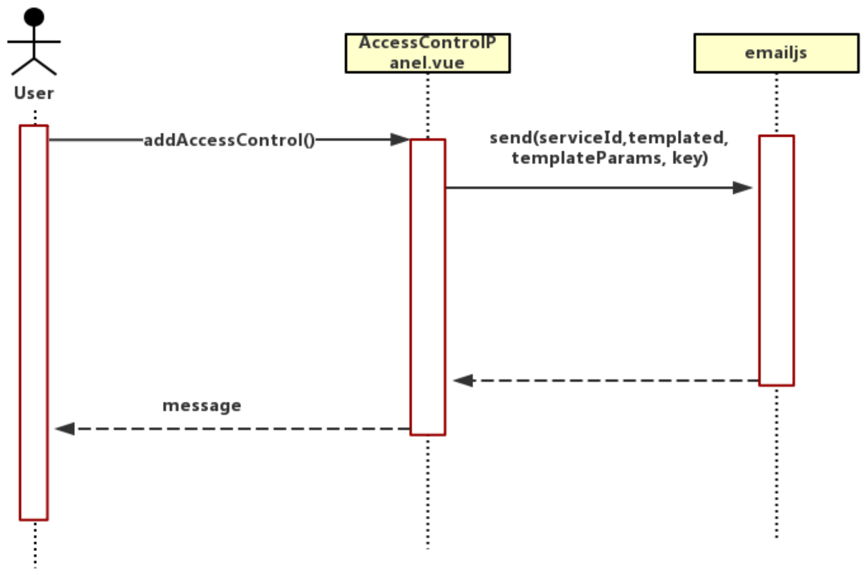
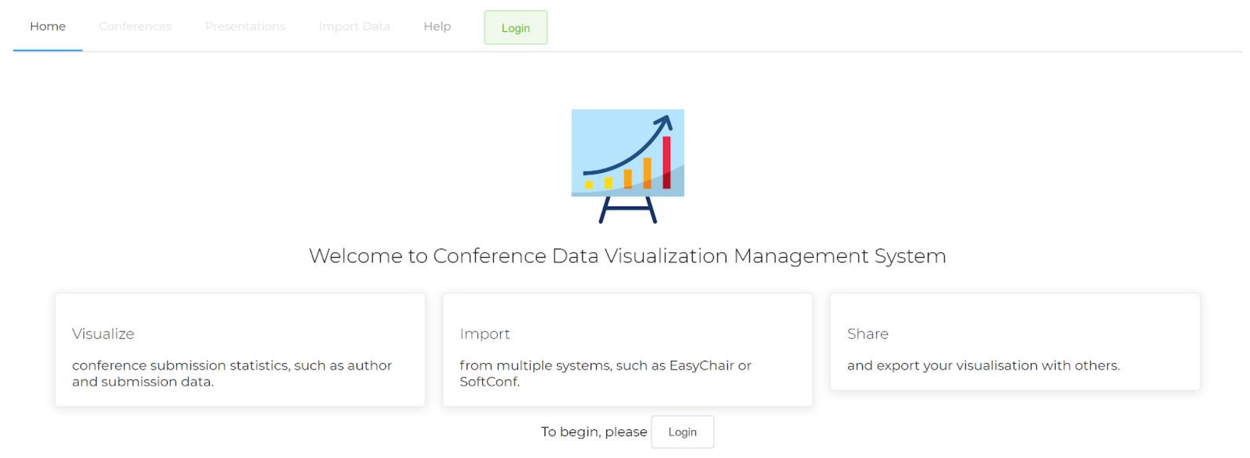


Figure 40: Sequential Diagram Illustrating the Sending of E-mail Notification

Figure 40 is the sequence diagram that illustrates the sending of E-mail notifications. When the user clicks the add button on `AccessControlPanel` Vue page. The `addAccessControl` method is called. The information of the presentation and organizer in the store object are wrapped into the `templateParams`. The send method in Emailjs is invoked with `serviceId`, `templated`, `templateParams` and `key`. Emailjs uses all these information to generate an e-mail to be sent to the recipient.

User Interface Enhancements

The first user interface enhancement was to create a dedicated **landing page** for ChairVisE, and move the current user guide to a dedicated help page. This allows new users to quickly understand what features ChairVizE provides, why they should use it, and how to start (by logging in).



Welcome to Conference Data Visualization Management System

Visualize

conference submission statistics, such as author and submission data.

Import

from multiple systems, such as EasyChair or SoftConf.

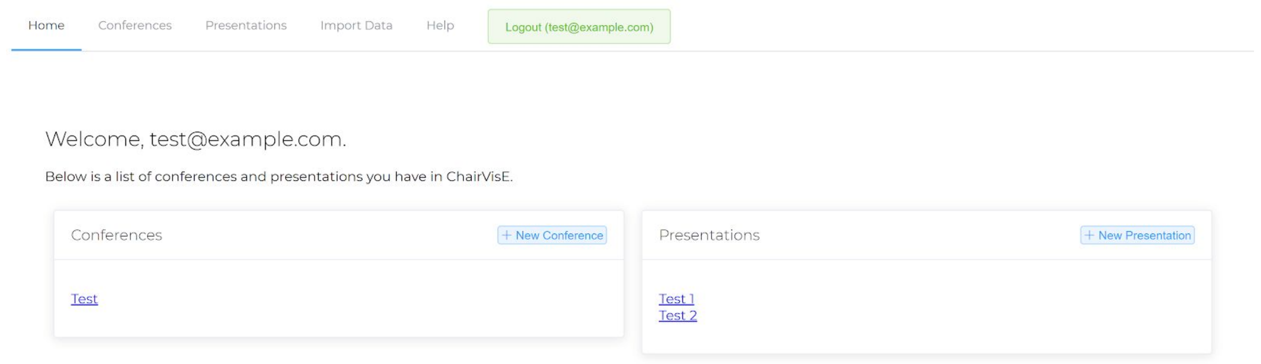
Share

and export your visualisation with others.

To begin, please Login

Figure 41: New Dedicated Landing Page

The second user interface enhancement is to provide a **dashboard** when a user is logged in as their home page. The dashboard shows them the conferences and presentations that they have created and navigate to them with one click, as well as to create new ones quickly. This reduces the effort needed to create new conferences and presentations, as well as to see all of their conferences and presentations at a glance.



Welcome, test@example.com.

Below is a list of conferences and presentations you have in ChairVisE.

Conferences + New Conference

Test

Presentations + New Presentation

Test 1
Test 2

Figure 42: Dashboard After Logging In

Finally, several small enhancements were also done.

- The Analyze tab has been renamed to Presentations, to better match the name of Conferences, as well as make it clearer to the user where to find presentations.

- Error messages have been added and shown when preconditions were not met eg. importing data before a conference is added. They clearly state what the problem is, and what should be done about it. This allows the user to use ChairVisE more effectively.

Future Improvements and Enhancements

Although ChairVisE v3.0 has provided a significant improvement over v2.0, there are still many possible enhancements that can be considered for a future iteration.

- **Compatibility with other conference management systems**

  The current version of ChairVisE only supports EasyChair and SoftConf. While these are some of the more widely used conference management systems, support for other conference management systems should be considered to be able to serve more users.

- **Asynchronous data importing via backend**

  ChairVisE pauses while conference data is being imported. If the conference data is sufficiently large, the user may be left waiting a long time before they can continue using ChairVisE. The import may also fail if the user closes the tab during the import process.

  A possible enhancement is to move the import process from the frontend. The user will upload their data, but can continue using ChairViseE while the backend processes the data. The user can then be notified (eg. via e-mail) when the import is complete. This will enhance the usability of our app, as well as improve user productivity.

- **Showing only possible mappings while importing data**

  ChairVisE allows using any mapping for any data imported, even those that may not be meaningful given the data currently being imported. For example, using an author mapping for a submission dataset may not yield meaningful data.

  ChairVisE should only show the mappings that are possible for a given data set, and not allow the user to use mappings that are not compatible with the current data set being imported.

- **Showing only possible visualizations given the type of data imported when creating presentations**

  ChairVisE allows adding any visualizations to a presentation, even those that may not be meaningful given the data currently imported. For example, adding an author visualisation to a conference without author data would show no results.

ChairVisE should only show the visualisations that are possible for a given data set, and not allow the user to create visualisations that they do not have data for.

- **Addition of more information to the Conference entity or creating new entities that can be associated with it**

  Currently the Conference entity only has a name and year field as a part of it. Other important information associated with this entity can be added after consultation with program committee members about their requirements. Furthermore, other new entities can be created and associated with this Conference as its “parent”. For example, a trivial hypothetical attendance entity could be created that tracks attendance for each day of the conference and this can be associated with a conference using a join column on the conference’s ID.

  Interesting visualizations and functionality that go beyond being related simply to the papers submitted can be generated thus adding further value to the overall system.

- **Support Multiple Types of E-mail Accounts**

  Currently the ChairVisE only supports login using a Google account. As such, the system is restricted to only creating shareable links between different Google accounts. In the future, to accommodate a greater number of users who may use different e-mail provider, we will try to support logins with e-mail addresses from different providers such as Hotmail, Yahoo! Mail, and Outlook. This would allow the user to share presentations for collaboration with a wider number of people, not just those who use Google.

Visualization Screenshots

The following are some sample visualization screenshots from ChairVisE3.0. Please note that this does not cover the full range of visualizations available on the system.

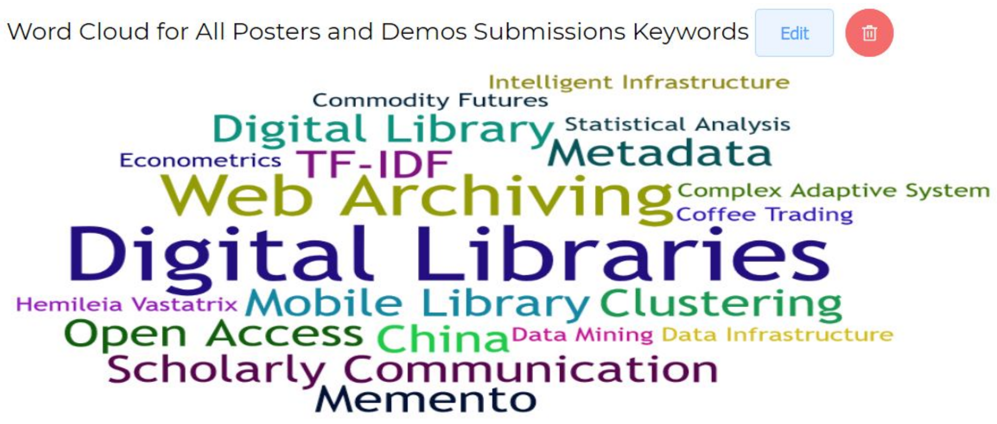


Figure 43: Word Cloud for All Posters and Demos Submissions Keywords

The above word cloud visualization in Figure xx shows a list of keywords found under the abstract section for all the submitted papers in Posters and Demos Track.

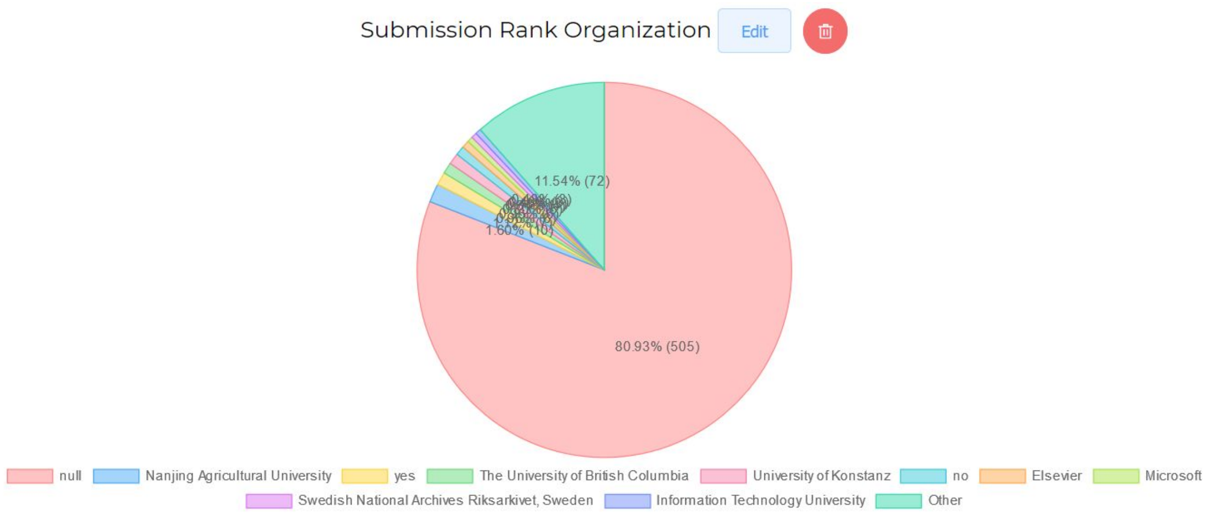


Figure 44: Submission Rank of Organizations

The above pie chart visualization shows the percentage and number of papers submitted from each organization, mainly showing which organization has more submissions than other organizations. The category 'Others' accounts for the total of all organizations whose individual percentages aren't significant enough to be included individually on the chart.

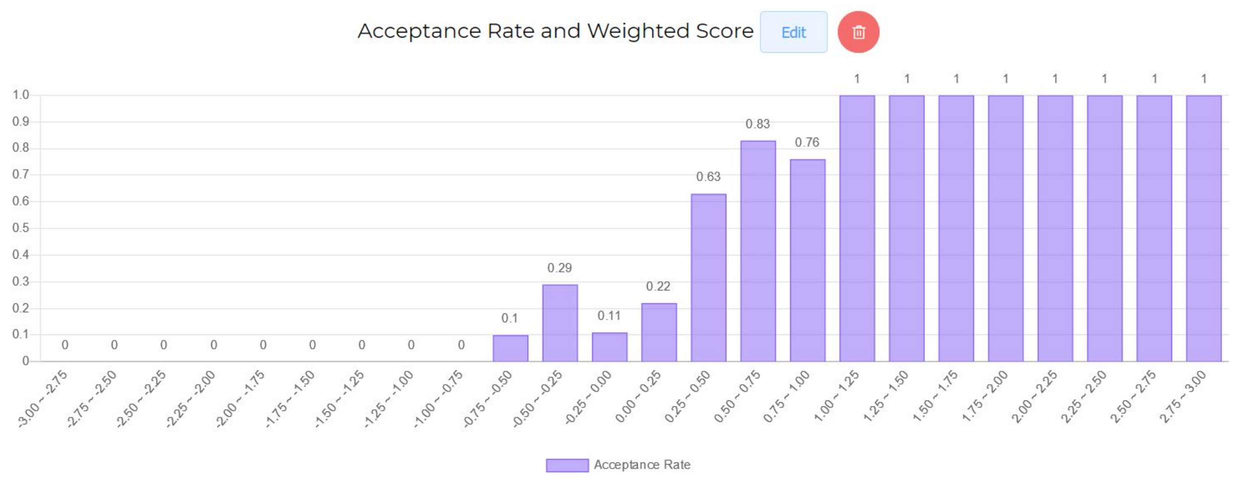


Figure 45: Acceptance Rate against Weighted Score

The above bar graph combines review and submission, showing the percentage of acceptance rate of based on the weighted score of the submissions. This thus gives an insight to what percentage of the papers will be accepted given a certain review score.

Code Implementation

This is a step-by-step guide for setting up a development environment on your local machine. Using this environment, you can contribute to the project by working on features, enhancements, bug fixes, etc.

Getting Started

To get started, you should have installed the latest version of IntelliJ on your local machine. Then please follow the steps.

Get a Copy

Clone the repo to your local machine.

Environment Setup

- Config your Google Cloud SDK. You can find the instructions at https://cloud.google.com/sdk/.
    - Prepare for app engine deployment under project root directory in terminal:
      `$ gcloud -q components install app-engine-java`
- Install npm. We suggest using NVM, which stands for Node Version Manager. You can follow instructions at https://www.npmjs.com/get-npm.
- Configure the connection to local database MySQL:

- Navigate to *src/main/resources/application-local.properties* for SQL server connection setup.
- Put the correct information to connect to the local database: `spring.datasource.username=YOUR_DATABASE_NAME` `spring.datasource.password=YOUR_PASSWORD`
  - If you are facing TIME ZONE issues, please do the following:
    - In `application.local.properties`, modify the `spring.datasource.url` value and add parameter `serverTimezone=UTC` as seen in this example - `spring.datasource.url=jdbc:mysql://127.0.0.1I3306/viz?useSSL=false&serverTimezone=UTC`
- Navigate to *src/main/webapp/WEB-INF/appengine-web.xml* and change the number of manual scaling instances to 1.

- Run application backend:
  - Go to the terminal, run the application by issuing cmd

    ```
    $ ./gradlew appengineRun (For non-Windows)
    $ gradlew appengineRun (For Windows)
    ```

  - Access the application through http://localhost:8080/_ah/admin.
- Run application frontend:
  - Navigate to *src/web/app*.
  - Install the dependency using "`npm install`"
    - If you are facing a checksum issue, please remove package-lock.json.
    - If you are facing other issues, try to first remove the *node_modules* folder first and redo `npm install`. If this still fails, try to clear the npm cache first by using the command `npm cache clean --force`.
  - Run "`npm run serve`".
  - Access the frontend application at http://localhost:4040.

Test and Deploy

Test

Here are two main components in this project's testing stage; front-end testing and backend testing.

- Frontend tests:
  - Make sure that npm is properly installed.
  - Run the following commands under root directory:
    - `$ cd src/web/app`
    - `$ npm run test:unit`
- Backend tests:

- Run the following commands under root directory:
- `$ ./gradlew check`

Deploy

You have two ways of deploying: locally and through continuous deployment.

- Local deployment:
  - Make sure that `npm` is properly installed on your local machine. You can verify it by calling `$ npm -v`.
  - Make sure that Google Cloud SDK is properly installed on your local machine. You can verify it by calling `$ gcloud --version`.
  - Authorize your Google Cloud SDK locally through `$ gcloud auth login`.
  - Make sure that application-prod.properties contains the correct configurations.
    - Modify the template with corresponding information.
  - Build the frontend for production with the following commands:
    - `$ cd src/web/app`
    - `$ npm run build`
  - Go to project root directory and run `$ ./gradlew appengineDeploy`.
- Continuous deployment:
  - Every time the master branch is updated, the new stable version will be automatically deployed to Google Cloud.