RS Handover

About

This document serves as a formal handover report of the project to another development team to take over and continue if needed.

Background

The RS project is a repository of learning resources for Melbourne University CIS Engineering students to self-learn about important emerging technologies such as programming frameworks, testing tools, and deployment tools. The product aims to provide a convenient and easy-to-use social learning platform as a learning community by sharing, discussing, and ranking the resources provided either by teaching staff or a student related to CIS subjects. Students need to login to get access to the resources and need permission to share a new resource. However, active participants will get rewards and possibly escalate to a moderator. The platform encourages interaction and communication among students by actively contributing to the platform.

System Implementation

Existing RS system

This learning resources system/platform was initially developed by the RS team from the subject SWEN90013 of the University of Melbourne in 2020, with the purpose to help university students to self-learn the important emerging technologies such as programming frameworks, testing tools, and deployment tools. The structure of this existing system can be separated into three components, which are front-end using React framework, back-end using node /express, and MongoDB for the database. The front-end and back-end deployment is supported by yarn and npm.

The existing functionalities of the system include

- Login
- Register account
- Account management
- Create new subjects
- Add sections to subjects
- Delete sections
- View pending articles list
- View subject details
- Create new articles
- Edit existing articles
- Add tags to articles
- Review pending articles
- Approve pending articles
- Reject pending articles
- Search articles by keywords
- Search articles by tags
- Comment an article
- Delete own comments
- Comment a comment
- Show homepage
- Show profile

Achieved RS System

As students from the subject COMP90082 software project in the first semester of 2021, we are responsible for maintaining and improving this system. For the maintenance aspect, we plan to thoroughly test the system to measure and validate the performance as well as stability and scalability. For the improvement aspect, we decided to enhance the UI design to improve the user experience and add more features to the existing functions so the system is easy to use and interact with. In addition, we also agree to provide more learning resources by adding more subject-related content into the system.

The additional features we added during this semester:

For Front-end:

- Allow users to edit personal information
- Add search filter in search page
- Add FAQ page to help users to get familiar with the system quickly
- Add Like and Bookmark buttons on the article page
- Fixed displaying subjects issue on student home page
- Improved page design and visual hierarchy:
    - Back button in register and article page
    - Cancel button in create article page and editor page
    - Breadcrumb in login and register pages
    - Updated button color and fixed button size

        - Increased font-weight to elements like titles and labels
        - Added margins and line spaces to different elements on one page
        - Updated hint messages for users' inputs
        - Added username in profile page
    - Code optimization
    - Completed user experience test

For Back-end:

- Organized the internal structure of the system
    - Simplified the structure of the source code
    - Eliminated Duplicated connection with database
    - Conducted manual tests and created a document where available API are listed and explained
- New features
    - Can use environment variables to decide the code to be executed based on different options (User Story 10.6)
        - Development: used for the purpose of development.
        - Test: run automation test under this model. This can be used for CI to run
        - Production: used for operation. Connecting to the real database.
    - Implemented automation test scripts for some RESTful API (User story 10.5)
        - Integration tests: login, register, and subject
- Like/Unlike feature in backend
    - Can set "liked" by sending a request via API PATCH "[host]/article/liked"
    - Can set "unliked" by sending a request via API PATCH "[host]/article/unliked"
- Consolidate API-level validation (validating the request body for POST method) in the backend part
    - Register a new student
    - Add a new subject
    - Add a new article
    - Sign in to the system.
- Student user can access all subjects
    - A newly registered student user can see all subjects
    - A newly added subject will be available by all existed student users

For Database and Deployment:

- Deployed Back-end on Oracle cloud
- Deployed Front-end on Oracle cloud
- Deployed database on MongoDB cloud
- CI/CD
    - GitHub Actions script
        - Rebuild on Commit/Pull Request
    - Server CI/CD API (in Python)
        - Service status
        - Service restart
        - Update from GitHub & Rebuild
- One-click deployment bash script
    - Install all the required dependencies and deploy the Backend service & CI/CD API
- Auto installation works for Nginx/Frontend, Backend, MongoDB, and CICD API.
    - `sudo bash -c "$(curl -fsSL https://raw.githubusercontent.com/ccarner/COMP90082-RS-2021/master/one-click-setup.sh)"`
    - Designed for & tested on **NeCTAR Ubuntu 20.04 LTS (Focal) amd64**.
- TLS/SSL now configured for all services (not automatically, guides provided in Confluence).
- Nginx reverse proxy configured.
    - All traffics go through Nginx now (with SSL)
- Database
    - Now locally deployed. (MongoDB cloud instance is preserved for data dumping).
    - Schema Validation.
    - Preliminary implementation of a Caching service: Redis.
        - The feature will not be part of the final release.
        - API/Docs provided in Confluence

Scope

In scope

The learning resource system aims to help CIS students at the University of Melbourne to obtain knowledge on emerging technologies such as programming frameworks, testing tools, and deployment tools. Users of this system can access subject-related resources that explain new tools and programming languages clearly. Apart from the contents that help users directly, it is also essential for the system to offer a user-friendly user interface and layout so that users can utilise the functions and access the contents more efficiently and conveniently.

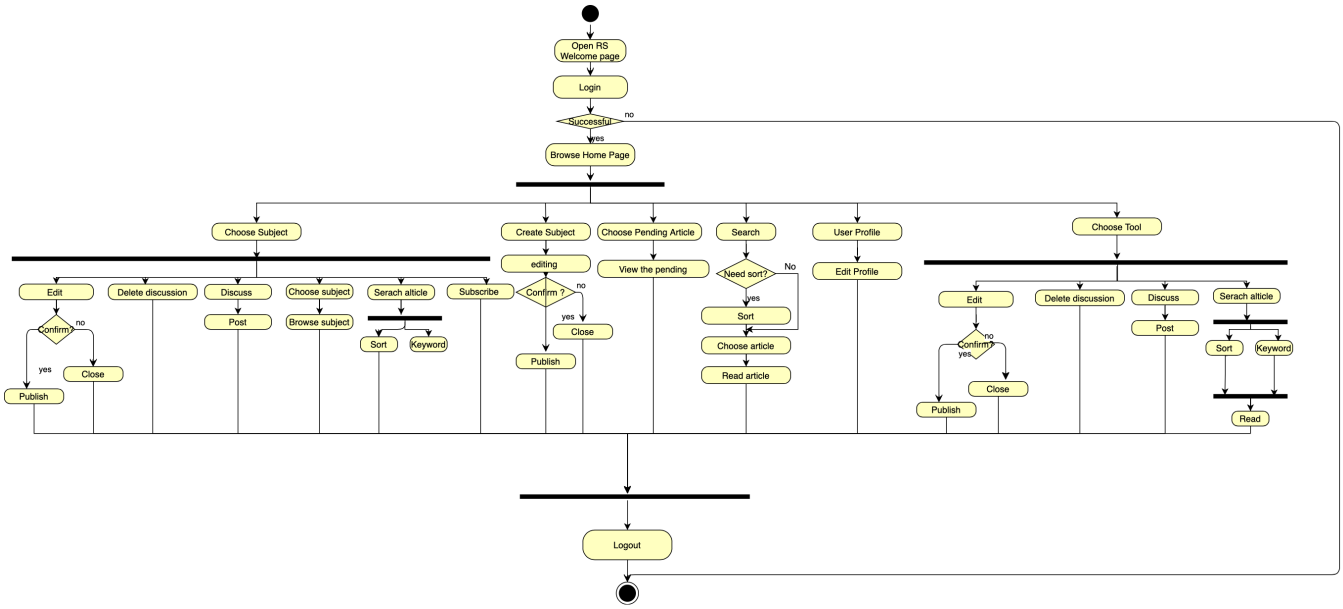
Out scope

The learning resource system is a platform only for CIS teaching staff and engineering students at the University of Melbourne to share and gain knowledge. It will be out of scope if it becomes accessible to other schools or the general public users.

Architecture

Activity diagram

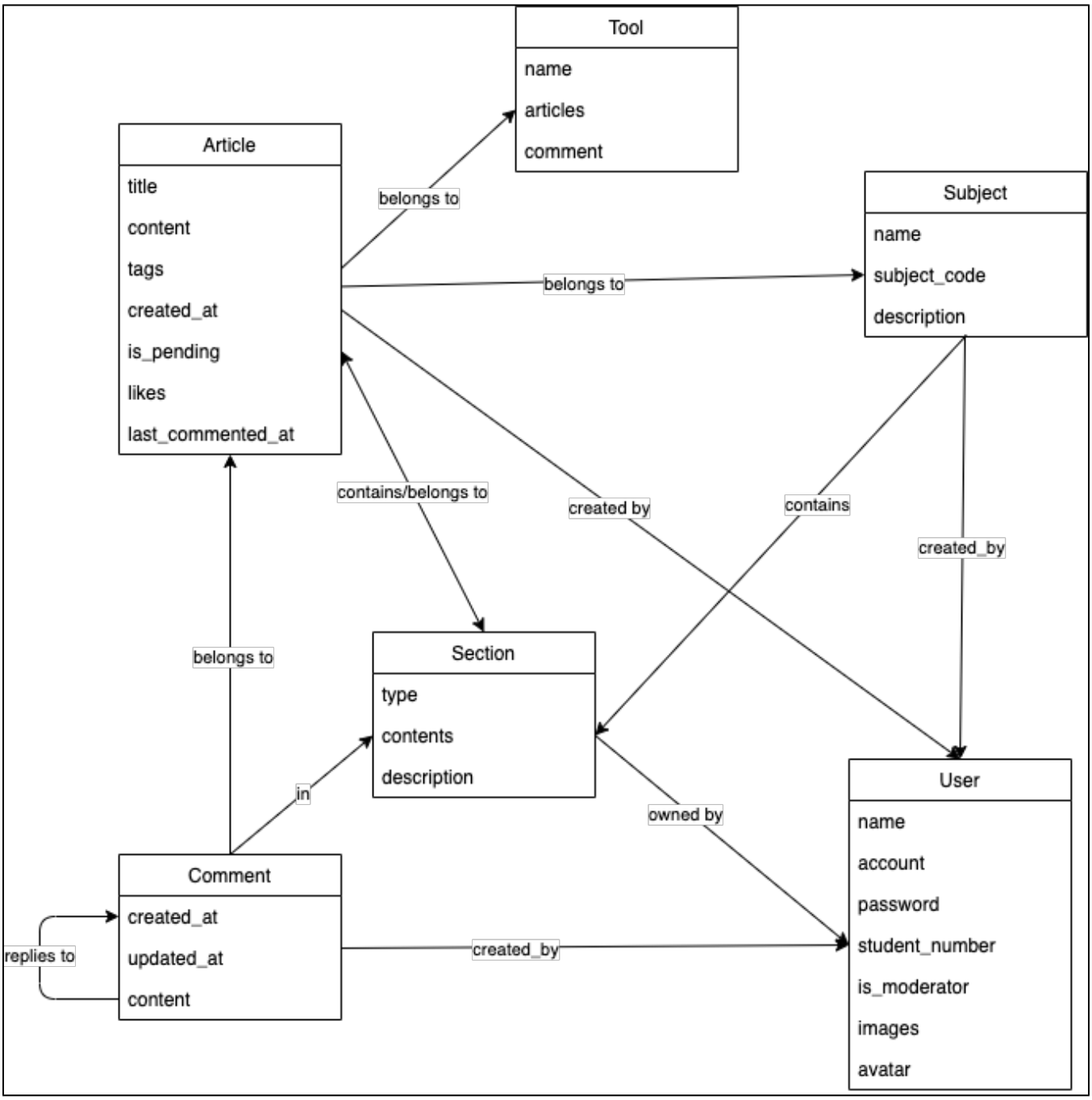
The following figure is the activity diagram of the RS system established based on the digital prototype and user story. Describes the work done during the execution of an operation.



Each user is required to log in with the correct username and password otherwise he cannot access the resource of the RS website. Once the user login successfully, he can choose what he wants to browse, including subjects and tools. The user can search what he wants to access by keyword in the search bar, and then he can sort the result according to some requirements. Besides, each user can edit his profile once he clicks the User Profile in the top-right of the webpage. The most important function of the RS website is to provide a useful resource to students and allow them to discuss freely. User with corresponding right can manage the discussion board once he accesses the Subject Page or Tool Page, and he also can post a new discussion in this section. Additionally, a user can subscribe to a subject that he likes on the subject page. Each article will move to the pending section before they publish, and those articles also can be deleted before they publish in the pending section. Moreover, a user with a legal right can create a new subject, edit it, then choose to publish or delete it.

Domain-layer design

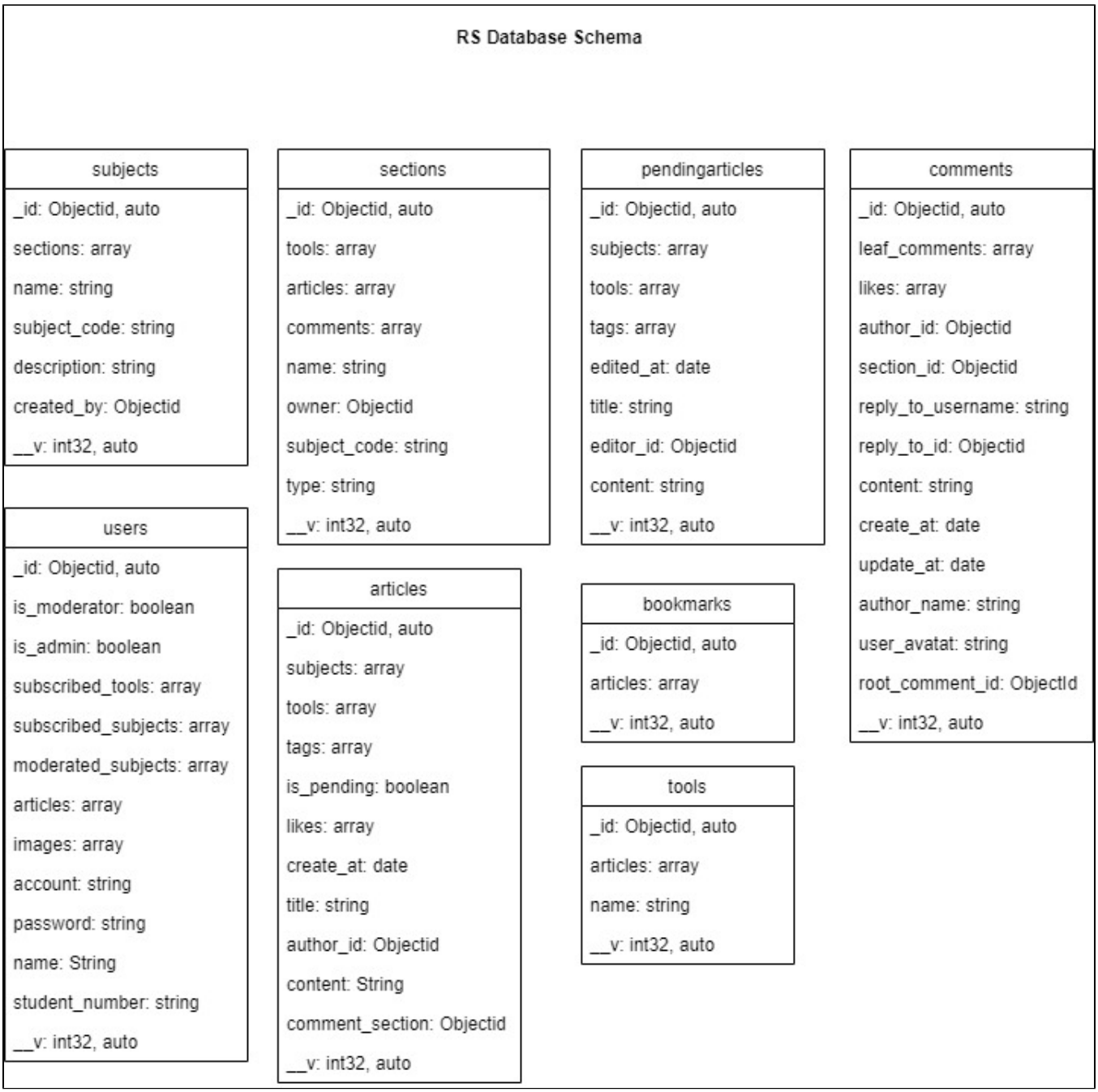
The mongoose framework allows us to treat the data in an object-oriented way. Therefore, we used the **Domain Model** pattern in our domain layer design. The domain model constructs a model of the business domain. It has the advantage of extensibility and reusability and thus matches our needs in the project. Our domain design is shown in the domain diagram below:



We categorized the objects in our system into six classes:

- **User:** A user can be a student, a moderator, or an administrator. Students are normal users of the website. Moderators have the additional responsibility to manage the website (e.g. create subject, approve articles,etc.). And administrators are able to manage the accounts of students and moderators.
- **Subject:** Each subject has its own page which contains different sections. Moderators can create and manage subjects. And students can enroll in a subject.
- **Tools**: Each tool has its own page which contains different sections.
- **Article:** An article can be an article or a pending article. A pending article is an article created by a student and has not been approved by the moderator to publish to the website. An article can be related to a subject or a tool. Tags can be attached to the articles.
- **Section:** Sections can be created by moderators under a subject/tool/article page. A section may be a comment section, an article section, a tool section, etc.
- **Comment:** Comments are contained in sections. They are made by students to comment on articles/subjects/tools.

Database schema



Deployment

Project repository: https://github.com/ccarner/COMP90082-RS-2021

Release tag: COMP90082_2021_RLSE_RS_FINAL

Working site: https://api.cervidae.com.au/

- For admin login, the username and password are both "admin2"
- For student login, you can either register a new account or use the username: student and password: 123456

Deployment overview: Deployment Overview

Deploy Front-end locally:

1. Clone Repo (**frontend/develop**):

```
git clone git@github.com:ccarner/COMP90082-RS-2021.git && cd Front_end
```

2. Update dependencies:

```
npm install
```

3. Start Dev server:

```
npm start
```

4. Open [http://localhost:3000] to view it in the browser. Before running the above command, please make sure your terminal path is under the Front_end folder

Deploy Back-end locally:

1. Clone Repo (**backend/develop**):

```
git clone git@github.com:ccarner/COMP90082-RS-2021.git && cd backend
```

2. Update dependencies:

```
npm install
```

3. Start server (**Unix-like**):

```
./start.sh
```

Start server (**Windows**):

```
npm start
```

Deploy the system online:

> If you are from the **Frontend or Database team**, then **DO NOT** set up your own local backend server, instead use our dev server: https://api.cervidae.com.au/api

More installation guide for online deployment about automated deployment to the production, MongoDB and enable SSL/TLS can be found in Deployment Guides.

Features

1. User

The student and moderators should be able to register an account and login. The administrator should be able to manage users' accounts.

2. Subject

The subject is a major component of the website. The subject page should contain a subject description and the related articles and tools recommand by the website moderators. All users should be able to view the subject pages that they subscribe to. And the moderators should be able to create, delete, edit subjects, add sections to subjects and review all pending articles under each subject.

3. Article

The article is a critical function for the website to thrive. All users can create new articles and edit existing articles to share their ideas. However, any editions made by students need to be reviewed by website moderators, and the moderators may approve or reject those editions. Besides users can attach tags to an article, in that case, other users will be able to search those articles based on their tags. Meanwhile, users will also be able to search articles base on the keywords in their titles.

4. Comment

The comment is a function for users to provide feedback to content in the website. Users should be able to leave comments on articles and reply to other comments.

5. Homepage

After login to the website, users will be directed to their homepage. From their homepage, users should be able to see all subjects enrolled by the user. And for moderators, should also be able to see the subjects moderated by them.

6. Profile

Users should have their own profile pages where they can check all the articles (and pending articles for students) they have created.

Frontend structure

Server *(Front_end/Front_end/src/)*

- app.js: The front end of the application, handles routing to different pages.

Utils *(Front_end/Front_end/src/utils/)*: Contains Utility classes for use by other classes.

- utils/AuthService.js: Performs the authentication to the backend and stores role and authorization details.
- utils/request.js: Formatted http request for use by other classes.
- utils/tools.js: Tools for adding and getting parameters to and from urls.

components *(Front_end/Front_end/src/components/)*: Contains components used on all pages.

- components/Footer.js: The footer used on every page.
- components/NavBar.js: The implementation for the Navbar used on every page.

container *(Front_end/Front_end/src/container/)*: Contains details for every page functionality.

The following sections list these pages inside container folder, presenting in the folder order:

addSection *(Front_end/Front_end/src/container/addSection/)*: Contains class with logic for adding sections.

- addSection/AddSection.jsx: Contains logic for adding sections.

addUserPage *(Front_end/Front_end/src/container/addSection/)*: Contains class with logic for adding users.

- addUserPage/AddUser.jsx: Handles logic for adding sections.

ArticlePage *(Front_end/Front_end/src/container/ArticlePage/)*: Contains the classes for the page for viewing articles.

- ArticlePage/Article.jsx: Handles the container for the viewer and associated buttons.
- ArticlePage/EditorComponent.jsx: Actual reader component, set up for read only access.
- ArticlePage/LikeButton.jsx: users can click the button to like or unlike the article.
- ArticlePage/Bookmark.jsx: users can click the bookmark button to bookmark or unbookmark the article.

Comment *(Front_end/Front_end/src/container/Comment/)*: Contains the comment components on pages.

- Comment/Comment.jsx: Handles the container for comment system and post function of users.
- Comment/SubComment.jsx:Represents individual comment item .
- Comment/SubCommentItem.jsx:Replys the comments of other users.

CreateArticlePage *(Front_end/Front_end/src/container/CreateArticlePage/)*: Handles the page for creating articles, set up differently from editing existing articles.

- CreateArticlePage/CreateArticlePage.jsx: Handles the container for the viewer and associated buttons.
- CreateArticlePage/EditorComponent.jsx: Actual reader component, set up for editing.
- CreateArticlePage/tagComponent.jsx: Handles adding and removing tags from articles.

EditArticlePage *(Front_end/Front_end/src/container/EditArticlePage/)*: Contains classes for editing articles.

- EditArticlePage/EditorPage.jsx: Handles the container for the viewer and associated buttons.
- EditArticlePage/EditorComponent.jsx: Actual reader component, set up for editing.
- EditArticlePage/tagComponent.jsx: Handles adding and removing tags from articles.

EditingSectionPage *(Front_end/Front_end/src/container/addSection/)*: Contains class for editing sections.

- EditingSectionPage/EditingSection.jsx: Contains logic for editing sections within subjects.

FAQPage *(Front_end/Front_end/src/container/FAQPage/)*: Contains the page for frequently asked questions (FAQ) to help use get familiar with the system.

- FAQPage/FAQ.jsx: FAQ page contains content to help users get familiar with the system.

HomePage *(Front_end/Front_end/src/container/addSection/)*: Contains the homepage.

- HomePage/Home.jsx: Contains logic for the homepage, displaying subjects for each user.

LoginPage *(Front_end/Front_end/src/container/LoginPage/)*: Contains class that displays the login page.

- LoginPage/Login.jsx: Contains logic for logging in.

PendingPage *(Front_end/Front_end/src/container/PendingPage/)*: Contains classes related to pending articles for moderators.

- PendingPage/detailedPendingPage.jsx: Handles the container for the viewer and associated buttons for a pending article.
- PendingPage/EditorComponent.jsx: Actual reader component, set up for read only.
- PendingPage/pendingArticleComponent.jsx: Component representing a pending article in the list.
- PendingPage/PendingPage.jsx: Displays list of pending articles for a subject, for moderator to approve.

ProfilePage *(Front_end/Front_end/src/container/ProfilePage/)*: Contains classes relating to the profile page, which displays all articles published by the user.

- ProfilePage/EditorComponent.jsx: Actual reader component, set up for editing.
- ProfilePage/ModeratorProfilePage.jsx: Profile page for moderator, Displays list of pending articles with options to approve.
- ProfilePage/ProfilePendingArticleComponent.jsx: Component representing a pending article in the list.
- ProfilePage/StudentProfilePage.jsx: Profile page for students, like moderator page but displaying unapproved articles with option to edit.

RegisterPage *(Front_end/Front_end/src/container/RegisterPage/)*: Contains class for registring a new user.

- RegisterPage/Register.jsx: Displays registration page.

SearchPage *(Front_end/Front_end/src/container/SearchPage/)*: Contains page for searching all articles.

- SearchPage/search.jsx: Displays search options and field.

SubjectPage *(Front_end/Front_end/src/container/SubjectPage/)*: Contains class for displaying details for a subject.

- SubjectPage/Subject.jsx: Displays the articles and details related to a subject.

WelcomePage *(Front_end/Front_end/src/container/WelcomePage/)*: Contains the landing splash page.

- WelcomePage/WelcomePage.jsx: Landing page.

Backend structure

Server *(backend/Back_end/)*

- app.js: Working as a web server, it listens to the request sent by each connected client and be ready to respond to it. It also builds a connection with database to allow data CRUD.

Routers *(backend/Back_end/routers/)*: Defines a series of routers that redirect HTTP requests from the front-end to the corresponding back-end controllers based on the suffix of the request.

- routers/article.js: Redirects the incoming HTTP requests with suffix of “/article” to their corresponding functions in the article controller.
- routers/auth.js: Redirects the incoming HTTP requests with suffix of “/article” to their corresponding functions in the article controller.
- routers/comment.js: Redirects the incoming HTTP requests with suffix of “/article” to their corresponding functions in the article controller.
- routers/search.js: Redirects the incoming HTTP requests with suffix of “/article” to their corresponding functions in the article controller.
- routers/section.js: Redirects the incoming HTTP requests with suffix of “/article” to their corresponding functions in the article controller.
- routers/subject.js: Redirects the incoming HTTP requests with suffix of “/article” to their corresponding functions in the article controller.
- routers/tool.js: Redirects the incoming HTTP requests with suffix of “/article” to their corresponding functions in the article controller.
- routers/user.js: Redirects the incoming HTTP requests with suffix of “/article” to their corresponding functions in the article controller.

Middlewares *(backend/Back_end/middlewares/)*: Contains classes that use to verify the identity of request owners and gurantee the security of the website.

- middlewares/auth.js: Contains a function to allow users to login safely, and generate an authentication token that can verity the users’ identity for their following operations.
- middlewares/verifyToken.js:

Controllers *(backend/Back_end/controllers/)*: Defines a series of functions that take the HTTP requests redirected by routers, process the requests, and generate HTTP responds to send back to the front-end.

- controllers/article.js: Contains functions that handles the incoming HTTP requests related to articles. (e.g. publish new articles, approve pending articles, etc.)
- controllers/comment.js: Contains functions that handles the incoming HTTP requests related to comments. (e.g. post new comments of an article, delete or update an existing comment, etc.)
- controllers/search.js: Contains functions that handles the incoming HTTP requests related to search. (e.g. search an article by keywords, etc.)
- controllers/section.js: Contains functions that handles the incoming HTTP requests related to section. (e.g. add an section of article to a subject, add a comment section to an article, etc.)
- controllers/subject.js:Contains functions that handles the incoming HTTP requests related to subjects. (e.g. create or update a subject, etc.)
- controllers/tool.js: Contains functions that handles the incoming HTTP requests related to tools. (e.g. add a new tool, etc.)
- controllers/user.js: Contains functions that handles the incoming HTTP requests related to users. (e.g. user subscribe a subject, etc.)

Proxies *(backend/Back_end/proxies/)*: Each class corresponds to a database document and works as an intermediate layer to provide the necessary functions for the system to communicate with that database.

- proxies/article.js: Provides necessary CRUD function for article documents in mongoDB.
- proxies/comment.js: Provides necessary CRUD function for comment documents in mongoDB.
- proxies/section.js: Provides necessary CRUD function for section documents in mongoDB.
- proxies/subject.js: Provides necessary CRUD function for subject documents in mongoDB.
- proxies/tool.js: Provides necessary CRUD function for tool documents in mongoDB.
- proxies/user.js: Provides necessary CRUD function for user documents in mongoDB.

Database

The database model is defined in the backend/Back_end/models/ folder using the Mongoose framework.

- models/article.js: Defines the Article and the PendingArticle mongoose model.
- models/comment.js: Defines the Comment mongoose model and some functions auto-triggered functions related to the model.
- models/section.js: Defines the Section mongoose model.
- models/subject.js: Defines the Subject mongoose model and some functions auto-triggered functions related to the model.
- models/tool.js: Defines the Tool model and some functions auto-triggered functions related to the model.
- models/user.js: Defines the User mongoose model.

Problem

1. Note that due to the limitation of unimelb database access, the feature of Login via unimelb account is dropped and replaced with the function of "Registe r account".

2. Note that the search bar in the navigation has a minor bug. When using it to search, it will go into the search page and it might require a second search in the search page.

3. Note that the approval function in the admin login has a glitch when approving the articles made by students.

4. Note that the like and bookmark buttons both worked well in the frontend and backend, but they require a linkage between the frontend and backend.

5. Due to the limitation of time, the cache service - Redis only finishes the preliminary implementation, but the API/docs have been documented on Confluence and can be found at Redis.

More information

More useful information can be found below:

Shared Goal Model

Shared User stories

Public Architecture

RS Shared Resources

RS team 1 (Frontend)

RS team 3 (Backend)

RS team 2 (Deployment + DB)

Confluence for the previous project: Home