



CS3219: Software Engineering Principles and Patterns

AY 2019/2020 Semester 1

Submission of ChairVisE 3.0 Report

Date of Submission

13 November 2019

Submitted By

Team 01	
Name	Matriculation Number
ANG ZHI KAI	A0169766H
DARREN ONG YUN KAI	A0169721Y
MUHAMMAD ZULQARNAIN BIN RAMLAN	A0168710E
LEE WEI HAO JONATHAN	A0166980R

Code Repository URL:

<https://github.com/CS3219-SE-Principles-and-Patterns/chairvise3-0-2019-team-01>

Table of Contents

1. Introduction	4
1.1 Purpose	4
1.2 Project Scope	4
1.3 Individual contributions to the Project	5
2. ChairVisE 2.0	7
2.1 Existing Features	7
2.2 Existing Design	8
2.2.1 Overall Architecture	8
2.2.2 General Sequence Diagram	9
2.2.3 Backend Architecture	10
2.2.3.1 UI component	11
2.2.3.2 Logic Component	13
2.2.3.2 Storage Component	14
2.2.3.3 Common Component	15
2.3 Frontend Architecture	16
3. ChairVisE 3.0 Developer Documentation	19
3.1 User Stories	19
3.2 Functional Requirements	20
3.3 Non Functional Requirements	20
3.3.1 Software Quality Attributes	20
Usability	20
Interoperability	20
System Constraints	21
3.4 Development Process	21
3.4.1 Agile Development Process Model	21
3.4.2 GitHub Issue Tracker	21
3.4.3 Workflow	21
3.5 Development Schedule	22
3.6 Design Decisions	23
3.6.1 Revamp of User Interface	23
3.6.2 Persistency of Records Uploaded	23
3.6.3 Co-authorship Queries	25
3.6.4 Adding Mail Support in Backend	27
3.7 Design and Implementation Constraints	28
3.7.1 Design constraints in Revamp of UI	28
3.7.2 Design constraints in Persistency of Records Uploaded	28

3.7.3 Design constraints in Adding Mail Support	28
3.8 Design Diagrams	30
3.8.1 Overall Architecture	30
3.8.2 General Sequence Diagram	31
3.8.3 Send Mail Backend Sequence Diagram	32
3.8.4 Backend Architecture	33
3.8.4.1 UI component	33
3.8.4.2 Logic Component	36
3.8.4.3 Storage Component	37
3.8.4.4 Common Component	38
3.9 Frontend Architecture	39
3.9 Setting up ChairVisE 3.0	42
3.10 Suggestions for improvements and enhancements	43
3.11 Visualisation Screenshots of ChairVisE 3.0	44
3.11.1 Mail Support	44
3.11.2 Persistency of Records Uploaded	45
3.11.3 Co-authorship Queries	46

1. Introduction

1.1 Purpose

The purpose of ChairVisE is to create a web application to enable conference program chairs to visualize and share the conference submission statistics. This can be achieved in two methods, namely (i) through parsing all the information for each of the submission that are undergoing review, or (ii) by using the metadata available through the conference management system (CMS). For the development of ChairVisE, method (ii) will be used.

The web application is expected to accept one or more CSV file(s) which contains metadata such as author, title, and time of submission organized as one row per submission. With these metadata, meaningful visualizations can be crafted and presented. The web application also supports features such as user authentication and persistent data storage.

1.2 Project Scope

The target audience of ChairVisE web application is conference program chairs.

1.3 Individual contributions to the Project

Contribution Chart (Technical & Non-Technical)	
ANG ZHI KAI (A0169766H)	
<u>Technical contributions</u>	<ul style="list-style-type: none">• Studied the use of Vuex and Element library, and provided assistance to team members wherever required for frontend development.• Revamp UI by standardizing usage of buttons and action dialogues.• Assisted in creation of UI for record group.
<u>Non-Technical contributions</u>	<ul style="list-style-type: none">• Acted on behalf of the team lead to ensure that sprints are completed on time.
DARREN ONG YUN KAI (A0169721Y)	
<u>Technical contributions</u>	<ul style="list-style-type: none">• Add new APIs for Record Group creation, reading, update and deletion• Create a UI for users to create, read, update and delete record group• Update the current record and presentation repository to include record group
<u>Non-Technical contributions</u>	<ul style="list-style-type: none">• Lead the team in project meetings and discussions.• Decide the main direction of how the project should be worked through.
MUHAMMAD ZULQARNAIN BIN RAMLAN (A0168710E)	
<u>Technical contributions</u>	<ul style="list-style-type: none">• Implemented co-authorship algorithm in backend• Added New API to handle co-authorship query in backend• Appended co-authorship query in UI frontend
<u>Non-Technical contributions</u>	<ul style="list-style-type: none">• Ensures that the team is well supported
LEE WEI HAO JONATHAN (A0166980R)	
<u>Technical contributions</u>	<ul style="list-style-type: none">• Added sending mail via SMTP support in backend.• Added sending of generated pdf via mail support in both frontend and backend for users to backup their presentation in pdf format.• Added sending of mail notification when a presentation is shared by presentation owner to another user to notify the user about the presentation being shared to

him/her such as the title and link to the shared presentation.

Non Technical contributions

- The updating of existing diagrams and descriptions in ChairVisE 2.0 for Week 6 presentation after analysing some minor errors and possible additions such as the association with gatekeeper in the Logic diagram, the frontend store component diagram description and the storage component diagram.
- The updating of the Setting up Section such as configuring the connection to SMTP server.
- The addition of Mail Support to presentation slides.
- Acting on behalf of the team as a point of contact with the lecturer/tutors.
- The updating of existing diagrams for ChairVisE 3.0 to match the addition of mail support in the backend.
- The addition of development process, design and implementation constraints send mail API sections, mail user stories section, development schedule, design decision section with adding mail support in backend design decision, development schedule section and suggestions for improvements and enhancements.
- Added Send Mail Backend Sequence diagram to showcase the interactions in the backend when a send mail API request is received from the frontend.
- Added design decisions section and adding mail support in backend design decisions with explanation.
- The updating of Setting up section which includes additional information including links on the usage of Google's gmail smtp server as additional steps are required for Google's gmail smtp server such as enabling less secure app access in the Google account or creating an App Password in the Google account with 2 step authentication enabled.

2. ChairVisE 2.0

2.1 Existing Features

1. The application allows users to upload csv file containing metadata from Conference Management System (EasyChair and SoftConf).
2. The application allows users to create multiple presentations based on the mix-and-match of data options provided by the system.
3. The application allows users to view various visualizations based on the author, review and submission records.
4. This application uses Google authentication to authenticate and authorize users to perform actions in the application
5. The application allows users to download the visualizations in a Portable Document Format (PDF).
6. This application allows customization of charts based on the user-defined SQL like query.
7. Sharing of presentation link with access permission.

2.2 Existing Design

2.2.1 Overall Architecture

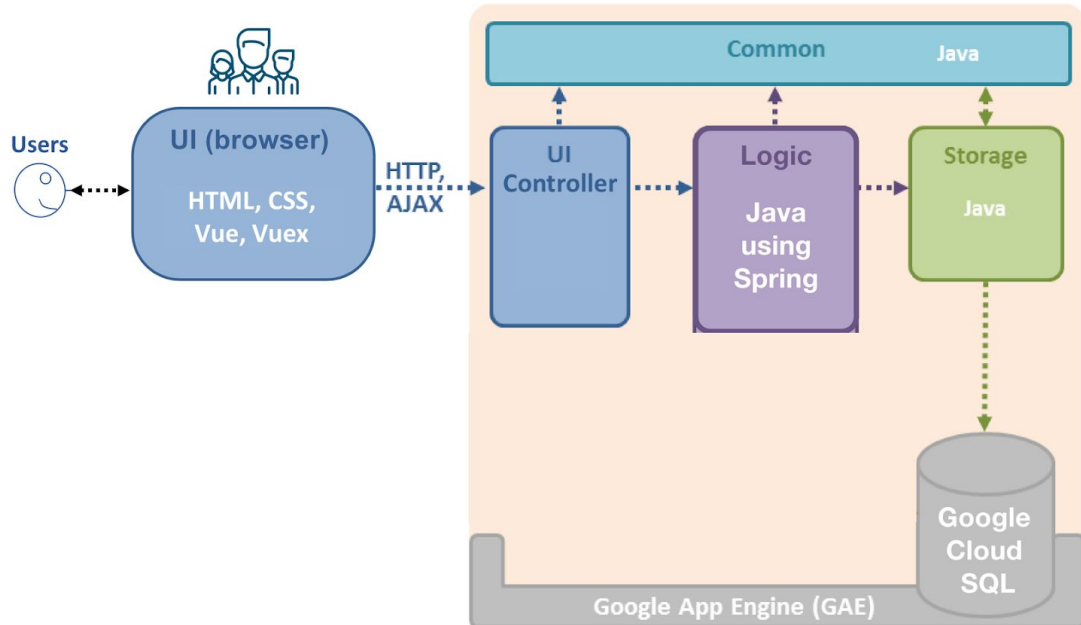


Figure 1: Overall Architecture For ChairVisE 2.0

The application uses the layered architecture design:

- **UI:** The component consists of API controllers and WebPage controllers. API controllers are responsible for handling API calls by the frontend. WebPage controllers are responsible for serving static production Vue files.
- **Logic:** The main logic of the application is in Java using the Spring framework.
- **Storage:** The storage layer of the application uses the persistence framework provided by Google App Engine, using MySQL 5.6.
- **Common:** The Common component contains utility code (data transfer objects, helper classes, etc.) used across the application.

2.2.2 General Sequence Diagram

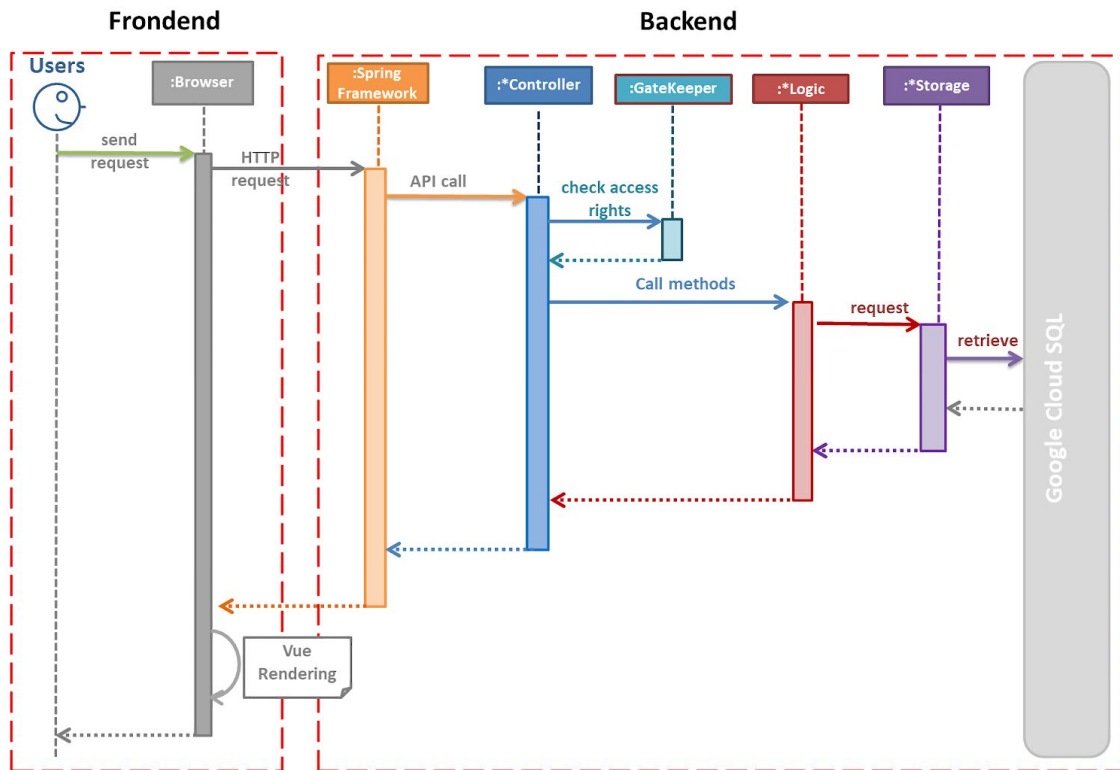


Figure 2: General Sequence Diagram For ChairVisE 2.0

The diagram above showcases the workflow in the system upon receiving a request from the user:

1. A user sends a request to the browser for certain data or visualisation.
2. The browser sends an HTTP request to the backend. Spring framework will handle the HTTP request and pass it to the correct controller based on URL mapping.
3. The controller checks the access rights by interacting with the GateKeeper.
4. The controller then calls the respective methods in the appropriate logic.
5. The logic processes and requests the data from the storage component.
6. The storage retrieves the data from the Google Cloud SQL (MySQL)
7. The response will be sent back to the browser where it will be used by Vue to render the web page.

2.2.3 Backend Architecture

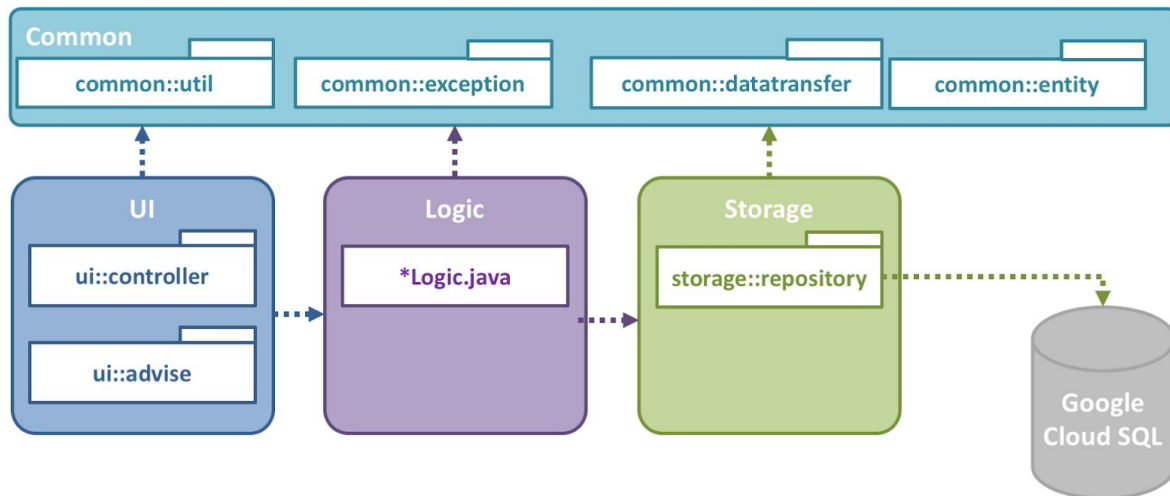


Figure 3: Backend Package Overview For ChairVisE 2.0

The above figure illustrates the backend package overview of the existing system.

2.2.3.1 UI component

All requests from the user browser are received and handled by the UI component in the backend of the web application.

- **ui.controller.api:** Provides backend Representational State Transfer (REST) API access to the users.
- **ui.controller.data:** Contains helper objects to be sent to the client in JSON.
- **ui.controller.webpage:** Handles static file requests for the users.
- **ui.advice:** Handles exception thrown by the application.

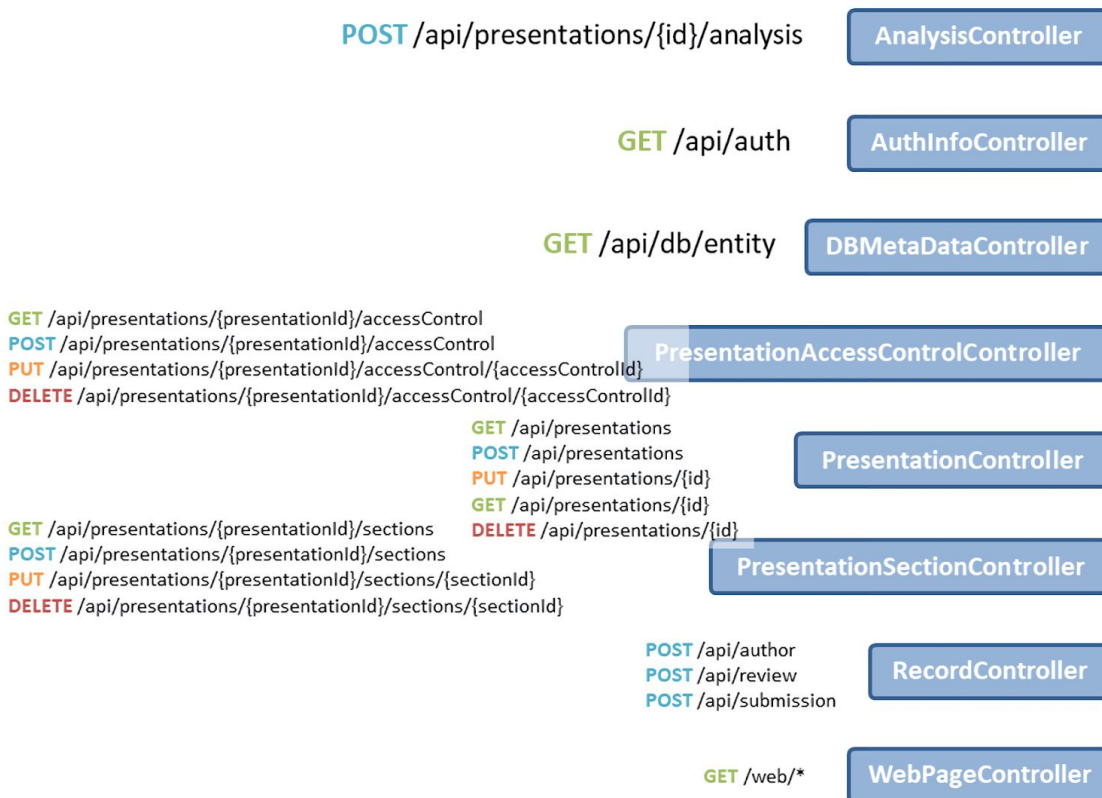


Figure 4: API requests to controller mapping For ChairVise 2.0

The above shows the mapping of the API requests to the appropriate controller. GET is to retrieve data. POST is to create data. PUT is to update data. DELETE is used to delete data.

- **AnalysisController** is used to handle the analysis request sent by the frontend and issue SQL aggregation query to the backend.
- **AuthInfoController** is used to check the current authentication status of user. For example, it will give login URL if the user is not logged in and logout URL if the user is logged in.
- **DBMetaDataController** exposes the metadata, including name and type of the standard data template used in the application. This is useful when users try to match their own CSV file to the data template used in the application.
- **PresentationController**, **PresentationSectionController** and **PresentationAccessControlController** are responsible for the CRUD operations.
- **RecordController** accepts the CSV data imported by users and stores them in the database.
- **WebPageController** serves the static production files built by Vue.

2.2.3.2 Logic Component

The Logic component handles the business logic. In particular, it is responsible for:

- Managing CRUD operations, ensuring the integrity of data.
- Providing a mechanism for checking access control rights.

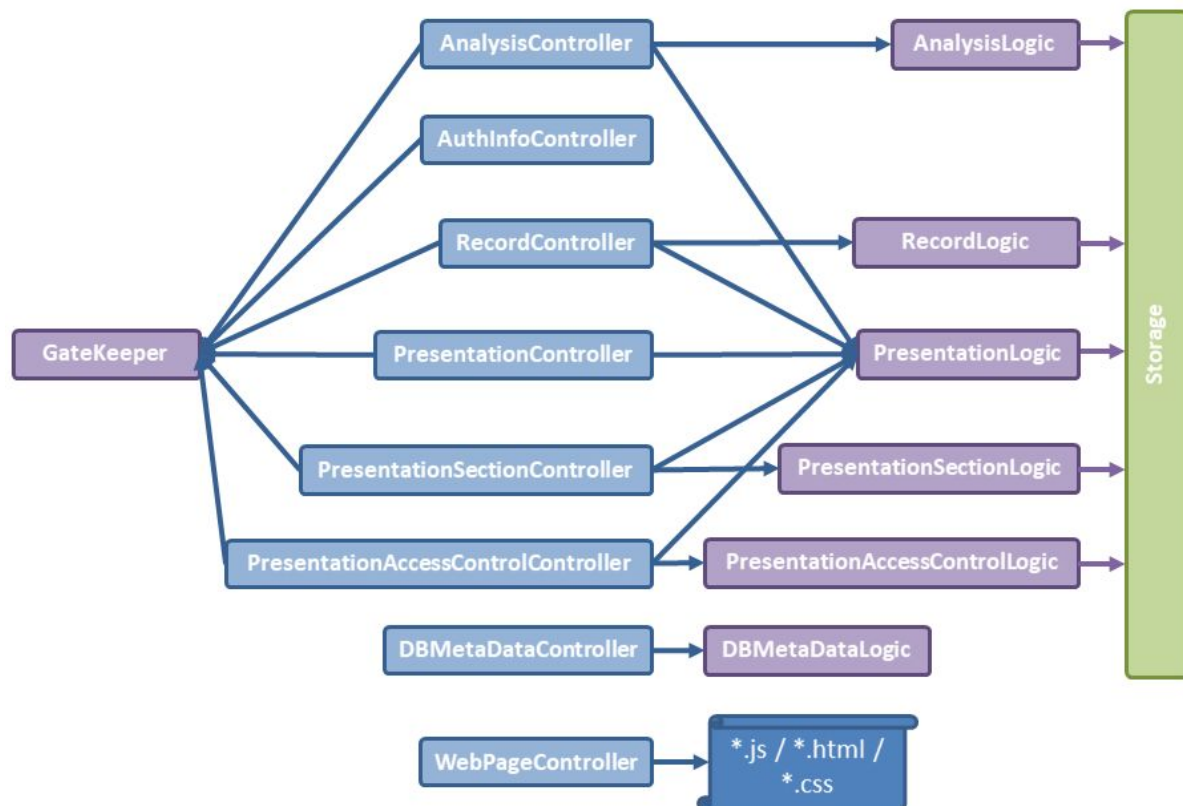


Figure 5: Backend Logic Storage For ChairVisE 2.0

The above figure illustrates how each controller uses its corresponding logic. The dependency is injected (Dependency Injection) by Spring framework automatically.

2.2.3.2 Storage Component

The Storage component performs create, read, update and delete (CRUD) operations on data entities individually. It contains minimal logic beyond what is directly relevant to CRUD operations.

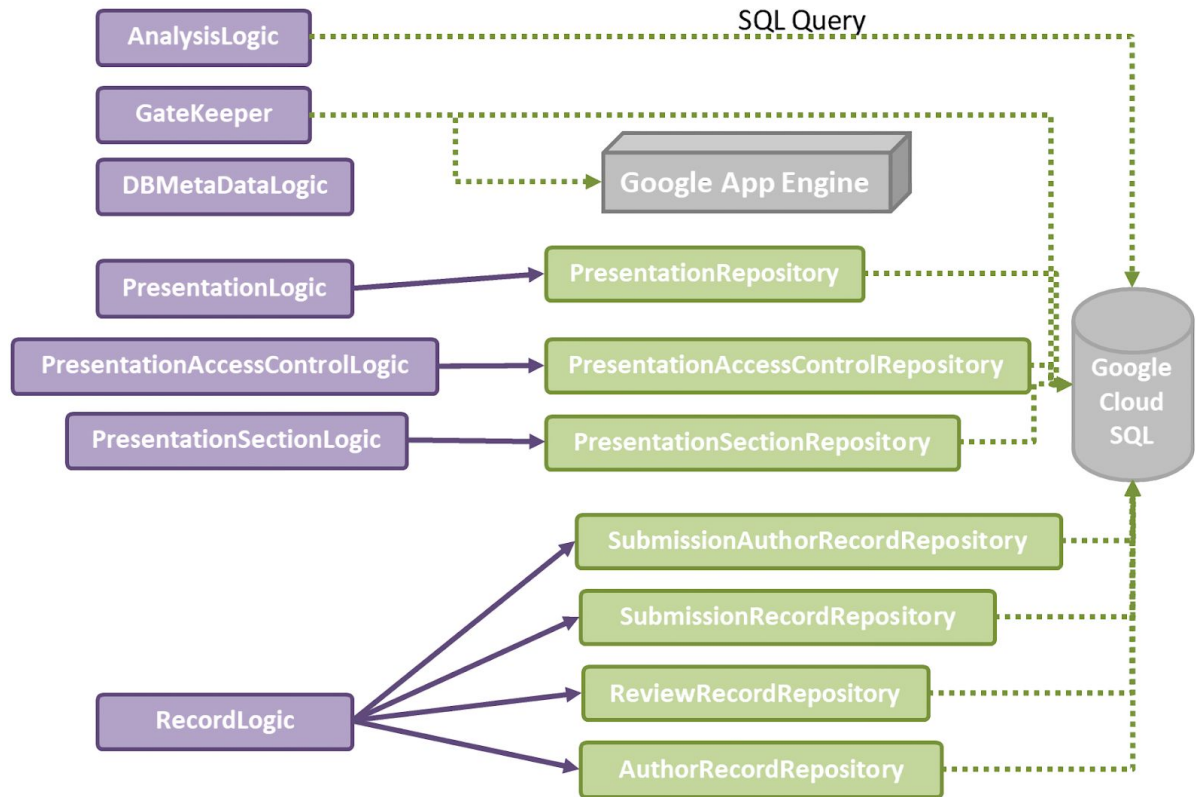


Figure 6: Backend Storage Overview For ChairVisE 2.0

The above illustrates how logic make use of the repository in the storage component.

- **AnalysisLogic** will issue SQL query directly to the Database to perform analysis query.
- **GateKeeper** will use GAE internal APIs to check logged in user and also issue queries to the database to check the access rights.
- ***RecordRepository** are responsible for persisting user uploaded CSV data to the database.
- **Presentation*Repository** are responsible for storing user generated presentation, access control list and sections.

2.2.3.3 Common Component

The Common component contains common utilities used across the web application.

- **common.util:** Contains utility classes.
- **common.exceptions:** Contains custom exceptions.
- **common.datatransfer:** Contains data transfer objects (DTOs).
- **common.entity:** Contains entity stored in the database.

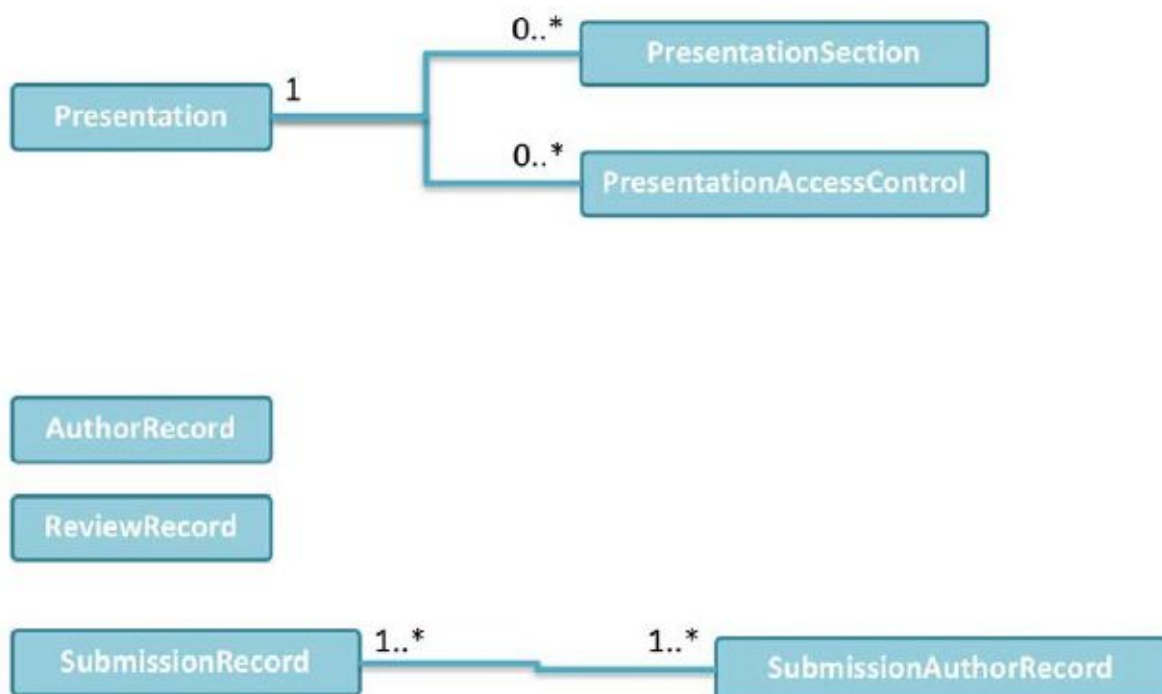


Figure 7: Backend Entities Overview For ChairVisE 2.0

The above illustrates the entity relationship. Each entity has its corresponding table in the relational database.

- One presentation can have multiple sections or access controls.
- ***Record** is the data imported by users. There is a many-to-many relationships between **SubmissionRecord** and **SubmissionAuthorRecord** because one submission can be contributed by multiple authors. Each author can also contribute to multiple submissions.

2.3 Frontend Architecture

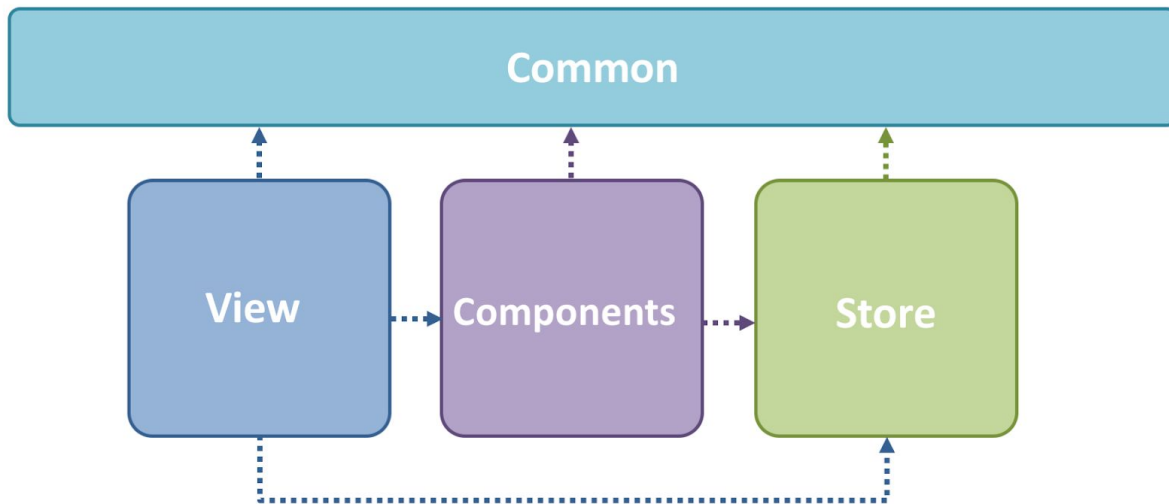


Figure 8: Frontend Package Overview For ChairVisE 2.0

The above figure illustrates the frontend package overview of the application.

- **view:** The view is mainly in charge of displaying pages of the application.
- **component:** The component contains reusable UI and display logic components which is called by multiple pages.
- **store:** The store contains core logic of the application. It shows the state of the application and handles mutation and action to interact within the application using Vuex.
- **common:** Contains utility function as well as constants used.

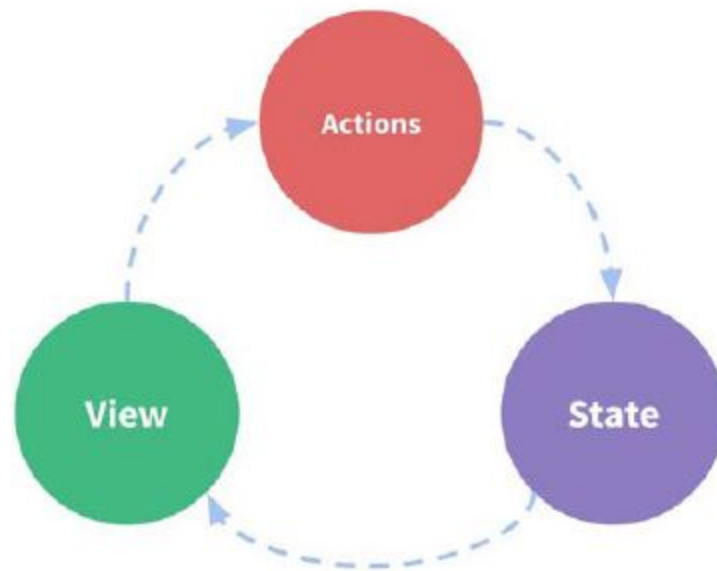


Figure 9: Source: Vuex Documentation

The above illustrates that the view (components) will fire actions and the actions change the state. After that, as view is subscribed to the state (Observable Pattern), the UI (view) will be notified and updated. In doing this, the state becomes the single point of truth in the whole application, which increases the maintainability a lot and make it is easy to trace the change of UI.

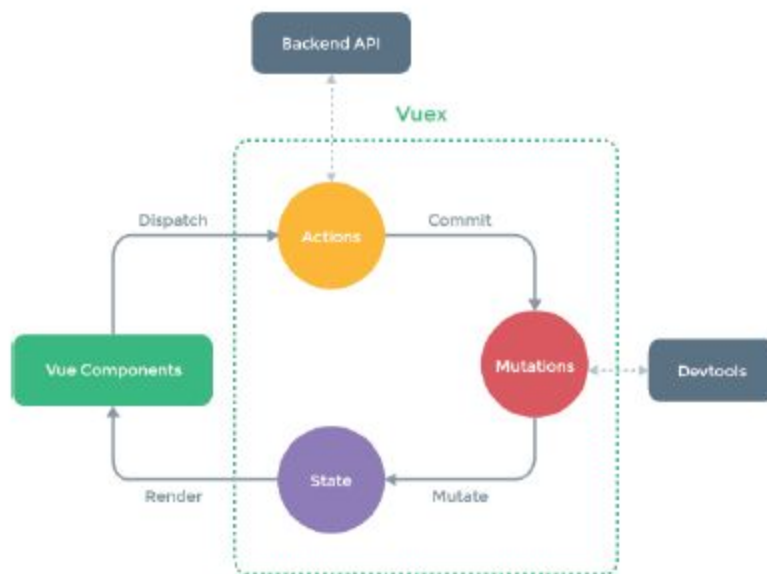


Figure 10: Source: Vuex Documentation

The above illustrates a more detailed. Typically, a vue component will dispatch an action. The action will communicate with backend API to do CRUD operations. After that, it will commit a mutation and the mutation will change the state. The change of state will be propagated to the component and component will render based on the data in the state. A devtool (e.g. Vuex.js devtools) can be used to monitor the state for easy debugging.

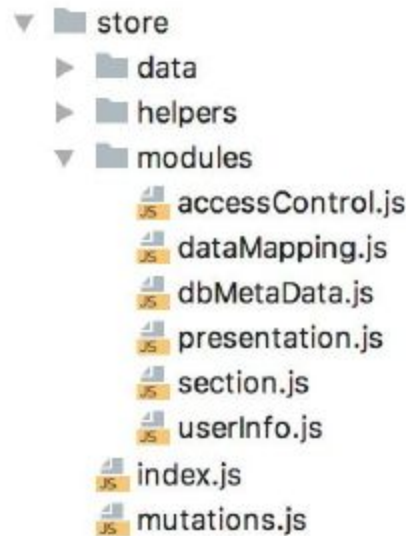


Figure 11: Frontend Store component for ChairVisE 2.0

The following modules are located in the store component:

- **accessControl.js** is responsible for handling logic related to the access control of a presentation.
- **dataMapping.js** is responsible for handling logic related to the mapping of CSV and built-in data schema.
- **presentation.js** is responsible for handling logic related to the presentation.
- **section.js** is responsible for handling logic related to the presentation section.
- **userInfo.js** is responsible for handling logic related to the authentication.

In summary, with the help of Vuex, the overall architecture of the frontend ensures that the state can only be mutated predictably.

3. ChairVisE 3.0 Developer Documentation

In ChairVisE 3.0, we decided to improve on and enhance the existing features and add new features which we believe would help improve the usability of the existing application which is one of quality attributes we are focusing on for ChairVisE 3.0.

3.1 User Stories

User Type/Role	Function	Benefit
As an organiser	I can receive email notification	So that I can be notified, if another organiser have shared his/her presentation to me. Similar to the sharing feature in Google documents.
As an organiser	I can receive an email with the generated pdf attached	So that I will be able to keep a copy of the generated pdf in my inbox.
As a user	I want my data to be persistent	So that I can keep a few presentations with different set of records.
As a user	I want to group my records uploaded together	So that I can update and identify different sets of records accordingly
As a user	I want to analyse co-authorship data	So that I can see the bigger picture and the network between the data
As a user	I want a confirmation button	Just in case I accidentally misclicked and do not want to actually remove the record.
As a user	I want an intuitive user interface	So that I am able to use the application with ease

Table 1: User Stories For ChairVisE 3.0

3.2 Functional Requirements

The following is the list of functional requirements **in order of highest to lowest priority**.

1. The application should be able to support the persistence of existing conference data.
2. The application should be able to add, view, remove and update a record group that group their author records, review records and submission records together.
3. The application should be able to allow users to choose the particular group of conference data they want to use for the visualisation
4. The application should be able to list the existing conference data uploaded for the group by the user.
5. The application should be able to replace the existing conference data uploaded by the user.
6. The application should be able to display data based on multiple author.
7. The application should be able to handle Co-authorship queries. (Country-Country etc.)
8. The application should support email service via Simple Mail Transfer Protocol (SMTP).
9. The application should allow a user to receive email notification when a presentation is shared to him/her. (Similar to sharing feature in Google document).
10. The user is able to send an email with the generated pdf attached to himself/herself as an attachment for backup purposes.
11. Revamp of UI to make elements more consistent across pages.

3.3 Non Functional Requirements

3.3.1 Software Quality Attributes

Usability

The application should be usable for users with no prior experience in using online conference programs to generate summary reports from existing data.

Interoperability

The application should use the CSV files generated by conference management systems (Easy Chair and SoftConf).

The application should support the 3 main CSV files generated from each conference system which are author.csv, submission.csv and review.csv.

System Constraints

The application should use Vue for frontend and Spring Boot for backend. In addition, it uses the Element library for the frontend and it does not natively support responsive design.

3.4 Development Process

3.4.1 Agile Development Process Model

Our team adopted the Agile Development Process Model specifically the SCRUM development process.

Before we begin working on ChairVisE 3.0, we plan a list of potential improvements and enhancements (Product Backlog). We then assign the potential improvements and enhancements from the list among ourselves based on our technical experience and expertise. After that, we set up three sprints for the development process and split the product backlog into the three sprints. Each sprint duration is 2 weeks. At the end of each sprint, there will be a working increment towards ChairVisE 3.0.

Meetings were conducted every week as needed, and also at the end of the sprint to discuss our progress and any impediments, and whether there is a need to adjust the backlog.

3.4.2 GitHub Issue Tracker

We take advantage of the GitHub Issue Tracker to track our progress and report bugs and add user stories. We also label each issue based on the nature of the issue such as priority level, backend or frontend, enhancement or bug.

3.4.3 Workflow

Our team adopted the Git branching workflow. For each feature that is being implemented, a branch will be created based off the latest master branch. When the feature is completed, a pull request shall be made and it will be subjected to a code review that will be conducted by other team members.

3.5 Development Schedule

Product Backlog		
Sprint 1	Sprint 2	Sprint 3
Modify existing backend logic of records uploading to ensure persistency	Redesign the interface of importing data and analyse components	Integrate the front end with backend logic Fix bugs caused by the updates
Implement algorithm of mapping and calculating the data collaboration in back-end	Optimize co-authorship algorithm, Implement front-end and add new api to handle co-authorship query	Integrate the presentation section front end with backend logic to display the co-authorship query visualization
Investigate the UI elements that requires updating based on ElementUI.	Update UI to be consistent across all pages. Assisted with implementing UI for record group.	
Basic <i>send-mail</i> backend API implementation through SMTP without file attachment support. (Test using Postman)	Improve <i>send-mail</i> backend API implementation with file attachment support (Test using Postman)	Add <i>send-mail</i> API in documentation. Add support for sending mail with generated presentation pdf to the current user's mail inbox in the frontend. Add support for sending mail to the user when a presentation is shared to him/her with the shared presentation link.

Table 2: Development Schedule For ChairVise 3.0

3.6 Design Decisions

3.6.1 Revamp of User Interface

Due to several constraints of the Element library, majority of the UI changes revolves around improving the consistency of the usage and arrangement of buttons, and the use of dialogs. The justification is to prevent any accidental misclicks which may result in a record being deleted from the application. In addition, the elements should be arranged in such a way that is intuitive for non-technical users to be familiar with.

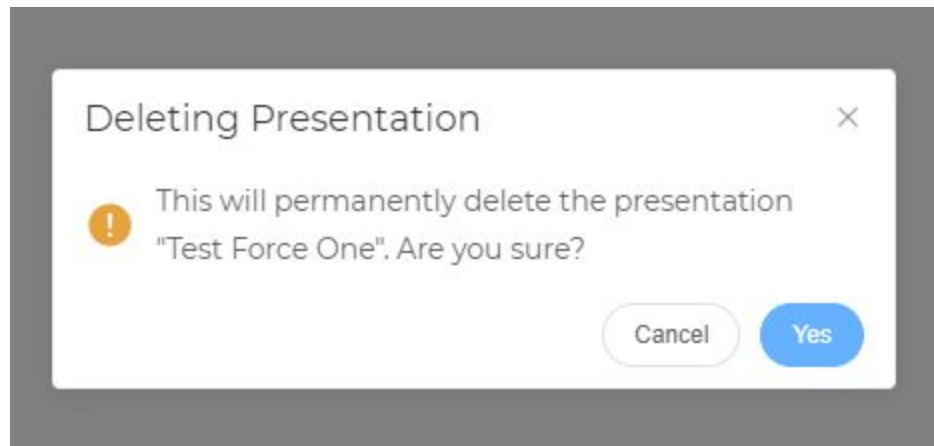


Figure 12: Confirmation before deletion

3.6.2 Persistency of Records Uploaded

The users often want to have multiple presentations using different set of records. However, the current ChairVise only allows one set of records per user. So, for example, whenever the users upload the author records and if there is an existing records for the user, it will be replaced. There are several ways to maintain the persistency of the records uploaded which are:

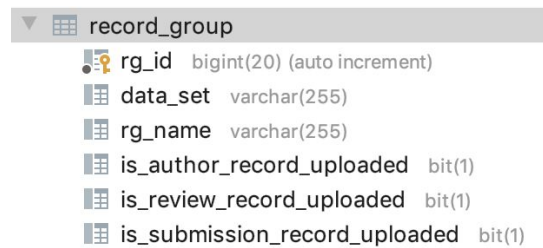
- Add a RecordGroup entity and change the existing entities of records
- Save the CSV files uploaded by the users to local server directory and load the data from it when it is being requested

Option 1 has been chosen as for the following reasons

- **Easier to implement:** Retrieving data from the database is easier and faster without reading the whole CSV file

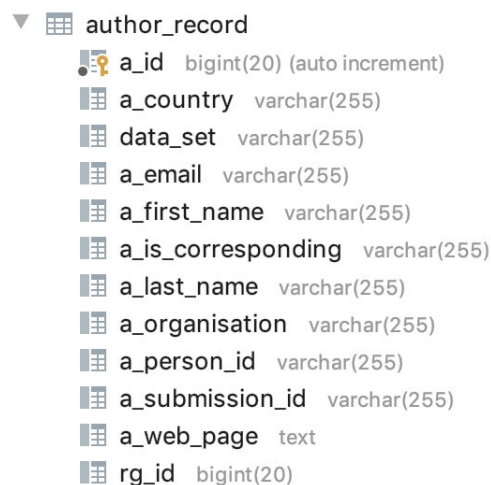
- **Less Memory Used:** Use less local harddisk memory as there is no need to save the file locally
- **More flexible:** Records can be retrieved and analysed from database using SQL

So, a new entity namely **RecordGroup** has been created and the records are being kept tracked with a new column, namely **record_group_id**, which is the unique identifier of the record group. With this new entity, user can now have multiple sets of data without being replaced and the UI for importing has been updated.



record_group	
rg_id	bigint(20) (auto increment)
data_set	varchar(255)
rg_name	varchar(255)
is_author_record_uploaded	bit(1)
is_review_record_uploaded	bit(1)
is_submission_record_uploaded	bit(1)

Figure 13: RecordGroup Database Table



author_record	
a_id	bigint(20) (auto increment)
a_country	varchar(255)
data_set	varchar(255)
a_email	varchar(255)
a_first_name	varchar(255)
a_is_corresponding	varchar(255)
a_last_name	varchar(255)
a_organisation	varchar(255)
a_person_id	varchar(255)
a_submission_id	varchar(255)
a_web_page	text
rg_id	bigint(20)

Figure 14: AuthorRecord Database Table

In order to handle the access the database table of record group, **RecordGroupRepository** has been created for retrieve record groups by user email / dataset and delete the record group by ID.

RecordLogic is now added a few methods to handle the logic of adding new record group, update existing record group, delete a record group, get a list of record groups by user email and get the details of a record group.

So, now **RecordController** has been added with a few new APIs for creating, reading, updating and deleting the record groups. The APIs will be listed as follows:

1. **GET** /api/record/record_groups
2. **POST** /api/record/record_groups
3. **PUT** /api/record/record_groups/{recordGroupId}
4. **GET** /api/record/record_groups/{recordGroupId}
5. **DELETE** /api/record/record_groups/{recordGroupId}

These APIs will be called by the front end to enable the users to add, update, view and remove the record group through the UI provided.

After all the changes made, the users are able to maintain the persistency of records uploaded through this newly added record group.

Thus, this option is chosen for an easier of implementation, reduce the memory space needed and maintain the flexibility of existing record entities.

3.6.3 Co-authorship Queries

Conference submissions often have multiple authors listed for a paper. With this additional information, the system could support queries and visualization on co-author data. This visualization would be beneficial for the user as the bigger picture and the network between the data can be seen. The following queries are implemented:

1. Overview of Country-Country Collaboration (Pie-chart)
2. Overview of Organization-Organization Collaboration (Pie-chart)
3. Top 10 Country Collaboration (Bar-chart)
4. Top 10 Organization Collaboration (Bar-chart)

There are multiple ways to implement the logic of this feature. Based on the current architecture of the system, the logic can be done either through creating new **SQL** queries directly in the **predefinedQueries** or through the back-end in the **AnalysisLogic**. The latter was chosen because of the following reasons:

1. **Extendability** : Able to add in more queries easily
2. **Flexibility**: Able to do more complex logic compared to the **SQL** approach

In the **AnalysisLogic**, a new method called **analyseCoauthor** will be created to specifically analyse co-authorship query. The method will only be using the **AuthorRecordRepository** because it has sufficient data to map the co-authorship. Each author record has a

submission_id that is tied to the unique paper submission. Duplicated **submission_id** in the author record means that the authors are co-authorship between each other.

a_submission_id	a_first_name	a_last_name	a_country	a_organisation
4	Mubxxxxx	Imxxx	Pakistan	Information Technology University
4	Saexxxxx	Haxxxx	Pakistan	Information Technology University
4	Sehxxxx	Iqxxxx	Pakistan	Information Technology University
4	Naxx	Aljxxxxx	United Kingdom	University of Southampton
4	Axx	Daxx	Saudi Arabia	King Abdulaziz University

Figure 15: Example of duplicated submission_id

The logic and algorithm involves the use of 2 data structures:

1. Map<SubmissionId, Set<Data>> **submissionIdToDataSetMap**
2. Map<Collaborators, NoOfCollaboration> **collaborationMap**

Pseudo-code of the algorithm:

```

Foreach of the authorRecord : a
  If submissionIdToDataSetMap contains a
    existingDataSet = submissionIdToDataSetMap.get(a)
    If a.Data does not contain in existingDataSet
      Foreach of the existingDataSet : e
        If collaborationMap contains "a-e"
          Increment noOfCollaboration of "a-e" by 1
        Else
          Add "a-e" into collaborationMap
      Add a.Data into existingDataSet
    Else
      Add a and a.Data into submissionIdToDataSetMap

```

Figure 16: Pseudo code of co-author algorithm

Time complexity: **$O(nk)$** | Space: **$O(n+m)$**

The final result will be stored inside the **collaborationMap** in the form of (country-country: noOfCollaboration, organization-organization: noOfCollaboration etc.)

Consequently, a new api will be added in the **AnalysisController** to analyse the co-author query: **POST** /api/presentations/{id}/analysis/coauthor

3.6.4 Adding Mail Support in Backend

The **MailController** in the controller component and **MailLogic** logic component adheres to the current Model View Controller (MVC) Pattern where the **MailController** handles send mail request and **MailLogic** handles the **Mail** domain logic, in this case, the construction of the message to be sent and sending the mail.

The **MailController** handles send-mail API requests from the frontend. This is to allow the controller to handle mail requests and future developers can use a single send-mail API endpoint (in this case */api/send-mail*) rather than relying on multiple different API endpoints to send mail.

The **MailLogic** contains the send mail logic as well as the preparation of the mail message to be sent such as how the message should be constructed with the **Mail** domain entity which represents a Mail message.

The **Mail** domain entity in the entity package of the commons component represents the **Mail** entity. By adhering to the Single Responsibility Principle, Separation of concerns as well as to maintain high cohesion, the **Mail** domain entity represents the characteristics of a Mail:

1. Who are the recipients of the mail? (Recipients)
2. What is the subject of the mail? (Subject)
3. What is the mail content of the mail/body of the mail?
4. The Mail Attachment if any

To ensure high cohesion, it does not include send mail logic as this represents the mail message and instead the send mail logic is in the **MailLogic** component as stated earlier.

The **JavaMailCommons** in the common component is a simplified class which contains methods based on *JavaMail* Library by encapsulating relevant methods in custom methods which are commonly used in sending mail using *JavaMail* Library such as the setting up of JavaMail session for sending mail. This reduces the complexity and provides an abstraction for the *JavaMail* Library.

The above design ensure a balance between cohesion and size complexity.

3.7 Design and Implementation Constraints

3.7.1 Design constraints in Revamp of UI

Due to how the Element library is developed, it does not support responsive or mobile design. This makes it extremely hard to design the application that fits for all devices. Understandably, this is because Element is meant to be used in a backstage management system.

In addition, to ensure extensibility and minimal changes to the frontend, our team decided to retain the use of the Element library as it already provide sufficient components needed, although there were discussions on whether a migration to other frontend framework such as Bootstrap should be used. This was later abandoned as it might be too time-consuming and not worth the risk.

3.7.2 Design constraints in Persistency of Records Uploaded

There are a few design and implementation constraints being taken into consideration while brainstorming for some possible design decisions.

The current design of importing data and analyse has been well implemented and hence it is not good to break the current design or implementation. This would be one of the constraints that it would cause some challenges to maintain the persistency. So, option 1 that is mentioned above is implemented since it is similar to a plug and play solutions where a new entity, RecordGroup, is created for adding maintaining the persistency.

Other than that, the JpaRepository is one of the other constraints where the only better way to access the database with the current architecture. Henceforth, SQL queries cannot be written to tackle the problem. So, I have to familiarise with JpaRepository and figure out how to add, modify and delete the data in database.

All in all, all the constraints are well handled, which resulted in the Persistency design implementation as mentioned in the previous section.

3.7.3 Design constraints in Adding Mail Support

Spring Boot provides a library known as [spring-boot-starter-mail](#) with easy abstraction for sending mail with auto configuration.

However, the library requires the dependency [spring-context-support](#) which cause a conflict with the *quartz* library bundled in *google-appengine* library, specifically *google-appengine-stubs* which is used in the test cases.

As the *quartz* library bundled with *google-appengine* is outdated compared to the actual [quartz](#) library, it resulted in missing functionalities which affected the test cases. Hence, in order to

avoid conflicts, [JavaMail](#) library by Java enterprise edition from Oracle is used which is lower level library compared to Spring Boot's library.

Attempts and efforts were made to emulate the Spring Boot's library's functionality and configuration while minimising any conflicts with *google-appengine* library, this resulted in the send-mail API backend design implementation as seen in the source code for ChairVisE 3.0.

3.8 Design Diagrams

3.8.1 Overall Architecture

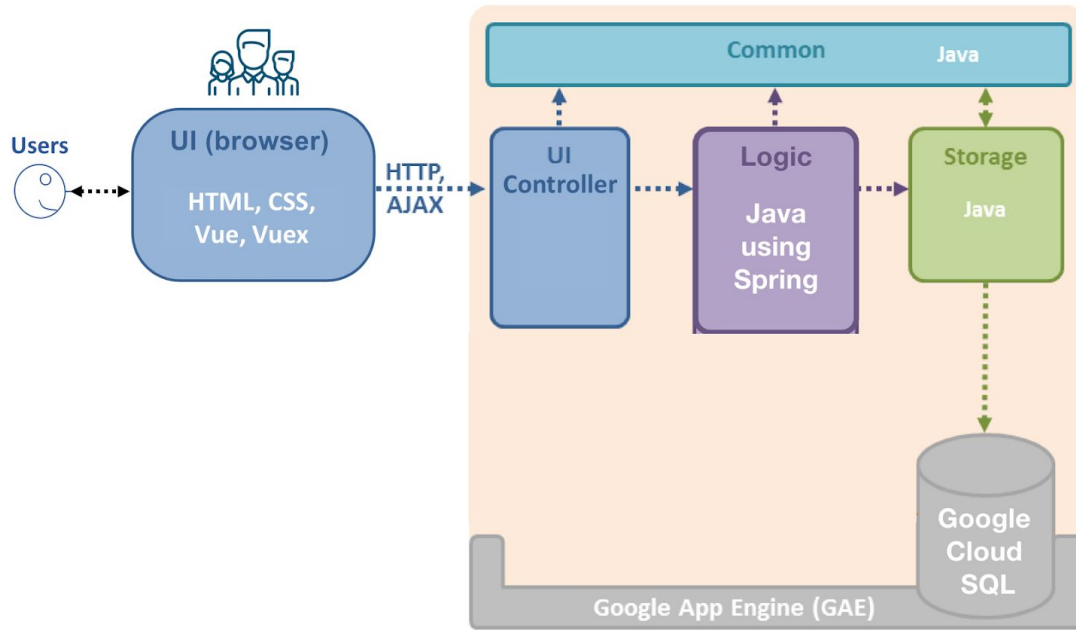


Figure 17: Overall Architecture For ChairVisE 3.0

The application uses the layered architecture design:

- **UI:** The component consists of API controllers and WebPage controllers. API controllers are responsible for handling API calls by the frontend. WebPage controllers are responsible for serving static production Vue files.
- **Logic:** The main logic of the application is in Java using the Spring framework.
- **Storage:** The storage layer of the application uses the persistence framework provided by Google App Engine, using MySQL 5.6.
- **Common:** The Common component contains utility code (data transfer objects, helper classes, etc.) used across the application.

3.8.2 General Sequence Diagram

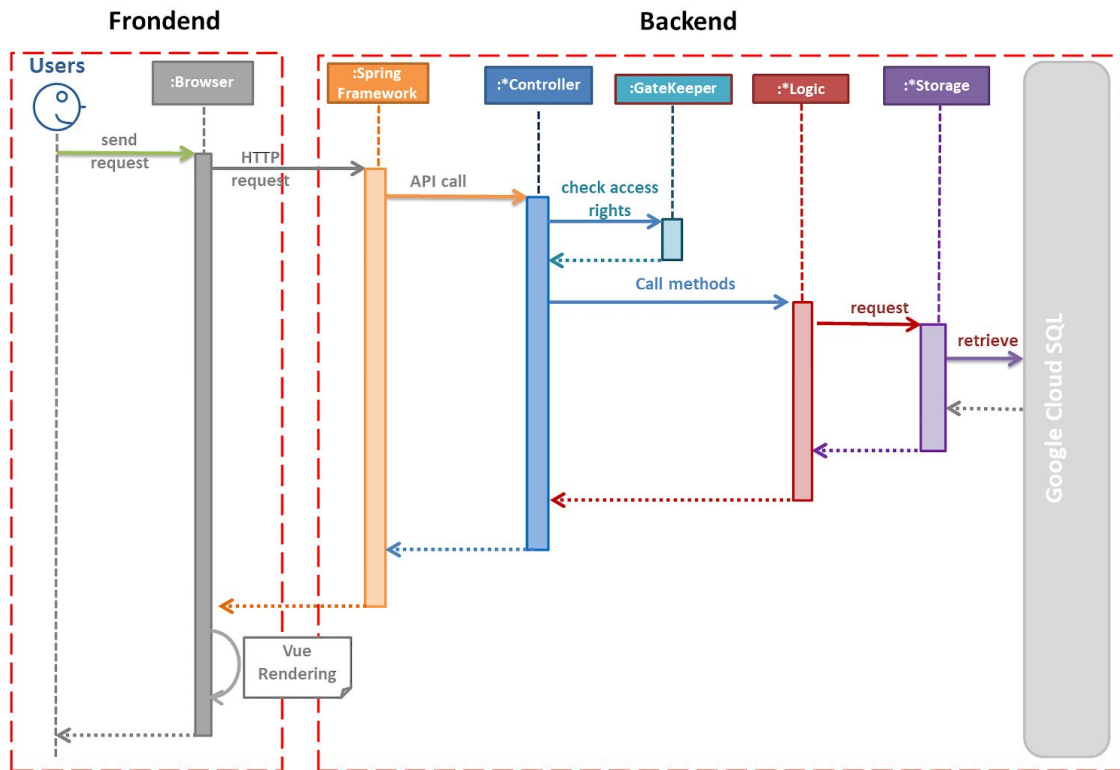


Figure 18: General Sequence Diagram For ChairVisE 3.0

The diagram above showcases the workflow in the system upon receiving a from the user:

1. A user sends a request to the browser for certain data or visualisation.
2. The browser sends an HTTP request to the backend. Spring framework will handle the HTTP request and pass to the correct controller based on URL mapping.
3. The controller checks the access rights by interacting with the GateKeeper.
4. The controller then calls the respective methods in the appropriate logic.
5. The logic processes and requests the data from the storage component.
6. The storage retrieve the data from the Google Cloud SQL (MySQL)
7. The response will be sent back to the browser where it will be used by Vue to render the web page.

3.8.3 Send Mail Backend Sequence Diagram

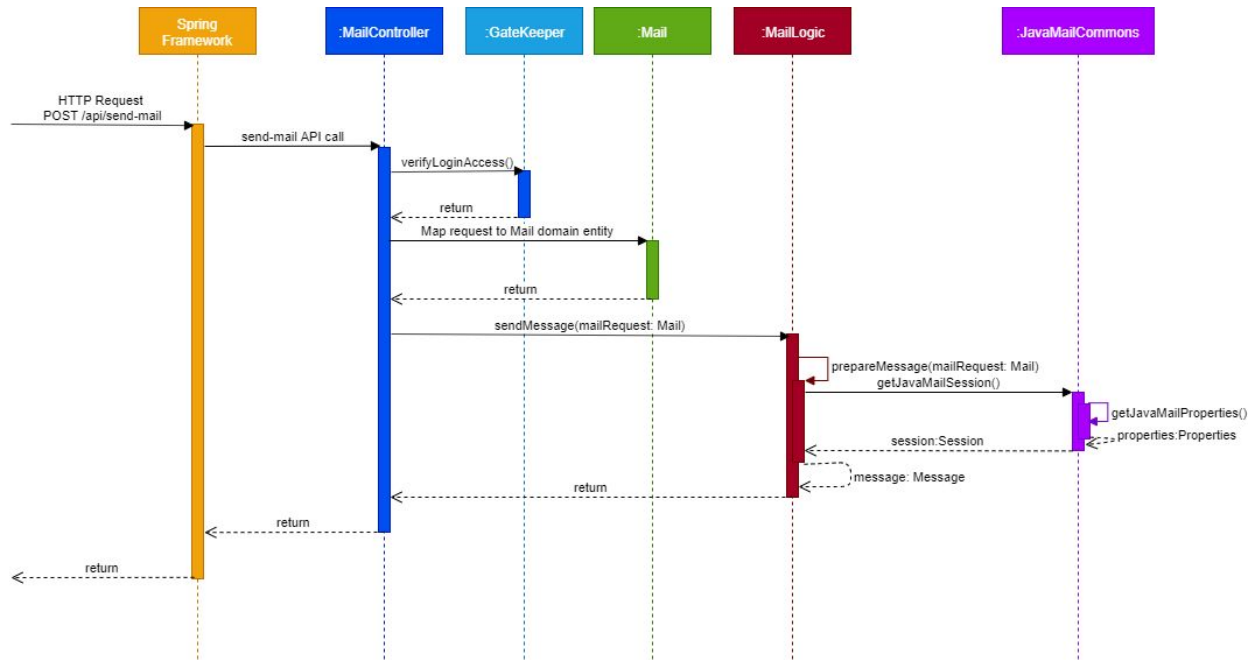


Figure 19: Send Mail Backend Sequence Diagram for ChairVisE 3.0

The send mail sequence diagram above shows the interactions in the backend when a send-mail API request is received from the frontend.

3.8.4 Backend Architecture

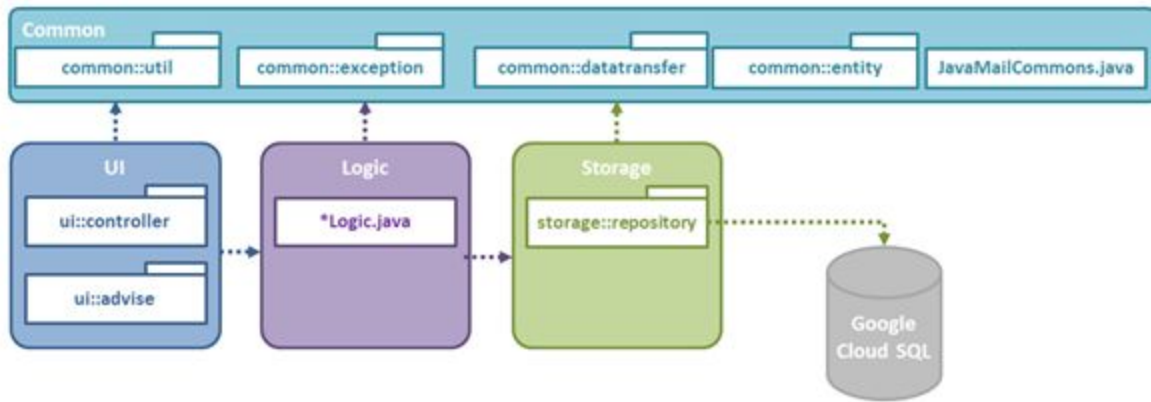


Figure 20: Backend Package Overview For ChairVisE 3.0

The above figure illustrates the backend package overview of the existing system.

3.8.4.1 UI component

All requests from the user browser are received and handled by the UI component in the backend of the web application.

- **ui.controller.api:** Provides backend Representational State Transfer (REST) API access to the users.
- **ui.controller.data:** Contains helper objects to be sent to the client in JSON.
- **ui.controller.webpage:** Handles static file requests for the users.
- **ui.advice:** Handles exception thrown by the application.



Figure 21: API requests to controller mapping For ChairVisE 3.0

The above shows the mapping of the API requests to the appropriate controller. GET is to retrieve data. POST is to create data. PUT is to update data. DELETE is used to delete data.

- **AnalysisController** is used to handle the analysis request sent by the frontend and issue SQL aggregation query to the backend.
- **AuthInfoController** is used to check the current authentication status of user. For example, it will give login URL if the user is not logged in and logout URL if the user is logged in.
- **DBMetaDataController** exposes the metadata, including name and type of the standard data template used in the application. This is useful when users try to match their own CSV file to the data template used in the application.

- **PresentationController**, **PresentationSectionController** and **PresentationAccessControlController** are responsible for the CRUD operations.
- **RecordController** allow record group CRUD operations and accepts the CSV data imported by users and stores them in the database.
- **WebPageController** serves the static production files built by Vue.
- **MailController** is used to handle send mail request from the frontend.

3.8.4.2 Logic Component

The Logic component handles the business logic. In particular, it is responsible for:

- Managing CRUD operations, ensuring the integrity of data.
- Providing a mechanism for checking access control rights.
- **MailLogic** handles the construction of the format of the mail to be sent and the sending of the mail.

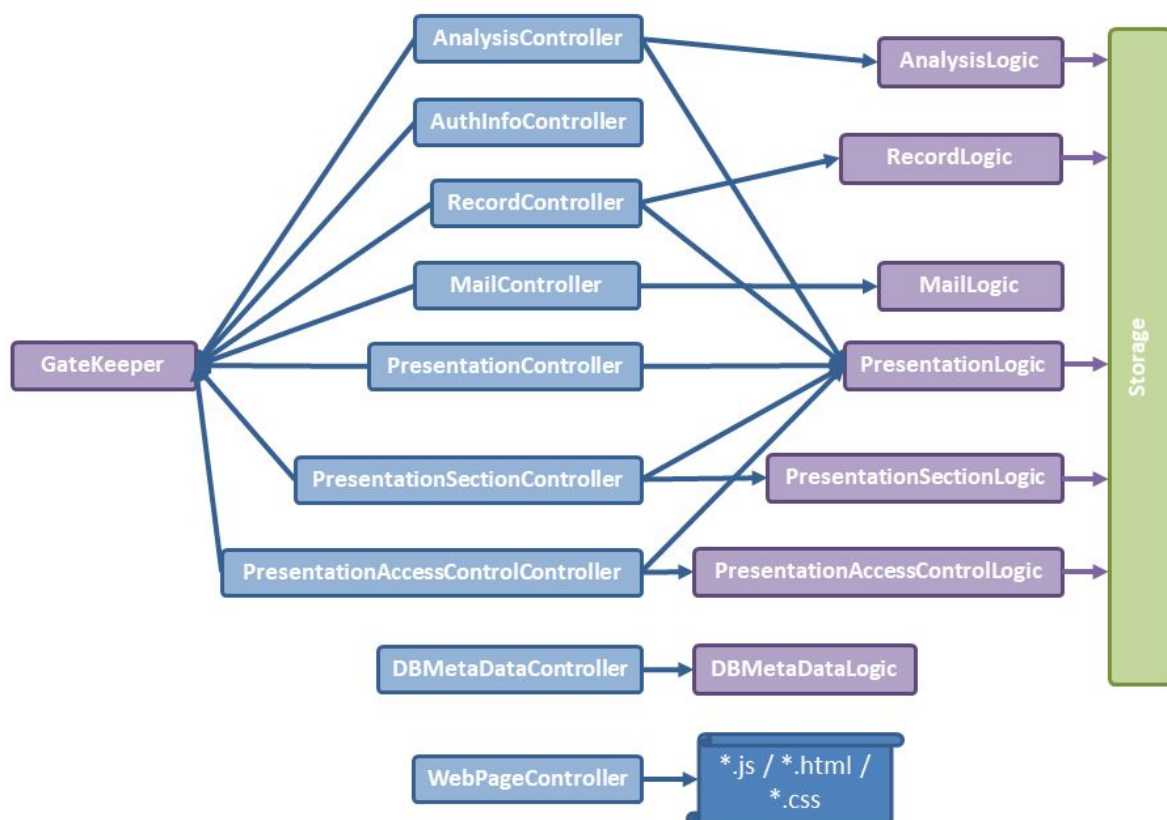


Figure 22: Backend Logic Storage For ChairVisE 3.0

The above figure illustrates how each controller uses its corresponding logic. The dependency is injected (Dependency Injection) by Spring framework automatically.

3.8.4.3 Storage Component

The Storage component performs create, read, update and delete (CRUD) operations on data entities individually. It contains minimal logic beyond what is directly relevant to CRUD operations.

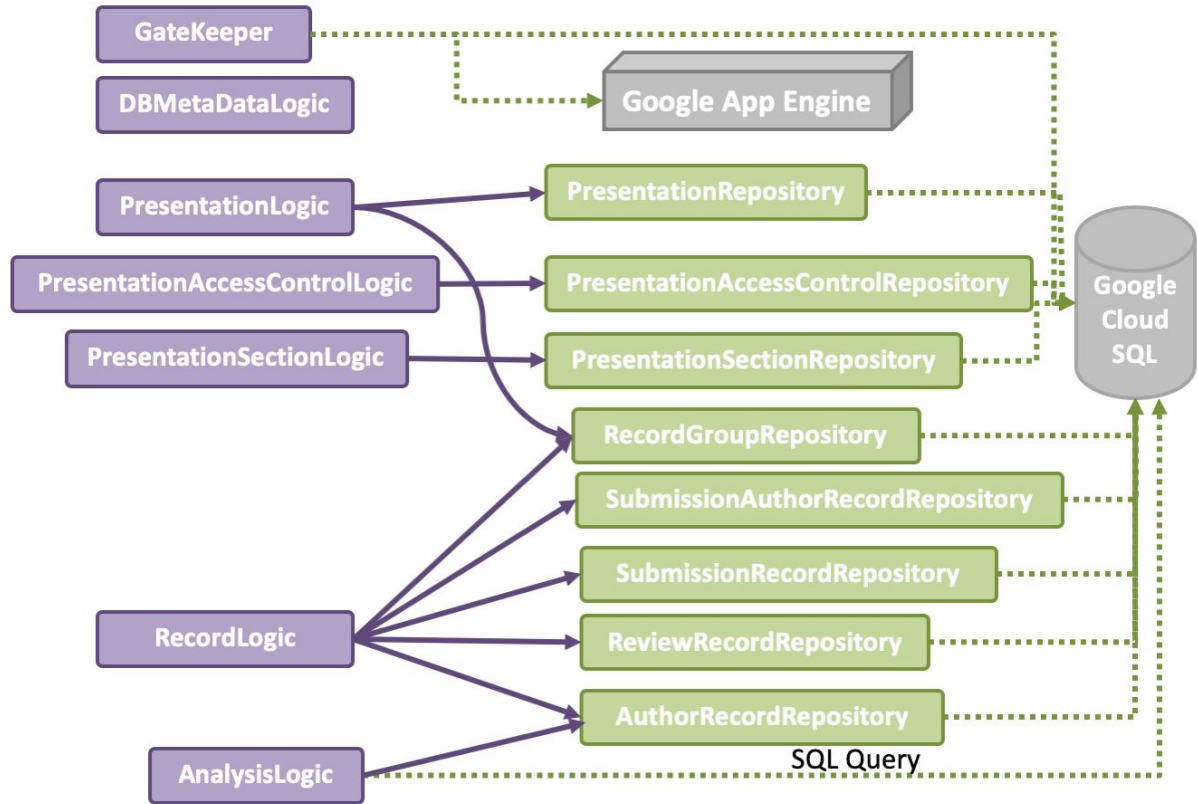


Figure 23: Backend Storage Overview For ChairVisE 3.0

The above illustrates how logic make use of the repository in the storage component.

- **AnalysisLogic** will issue SQL query directly to the Database to perform analysis query.
- **GateKeeper** will use GAE internal APIs to check logged in user and also issue queries to the database to check the access rights.
- **RecordGroupRepository** is responsible for grouping and persisting the data uploaded to the database.
- ***RecordRepository** are responsible for updating user uploaded CSV data to the database.
- **Presentation*Repository** are responsible for storing user generated presentation, access control list and sections.

3.8.4.4 Common Component

The Common component contains common utilities used across the web application.

- **common.util:** Contains utility classes.
- **common.exceptions:** Contains custom exceptions.
- **common.datatransfer:** Contains data transfer objects (DTOs).
- **common.entity:** Contains domain entity such as Mail domain entity and entity stored in the database.
- **JavaMailCommons:** A simplified class which contains methods based on *JavaMail* Library by encapsulating relevant methods in custom methods which are commonly used in sending mail using *JavaMail* Library such as the setting up of *JavaMail* session for sending mail. This reduces the complexity and provides an abstraction for the *JavaMail* Library.

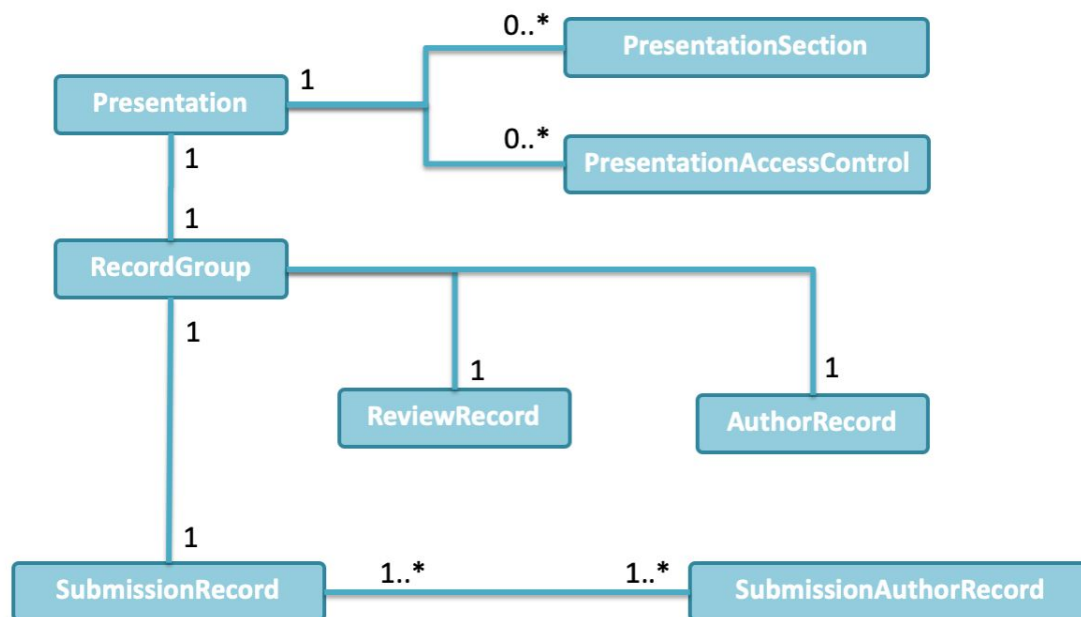


Figure 24: Backend Entities Overview For ChairVisE 3.0

The above illustrates the entity relationship. Each entity has its corresponding table in the relational database.

- One presentation can have multiple sections or access controls.

- One presentation can only have one record group.
- One record group can only have one author record, one review record and one submission record.
- ***Record** is the data imported by users. There is a many-to-many relationships between SubmissionRecord and SubmissionAuthorRecord because one submission can be contributed by multiple authors. Each author can also contribute to multiple submissions.

3.9 Frontend Architecture

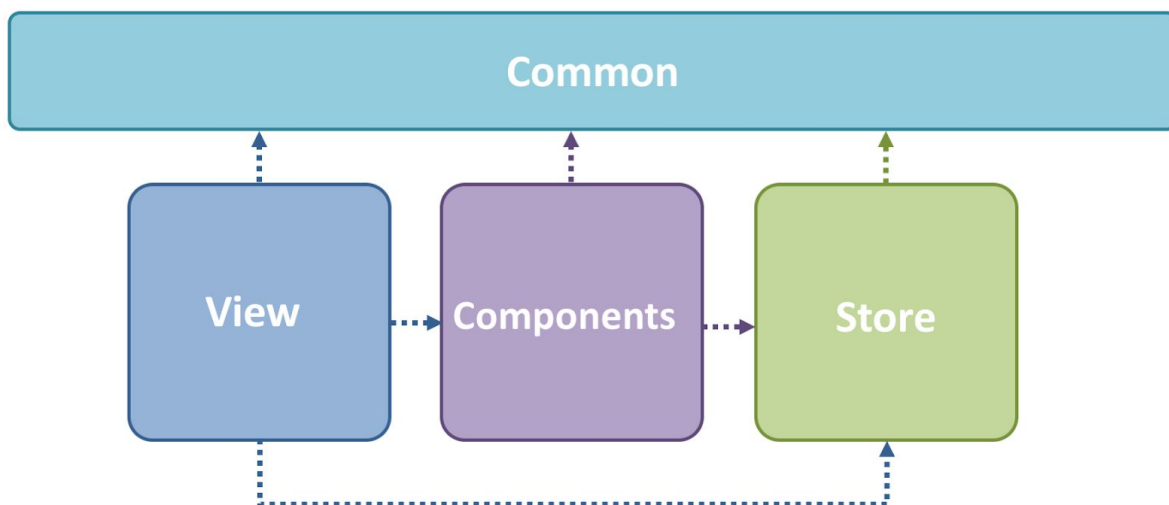


Figure 25: Frontend Package Overview For ChairVisE 3.0

The above figure illustrates the frontend package overview of the application.

- **view:** The view is mainly in charge of displaying pages of the application.
- **component:** The component contains reusable UI and display logic components which is called by multiple pages.
- **store:** The store contains core logic of the application. It shows the state of the application and handles mutation and action to interact within the application using Vuex.
- **common:** Contains utility function as well as constants used.

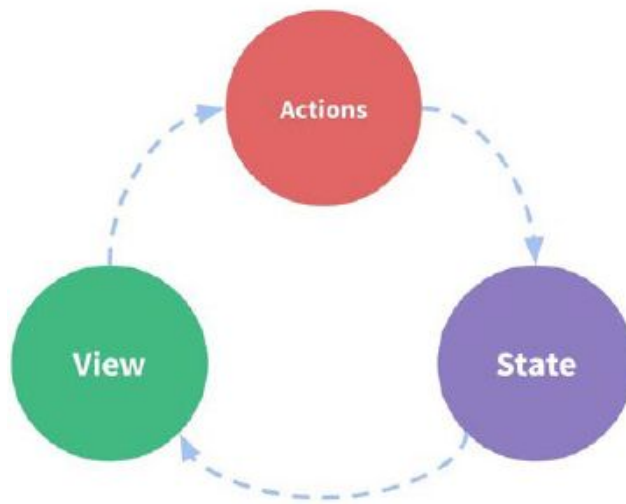


Figure 26: Source: Vuex Documentation

The above illustrates that the view (components) will fire actions and the actions change the state. After that, as view is subscribed to the state (Observable Pattern), the UI (view) will be notified and updated. In doing this, the state becomes the single point of truth in the whole application, which increases the maintainability a lot and make it is easy to trace the change of UI.

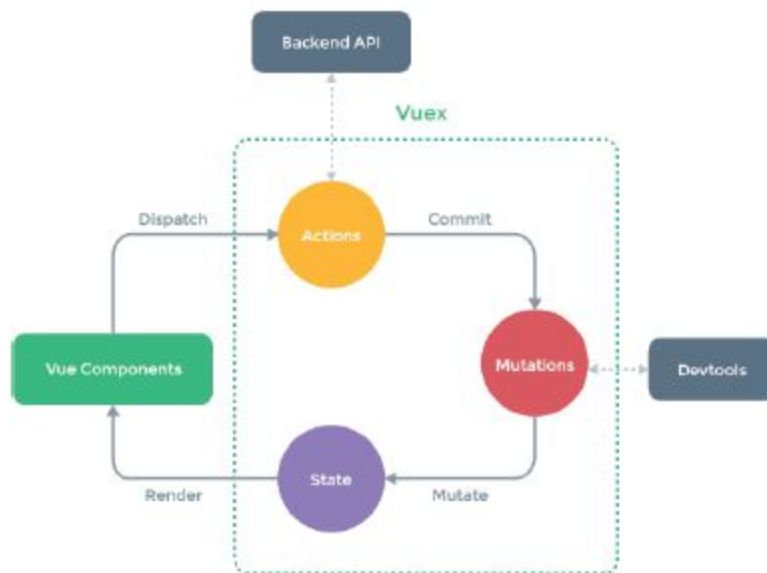


Figure 27: Source: Vuex Documentation

The above illustrates a more detailed. Typically, a vue component will dispatch an action. The action will communicate with backend API to do CRUD operations. After that, it will commit a mutation and the mutation will change the state. The change of state will be propagated to the component and component will render based on the data in the state. A devtool (e.g. Vuex.js devtools) can be used to monitor the state for easy debugging.

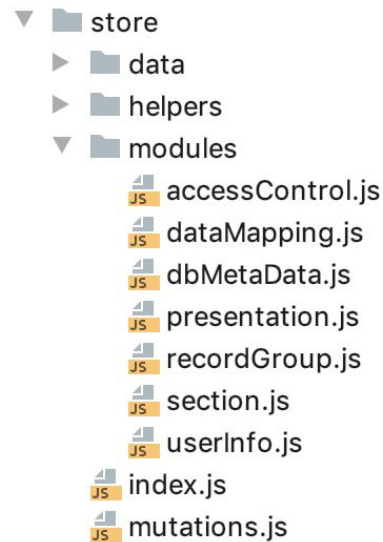


Figure 28: Frontend Store component For ChairVisE 3.0

The following modules are located in the store component:

- **accessControl.js** is responsible for handling logic related to the access control of a presentation.
- **dataMapping.js** is responsible for handling logic related to the mapping of CSV and built-in data schema.
- **presentation.js** is responsible for handling logic related to the presentation.
- **recordGroup.js** is responsible for handling logic related to the record group.
- **section.js** is responsible for handling logic related to the presentation section.
- **userInfo.js** is responsible for handling logic related to the authentication.

In summary, with the help of Vuex, the overall architecture of the frontend ensures that the state can only be mutated predictably.

3.9 Setting up ChairVisE 3.0

Follow the step by step guide below to set up ChairVisE 3.0 on your local machine.

1. Clone the GitHub repository to your local machine.
2. Configure Google Cloud SDK on your machine. You can follow the instructions on <https://cloud.google.com/sdk/>

- a. Prepare for app engine deployment under project root directory in terminal:

```
gcloud -q components install app-engine-java
```

- b. Install npm. We suggest using Node Version Manager (NVM). You can follow the instructions <https://www.npmjs.com/get-npm>

- c. Configure the connection to your local MySQL database:

- i. Navigate to *src/main/resources/application-local.properties* for SQL server connection setup.
- ii. Input your SQL server credentials:

```
spring.datasource.username=<SQL server username>
```

```
spring.datasource.password=<SQL server password>
```

- d. Configure the connection to your Simple Mail Transfer Protocol (SMTP) server.

- i. Navigate to *src/main/resources/application-local.properties* for SMTP server connection setup.
- ii. Input your SMTP server credentials:

```
smtpconfiguration.mail.host=<SMTP server host>
```

```
smtpconfiguration.mail.port=<SMTP server port>
```

```
smtpconfiguration.mail.username=<SMTP server username>
```

```
smtpconfiguration.mail.password=<SMTP server password>
```

```
smtpconfiguration.mail.mailaddress=<SMTP server mail address>
```

NOTE: If you are using a Gmail account, you have to **enable less secure app access** in your Google account [here](#). You can learn more about Google less secure app access [here](#). Google disable less secure app

access by default. Alternatively, you could create a Google App Password using your Google account with 2 step authentication enabled as described [here](#).

3. Run application backend

- a. Go to the terminal, run the application by inputting the command:

gradlew appengineRun

- b. Access the backend application through http://localhost:8080/_ah/admin

4. Run application frontend

- a. Navigate to *src/web/app*
- b. Install the dependencies using the command: *npm install*
- c. Run *npm run serve* in your terminal to start the application.
- d. Access the frontend application through <http://localhost:4040>

3.10 Suggestions for improvements and enhancements

Potential suggestions for future versions of ChairVisE:

1. Add form support so that users can choose which custom mail addresses content they want to send the generated pdf to, custom mail subject and custom mail content for personal backup or for distribution (frontend).
2. Add support for notifications when access controls are changed such as new users that are being added to the shared presentation or the permissions for each user (frontend).
3. Add support for custom mail message content when a new user is added to access control for shared presentation (frontend).
4. Add support for third party enterprise/third party mail services API such as mailchimp and sendgrid to provide alternative mail configuration methods for administrators in the backend.
5. Add an option for users to anonymise data such as names before sharing, exporting or sharing presentation to other users such as for general distribution purposes.

3.11 Visualisation Screenshots of ChairVisE 3.0

3.11.1 Mail Support

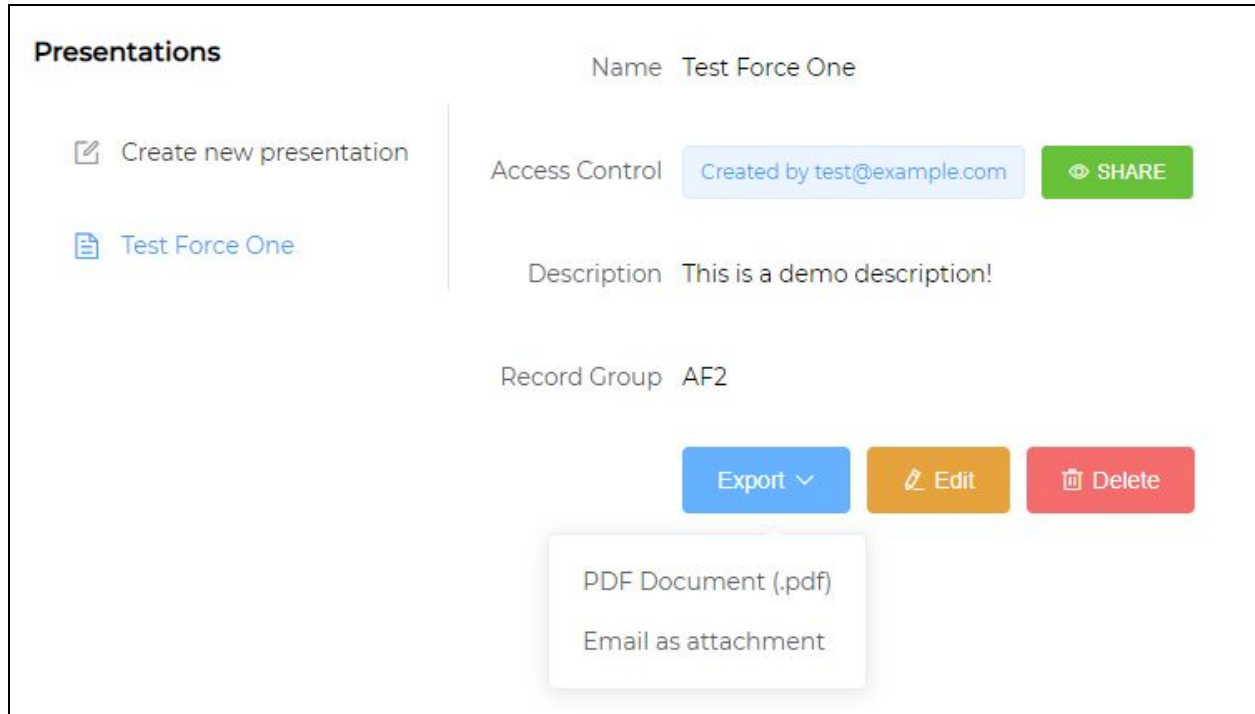


Figure 29: Shows a button to allow users to export presentation to PDF or email as attachment

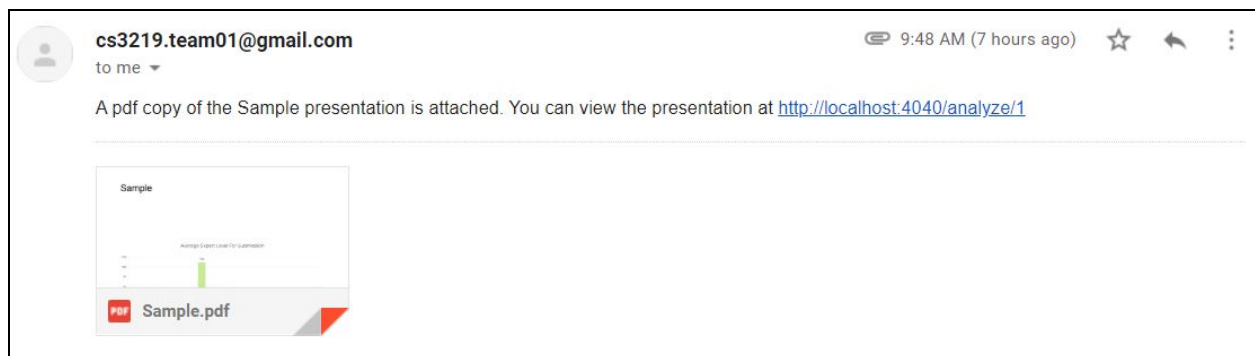


Figure 30: Shows mail notification with generated PDF for backup received by the current user

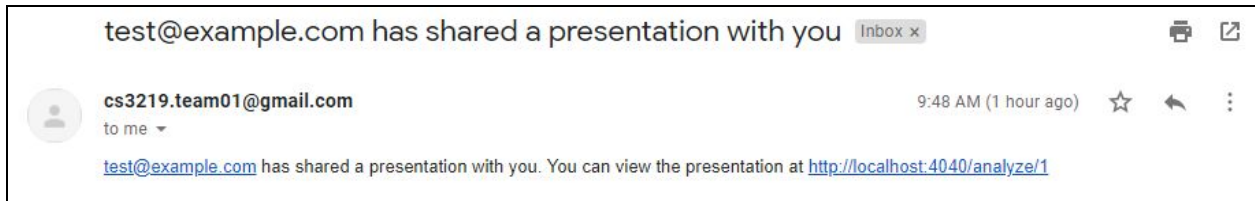


Figure 31: Shows mail notification received by users

3.11.2 Persistency of Records Uploaded

The image shows a web application interface for adding a new record group. At the top, there are navigation tabs: "Home", "Analyze", and "Import Data". On the right, there's a "Logout (test@example.com)" button. Below the tabs, there are two buttons: "Add New Record Group" and "Select Existing Record Group". The "Add New Record Group" button is highlighted. Below these buttons, there's a text input field labeled "Record Group Name". Below the input field, there's a green button labeled "Add New Record Group" and a red error message that says "Please enter record group name".

Figure 32: UI for Add Record Group

The image shows a web application interface for viewing, updating, deleting, and uploading records for a record group. At the top, there are navigation tabs: "Home", "Analyze", and "Import Data". On the right, there's a "Logout (test@example.com)" button. Below the tabs, there are two buttons: "Add New Record Group" and "Select Existing Record Group". The "Select Existing Record Group" button is highlighted. Below these buttons, there's a dropdown menu showing "AF1". Below the dropdown, there's a green button labeled "Select". Below the "Select" button, there's a text input field labeled "Record Group Name" with "AF1" entered. Below the input field, there are two buttons: "Edit" and "Delete". Below these buttons, there are three status indicators: "Author Record: Not Uploaded", "Review Record: Uploaded", and "Submission Record: Not Uploaded". Below the status indicators, there are four steps for uploading a record: "Step 1: Select conference paper" with a "Format Type" dropdown, "Step 2: Select record type" with a "Table Type" dropdown, "Step 3: Does the file has headers?" with a "Has header?" dropdown, and "Step 4: Select predefined mapping" with a "Predefined Mapping" dropdown.

Figure 33: UI for View, Update, Delete and Upload Record for Record Group

Other than that, presentation interface has also been updated to cater for the record group updates.

Home Analyze Import Data Logout (test@example.com)

Presentations created by me

☒ New

☐ test

☐ testII

☐ aaaa

* Name

Description

Record Group

Save

Please create presentation before adding sections

Figure 34: UI for Add New Presentation

3.11.3 Co-authorship Queries

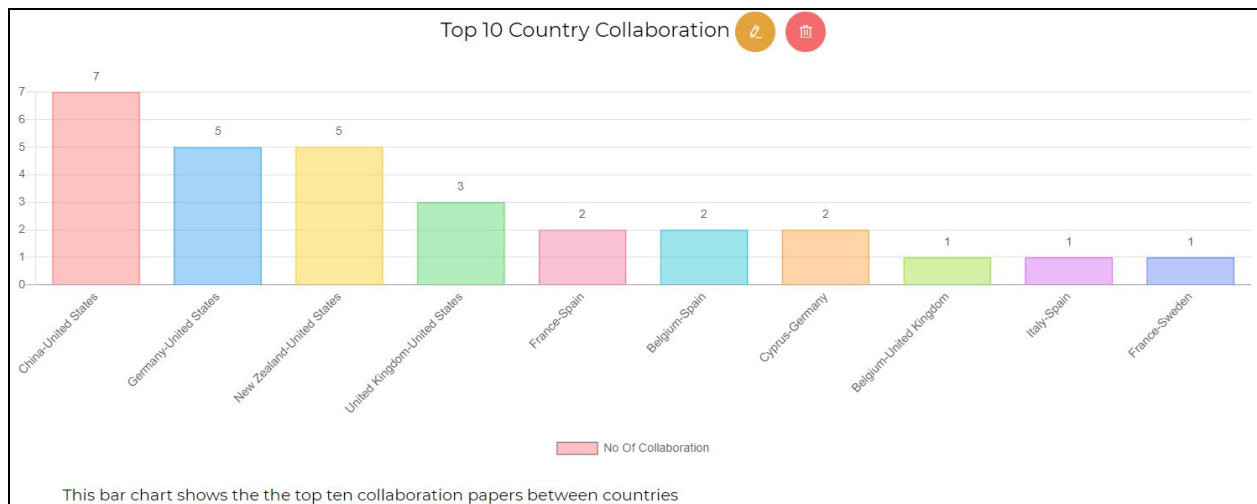


Figure 35: Top 10 Country Collaboration visualization



Figure 36: Top 10 Organization Collaboration visualization

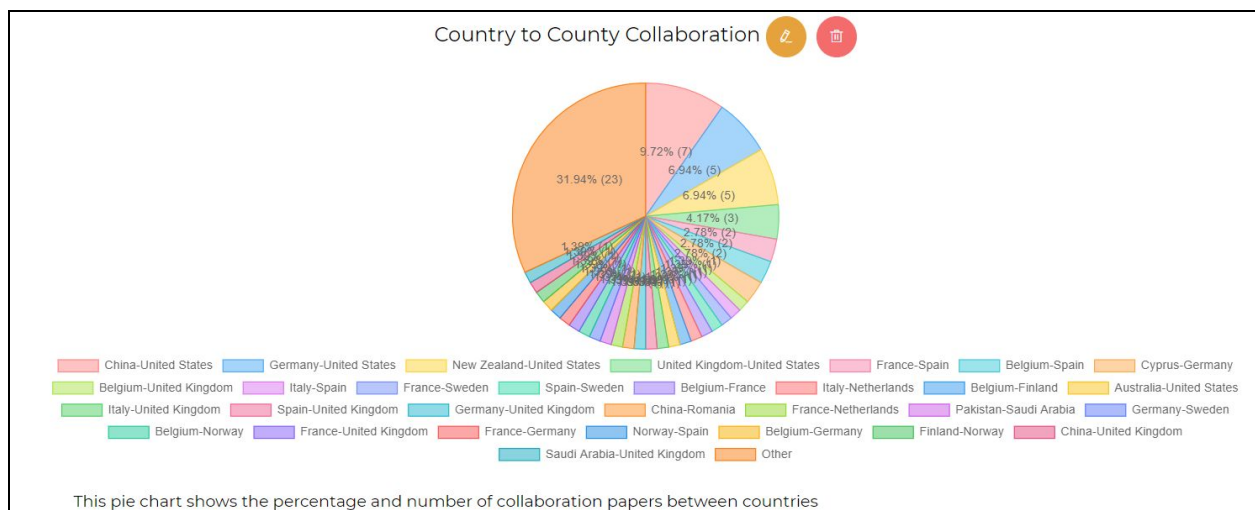


Figure 37: Country-Country Collaboration visualization