



# CS3219

## RottenMods Documentation

Group 1

**Code Repository URL:**

<https://github.com/CS3219-SE-Principles-and-Patterns/cs3219-ay2021-s1-project-2020-s1-g01>

**Team Members:**

<b>Student Name</b>	Wee Woon Wayne	Chester Sim (Shen Dingyi)	Low Zhang Xian	Chan Jing Rong
<b>Matriculation No.</b>	A0173225L	A0185095W	A0185078U	A0185806W

## Table of Contents

<b>1. Introduction</b>	<b>5</b>
1.1. Purpose	5
1.2. Document Conventions	5
1.2.1. Terms	5
1.2.2. Syntax Conventions	5
1.2.3. Naming Conventions	6
1.3. Intended Audience	6
1.4. Scope	6
1.5. References	7
1.6. Individual Contributions	8
<b>2. General Description</b>	<b>9</b>
2.1. Product Features	9
2.2. Target Audience	10
2.3. Operating Environment	10
<b>3. User Stories</b>	<b>11</b>
<b>4. Functional Requirements</b>	<b>13</b>
<b>Figure 4.1 Functional Requirements Table</b>	<b>15</b>
<b>5. User Interface Requirements</b>	<b>16</b>
<b>Fig 5.1 User Interface Requirements Table</b>	<b>16</b>
<b>6. Non Functional Requirements</b>	<b>17</b>
<b>Fig 6.1 Non-Functional Requirements Table</b>	<b>17</b>
<b>7. Development Process</b>	<b>18</b>
7.1. Overview	18
7.2. Development Schedule	18
<b>Fig 7.1 Development Schedule Table</b>	<b>19</b>

7.3. Project Management	19
<b>8. Technology Stack</b>	<b>20</b>
8.1. Frontend	20
8.1.1. Frameworks	20
8.1.2. Libraries	20
<b>Fig 8.1 Frontend Libraries Table</b>	<b>20</b>
8.2. Backend	21
8.2.1. Frameworks	21
8.2.2. Libraries	22
<b>Fig 8.2 Server Libraries Table</b>	<b>22</b>
8.3. Database	22
<b>9. Design Overview</b>	<b>23</b>
9.1. Architecture	23
9.1.1. Overview	23
9.2. Client/Frontend	24
9.2.1. Overview	24
9.2.2. Architecture	24
9.2.3 Pages	25
9.2.4 Components	25
9.2.5 API	29
9.2.6 State	30
9.2.7 Breakdown of Components	34
9.3. Server	39
9.3.1. Overview	39
9.3.2. Architecture	40
9.3.3. Routers	42
9.3.4. Handlers	45

9.3.5. Makers	47
9.3.6. Models	49
9.3.7. Observers	49
9.3.8. Modules, Helpers and Validators	50
9.3.8.1. Modules	50
9.3.8.2. Helpers	50
9.3.8.3. Validators	50
9.4. Recommendations	50
9.4.1. User-based Recommendation	50
9.4.1.1 Advantages	52
9.4.1.2 Limitations	52
9.4.1.3 Computation Time	53
9.4.2. Other Algorithms	54
9.4.2.1 Recommendation of Most Viewed Modules	54
9.4.2.2 Recommendation of Most Rated Modules	54
9.4.2.3 Recommendation of Top Rated Modules	54
9.4.2.4 Advantages	55
9.4.2.5 Limitations	55
9.4.2.6 Computation Time	56
9.4.3. Optimizations	56
9.5. Other Design Considerations	58
9.5.1. Frontend Design Considerations	58
9.5.1.1. FormModalItem.tsx	58
9.5.2. Backend Design Considerations	62
9.5.2.1. Functional over Object-Oriented	62
9.5.2.2. NoSQL over SQL Database	64
9.5.2.3. Monolith over Microservices	66

<b>10. API Documentation</b>	<b>67</b>
<b>11. Deployment</b>	<b>68</b>
<b>12. Possible Enhancements</b>	<b>69</b>
12.1 Academic Study Planner	69
12.2 University Reviews	70
12.3 Others	70
12.3.1 Direct Messaging	70
12.3.2 Multi-Module Comparison	71
12.3.3 Public Profiles	71

# 1. Introduction

## 1.1. Purpose

The purpose of this document is to present a detailed description of the web application RottenMods. It serves to explain the purpose and features of the application, its components, interfaces, functionalities and operation constraints. This document is intended for stakeholders and developers of this web application.

## 1.2. Document Conventions

### 1.2.1. Terms

Some terms that were used in this document have meanings that might differ from the reader's original and are therefore further elaborated on in the table below

Term	Definition
Module	A self-contained course taken in university that typically lasts one term. Not to be confused with a <i>module in software engineering</i>

Fig 1.1 Terms Table

### 1.2.2. Syntax Conventions

Different font styles are used throughout this document to indicate specific information. The following table details these syntax conventions

Font Style	Use
<b>Bold</b>	Table headers, Important information
Condensed	Code snippets
<i><b>Italics Maroon Bold</b></i>	Class/Object names

<i>Italics Maroon</i>	Function/Variable names
<i>Italics</i>	Figure names

*Fig 1.2 Syntax Convention Table*

### 1.2.3. Naming Conventions

Different naming conventions are also followed throughout this document. The following table details these naming conventions

<b>Naming Style</b>	<b>Use</b>
<i>Fig X.Y NAME</i>	Naming of tables and figures where X is the number of the top-level section, Y is the order of the figure in the section and NAME is the name of the figure.

*Fig 1.3 Naming Convention Table*

## 1.3. Intended Audience

The intended audience for the web application is current and prospective university students.

## 1.4. Scope

This web application will be a module review application for current and prospective university students. This application will be designed to allow users to leave reviews for modules they have taken and to browse reviews for modules in order to decide which module to register for, which would otherwise have been done by visiting multiple sites in order to gather the relevant information.

## 1.5. References

Rees, L. (2018, November 19). Creating a Simple Recommendation Engine Using JavaScript and Neo4j. Retrieved from Medium:

<https://medium.com/@lgrees/creating-a-simple-recommendation-engine-using-javascript-and-neo4j-c0fe9859c469>

Ridwan, M. (n.d.). Predicting Likes: Inside A Simple Recommendation Engine's Algorithms.

Retrieved from Toptal:

<https://www.toptal.com/algorithms/predicting-likes-inside-a-simple-recommendation-engine>

Oni, S. (2018, April 19). Introduction To Recommendation system In Javascript. Retrieved from Medium:

<https://becominghuman.ai/introduction-to-recommendation-system-in-javascript-74209c7ff2f7>



## 1.6. Individual Contributions

Student Name	Contributions
Wee Woon Wayne	<p><b>Technical:</b></p> <ul style="list-style-type: none"><li>• Implemented everything server-side other than the recommendation algorithms</li><li>• Deployment of Frontend, Backend and database on AWS</li></ul> <p><b>Non-Technical:</b></p> <ul style="list-style-type: none"><li>• API Documentation</li><li>• Documentation for BackEnd architecture</li><li>• Created diagrams for BackEnd architecture</li></ul>
Chester Sim (Shen Dingyi)	<p><b>Technical:</b></p> <ul style="list-style-type: none"><li>• FrontEnd UI Design</li><li>• Implemented home and search page</li><li>• Implemented module comparison page</li><li>• Implemented module bookmarkers</li><li>• Implemented reviews system</li></ul> <p><b>Non-Technical:</b></p> <ul style="list-style-type: none"><li>• FrontEnd writeup for documentation</li><li>• Created diagrams for FrontEnd architecture</li></ul>
Low Zhang Xian	<p><b>Technical:</b></p> <ul style="list-style-type: none"><li>• FrontEnd UI Design</li><li>• Implemented authentication flow</li><li>• Implemented profile page</li><li>• Implemented study planner</li><li>• Integrated recommendation system into front end</li></ul> <p><b>Non-Technical:</b></p> <ul style="list-style-type: none"><li>• Requirement specifications and possible enhancements write-up in documentation</li></ul>
Chan Jing Rong	<p><b>Technical:</b></p> <ul style="list-style-type: none"><li>• Created recommendation algorithms<ul style="list-style-type: none"><li>○ User-based recommendations</li><li>○ Recommendations based on ratings</li><li>○ Recommendations based on views</li></ul></li></ul> <p><b>Non-Technical:</b></p> <ul style="list-style-type: none"><li>• Documentation for recommendation algorithms</li><li>• Presentation slide deck</li></ul>

Fig 1.4 Syntax C

## 2. General Description

### 2.1. Product Features

Below is a high-level summary table of the features that will be included in the RottenMods Web Application

Index	Feature	Description
F1	Module Search	A system that allows search based on module titles or module code, allowing users to view respective module pages, along with its official information.
F2	Module Reviews	A system which allows registered users to leave and view reviews and ratings for modules and engage in discussion around specific modules
F3	Module Comparison	A system which allows users to easily compare multiple modules according to relevant attributes
F4	Study Planner	A system which allows users to add modules to a system which assists in module planning i.e. allows allocation of modules to specific semesters
F5	Modules Bookmarker	A system which allows users to bookmark favourite modules and come back to them later
F6	Module Recommendation System	A system which provides recommendations of modules by utilizing different algorithms
F7	Management System	A system which allows registered users and administrators to add, remove or edit existing modules in the database

*Fig 2.1 Product Feature Table*

## 2.2. Target Audience

The target users of RottenMods can be split into two main groups - visitors, which include mainly **prospective university students** waiting to enter university, and contributors, which include **current university students** and **graduates**. Graduates have to have had created a RottenMods account when they still had access to the email provided by the university.

The difference between the two groups is that while visitors have read-only access to the entire site, contributors are able to add information such as Schools, Courses and Modules. Contributors are also able to give ratings and leave reviews.

The rationale for this was to provide more accurate information on RottenMods.

## 2.3. Operating Environment

The current version of the web application is operable on most desktop web browsers. These include:

- Chrome
- Edge
- Firefox
- Safari

The functionality of the web application might be limited depending on the device

Type of Device	Degree of functionality
Any laptop or desktop	Full
Any landscape display tablets	Limited
Any mobile device	Very Limited

*Fig 2.2 Device Functionality Table*

### 3. User Stories

In order to better understand each specified feature, we break them down into user stories as follows. The feature(s) that each user story is related to is also attached below.

Index	Description	Feature
<b>As a normal user, I want to be able to...</b>		
US1.1	search for the module I am interested in	F1
US1.2	see the ratings of the module I am interested in	F2
US1.3	see the reviews of the module I am interested in	F2
US1.4	see the official module information of a module that I am interested in	F1
US1.5	sort and filter reviews based on likes and comments	F2
US1.6	have an account so I can take advantage of recommendations	F4, F5, F6
<b>As a contributing student user, I want to be able to...</b>		
US2.1	give a rating for a module that I have taken	F2
US2.2	give a review for a module that I have taken	F2
US2.3	to add new modules that I do not find on the platform easily	F7
<b>As a conscientious student user, I want to be able to...</b>		
US3.1	see even external reviews for a module that I am interested in	F2
US3.2	be recommended modules tailored to my liking	F6
US3.3	save modules that I am interested in	F5
US3.4	easily access all my previously saved modules	F5
US3.5	compare modules using multiple factors	F3

<b>As an inquisitive user, I want to...</b>		
US4.1	compare two different modules that I am interested in	F3
US4.2	compare two different schools that I am interested in	F3
US4.2	compare how similar two different modules are	F3
US4.3	compare different reviews that two different modules have on one page	F3
US4.4	compare the workload between two different modules according to students	F3
US4.5	ask other users for information on a module	F2
US4.6	see similar modules to the one I am currently looking at.	F6
<b>As an involved student user, I want to be able to...</b>		
US5.1	engage in discussions about a module	F2
<b>As a user concerned with privacy issues, I want to be able to...</b>		
US6.1	have a secure way of obtaining access to my account	F2
<b>As a forgetful user, I want to be able to...</b>		
US7.1	make sure that I can easily gain access to my account if I forgot my password	F7
<b>As a lazy user, I want to be able to...</b>		
US8.1	have my data saved even if I am too lazy to create an account	F7
US8.2	leave a module review with little effort	F2
<b>As a careful user, I want to be able to...</b>		
US9.1	ensure that module ratings and reviews are legitimate and sincere	F2
US9.2	ensure that module details are legitimate and up-to-date	F1

*Fig 3.1 User Stories with mapped feature Table*

## 4. Functional Requirements

For each product feature that we have provided under Section 2.1, we list the functional requirements that our system should provide below. **Note that each section is labelled under which feature they correspond to.** For example, F1: Module Search refers to all the functional requirements specified for Feature 1.

Index	Description	Priority
<b>F1: Module search</b>		
FR1.1	The web application should provide a search component for users to search for a module by their module code.	High
FR1.2	The web application should provide a module search results page which contains relevant search results with a quick overview of the module.	High
FR1.3	The web application should provide a search component for users to search for a module by their module title.	Medium
FR1.4	The web application should allow users to see their search history.	Low
<b>F2: Module review system</b>		
FR2.1	The web application should show an average tally of reviews obtained from users containing: expected workload, numerical rating and expected difficulty in the respective module page.	High
FR2.2	The web application should show the official module information from its university on each module page.	High
FR2.3	The web application should have a form that allows logged in users to add a review for a module.	High
FR2.4	The web application should show all reviews that are submitted by other users on the module page.	High
FR2.5	The web application should provide a component for users to sort reviews based on different criteria including number of likes and number of comments in the module page.	High
FR2.6	The web application should provide a component for logged in users to view modules that they have left a review for in their profile page.	High
FR2.7	The web application should provide a component for logged in users to post comments for each review in the module page.	Medium

<b>F3: Module Comparison</b>		
FR3.1	The web application should provide a page for users to compare information from two modules side by side.	High
FR3.2	The web application should provide a module comparison page that allow users to compare official module information from universities of two modules.	High
FR3.3	The web application should provide a module comparison page that allow users to compare an average tally of user obtained information such as expected workload, difficulty and ratings of two modules.	High
FR3.4	The web application should have a component in the home page and search page to allow users to select 2 displayed modules for comparison in the module comparison page.	Medium
<b>F4: Study Planner</b>		
FR4.1	The web application should show logged in users what modules have been added to their study plan.	High
FR4.2	The web application should provide a form for users to add modules into a specified year and semester.	High
FR4.3	The web application should provide a component for logged in users to add modules to their study plan in the module page.	High
FR4.4	The web application should provide a component for logged in users to delete modules from their study plan.	Low
<b>FR5: Modules Bookmarker</b>		
FR5.1	The web application should provide a component for logged in users to bookmark modules in the module page.	High
FR5.2	The web application should show logged in users what their bookmarked modules are on their profile page.	High
<b>F6: Modules Recommendation System</b>		
FR6.1	The web application should suggest similar modules to a user based on their searches on the search page.	High
FR6.2	The web application should show trending modules that are generating a lot of reviews or comments for users on the home page.	High
<b>F7. Others</b>		
FR.7.1	The web application should provide a form that allows students with an official school email to sign up for an account.	High
FR7.2	The web application should provide a component for users to log in	High

	and out.	
FR7.3	The web application should ensure that a logged-in user stays logged in even after a refresh in the page.	Medium
FR7.4	The web application should allow a component for users to add new universities or courses into the system.	Low

*5. Figure 4.1 Functional Requirements Table*



## 6. User Interface Requirements

We specify the user interface requirements for the system which includes the requirements for buttons, functions and messages that will be displayed on any page.

Index	Description	Priority
<b>UIR1: Login and Sign Up</b>		
UIR1.1	The web application should provide the user with a popup notification upon successful login, signup, and logout.	High
UIR1.2	The web application should provide a link on the signup modal to redirect existing users to the login modal, and a link on the login modal to redirect new users to the signup modal.	High
UIR1.3	The web application's login and signup modal should present relevant error hints whenever an invalid input is entered in any form field.	High
<b>UIR2: General</b>		
UIR2.1	The web application should provide a pop up modal with form fields whenever user input is required.	High
UIR2.2	The web application should display components that display the same type of information with the same colour scheme.	High
UIR1.3	The web application should provide either hints or titles in all form fields that require user input.	High

7. Fig 5.1 User Interface Requirements Table

## 8. Non Functional Requirements

The performance requirements listed below specify how fast the app should respond to the user under a certain workload. The specific timings were referenced from [Response Times: The 3 Important Limits](#) by Jakob Nielsen

Index	Description	Priority
<b>NFR1: Performance</b>		
NFR1.1	The web application should respond within 0.1 seconds to a user action that requires no data transfer	High
NFR1.2	The web application should respond within 0.1 seconds to a user action that makes API requests without a request body	High
NFR1.3	The web application should respond within 1 second to a user action that makes API requests with a request body	High
NFR1.4	Each page of the web application should take no more than 1 second to load for all users with a stable internet connection.	High
<b>NFR2: Security</b>		
NFR2.1	The web application should not give users access to restricted pages such as the profile page unless they have logged in.	High
NFR2.2	The web application should not allow sign-ups to users that do not have a student email account that terminates in ".edu"	High
NFR2.3	The web application should only store the email address and password of the user in the browser's web storage for persistent logins.	High

9. Fig 6.1 Non-Functional Requirements Table

## 10. Development Process

### 10.1. Overview

Our team followed the Waterfall process to develop RottenMods. Since our features and project requirements were approved before development commenced, they were well-understood and fixed. Hence, our team planned a development schedule to ensure that all requirements and features were met. Apart from code implementation, we also factored in time for documentation. Weekly meetings with set agendas were done to update our progress and to set and adjust the goals for the following week.

### 10.2. Development Schedule

Week	Implementation	Stakeholders
7	Setting up Frontend and Backend with necessary frameworks	All team members
8	Creation of database with MongoDB	Wayne
	Hi-fi Figma prototypes design for Frontend	Chester, Zhang Xian
9	Implemented crucial routes - Module, User, Review	Wayne
	Implemented recommendation algorithms	Jing Rong
	Set up Login, Home page, Search Page	Chester, Zhang Xian
10	Implemented Login Authentication	Wayne
	Implemented non-crucial Routes - Bookmark, Course, Event, School, Rating	Wayne
	Implemented routes for Study Planner	Wayne
	Added Reaction Routes	Wayne
	Fixed bugs for recommendation algorithms	Jing Rong
	Set up Module Page, Profile Page	Chester, Zhang Xian
11	Integration between Recommendation Algorithms and	Wayne, Jing Rong

	Server	
	Integration between Frontend and Server	Wayne, Chester, Zhang Xian
12	Documentation/Bug Fixes	All
	Deployment of site	Wayne
13	Documentation/Bug Fixes	All

11. Fig 7.1 Development Schedule Table

## 11.1. Project Management

Management of tasks was done on Clubhouse, a project management tool for software teams. After our weekly meetings, we will update our individual tasks as Stories on Clubhouse to better track the progress of the application. Bugs discovered were also placed as Stories and assigned to the relevant stakeholders.

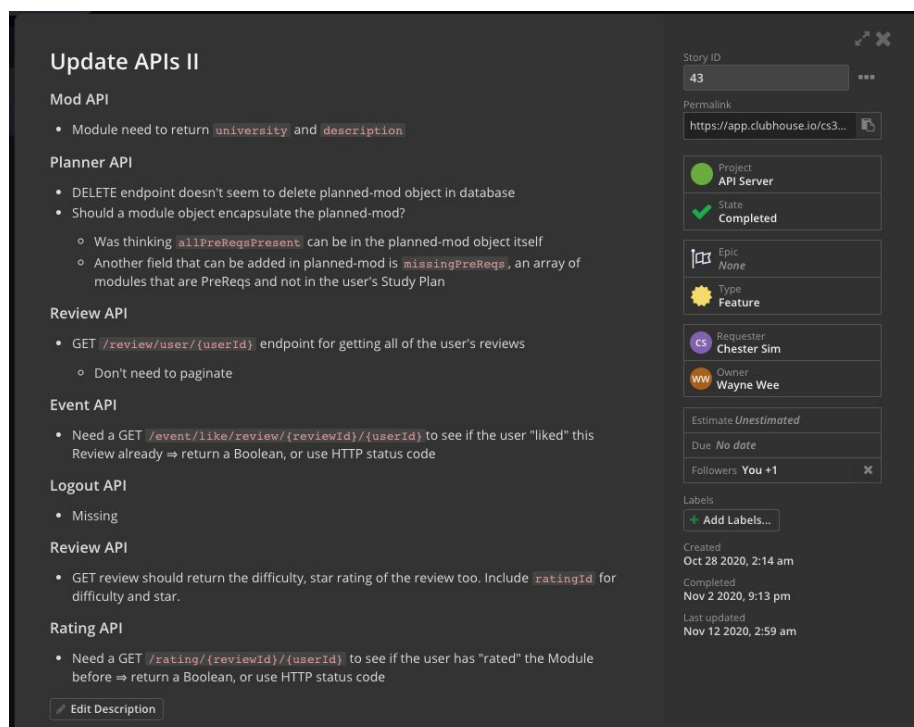


Fig 7.2 Example of a Story on Clubhouse

## 12. Technology Stack

### 12.1. Frontend

#### 12.1.1. Frameworks

The following are the frameworks used in the development of the Frontend of RottenMods.

##### **Next.js**

A Javascript framework for building React web applications.

**Rationale:** Next.js allows server-side rendering (SSR), allowing us to integrate easily with the middleware used by our server to run our database. SSR proves to be able to render UI components faster since the pre-rendered HTML files are sent directly to the client's browser.

#### 12.1.2. Libraries

The following are the libraries that were used in the development of the Frontend for the web application.

v	Name	Description
4.0.5	Redux	Predictable state container for managing application state
4.8.0	antd (Ant Design)	React UI library
3.11.2	react-modal	Highly customizable Modals for React
0.20.0	axios	Promise based HTTP client
6.13.6	query-string	Stringify JS Objects into URL queries

13. Fig 8.1 Frontend Libraries Table

## 13.1. Backend

### 13.1.1. Frameworks

The following are frameworks that were used in the development of the RottenMods server

#### **TypeScript v3.9.5**

Javascript-based language which allows for static type definitions

**Rationale:** TypeScript does static type checking which prevents run-time errors that occur due to incorrect typing. Files are written in TypeScript and are then compiled into JavaScript to be run. In the current version of the application however, the benefits of TypeScript are not fully utilized due to the time constraint faced during development.

#### **npm v6.13.4**

Default package manager for Node.js. Manages dependencies and develops Node programs

**Rationale:** npm provides an easy way to start a Node.js application

#### **Express 4.17.1**

Backend web application framework that provides management of HTTP API routes.

**Rationale:** It is able to handle requests, responses and middleware from clients, and easily integrates with the MongoDB database.

### 13.1.2. Libraries

The following are libraries that were used in the development of the RottenMods server

<b>v</b>	<b>Name</b>	<b>Description</b>
1.19.0	body-parser	Extracts the body from the request and exposes in on the request body object provided by <i>express</i>
1.4.5	cookie-parser	Handles the setting and parsing of cookies
2.8.5	cors	Middleware library that handles cross-origin resource sharing
4.17.1	express	Provides management of routes; handles requests, responses and middleware
5.10.6	mongoose	Object data modelling library that provides data modelling and wrapper functions for MongoDB methods
1.1.5	randomstring	Generates random strings

*14. Fig 8.2 Server Libraries Table*

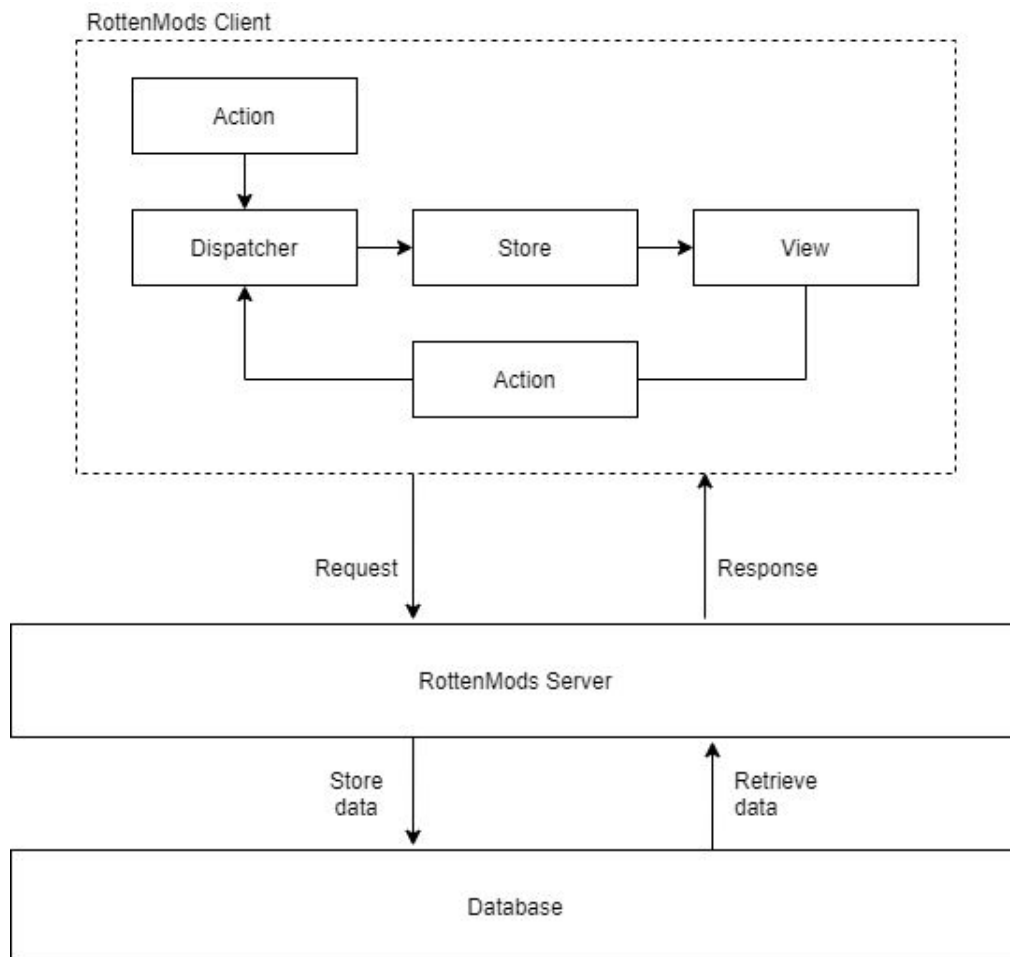
## 14.1. Database

MongoDB (v4.2) is a NoSQL database that stores JSON-like documents. It was chosen for its popularity, the document data model and its powerful and flexible query format.

## 15. Design Overview

### 15.1. Architecture

#### 15.1.1. Overview



*Fig 9.1 System Architecture of RottenMods*

RottenMods uses a two-service architecture consisting of a client and a server with a database, typical of a WebMVC. When the client makes requests to the server, the server processes the request and handles the business logic of the application. The server also communicates directly with the database to store and retrieve data. Data which is retrieved is then passed up the chain and returned to the client via a Response.

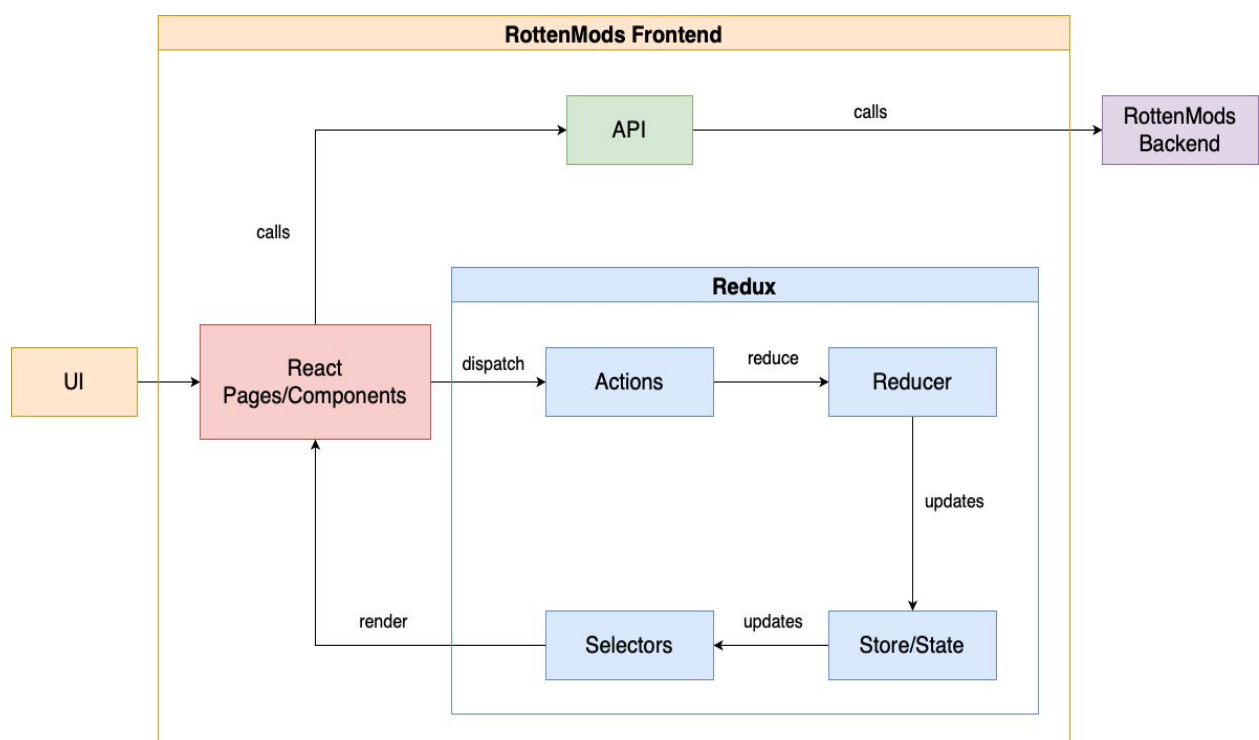


## 15.2. Client/Frontend

### 15.2.1. Overview

The frontend of the application is built using Next.js, a ReactJS framework that allows for Server-Side Rendering (SSR). In this document, we will focus on ReactJS instead of Next.js. Its responsibility includes rendering UI components/views and also making RESTful API calls to the Server to retrieve data to display. State management is supported by the Redux library, which allows for an immutable, predictable and performant state container.

### 15.2.2. Architecture



*Fig 9.2 Architecture of Front End*

The figure depicts the frontend architecture of the application. The main modules of the architecture are Pages, Store, Assets and API, while there are sub-modules including Components and Reducers. The whole process starts with a React Component calling an

API to fetch some data. When the data is received successfully from the server, the Component will dispatch “actions” that will update the App’s state with the newly retrieved data. The Component will detect changes in the App’s state, and rerender the relevant UI elements on the client’s browser. More of this process will be talked in-depth in the next few sections.

### 15.2.3. Pages

Pages are the main representation of the UI. They are broken down into smaller components that abstract the implementation of each different section of the page.

### 15.2.4. Components

Components are the building blocks of Pages, and they abstract both the business and UI logic away from the Pages. More importantly, many components are reusable throughout the application.

```
const Home: NextPage = ({}) => {
  return (
    <>
      <ModuleCompareModal />
      <SectionTitle title="Trending Modules" />
      <HomeModuleList />
      <SectionTitle title="Popular Modules" />
      <HomeModuleList />
      <SectionTitle title="Recommended For You" />
      <HomeModuleList />
    </>
  );
};
```

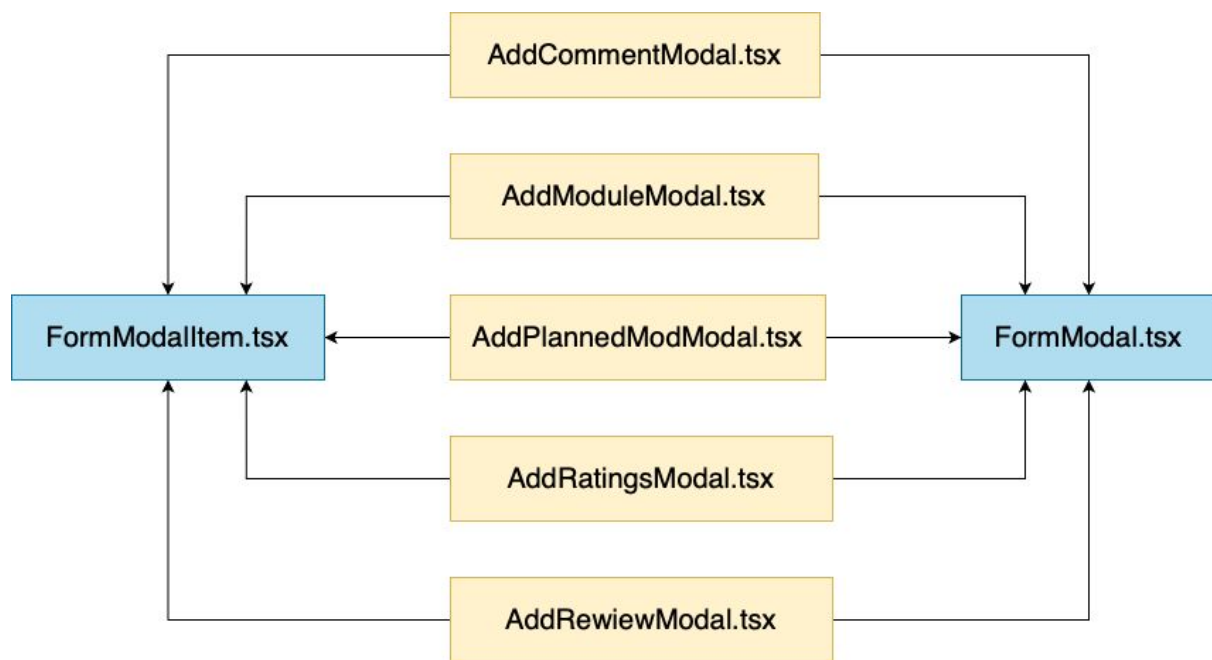
*Fig 9.3 Modularize code for Home Page*



*Fig 9.4 Modularize UI Components*

From the above two figures, we are able to see how abstracting sections of the pages can lead to a more reusable codebase. Although `HomeModuleList` is only used in the Home page, general components like `SectionTitle` can be used in other pages and components as well.

Although UI components are not usually seen as “extensive” in nature (e.g. OOP Class extension), we have developed some custom components that follow such design patterns, to take advantage of the benefits of the Open-Closed Principle (open for extension, close for modifications). One example of such UI components is the Modals with Forms used in the application.



*Fig 9.5 Modals with Forms are built using FormModal and FormModalItem*

From the figure above, we can see that the respective Modals are built on top of `FormModal` and `FormModalItem`. To keep the consistency of the look of Modals across the application, creating a base component (similar to a base Class) was an obvious choice for us. `FormModal` will take in a number of `FormModalItems` as children components, and render them accordingly. These `FormModalItems` are input UI elements as part of the form to be created (e.g. text, number, ratings etc.).

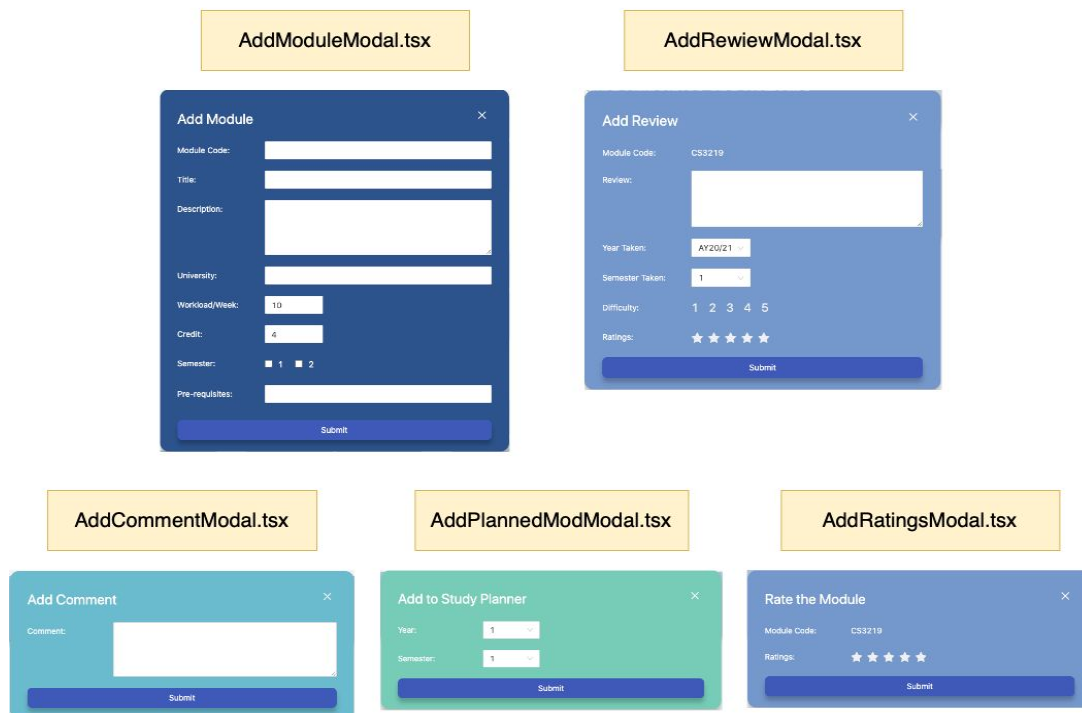
```

return (
  <FormModal
    backgroundColor={reviewBlue}
    submitColor={submitColor}
    title={` ${reviewByUser ? "Edit" : "Add"} Review`}
    isModalVisible={isModalVisible}
    setModalVisibility={closeModal}
    onSubmit={onSubmit}
    submitText={submitText}
  >
    <FormModalItem label="Module Code" type="text" value={code} />
    <FormModalItem label="Review" type="textarea" value={text} setValue={setText} />
    <FormModalItem label="Year Taken" type="annualYear" value={year} setValue={setYear} />
    <FormModalItem label="Semester Taken" type="semester" value={semester} setValue={setSemester} />
    <FormModalItem label="Difficulty" type="difficulty" value={difficulty} setValue={setDifficulty} />
    <FormModalItem label="Ratings" type="rate" value={ratings} setValue={setRatings} />
  </FormModal>
)

```

*Fig 9.6 Code for AddReviewModal*

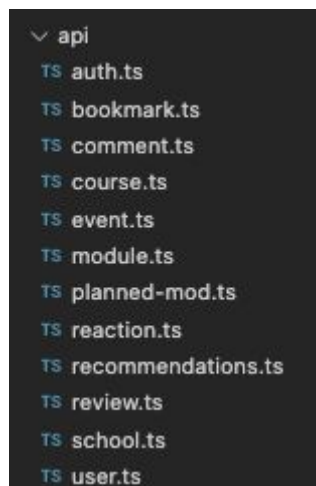
The responsibility of the extended Modal components (e.g. AddReviewModal, etc.) is then to only determine what types of FormModalItems to render in the FormModal. How the underlying Modal is displayed is abstracted by both FormModal and FormModalItem, improving the separation of concerns, while also maintaining the looks across the Modals.



*Fig 9.7 Actual screenshots of the Modals*

### 15.2.5. API

All RESTful API calls are implemented inside the API modules. They are broken down into the individual entities and are responsible for calling the different endpoints made available by the server only.



*Fig 9.8 APIs abstracted into their respective entities*

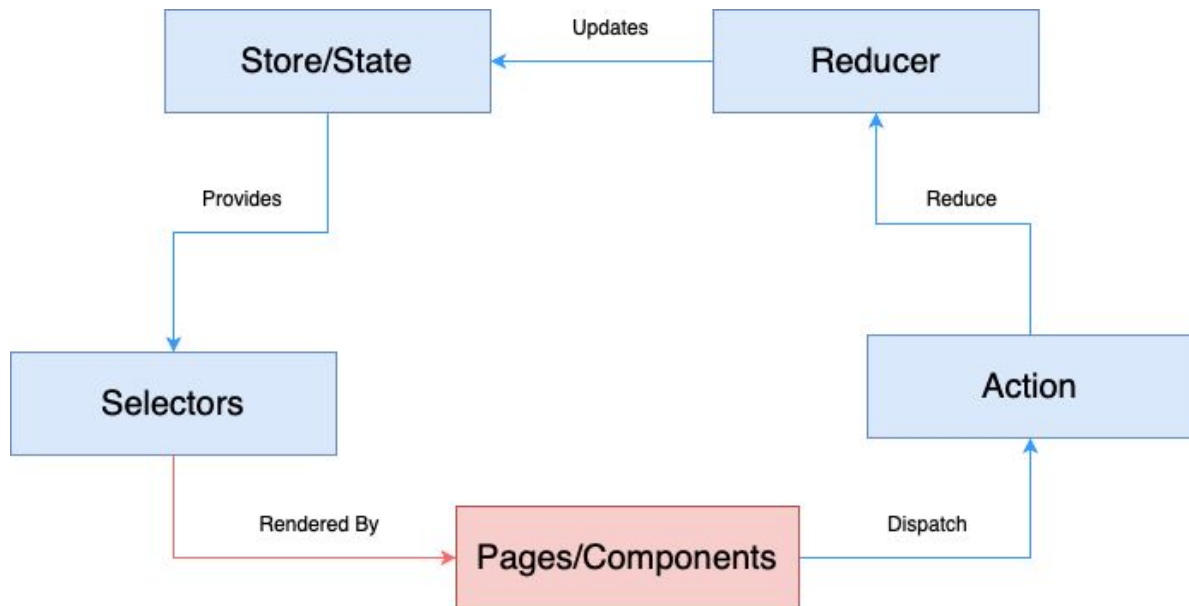
The different entities have the following responsibilities:

Entity	Responsibilities
auth.ts	Enabling authentication for the user
bookmark.ts	Adding, fetching and removing bookmarks of modules for the user
comment.ts	Adding and fetching comments for module reviews
course.ts	Adding and fetching courses of universities and users
event.ts	Adding events (e.g. views) to fuel data analytics for recommendations
module.ts	Adding, fetching, searching, and updating modules of the application
planned-mod.ts	Adding and removing modules from users' study plan
reaction.ts	Adding and removing reactions (e.g. review likes)
review.ts	Adding, fetching and updating module reviews
school.ts	Adding and fetching universities on the application
user.ts	Fetching user details
recommendation.ts	Fetching recommended modules

*Fig 9.9 Entity Responsibility Table*

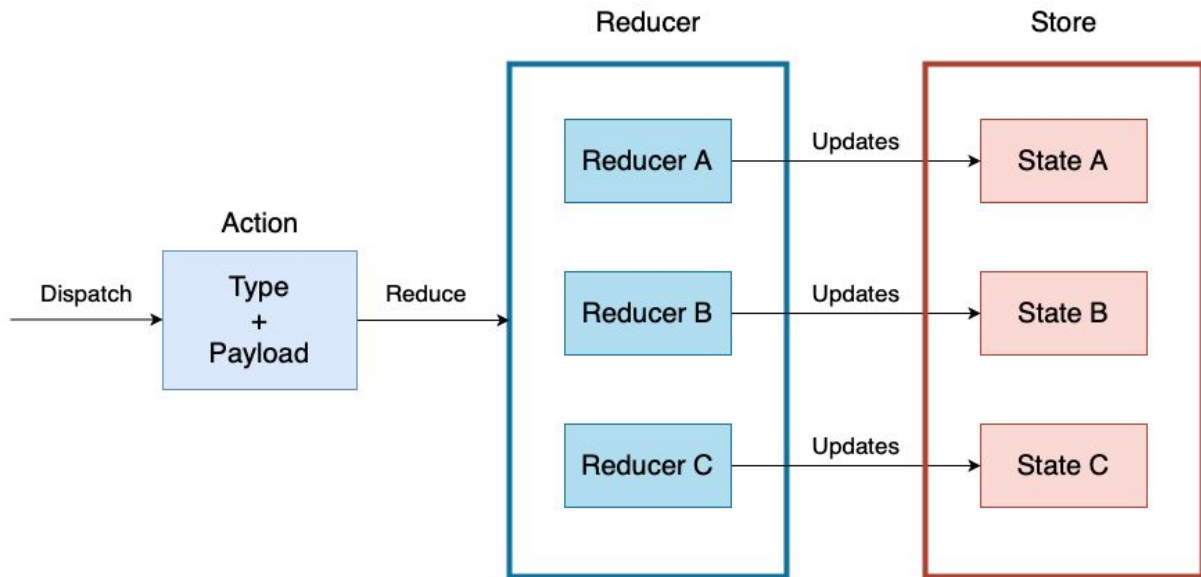
### 15.2.6. State

App-wide state is managed by the Redux Library, to allow any components and pages to access state that is commonly used across the app.



*Fig 9.10 Diagram of how React UI components (red) interacts with Redux (blue)*

The diagram above depicts how Components and Pages are able to trigger changes in the Redux Store using Actions. Actions carry two important information: Type and Payload. The Type represents the type of the Action being dispatched. This is important as the role of the Reducer is to reduce (in functional programming terms) the Payload and State into a new State, and the Type of the action allows the Reducer to identify the method of how to perform the reduction. This architecture is in fact inspired by the Flux architecture.



*Fig 9.11 The relationship between Action, Reducer, and Store*

As we can see from the diagram above, the Reducer is actually made up of smaller Reducers, who are then responsible for different sub-states of the Store. Inside each Reducer, they are also responsible for a few Types of Actions that should trigger changes in their respective States.



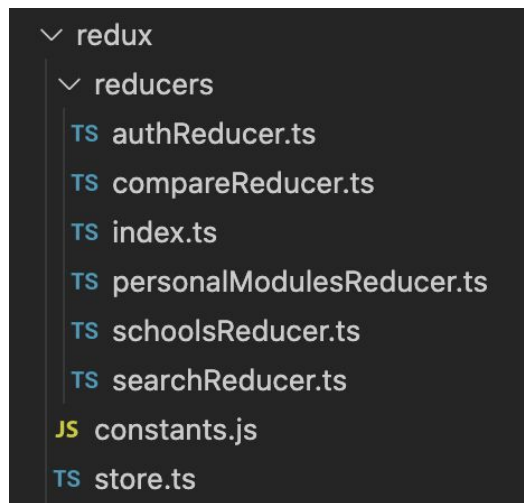


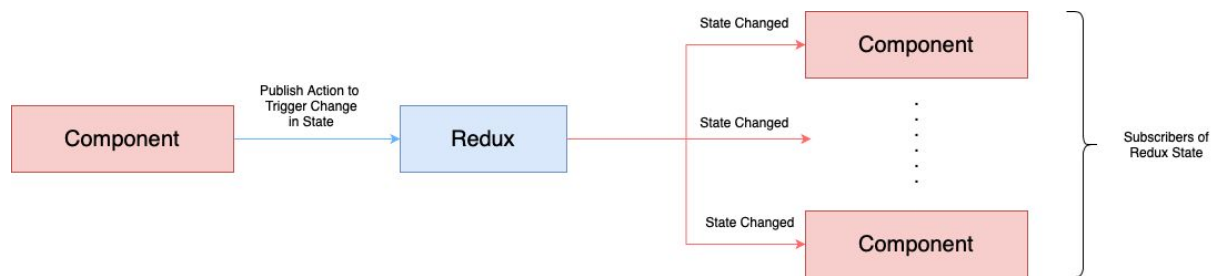
Fig 9.12 RottenMods Redux Reducers

The individual responsibilities of the Reducers are as follows:

Reducers	Responsibilities
authReducer.ts	Maintains the state of whether the user is logged in, and the logged-in user's details
compareReducer.ts	Maintains the state of the two modules to be compared, allowing for the choosing of modules to compare across the application
personalModuleReducer.ts	Maintains the state of bookmarked, reviewed and planned modules of the user, to allow for a better user experience across the app
schoolsReducer.ts	Maintains the state of all available schools on the application
searchReducer.ts	Maintains the state of the search term to allow for a better user experience across the app

Fig 9.13 Reducer Responsibilities Table

One notable design pattern we can see from these interactions is the Publish-Subscribe messaging pattern.



*Fig 9.14 Pub/Sub Messaging Pattern using Redux*

Data (message) from the Components themselves can be published into the Redux store. This publishing is abstracted by Redux itself, seen as the blue components and interactions in Figure 9.2.2.4 (a). Once the Redux Store is updated, subscribers of the Redux Store will then be notified of the changes. These Components are allowed to subscribe to individual sub-state of the Redux Store using Selectors, and will only be notified when there are changes to these sub-states.

```
const isLoggedIn = useSelector((state) => state.auth.isLoggedIn);
```

*Fig 9.15 Example of subscribing to the state of whether the user is logged in*

### 15.2.7. Breakdown of Components

In this section, we will break down the pages of the application, to allow quick mapping of code components and their actual UI counterparts.

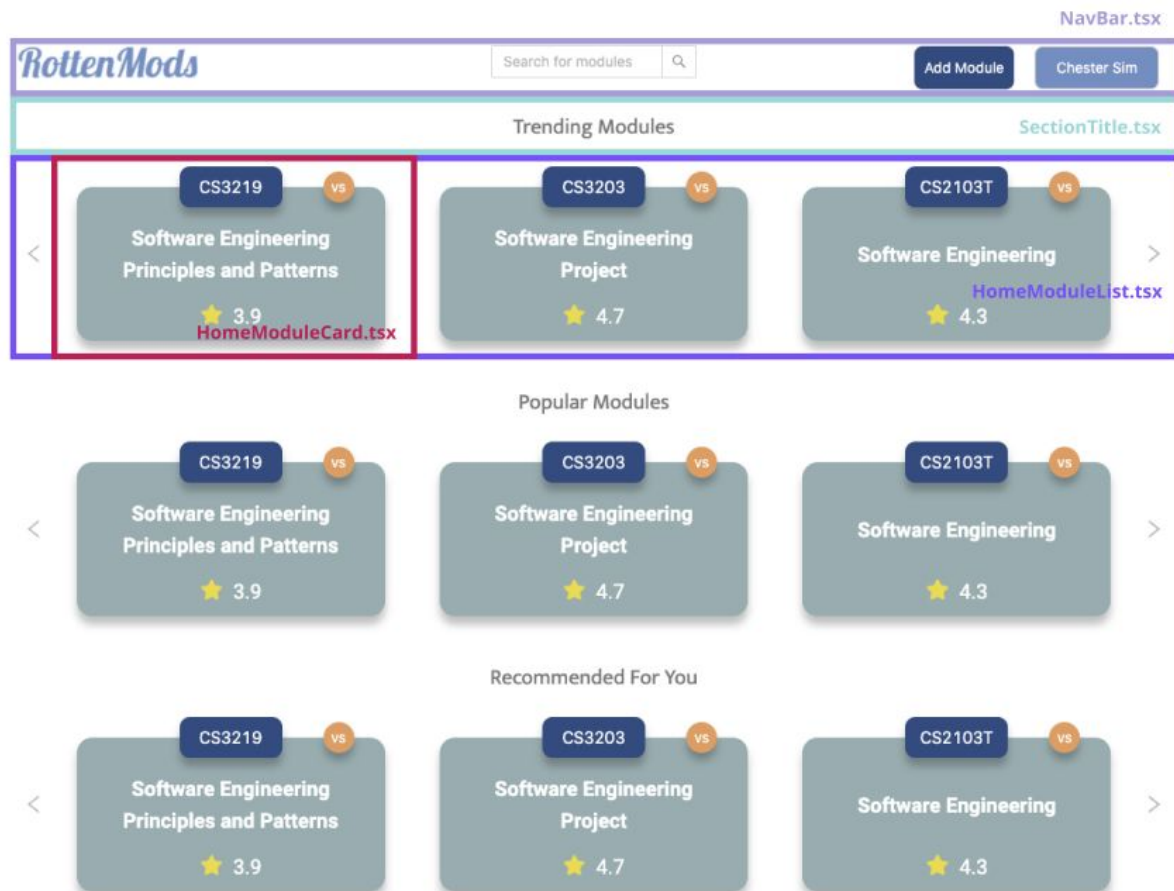


Fig 9.16 Home Page (home.tsx)

**RottenMods**  [Add Module](#) [Chester Sim 2](#)

CS3219

University: NUS   Credit: 4   Pre-reqs: None  
 Semester(s) offered: 1   Workload/Week: 10 hours   Difficulty: 4.3/5

3.6  
 [Bookmarks] [Star] [Edit]

### Software Engineering Principles and Patterns

This module provides an in-depth, hands-on experience in key aspects of software engineering that accompany the development of software. Based on proven principles and best practices, this module focuses on software architectural design from the perspective of the software process. It covers techniques for requirement elicitation and specification that provide sound base for architectural design. The module covers design decision exploration as well as patterns that explicate principles and best practices in replicable form.

[ModuleInformation.tsx](#)

---

Reviews

Sort ▾

Chester Sim | AY19/20, Sem 2 | Difficulty: 2.0 | Rating: 4.0  
 This is a good module!

2 Likes | 1 Comments  
[ReviewCard.tsx](#)

Chester Sim 2 | AY20/21, Sem 1 | Difficulty: 5.0 | Rating: 5.0  
 I have learnt plenty of useful Design Patterns that I am sure I can apply in my future work!

0 Likes | 0 Comments  
[ReviewList.tsx](#)

Fig 9.17 Module Review Page (module-review.tsx)

CS3219 X

vs

CS3203 X

→

Fig 9.18 Module Comparison Component (ModuleCompareModal.tsx)

**CS3219**

VS

**CS3203**

**Software Engineering Principles and Patterns**

This module provides an in-depth, hands-on experience in key aspects of software engineering that accompany the development of software. Based on proven principles and best practices, this module focuses on software architectural design from the perspective of the software process. It covers techniques for requirement elicitation and specification that provide sound base for architectural design. The module covers design decision exploration as well as patterns that explicate principles and best practices in replicable form.

**Software Engineering Project**

This module provides students with hands-on experience in working in project groups through a complete SDLC to develop a well-designed, well-tested, large-scaled software system. The students will apply the current best software engineering practices on the analysis, design, implementation, and testing of software system. Through the project, students will practise analysis of user's needs, formulation of computing requirements to meet the user's needs, modelling and design of the computer systems according to the requirements, evaluation of the design, efficient implementation of software components, system integration, software version control, and rigorous testing.

[ModuleInfoComparison.tsx](#)

10	Expected Workload/Week	20
3.0/5	Difficulty	3.0/5
3.0	Ratings	3.0
1	Semester(s) offered	1, 2
4	Credits	4
National University Of Singapore	University	National University Of Singapore

[MetaCompareRow.tsx](#)

[ModuleMetaComparison.tsx](#)

Chester Sim | AY20/21, Sem 1 | Difficulty: 5.0 | Rating: 5.0

This is a good module!

Chester Sim | AY20/21, Sem 1 | Difficulty: 5.0 | Rating: 5.0

This is also a good module!

Reviews

[ReviewList.tsx](#)

[ModuleReviewsComparison.tsx](#)

Fig 9.19 Module Comparison Page (module-comparison.tsx)

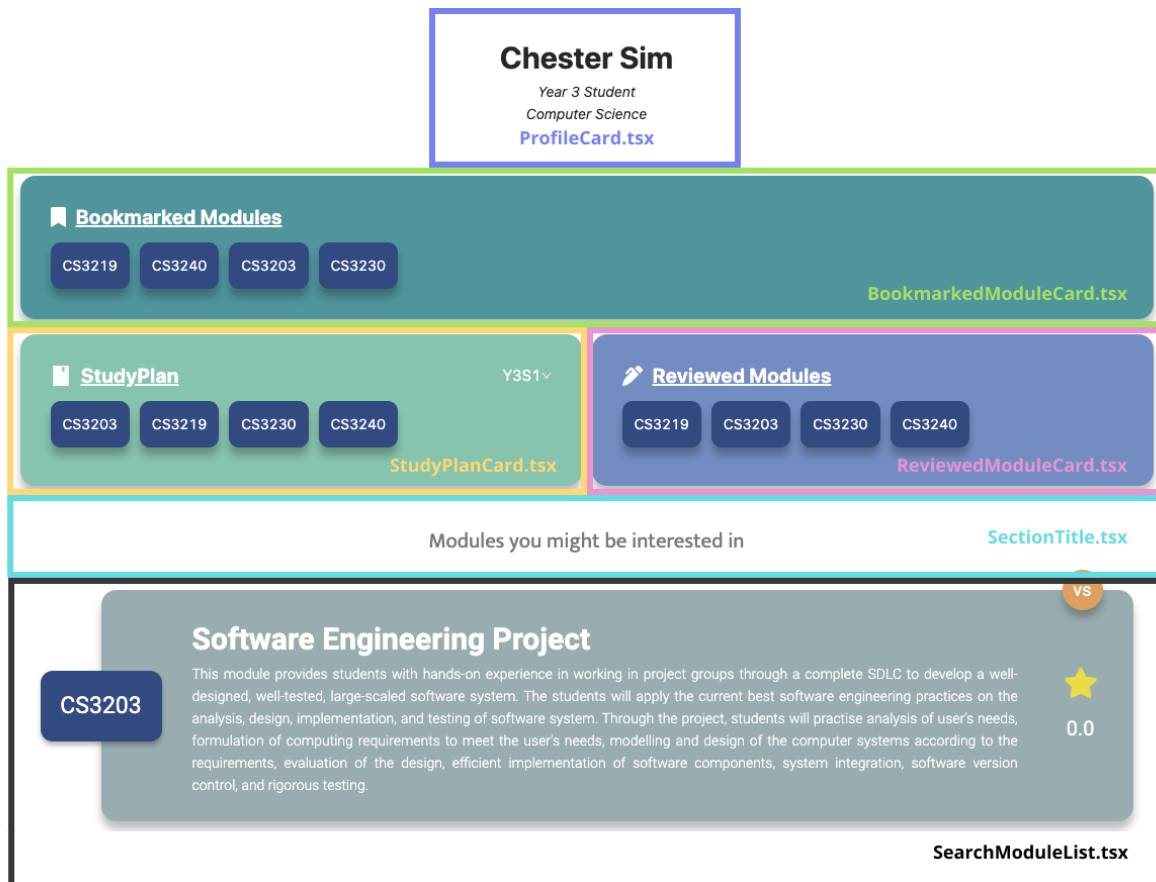


Fig 9.20 Profile Page(profile.tsx)

Search results for "cs32"

SectionTitle.tsx

School:

NUS

Semester:

☐ 1☐ 2☐ 3

Credits:

CS3201R

### Software Engineering Project I

This 1-MC module adds a research component to the host module, enabling students to acquire more in-depth understanding of the research issues pertaining to the subject matter.

VS



0.0

SearchModuleListCard.ts

CS3201

### Software Engineering Project I

This module is the first part of a two-part series on the practice of software engineering in Software Development Life Cycle (SDLC). These two modules together provide the students with hands-on experience in working in project groups through a complete SDLC to develop a well-designed, well-tested, large-scaled software system. This first part focuses on applying best software engineering practices on the analysis and design of software system. The students will practice analysis of user's needs, formulation of computing requirements to meet the user's needs, modeling and design of the computer systems according to the requirements, and evaluation of the design.

VS



0.0

CS3202

### Software Engineering Project II

This module is the second part of a two-part series on the practice of software engineering in Software Development Life Cycle (SDLC). These two modules together provide the students with hands-on experience in working in project groups through a complete SDLC to develop a well-tested, large-scaled software system. This second part focuses on applying best software engineering practices on the implementation and testing of the software system. The students will practice efficient implementation of software components, system integration, software version control, and rigorous testing.

VS



0.0

CS3211

### Parallel and Concurrent Programming

A concurrent system consists of a set of processes that executes simultaneously and that may collaborate by communicating and synchronising with one another. Examples of concurrent systems are parallel programs that describe sets of collaborating processes. This module introduces the design, development and debugging of parallel programs. It will build on the concurrency concepts gained from the Operating Systems module. It covers concepts and modelling tools for specifying and reasoning (about the properties of) concurrent systems and parallel programs. It also covers principles of performance analysis, asynchronous and asynchronous parallel programming, and engineering concurrent systems and parallel programs.

VS



0.0

CS3210

### Parallel Computing

The aim of this module is to provide an introduction to the field of parallel computing with hands-on parallel programming experience on real parallel machines. The module is divided into four parts: parallel computation models and parallelism, parallel architectures, parallel algorithm design and programming, and new parallel computing models. Topics includes: theory of parallelism and models; shared-memory architectures; distributed-memory architectures; data parallel architectures; interconnection networks, topologies and basic of communication operations; principles of parallel algorithm design; performance and scalability of parallel programs; overview of new parallel computing models such as grid, cloud, GPGPU.

VS



0.0

SearchModuleList.tsx

See More

SeeMoreButton.tsx

Fig 9.21 Search Page (search.tsx)

## **15.3. Server**

### **15.3.1. Overview**

The API server handles the requests made from the client, contains the business logic of the application and manages access to the database. The server is written in TypeScript, which is then compiled and run using Node.js. The Express library is used to handle the routing. The data storage is done using MongoDB, a NoSQL database program.



### 15.3.2. Architecture

The server is made up of a number of sub-components which are listed, along with their functions, in the following table.

Component	Function
Routers	Receives the request made by the client and decide which method of the Handler needs to be called
Handlers	Calls the Maker in order to retrieve a valid object and calls the Model to access the database
Makers	Transforms, validates and normalizes information passed in by the Handler and returns a valid object
Models	Calls the database functions
Observers	Notifies a list of subscribers on event triggers
Modules	Contains large auxiliary functions such as the recommendation methods
Helpers	Contains small auxiliary functions
Validators	Contains validation functions

*Fig 9.22 Component Functions Table*

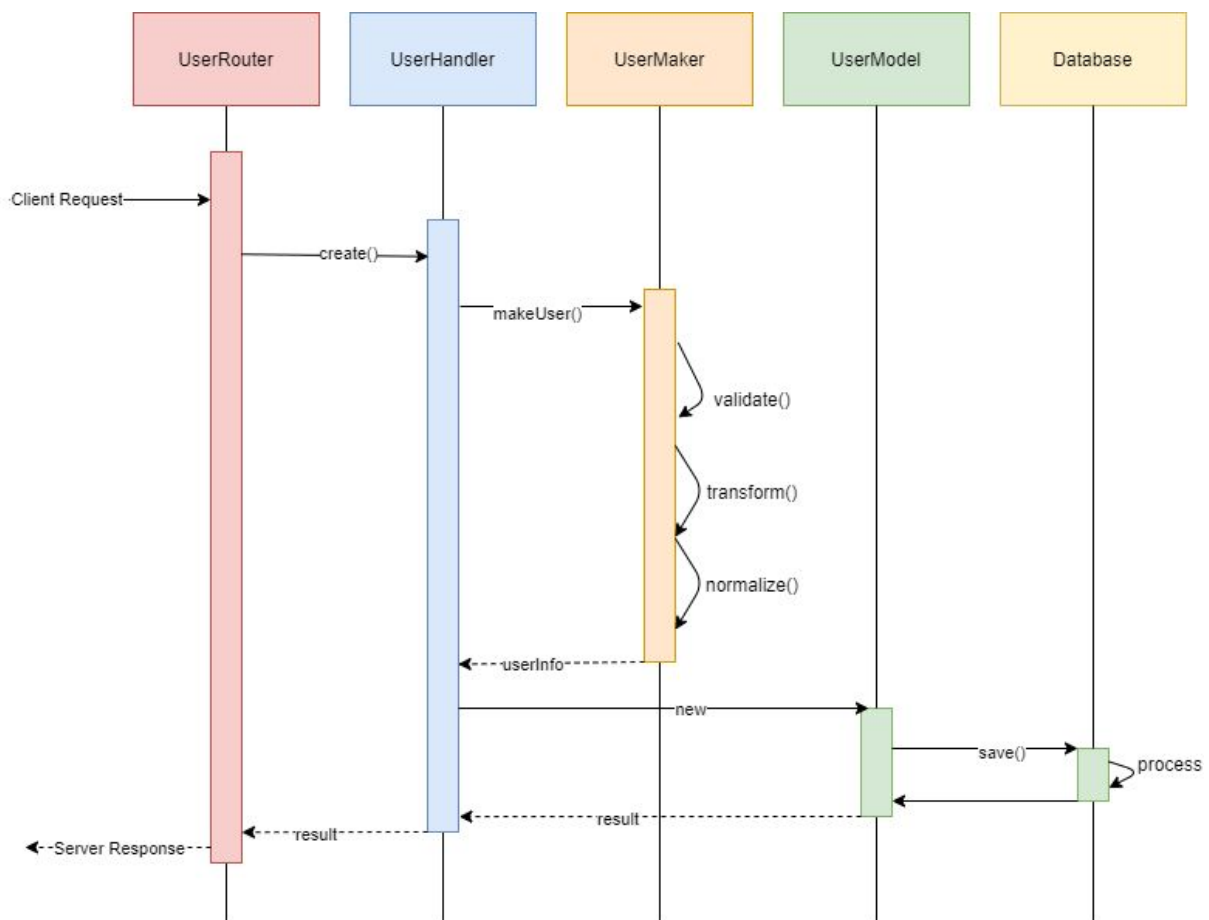


Fig 9.23 Sequence diagram of how Server handles Client Requests

To illustrate the server architecture more clearly, consider a scenario where a request is made to add a new user to the database. Firstly, the **UserRouter** would receive the request from the client and, based on the request method and path, decide which function of the **UserHandler** to call and eventually calls **UserHandler.create()**. The **UserHandler** then calls the function **makeUser()** with the request body, which transforms the body, validates its parameters and normalizes the data. After receiving a valid object body from **makeUser()** The **create()** function then calls the **save()** method of the **UserModel**, which saves the object to the database. If successful, a success message is returned to the **UserRouter** which sends it in the response to the client. The following sections describe these components in a greater level of detail.

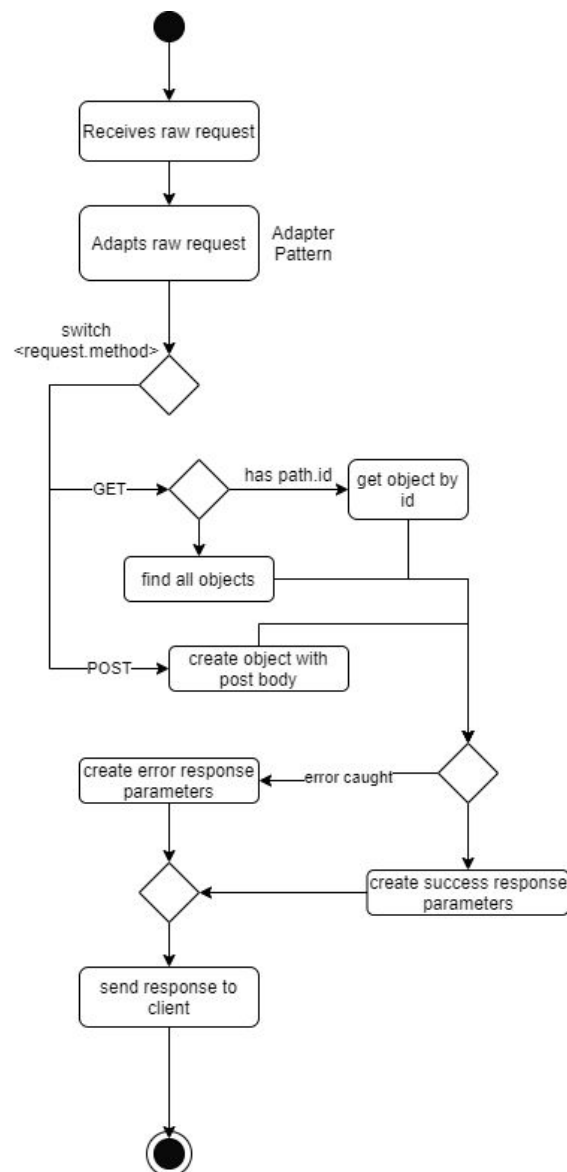
### 15.3.3. Routers

The Routers handle the requests from and the responses to the client.

The file `routers/index.ts` exposes the API endpoints to the client but does not control the routing itself. When one of the endpoints is hit, the corresponding Router function is called to decide what method should be run.

There were two main options to handle the routing of client requests to the server. The first of these was to rely on the Express framework completely for deciding which method to call, adding middleware and error handling. The alternative to this decision was to handle the request directly by adapting the request body- keeping only the required fields and retaining control of the selection of methods executed through switching on the request method and path parameters. Whilst the underlying functionality remains the same, the second method was utilized in order to reduce coupling between the application server and the Express framework. By doing so, the Dependency Inversion Principles is followed. If the Express framework were to change, only the `routers/index.ts` file, which uses Express to expose the API endpoints would need to be modified.

The following activity diagram shows the decisions a Router makes to decide which method of its corresponding handler should be called:



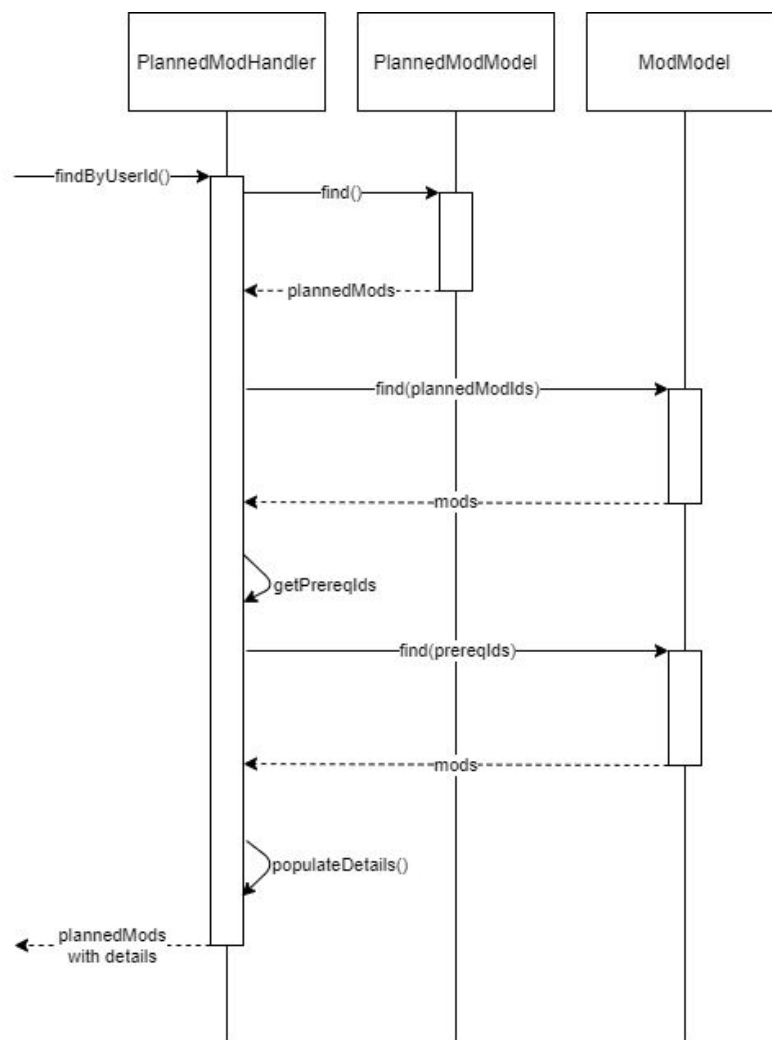
*Fig 9.24 Activity Diagram of decisions made by the Router*

When the Router receives a request from the client, it adapts the raw request by retrieving only the request body, path and query parameters, path and method, ensuring that nothing else is passed to the handlers. Next, it switches on the request method. For the illustration above, if the method is a GET and the parameter *id* is present in the path, the Router calls the

*GetObjectById()* method of the handler. If the method succeeds, a success response is created and the response is sent to the client. Otherwise, the client receives an error response.

### 15.3.4. Handlers

Handlers are functions that are composed of multiple functions and also handle the parsing of the request body, if necessary. When the Handler function is called, it first calls the Maker function to transform, validate and normalize the body of the request, if the body is present. Afterwards, it parses the body and retrieves the appropriate parameters, before passing it to the functions of the Model. It then returns the results it receives from the Model to the Router.



*Fig 9.25 Sequence Diagram of the PlannedMod Handler*

For example, to retrieve a list of modules from the user's study plan, the method *findByUserId()* from the **PlannedModHandler** is called. First, it calls the *find* method of the **PlannedModModel** in order to get a list of planned modules. In addition to returning a list of planned modules, the planned modules also have to be populated with the module data and the module data of each planned module's prerequisites. Therefore, the *findByUserId()* function next gets a list of modules corresponding to each planned module in the list of planned modules. A list of prerequisite ids is obtained from this list of modules, and the originally planned module list is populated.

### 15.3.5. Makers

As briefly described in the architecture overview, Makers are composite functions consisting of three functions:

*validate()*: The validation function takes in a request body and ensures that they meet specific criteria. Firstly, it checks for the required parameters. If a required parameter is missing, an error is thrown. Then, it checks that the parameters that are present meet certain criteria.

*transform()*: The transform function takes in a request body and transforms some parameters if necessary. It then returns the request body with the transformed parameters.

*normalize()*: The normalize function normalizes parameters in the body and returns the request body with the normalized parameters.

The Maker function then returns a validated, transformed and normalized request body to the Handler.



For example, when *makeUser()*, a Maker function, is called, the user information in the request body is validated. The validator checks that the name, email and password are present. It also performs the following checks:

Parameter	Check
email	Checks if the email ends with .edu to verify that a user is from an academic institution
schoolId	Checks if the school specified by the identifier exists
courseId	Checks if the course specified by the identifier exists
schoolStartDate	Checks if the date is a valid date string
currentYear	Checks if the year is a number between 1 and 8 (8 years being the maximum number of years of study)

*Fig 9.26 Checks for Parameters Table*

If all checks pass, the transform function is called. In order to secure the password, hashing is used and only the hashed password is stored in the database. When the user logs in, the password they provide is hashed with the same hash function and compared with the stored hashed password in the database. The transform function thus hashes the password and replaces the original password with its hashed version.

Finally, the normalize function normalizes the email to contain lowercase letters only, as emails are not case sensitive. This prevents duplicate emails from being created.

### 15.3.6. Models

Models are constructors provided by Mongoose which aid in the creation and reading of documents from MongoDB. To create a model, a document schema must first be defined. The schema contains the fields of a document and the types of each field. Mongoose provides wrapper functions for MongoDB's native methods.

For example, in order to retrieve all Schools from the database, *SchoolModel.find()* is called, with an empty JSON object - {}.

### 15.3.7. Observers

An observer, as the name suggests, implements the observer pattern for Events, Ratings and Reactions. These three are similar in a way that they can be added to various types in the application. For example, a School, Course and Module can have a rating that reflects how much they are liked by students. These three, along with Reviews, can have *click events* that serve as logs of when a user clicks the object. These events can then be used in current and future implementations of the RottenMods recommendation algorithm.

For illustration, consider when a user clicks the *like* button on a module. A request is made and handled by the *ReactionRouter* and *ReactionHandler*. After adding the reaction object to the database, the *ReactionHandler* then calls the *notify()* function from the *ReactionObserver*. The *ReactionObserver* file contains a list of subscribers, namely the Module, School, Course, Review and Reply models. Whenever a *notify()* function is called, the list of subscribers is checked and if the *subType* - subscription type - of the *Reaction* object is found to match a subscriber, such as Module, the Module document will be retrieved using the *subId* of the *Reaction* object. The Module document will then be modified accordingly and in this case, its *like reaction* count will be increased by one.

The use of the observer pattern in this situation allows Ratings, Events and Reactions to be easily added to any new object in the application. Additionally, the values stored in the documents themselves can be updated automatically on any creation, update or deletion of a Rating, Event or Reaction.

### 15.3.8. Modules, Helpers and Validators

These components are files containing helper functions that can be called from anywhere.

#### 15.3.8.1. Modules

Modules contain complex helper functions such as those that produce module recommendations for a user.

#### 15.3.8.2. Helpers

Helpers contain simple helper functions such as the hashing of a password.

#### 15.3.8.3. Validators

Validators contain simple validation functions such as validating an email string using a regular expression.

## 15.4. Recommendations

### 15.4.1. User-based Recommendation

Modules are recommended to users through applying the concept of **collaborative filtering**, where a user's interest for a particular module is predicted based on other similar users' interest on the same module. The algorithm operates on the assumption that if User B likes at least one same module as User A, while User C does not like any modules that User A has liked, User A is more likely to be interested in a module that User B has liked rather than one

of User C's. To create a collaborative filtering algorithm, the **Jaccard Index** is used to evaluate the similarity between users. The equation for the Jaccard Index is as follows:

$$\text{Jaccard Index (User 1, User 2)} = \frac{|\text{Modules liked by User 1} \cap \text{Modules liked by User 2}|}{|\text{Modules liked by User 1} \cup \text{Modules liked by User 2}|}$$

*Fig 9.27 Equation for Jaccard Index*

A Jaccard Index closer to 1 means that User 1 has largely similar interests to User 2, while an index closer to 0 means that the two users bear little similarities based on the modules liked by each other. During the computation of recommendations, modules of User B who has a higher Jaccard Index with User A would be recommended rather than those of User C who has a lower Jaccard Index with User A.

The overall recommendation index is then computed using the Jaccard Index. To find modules suitable to recommend to a particular user, for every module in the database, the Jaccard Index between the user and every user who liked the module is computed. The average of the indexes returns the recommendation index for the user for the module. The equation used to generate the recommendation index is as follows:

$$\text{Recommendation Index (User, Module)} = \frac{\sum \text{Jaccard Index (User, User who liked module)}}{\text{Number of users who liked module}}$$

*Fig 9.28 Recommendation Index Equation*

The recommendation indexes computed for each module are then sorted in descending order, where similar to the Jaccard Index, a recommendation index closer to 1 would imply that the user is more likely to be interested in the module, and the reverse for a recommendation index closer to 0.

#### 15.4.1.1. Advantages

With the assumption that similar users would like similar modules, the implemented algorithm ensures that modules are recommended to the user based on logical computations. In addition, modules that return a recommendation index of 0, meaning that there were no similar users who liked the module, would be filtered before the list of recommendations is returned. This ensures that only relevant modules would be recommended to the users.

The algorithm is also **flexible** and can be applied without the need for any changes when there are new modules or new users in the system. Since the algorithm considers all modules in the system for the generation of recommendation indexes, new modules are automatically included and would be recommended as long as the modules were liked by some users.

When considering future extensions of the web application, the algorithm can be **easily reused** to compute similarity based on other reasonable criteria apart from likes for a module, such as module views or ratings. Little modification would be required when applying the existing algorithm for new features of the web application.

As the algorithm is based on the user interactions in the web application, it is expected that the recommendations would become more refined and accurate with increased user activity.

#### 15.4.1.2. Limitations

One observed limitation of the algorithm is the preference towards older modules compared to newer modules. Considering that older modules in the database may have a higher number of likes than new modules, older modules have a higher tendency and probability to be recommended over newer modules in the database.

As a collaborative filtering algorithm, the algorithm is also largely dependent on the existing databases of users and modules. As such, in the case where there are no liked modules in the database, the recommendation index could not be generated. In addition, if the target user for a recommendation has not liked any modules in the database, the algorithm is unable to generate recommendations since the user would bear no similarities to other users in the database.

While there were limitations observed, these limitations are mostly relevant to when the web application is newly released with a lower amount of user traffic, where it may be difficult to generate recommendations considering the possibly low number of module likes recorded. However, the recommendation algorithm would only become more refined and reliable over time when the web application has received a reasonable amount of user traffic. Considering that RottenMods is meant to be used on a regular basis by a large demographic, the limitations would be mitigated over time as the database of users grows.

#### **15.4.1.3. Computation Time**

To generate the recommendations, the algorithm iterates through all modules currently available in the database and performs Mongoose *aggregation pipeline* functions to compute the recommendation index for each module. As such, the time complexity of the algorithm is  $O(n) * \text{time complexity of aggregations}$ . When there is a large number of modules in the database, the computation required to generate the recommendations increases significantly. When tested with 5687 modules in the web application, the computation time is on average 10 seconds per computation.

## 15.4.2. Other Algorithms

Modules are also recommended based on overall statistics for each module. Recommendations are generated by assessing a module's total rating and view count, as well as its average rating.

### 15.4.2.1. Trending: Recommendation of Most Viewed Modules

In the case of this project, trending modules refer to modules that were most frequently viewed in the database. Modules are recommended to the user based on the total number of view counts for each module, where the top 10 most viewed modules would be recommended. The algorithm filters the view events by module and counts the total number of views received per module. This result is then sorted and limited to 10 module entries before returning.

This algorithm assumes that users would be more interested in modules with higher view counts as compared to those with lower view counts.

### 15.4.2.2. Popular: Recommendation of Most Rated Modules

In the case of this project, popular modules refer to modules that were most frequently rated in the database. Modules are recommended to the user based on the total number of ratings for each module, where the top 10 most rated modules would be recommended. The algorithm filters the rating events by module and counts the total number of ratings received per module. This result is then sorted and limited to 10 module entries before returning.

This algorithm assumes that users would be more interested in modules with higher number ratings as compared to those with a lower number of ratings.

#### **15.4.2.3. Top: Recommendation of Top Rated Modules**

In the case of this project, top modules refer to modules that have the highest ratings in the database. Modules are recommended to the user based on the average rating of each module, where the top 10 top-rated modules would be recommended. The algorithm filters the rating events by module and computes the average ratings received by the modules. The average is computed by dividing the total sum of rating values by the number of ratings received for the module. This result is then sorted and limited to 10 module entries before returning.

This algorithm assumes that users would be more interested in modules with higher average ratings as compared to those with lower average ratings.

#### **15.4.2.4. Advantages**

As these algorithms are based on the overall statistics for each module, they are highly responsive to user interactions on the web application, where an increase in views or ratings for a particular module would result in the module being reflected in the recommendations after evaluation. In comparison to the user-based recommendation algorithm, these algorithms may be useful in helping users discover new modules that they previously may not have considered based on their preferences and interests.

While there are only three such algorithms utilised in the web application, they can be easily modified and reused to retrieve recommendations based on other statistics, such as the number of reviews or likes. When there are extensions made to RottenMods, the algorithms could be applied to generate recommendations for newly implemented criteria.



#### 15.4.2.5. Limitations

As the recommendations are not user-specific, they may not be entirely relevant to every user in the database. These recommendations are mainly used to show users modules that are frequently accessed and rated by users, and not tailored to each user's module preferences. While the above-mentioned user-based recommendation algorithm is used to specifically match modules to a user's interests, these generic recommendation algorithms serve more for users who are interested to find modules that other users are viewing and rating.

Reviewing the algorithm to recommend modules based on their average ratings, it can be observed that modules with a small number of ratings but high rating values would be preferred over modules with a large number of ratings but low rating values. For example, if Module A has only one rating of value 5, it would be highly recommended in comparison to Module B with 10 ratings of values ranging from 4 to 5. Even though Module B received consistently high ratings from users, it would be less recommended as compared to Module A based on the algorithm. As such, the recommendations would not reflect an objective view of modules that are popular among users.

Similar to the user-based algorithm, the recommendations generated by these algorithms would be refined over time when there are more ratings and views by users in the database. As such, while the recommendations may be inaccurate at the start of the web application, the accuracies would improve with usage.

#### 15.4.2.6. Computation Time

As the algorithms are based on Mongoose aggregation pipelines, results are easily and quickly retrieved from the database. The average computation time to generate the recommendations is 2 seconds.

#### 15.4.3. Optimizations

In order to achieve the response times specified in the non-functional requirements, optimization had to be done. Due to limited time and knowledge to implement more optimal algorithms, the system used cron jobs in order to ensure that recommendation response times were no different from other responses.

A cron job is a time-based job scheduler and for RottenMods, runs the recommendation algorithms at 0200 and 0300 SGT (GMT +8) every day. At 0200, the general recommendations are computed. At 0300, the recommendations based on similarity are computed. These timings were chosen as RottenMods is mainly meant for Singaporeans and it is likely that traffic would be at the lowest at these durations. Once the recommendations are computed, they are stored in the database. When a request is made to get a recommendation, the stored recommendation for the day is retrieved and returned.

In making these decisions, the main two criteria considered was the speed of response and accuracy of responses. By computing recommendations only once a day, the results returned to the user would not have the same degree of accuracy as if the recommendations were computed at the point of request. However, this was deemed to be insignificant as it is unlikely that the amount of new data collected in a day would result in noticeable changes to the recommendation results. On the other hand, if requests for recommendations took 10s

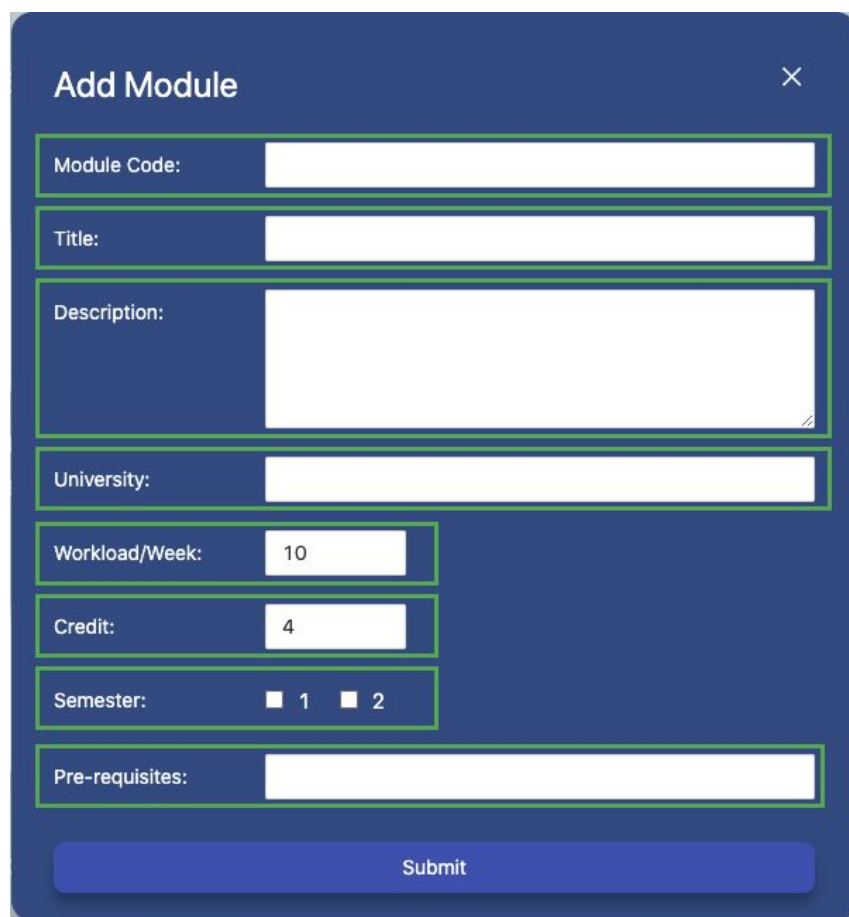
to complete, the user would most likely believe that a bug was encountered and leave the site.

## 15.5. Other Design Considerations

### 15.5.1. Frontend Design Considerations

#### 15.5.1.1. FormModalItem.tsx

In the earlier section 9.2.4, we talked about how Form Modals are abstracted by the underlying FormModal and FormModalItem components. Different Form Modals are created using the FormModal component as a base and then inserting the specific FormModalItems needed to create a customized Form Modal.



The image shows a 'Add Module' form modal. It has a dark blue header with the title 'Add Module' and a close button (X). The form contains several input fields: 'Module Code:', 'Title:', 'Description:' (a larger text area), 'University:', 'Workload/Week:' (with the value '10'), 'Credit:' (with the value '4'), 'Semester:' (with radio buttons for '1' and '2'), and 'Pre-requisites:'. A 'Submit' button is at the bottom.

*Fig 9.29 AddModuleModal encompassing the many different FormModalItems*

The components highlighted in green boxes are actually the one same component: `FormModalItem`. While we discussed earlier that `FormModal` itself abides by the Open-Closed Principle, `FormModalItem` was deliberately chosen to do the opposite; open for modifications, closed for extensions. There were 2 options to choose from:

### Option 1

Create a base `FormModalItem` that keeps the format of the overall component consistent, and take in children components and render them accordingly. (Similar to the design of `FormModal`). Then create different components that use `FormModalItem` as the base component, and render specific form input UI elements inside `FormModalItem`.



*Fig 9.30: Example Components and how their code would look like*

As we can see from the example above, we can “extend” `FormModalItem` into many different variations, customized to specific needs.

## Option 2 (chosen for RottenMods)

Contain all the different types of input UI elements inside FormModalItem, and pass into it a "type" property to determine the UI element to be rendered.

```
const renderInputType = {
  "input": renderInput,
  "textarea": renderTextArea,
  "text": renderText,
  "rate": renderRate,
  "difficulty": renderDifficulty,
  "number": renderNumber,
  "year": renderYear,
  "semester": renderSemester,
  "semesters": renderSemesters,
  "annualYear": renderAnnualYear,
  "university": renderUniversity,
  "autocomplete": renderAutoComplete,
  "prereq": renderPrereq,
}

+

const renderTextArea = () => (
  <TextArea
    style={styles.input}
    rows={4}
    autoSize={{ minRows: 4, maxRows: 4 }}
    value={value as string}
    onChange={(e) => setValue(e.target.value)}
  />
);
```

*Fig 9.31 Different UI render methods in FormModalItem, with an example of how to render a Text Area.*

Objects in Javascript are technically Hashes that allow  $O(1)$  average time for lookup of a key. By storing the "type" as the key, and the render method of the UI element as the value, there is little to no performance difference between Option 1 and 2 when rendering the different types of FormModalItem. However, this violates the Open-Closed Principle, since FormModalItem is now open for modifications so that we must add new input UI elements in FormModalItem.tsx itself.

To justify our decision, we felt that there are many different types of input UI elements available, and if we were to abstract all of them into their own Component based off FormModalItem, there would be too many files to maintain. Many of these input UI elements are very reusable, which makes it very hard for the developer to look through each different Component to determine what and how it renders.

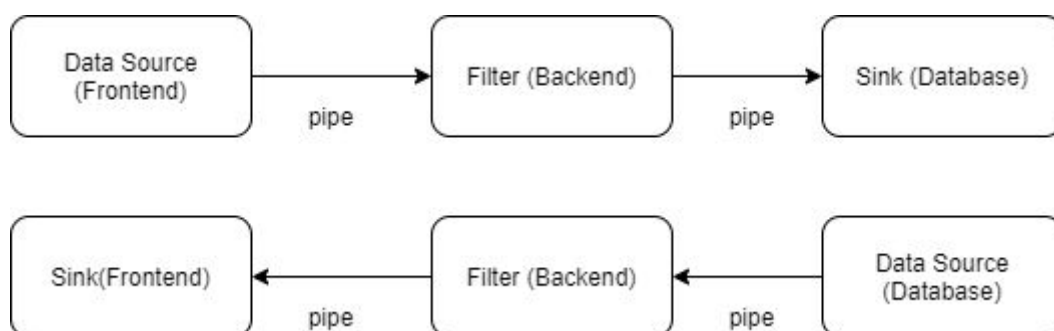
Instead, by having all the render methods of the different UI elements in `FormModallItem`, the developer can easily scroll through them to find the one that suits his needs. Of course, if there are hundreds of variations for input methods, this may not be feasible too. However, RottenMods specifically only require ~15 of such elements, and at this range of numbers, we feel that Option 2 is best for current practice.

## 15.5.2. Backend Design Considerations

### 15.5.2.1. Functional over Object-Oriented

JavaScript classes were introduced with ECMAScript 2015 (ES6) which allowed a JavaScript application to use object-oriented programming. One of the important decisions at the start was whether to use classes or functions. Aside from Mongoose “Model” objects which were used in order to manipulate the objects in the database, functions were used for everything else.

The backend performs three primary functions. First, it receives data from requests made by the client. Secondly, it processes the data it receives or sends. Lastly, it sends the processed data back to the client or stores it in the database. The first consideration, thus, was whether there was a need for classes at all. Object-oriented programming is mainly based on the idea of storing data and code in “objects”. As much of what the backend does is data processing, there was no need for the objects provided by OOP. Instead, having identified a pipe-and-filter model in the backend architecture, it was concluded that the composition of functions using functional programming would be more appropriate.



*Fig 9.32 Pipe and Filter Model*



The second consideration was adherence to software principles and, more specifically, the SOLID principles of software design for OOP. The concern was whether these principles could be easily adhered to in functional programming. Ultimately, it was decided that they could and have listed the reasons in the table below.

<b>SOLID Principles in Functional Programming</b>	
Single-Responsibility Principle (SRP)	FP can achieve SRP by ensuring that each function is small and does only one thing.
Open-Closed Principle (OCP)	FP may achieve the OCP by the use of function composition and higher-order functions.
Liskov Substitution Principle (LSP)	FP achieves LSP by the use of generic types. This allows inputs of different types to be substitutable for each other in a function which takes in generic parameters.
Interface Segregation Principle (ISP)	FP achieves ISP by the modularity of functions. It is simple to remove an unused function from a composition.
Dependency Inversion Principle (DIP)	FP achieves DIP by default. Functions are abstractions and the lack of classes removes the possibility of “concretions” anyway.

*Fig 9.33 SOLID Principles Table*

In view of the above considerations, functional programming was chosen over object-oriented programming for the backend.

### 15.5.2.2. NoSQL over SQL Database

Two main categories of databases were considered for our implementation - SQL databases and NoSQL databases. SQL stands for Structured Query Language while NoSQL simply means non-SQL. The following table lists the relevant differences for the application between the two categories.

SQL databases are relational and table-based. In SQL databases, pieces of data are stored as rows in a table. In contrast, NoSQL databases are non-relational and document-based. In NoSQL databases, pieces of data are stored as JSON-like documents. The various criteria considered for this decision include flexibility, scalability and ease of use.

In terms of flexibility, NoSQL databases edge out SQL databases in allowing data models with flexible schemas. This means that when a new property is required in, for example, the User Model, simply updating the schema of the model would be sufficient. In contrast, SQL databases use rigid data models which are highly resistant to change. Much more work needs to be done in order to add a new property. For the purposes of RottenMods, a tight development schedule meant that the extra time incurred by this action, multiplied by the number of times the action had to be taken, was not viable.

Furthermore, NoSQL databases offer better horizontal scalability. Specifically, MongoDB supports horizontal scaling by a process called “Sharding”. This process distributes data over multiple machines and simply adding shards to a collection can scale the database horizontally. This distribution of data also results in NoSQL databases having multiple points of failure. In contrast, SQL databases do not have the capacity to easily scale horizontally. The ease of scaling that NoSQL databases provide is beneficial for web applications like RottenMods and the multiple points of failure help reduce the amount of downtime the site goes through.

Finally, NoSQL databases, as seen in the context of RottenMods, are easier to use. In a NoSQL database document, multi-level nesting is supported and allows for easier storing and fetching of data. For example, to retrieve all the module identifiers tied to a user recommendation, only a single Recommendation document containing an array of module identifiers would need to be retrieved. The document can then be directly sent to the client. In a SQL database, however, each module identifier tied to a user recommendation would be stored as one line of data and all would need to be retrieved. Further work would have to be done to transform the data before sending it to the client.

### 15.5.2.3. Monolith over Microservices

A monolithic application is an application with only one tier, a single-unit of software. A microservices architecture, on the other hand, specifies a collection of small, independent applications communicating through light-weight communications mechanisms. A comparison of these two architectural styles are made in the table below

Criteria	Monolithic	Microservices
Scalability	Difficult to scale as components cannot be scaled independently. Scaling difficulty increases as the application get larger and components become more tightly coupled	Easy to scale as each microservice contains a small application independent from the rest of the system. Additional functionality can be added by adding another independent microservice. Loose coupling
Ease of Deployment	Only one deployment per application	Many deployments per application
Ease of Development	Easy to develop as the monolithic approach has been the default way of building applications	Difficult to develop as many of the technologies required for the development of microservices are less understood

*Fig 9.34 Comparison of Architectural Styles Table*

For the development of RottenMods, a monolithic architecture was chosen. This was due to the combined constraints of time, manpower and knowledge. The development time for the application was approximately 6 weeks, leaving not much time to learn and implement a working application using a microservices architecture. In addition, DevOps was handled by only one person in the team. Finally, the team did not have any pre-existing knowledge of implementing microservices prior to the development of this application.

Furthermore, as RottenMods serves only as a simple module review website with little business logic and small scaling potential, a monolithic architecture sufficed.

## 16. API Documentation

The API documentation is available [here](#).

## 17. Deployment

The site was deployed using [Amazon Elastic Compute Cloud](#). There were a number of web hosting options available but the choices were narrowed down to Google Cloud and Amazon Web Services (AWS) due to their widespread use in the industry and the flexibility offered. In the end, AWS was chosen over Google Cloud as AWS provides a free tier, which allows for hosting up to a certain limit. In addition, AWS was felt to be easier to use.

Nginx was used as the HTTP server and reverse proxy.

The frontend, backend and database all live on the same server.

Component	Port
Frontend	3000
Backend	8080
Database	27017

*Fig 11.1 Component Port Table*

The frontend, backend and cron job processes are managed by PM2, a node process manager.

The site can be accessed [here](#).

## 18. Possible Enhancements

The current implementation of RottenMods can be further enhanced to provide a more comprehensive platform for university students by repurposing or extending certain features.

### 18.1. Academic Study Planner

Instead of only allowing users to save modules that they have taken before in their study plan, we can implement a complete academic planner that allows users to plan what modules that they intend to take in the future. Our current implementation allows extension for this since it already collects and displays useful information that a user needs to decide what modules they should take in the future.

Here is a mapping of some of our current implemented features and why they can be extended for integrating this potential enhancement.

Current Implementation	Planning for future modules
Reviews(expected workload, difficulty ratings, comments and discussion)	With this information, the user can see the real feedback from other students that have taken the particular module before to decide whether it fits their learning expectations.
Official module information(pre-requisites, module description)	While adding modules into the academic planner, they can easily see whether they are eligible to take a module at a particular semester(since some modules are available only in one semester). They can also plan the order to take their modules based on their prerequisites.
Module recommendation system	Similar modules can be recommended based on what students intend to add to their academic planner. This allows students to discover modules that they have no idea about.

*Fig 12.1 Current Implementation Extensions Table*

## **18.2. University Reviews**

This possible enhancement will also benefit a different target group of the audience while taking advantage of our current implementation. This target group will be pre-university students deciding which university that they want to enrol into.

Apart from collecting information from official university ranking websites like QS World University Rankings, our enhancement will extend our current implementation by allowing users to also leave reviews for a university. Pre-university students will be able to see what others are saying about a particular university, and also go further and see what others are saying about core modules that they will take if they enrol there.

Current university students can benefit from this as well, allowing them an alternative platform to see which universities they should visit for their exchange programme and plan their mapping modules for that university.

## **18.3. Others**

We discuss more possible enhancements briefly below.

### **18.3.1. Direct Messaging**

Our current implementation also allows for students to discuss using comments on reviews posted on a module page. However, students might find it more meaningful to directly reach out to another student to privately discuss their thoughts on a particular module through direct messaging.



### **18.3.2. Multi-Module Comparison**

Our current implementation only allows for a one-page comparison of 2 modules. However, students may find it more helpful to be able to compare up to 5 modules at one point in time since they are usually more than 2 modules in a particular field that they may be interested in.

### **18.3.3. Public Profiles**

Our current implementation only allows a user to manage and see their own study plans, bookmarks and reviews they wrote. However, students may want to reference seniors' study plan to get a better idea of how to plan their own study plan or see how other users have reviewed other modules if the current user chanced upon a good review written by these other users.