CS615 Group 3 Project Report

Yosyp Mykhailiv   
*Department of Computer Science*  
*Maynooth University*

Maynooth, County Kildare, Ireland  
yosyp.mykhailiv.2020@mumail.ie Anurag M.A. Babu  
*Department of Computer Science*  
*Maynooth University*Maynooth, County Kildare, Ireland  
anurag.madnayakanahalliara.2020@mumail.ie Akshay S. Menon  
*Department of Computer Science   
Maynooth University*Maynooth, County Kildare, Ireland  
akshay.menon.2020@mumail.ie

# Introduction

The project is a user study tool that incorporates users, tasks, and experiments. The app has authentication and implements 2 user scenarios:

* A researcher logs into the application and creates a task (title, description, link to the platform where the task can be performed)
* A researcher signs into the application after what they create/edit an experiment (title, description) and assign previously created tasks to the experiment

The application consists of 3 main parts:

* Database (MySQL)
* Backend (NodeJS REST API)
* Frontend (React application)

Each of the above parts is described in a separate section with a corresponding title. As a code management tool Git is used and the remote repository is hosted on GitHub.

# Database

As a persistent data storage MySQL database is used. The whole database structure (tables, indexes, keys, etc) except the database itself can be created by running migrations that are part of the backend logic. The detailed guidelines on how to setup the database schema are provided in the “Deployment” section. The key feature of the migrations, which are implemented using “Sequelize ORM” package, is that the final database can be easily changed (for example, PostgreSQL can be used instead of MySQL). However, such a change has not been tested. For more information about the supported databases and other details please refer to the ORM documentation [1].

The database has 4 tables which are described below.

1. Database table “Users”

| **Column name** | **Data type** | **Other** |
| --- | --- | --- |
| id | INT(11) | Primary key, Unique key |
| fullName | VARCHAR(255) |  |
| email | VARCHAR(255) | Unique key |
| password | VARCHAR(255) |  |
| createdAt | DATETIME |  |
| updatedAt | DATETIME |  |

1. Database table “Tasks”

|  |  |  |
| --- | --- | --- |
| **Column name** | **Data type** | **Other** |
| id | INT(11) | Primary key, Unique key |
| title | VARCHAR(255) |  |
| description | TEXT |  |
| link | VARCHAR(255) |  |
| createdAt | DATETIME |  |
| updatedAt | DATETIME |  |

1. Database table “Experiments”

|  |  |  |
| --- | --- | --- |
| **Column name** | **Data type** | **Other** |
| id | INT(11) | Primary key, Unique key |
| status | TINYINT(4) |  |
| title | VARCHAR(255) |  |
| description | TEXT |  |
| createdAt | DATETIME |  |
| updatedAt | DATETIME |  |

1. Database table “ExperimentTasks”

|  |  |  |
| --- | --- | --- |
| **Column name** | **Data type** | **Other** |
| id | INT(11) | Primary key, Unique key |
| experimentId | INT(11) | Foreign Key (Experiments.id), Unique key with taskId |
| taskId | INT(11) | Foreign Key (Tasks.id),  Unique key with experimentId |
| createdAt | DATETIME |  |
| updatedAt | DATETIME |  |

# Backend

Backend part is built using NodeJS and its framework Express.js. The backend is basically only REST API which is consumed by the frontend app. The app implements MVC structure even though the server-side app doesn’t render any views. Let’s look through the project structure to see what is stored where.

|  |  |
| --- | --- |
| **Folder/file name** | **Description** |
| config | The folder holds config example files, for example “db\_example.json” which is an example of the db config. |
| controllers | This folder contains files with the domain logic, for example “tasksController.js” contains all the logic related to the tasks (creation, deletion, update, etc). |
| doc | The folder contains “swagger.js”, the file which configures Swagger API documentation. Also, there are files with “yaml” extension that are reusable API doc components. |
| logs | The folder holds all the log files. A separate log file is created for each day. In addition, it has “logger.js” file which creates the logger that can be configured of course. |
| middlewares | The folder contains middlewares, for example “validateRequest.js” which is used for validating API requests. See its usage in routes. |
| migrations | The folder contains all the migrations., files-scripts that are used for setting up the database structure. |
| models | This folder has ORM models. A separate model is created for each database table. For example, “task.js” contains the model that can be used for creating, fetching, updating task records. |
| react | The folder contains React application, the detailed documentations is available in the “Frontend” section. |
| routes | The folder holds the route files, a separate file is created per each domain. For example, file “task.js” contains all the task related routes. Also, “task.js” contains special comments that are converted to Swagger docs. |
| validation-rules | The folder with validation rules for API endpoints. Rules are grouped by domain folders. For example, the validation rules that are related to task endpoints are stored in the “task” subfolder. General rules, like “paginatioRules.js” are stored in the folder itself. See route files to learn how those rules are used. |
| index.js | Entry point to the application. |

There are several libraries that are quite important for understanding how the application works. Let’s have a closer look at them.

|  |  |
| --- | --- |
| **Package name** | **Description** |
| express | The main package, framework itself. It provides many utilities for working with requests, responses, etc. |
| sequelize,  sequelize-cli,  mysql2 | Group of packages that do all the database related “magic”. These packages provide migrations, and models support and many other features that are not used in the project. Check their docs for more information. |
| winston,  winston-daily-rotate-files | Packages that make logging easy. |
| swagger-ui-express,  swagger-jsdoc | The packages that provide Swagger UI and parse route files for generating documentation. |

The important moment is how the frontend and backend apps are combined. To understand it, the main file, “index.js” should be checked. This files has 2 static routes which actually return build react app entry points and all the dependencies.

# Frontend

The frontend is implemented using a combination of ReactJS and Redux, they are both Javascript libraries used to build user interfaces. Material-UI, another popular React UI framework was used to style and theme our application. The frontend consumes the REST API’s implemented in the backend, presenting the data to the end user in a web application. React is mainly used to build ‘single page’ web applications and is mainly concerned with rendering data on the browser DOM. Therefore, it is used in combination with a number of other libraries for routing, maintaining state, etc. Redux is a library that is used in combination with other client-side scripting libraries to manage the state of the application. The application uses an MVC architecture, let us look through the project structure to gain an overall.

|  |  |
| --- | --- |
| **Folder/file name** | **Description** |
| Public | Contains the index.html which is the starting point for the client-side application. It contains the main container through which react renders the DOM. It can be edited to set the title of the applications and setting the icons. |
| package. json | Contains the list of dependencies that are needed to run the application. Using the ‘npm install’ command installs all the required dependencies |
| . env | Contains the environment variable which are accessible throughout the lifecycle of the application |
| src | The scripts for the operational logic is contained in this folder. A basic breakdown of the contents in this folder are given below |
| src/index.js | Is where the react application starts it’s execution from. The base component for the application is loaded into this file. |
| src/components | The components which contain the operational logic and various features of the application. For example components/experiment contains the files which handles the operational logic for the ‘experiment’ section of the web application. |
| src/redux | Contains actions, reducers and redux store which are used to maintain a global state of the application. Individual feature components can access the redux store to get their slice of the state for its operations. Reducers accept changes to the state and updates the store. Actions are used to trigger the reducers and passes it the new state to be updated. |

The components are organized into smart and dummy components. Dummy components focus on the UI and accepts properties to display them. Smart components manipulate the data, makes api calls and accesses the redux store and lifecycle methods. This improves the reusability and readability of the code. For example, components/experiment/ExperimentController.js is a smart component. It maintains the state of the list of experiments to be displayed and makes API calls to the server to update them. Whereas, component/experiment/ExperimentList.js merely accepts the list of experiments as props from the ExperimentController.js and displays it. ‘ExperimentList’ is rendered as a child of ‘ExperimentController’. The same kind of architecture is followed for form inputs and in the ‘Task’ feature as well. Example, components/task/ManageTask.js is the smart component which contains the dummy component TaskForm.js. TaskForm.js simply accepts the values as props from the parent component and the state of the entire form is managed in ManageTask.js. Another important feature to note here is the client-side routing. Since this is a single page application all page routing occurs on the client side, there is no communication to the server. We have used the library ‘react-router-dom’ for implementing this routing. Once the user has logged in, all the routes are made relative to ‘/dashboard’. The routes are configured in component/dashboard/DashboardController.js. Any new routes(pages) that need to be added can be configured in this file. There are also a few libraries that are used in combination with react and redux and is important to know them too. These libraries can be found in ‘packages.json’ located in the root folder. Below is a short description of them.

|  |  |
| --- | --- |
| Package-name | Description |
| axios | Promise based http-client for communicating with the server. Used for making XMLHttpRequests from the browser. It is used to make the server-client communication easier and to set authorization headers. |
| react-redux | Is a javascript library that is used to connect redux to the react framework so that the react components can access the redux store and dispatch actions to the reducer |
| redux-thunk | Is a middleware used in redux and in our application it is used to dispatch actions to the reducer after getting a response from the API |
| react-toastify | Is a simple library that is used to show confirmation messages upon triggering it. It is used as a message delivery to the user upon successful completion of an API and also used to display any errors that occurs on communicating with the server. |

Another important feature that is implemented in the front end is the private-routing component and can be found in components/common/PrivateRoute.js. This component checks if a user is authenticated to access a particular page by checking the redux store for user authentication status. The route ‘/dashboard’ is the only one included in this component as all other routes are relative to this route. Finally, a few redux functionalities that are specific to this web application needs to be discussed. Redux/configurestore.js contains the initialization of the store. Redux/reducers/initialState.js contains the initial state of the store when the application starts up. Redux/reducers/index.js contains the root reducer to which all the reducers are added.

# Deployment

Prerequisites:

* MySQL server (version 5.7 is preferred)
* NodeJS (version 12 is preferred)
* Git CLI should be installed

Note: “preferred” means that other versions have not been tested but should work with a lot of other versions as well.

Please follow the instruction in the “REAME.md” file for setting up and running the project.

##### References

1. Sequelize.org, ‘Manual | Sequelize’, 2019. [Online]. Available: https://sequelize.org/v5. [Accessed : 4-May-2020]