Database and Web Technique Term Paper
Summer Semester 2021

Khac Hieu Nguyen

Master of Automotive Software Engineering

Matriculation number: 678452

Chemnitz, 05.07.2021

Table of Contents

1. MERN Stack Overview 3
2. Project Architecture 4
3. Front-end 4
3.1. Technology 4
3.2. Implementation 7
3.2.1. Front-end app structure 7
3.2.2. Working Flow 8
4. Back-end 9
4.1. Technology 9
4.2. Implementation 11
4.2.1. Back-end app structure 11
4.2.2. Working Flow 12
5. Database diagram 13
References 14
APPENDIX 15

1. MERN Stack Overview

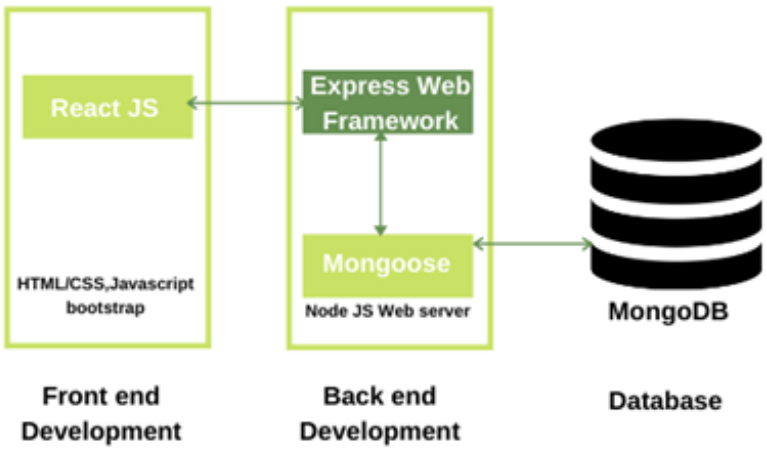
This project is developed by using MERN stack (Mongoose, Express JS, React, Node JS). MERN stack is a collection of robust and powerful technologies used to develop scalable master web applications, comprising front-end, back-end, and database components. It is a technology stack that is a user-friendly full-stack JavaScript framework for building dynamic websites and applications

MongoDB: MongoDB is an open source, cross platform, NoSql DBMS. It is a document-oriented database, which means that data is saved using collections and documents. MongoDB stores the data in binary JSON format that allows the fast exchange of data between client and server. It can also be used for the storage of large volumes of data, which makes it highly scalable.

Express: Express JS is a modular, lightweight framework of Node JS that helps in building web applications. It is a server-side, back-end, JavaScript-based framework that is designed to write simplified, fast, and secure applications.

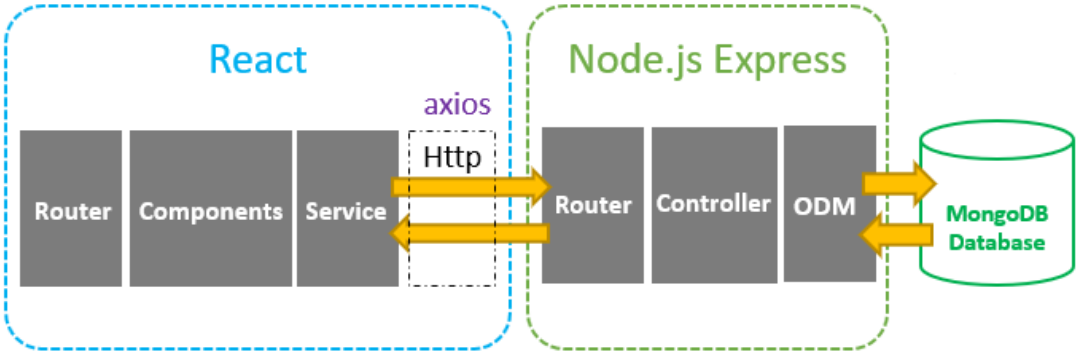
React: React JS is an open source JavaScript library used to build user interfaces, typically for single page applications. It offers the facility of code reusability on multiple platforms. It is fast and scalable.

Node.js: Node JS is an open-source, cross-platform, JavaScript runtime environment. It is designed to run the JavaScript code outside the browser, on the server side.



2. Project Architecture

MERN stack application will follow this architecture



React client app sends HTTP requests and receives responses using axios, data then will be forward to each components. React Router is used for navigating between different pages.

Express app defines all routes that matching API calls and corresponding controller. Controllers interact with MongoDB database using Mongoose ODM.

3. Front-end

3.1. Technology

React

React is a JavaScript library for creating user interfaces. The react package contains only the functionality necessary to define React components. It is typically used together with a React renderer like react-dom for the web, or react native for the native environments.

React Router

React Router is a collection of navigational components that compose declaratively with the application. This library supports dynamic routing that takes place as the app is rendering not in a configuration or convention outside of a running app. That means almost everything is a component in React Router.

React Hooks

Hooks are functions that let you “hook into” React state and lifecycle features from function components. Hooks don’t work inside classes — they let you use React without classes.

- **useState():** enables you to add state to function components. useState() returns a pair: the current state value and a function that lets you update it.

```
const [classList, setClassList] = useState([]);
const [subjectList, setSubjectList] = useState([]);
const [subjectListByTeacher, setSubjectListByTeacher] = useState([]);
const [subjectListByClass, setSubjectListByClass] = useState([]);
const [teacherList, setTeacherList] = useState([]);
const [studentList, setStudentList] = useState([]);
const [testResultList, setTestResultList] = useState([]);
const [currentUser, setCurrentUser] = useState(undefined);
```

- **useEffect():** adds the ability to perform side effects from a function component. It serves the same purpose as componentDidMount, componentDidUpdate, and componentWillUnmount in React classes, but unified into a single API.

```
useEffect(() => {
  fetchRandomUsers();
  fetchTeacherList();
  fetchRandomClasses();

}, [fetchRandomUsers, fetchRandomClasses, fetchTeacherList]);
```

Proptypes

A runtime type checking for React props and similar objects. Using prop-types to document the intended types of properties passed.

```
CustomTable.propTypes = {
  classes: PropTypes.object.isRequired,
  currentUser: PropTypes.object.isRequired,
  studyingClass: PropTypes.object.isRequired,
  subjectList: PropTypes.arrayOf(PropTypes.object).isRequired,
  selectStudentPage: PropTypes.func.isRequired,
  pushMessageToSnackbar: PropTypes.func.isRequired
};
```

Axios

A promised based HTTP client for the browser and node.js. Axios deals with responses using *Promises*, so it's streamlined and easy to use in our code. Axios uses methods like get() and post() that perform http GET and POST requests for retrieving or creating resources.
Axios provides some features like:

- Make XMLHttpRequests from the browser
- Make http requests from node.js
- Supports the Promise API
- Intercept request and response
- Transform request and response data

- Cancel requests
- Automatic transforms for JSON data
- Client side support for protecting against XSRF

An example get request to get all user list from server

```
import axios from 'axios';

export default axios.create({
    baseURL: "http://localhost:8080/api/manage",
    headers: {
        "content-type": "application/json"
    }
});
```

```
    getUserList() {
        return api.get('/users');
    }
```

Material UI

Material-UI is one of the most popular library for React project. It aligns with Material Design by Google with a large library of components, fully customizable theme, regularly updated and extensive support for issues and bugs.

Material-UI components work in isolation. They are self-supporting, and will only inject the styles they need to display. They don't rely on any global style-sheets such as normalize.css.

Material-UI components work without any additional setup and don’t pollute the global scope.

Define a button Hello world by import Component Button and it is ready to use.

```
import React from 'react';
import { Button } from '@material-ui/core';

function App() {
  return <Button color="primary">Hello World</Button>;
}
```

Date-fns

Date-fns is a modern JavaScript date utility library. It provides the most comprehensive, simple and consistent toolset for manipulating JavaScript dates in a browser and Node.js. It works well with modern module bundlers such as **webpack**, **Browserify**, and **Rollup** and also supports tree-shaking. Date-fns uses the native Date type and doesn't reinvent the wheel. It doesn't extend core objects for safety's sake. Functions in date-fns work predictably and stick to ECMAScript behavior in edge cases.

Jspdf-autotable

This jsPDF plugin adds the ability to generate PDF tables either by parsing HTML tables or by using Javascript data directly.

My Result

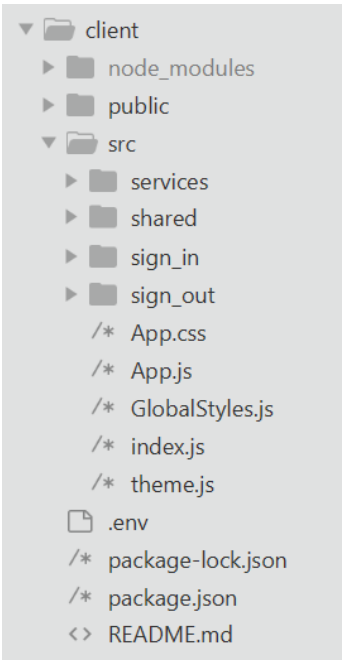
SUBJECT NAME	GRADE
Automotive Sensor System	1
Formal Specification and Verification	1
Practical Automotive Software Engineering	1

3.2. Implementation

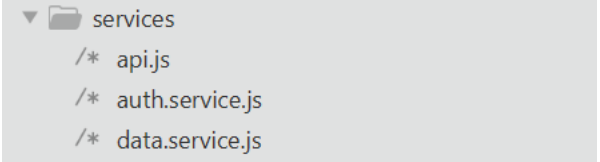
3.2.1. Front-end app structure

Front-end app is in client folder.

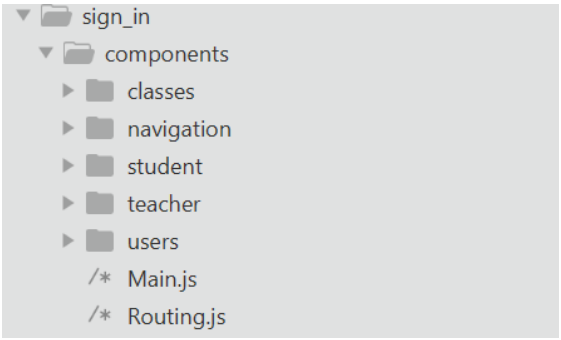
- **package.json**: contains modules of the project.
- **App.js** is the container that has Router and Navigation bar.



- **public**: stores some images for account avatar, home image.
- .**env**: stores environment variables.
- **services**: defines api calls, communicating between client and server. Those apis are used for authenticate user when log in (AuthService) and fetching data from server (DataService).



- **shared:** define some common components and functions can be used between all components in the project**.**
- **sign_in:** contains components of each view based on user role. Admin will manage classes and users view. Student and teacher are the corresponding view of student and teacher when they log in to the system.

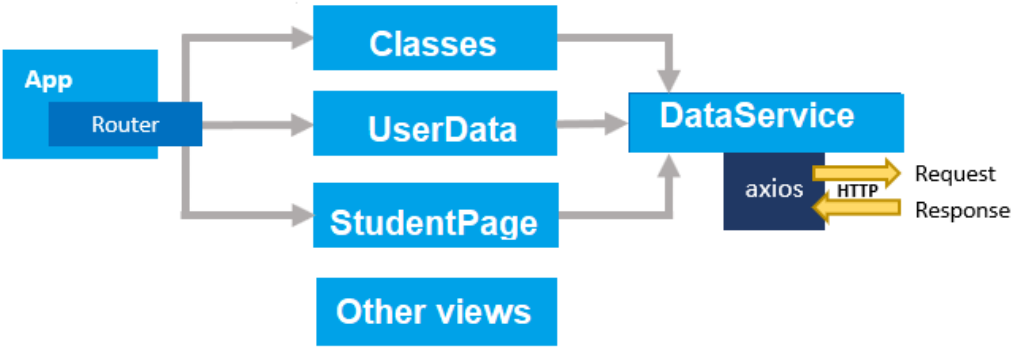


- **sign_out:** show home page with log in dialog.
- **GlobalStyles and theme:** define style, color of the project.

3.2.2. Working Flow

The App component is a container with React Router. It has a Navigation bar that links to routes path. Classes, UserData, StudentPage are components that manage information of class, user, studying subjects of student. These components will use DataService methods which use axios to make HTTP requests and receive responses. Other views will work the same.

All requests using API calls from React app will be delivered to Node JS server by configuring proxy in package.json: **"proxy": "http://127.0.0.1:8080"**



4. Back-end

4.1. Technology

Express

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. Express provides us a lot of features that help create a robust API is quick and easy:

- Robust routing
- Focus on high performance
- Super-high test coverage
- HTTP helpers (redirection, caching, etc)
- View system supporting 14+ template engines
- Content negotiation
- Executable for generating applications quickly

Async

Async is a utility module which provides straight-forward, powerful functions for working with asynchronous JavaScript. Most of the methods we use in *Express* are asynchronous—you specify an operation to perform, passing a callback. The method returns immediately, and the callback is invoked when the requested operation completes.

Async has a lot of useful. Some of the most important functions are:

- async.parallel() to execute any operations that must be performed in parallel.
- async.series() for when we need to ensure that asynchronous operations are performed in series.
- async.waterfall() for operations that must be run in series, with each operation depending on the results of preceding operations.

```
async.parallel({
    class: function (callback) {
        Class.findById(req.params.id)
            .exec(callback)
    },

    class_subjects: function (callback) {
        Subject.find({ 'class': req.params.id })
            .populate('teacher')
            .exec(callback)
    },
}, function (err, results) {
```

Bcryptjs

Bcrypt is a secured way to store password in database regardless of whatever language app's backend is built in. By using hash algorithms (one-way function), the resulting hash cannot be reversed.

Auto-gen a salt and hash password:

```
password: bcrypt.hashSync(req.body.password, 8),
```

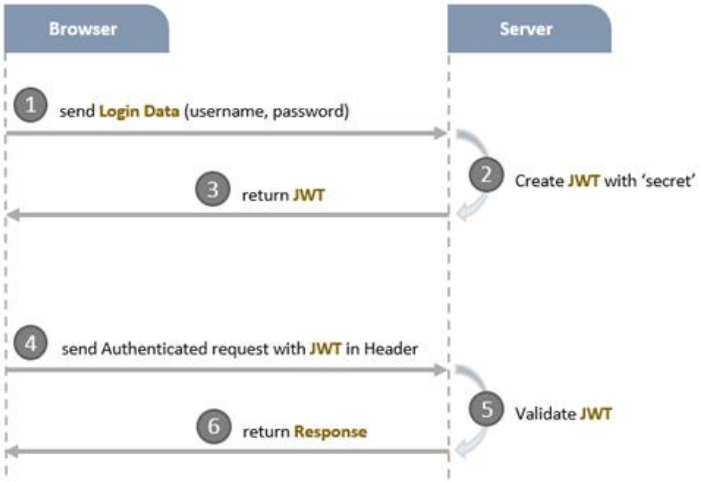
Check password:

```
var passwordIsValid = bcrypt.compareSync(
  req.body.password,
  user.password
);
```

Jsonwebtoken

Jsonwebtoken is used to authorize a user when login to our system for accessing features of the app.

Once user is logged in, each subsequent request will include the JWT, allowing the user to access routes, services, and resources that are permitted with that token



Mongoose

Mongoose is a **MongoDB** object modeling tool designed to work in an asynchronous environment. Mongoose supports both promises and callbacks.

Mongoose provides a straight-forward, schema-based solution to model application data. It includes built-in type casting, validation, query building, business logic hooks and more, out of the box.

Nodemon (Only used in development)

Nodemon is a tool that helps develop node.js based applications by automatically restarting the node application when file changes in the directory are detected.

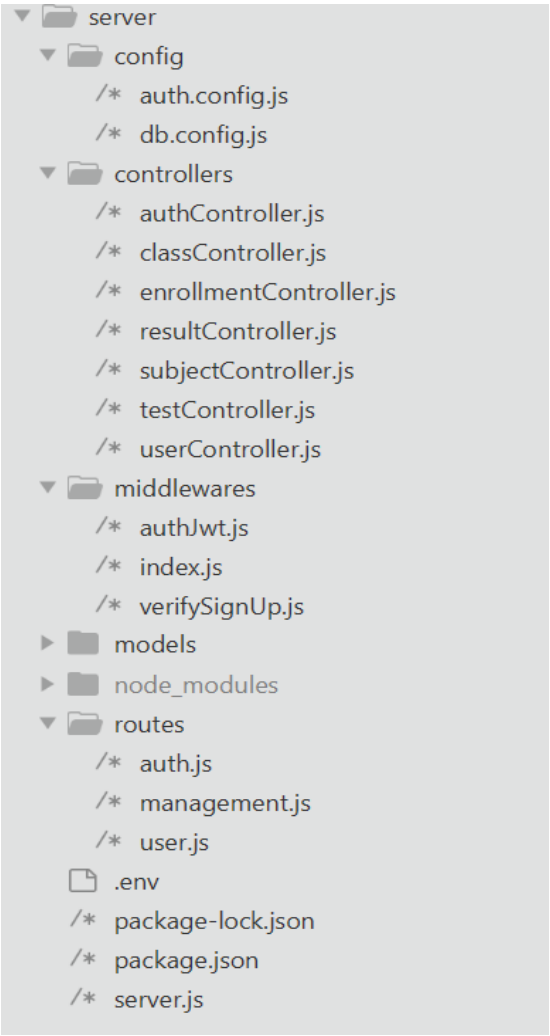
```
"scripts": {
  "dev": "nodemon ./server.js"
},
```

4.2. Implementation

4.2.1. Back-end app structure

Back-end app is in server folder.

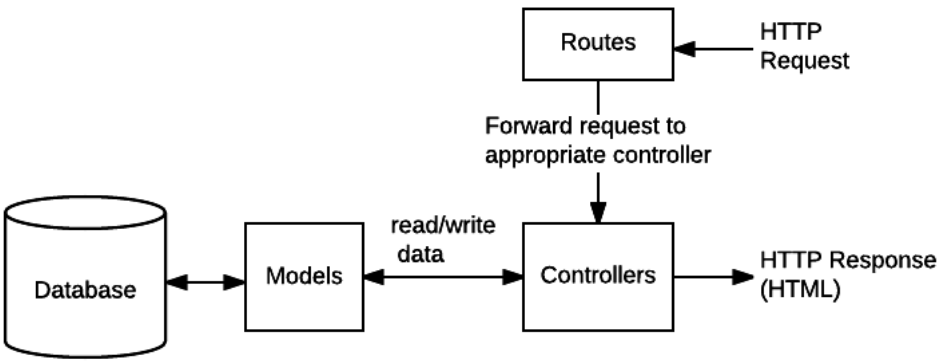
- **config:** stores some configurations for authentication (store secret key) and database (host, port, database name)
- **controllers:** contain all functions for APIs
- **middlewares:** contain all middleware function for authentication and verying users
- **models:** contain all schema files
- **routes:** contain all defined routes using Express Router
- **.env:** stores some environment configuration variables
- **server.js**: the entry point of the Express application
- **package.json**: contain all packages, scripts and dependencies



4.2.2. Working Flow

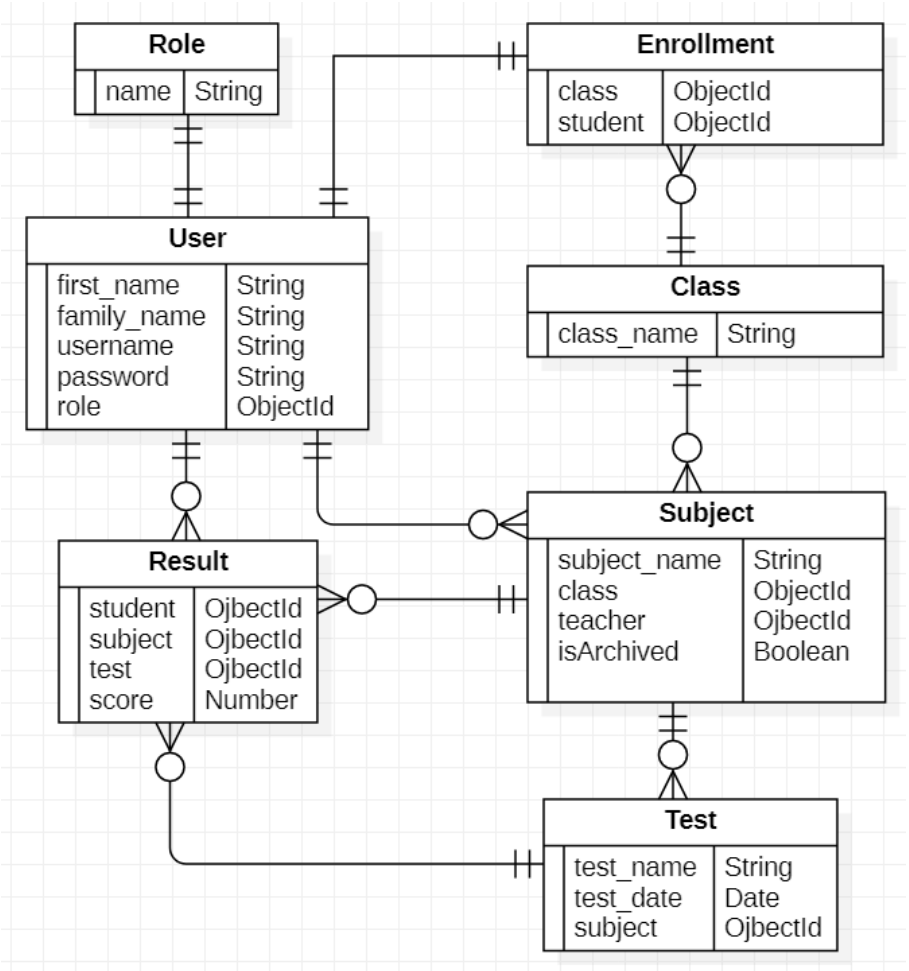
Routes: forward the supported requests to the appropriate controller functions

Controller functions will get the requested data from the models and return it to the user view in the browser.



5. Database diagram

This is the database design of the project based on Mongoose Schema.



References

- Why using MERN stack
  - https://www.classicinformatics.com/blog/why-is-mern-stack-our-preferred-platform-for-startups-apps
- React
  - https://www.npmjs.com/package/react
- React Router
  - https://reactrouter.com/web/guides/philosophy
- React Hooks
  - https://reactjs.org/docs/hooks-intro.html
- Axios
  - https://www.npmjs.com/package/axios
  - https://dev.to/cesareferrari/working-with-axios-in-react-540c
- Material-UI
  - https://material-ui.com/
  - https://material-ui.com/getting-started/usage/
- Date-fns
  - https://date-fns.org/
- Jspdf-autotable
  - https://www.npmjs.com/package/jspdf-autotable
- Express
  - http://expressjs.com/
- Bcryptjs
  - https://www.npmjs.com/package/bcryptjs
  - https://javascript.plainenglish.io/how-bcryptjs-works-90ef4cb85bf4
- Jsonwebtoken
  - https://jwt.io/introduction
  - https://bezkoder.com/jwt-json-web-token/
- Mongoose
  - https://mongoosejs.com/
  - https://www.npmjs.com/package/mongoose
- MERN stack example application
  - https://bezkoder.com/react-node-express-mongodb-mern-stack/
  - https://bezkoder.com/react-crud-web-api/
  - https://github.com/mdn/express-locallibrary-tutorial
  - https://github.com/dunky11/react-saas-template
  - https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/routes

APPENDIX

Rest API used in the project

Function	URL	Method	URL params	Data params	Success Response	Error Response
Sign in	/api/auth/signin	POST		username=[string] password=[string]	200 User data: _id=[String] first_name=[String] family_name=[String] username=[String] password=String] role=[String] accessToken=[String]	500: user not found 404: invalidUsername 401: invalidPassword
Get list of user	/api/manage/users	GET			200, list of user Each user contains: _id=[String] first_name=[String] family_name=[String] username=[String] password=String] role=[String]	500: Cannot retrieve user list
Create new user	/api/manage/user/create	POST		first_name=[String] family_name=[String] username=[String] password=[String] role=[String]	200, “User was registered successfully”	500 “Cannot create user”
Update user informatio n	/api/manage/user/:id/update	POST	Id=[String]	first_name=[String] family_name=[String] username=[String] password=[String] role=[String]	200, “User was updated successfully”	404 “User not found” 500 “Cannot update user”
Delete a user	/api/manage/user/:id/delete	POST	Id=[String]		200, “User was deleted successfully”	500 “Cannot retrieve user detail”
Get user detail	/api/manage/user/:id	GET	Id=[String]			
Get list of classes	/api/manage/classes	GET			200, list of class	500 “Cannot retrieve class list”
Create new class	/api/manage/class/create	POST		class_name=[String]	200, “Class was created successfully”	500 “Cannot create new class” 400 “Class with name ${class_name} is already existed”
Update a class	/api/manage/class/:id/updat e	POST	Id=[String]	class_name=[String]	200, “Class was updated successfully	500 “Cannot update this class” 404 “Class not found”
Delete a class	/api/manage/class/:id/delete	POST	Id=[String]		200, “Class was updated successfully	

Get class detail	/api/manage/class/:id	GET	Id=[String]		200, class detail
					_id=[String]
					class_name=[String]
					subject list of the class
					_id=[String]
					subject_name=[String]
					teacher={
					_id=[String]
					first_name=[String]
					family_name=[String]
					
					username=[String]
					
					password=String]
					role=[String]
					
					}
						500 “Cannot retrieve class detail”
						404 “Class not found”

Get list of enrollment	/api/manage/enrollments	GET			200, list of enrollment
					_id=[String]
					class=[String]
					student={
					_id=[String]
					first_name=[String]
					family_name=[String]
					username=[String]
					password=String]
					role=[String]
					
					}
						500 “Cannot get enrollment information”

Get enrollment of provided student id	/api/manage/enrollment/student	GET		studentId=[String]	200, enrollment detail of provided student id _id=[String] class=[String] student=[String]	500 “Cannot get enrollment information”
Register a student to specific class	/api/manage/enrollment/register	POST		class=[String] student=[String]	200, “Student was added successfully”	500 “Cannot add student to this class”
Deregister student	/api/manage/enrollment/deregister	POST		studentId=[String]	200 “Deregister student successfully”	500 “Cannot deregister this student”

Get enrollment detail	/api/manage/enrollment/:id	GET	Id=[String]		200, list of enrollment
					_id=[String]
					class=[String]
					student={
					_id=[String]
					first_name=[String]
					family_name=[String]
					username=[String]
					password=String]
					role=[String]
					}
						500 “Cannot get enrollment information”

Get list of subject	/api/manage/subjects	GET			200 subjec list
					_id=[String]
					subject_name=[String]
					isArchive=[Boolean]
					class={
					_id=[String]
					class_name=[String]
					}
					teacher={
					_id=[String]
					first_name=[String]
					family_name=[String]
					username=[String]
					password=String]
					role=[String]
					}
						500 “Cannot retrieve subject list”

Create new subject	/api/manage/subject/create	POST		subject_name=[String] assigned_teacher=[String] assigned_class=[String]	200 “Subject was created successfully”	500 “Cannot create new subject” 400 “Subject with name ${subject_name} is already existed”
Update a subject	/api/manage/subject/:id/update	POST	Id=[String]	subject_name=[String] teacher=[String]	200 “Subject was updated successfully”	500 “Cannot update subject” 404 “Subject not found”
Delete a subject	/api/manage/subject/:id/delete	POST	Id=[String]		200 “Subject was deleted successfully”	500 “Cannot delete subject” 401 “Cannot delete subject has dependent test”
Get subject detail	/api/manage/subject/:id	GET	Id=[String]		200 detail of subject _id=[String] subject_name=[String] class=[String] isArchive=[Boolean] list of test _id=[String] test_name=[String] test_date=[Date] list of test result _id=[String] test=[String] subject=[String] student=[String] score=[Number]	500 “Cannot get detail of this subject”
Archive a subject	/api/manage/subject/archive	POST	Id=[String]		200 “Subject was archived successfully”	500 “Cannot archive this subject”

						400 “Cannot archive subject with no test”
Get list of test	/api/manage/tests	GET			200 list of test _id=[String] test_name=[String] test_date=[Date]	500 “Cannot retrieve test”
Create a new test	/api/manage/test/create	POST		test_name=[String] test_date=[String] subject=[String]	200 “Test was created successfully”	500 “Cannot create new test” 400 “Test is already existed”
Update a test	/api/manage/test/:id/update	POST	Id=[String]	test_name=[String] test_date=[String]	200 “Test was updated successfully”	500 “Cannot update this test” 404 “Test not found”
Delete a test	/api/manage/test/:id/delete	POST	Id=[String]		200 “Test was deleted successfully”	500 “Cannot delete this test” 404 “Test not found”
Get test detail	/api/manage/test/:id	GET	Id=[String]		200 test detail _id=[String] test_name=[String] test_date=[Date] list of test result _id=[String] test=[String] subject=[String] student=[String] score=[Number]	500 “Cannot retrieve test details” 404 “Test not found”

Get list of result	/api/manage/results	GET			200 list of test result
					_id=[String]
					test={
					_id=[String]
					test_name=[String]
					test_date=[Date]
					list of test result
					}
					subject=[String]
					student=[String]
					score=[Number]
						500 “Cannot retrieve test result”

Get result by student id	/api/manage/result/student	GET		studentId=[String]	200 test result by student id
					_id=[String]
					subject={
					_id=[String]
					subject_name=[String]
					isArchive=[Boolean]
					}
					test=[String]
					student=[String]
					score=[Number]
						500 “Cannot retrieve test result of this student”

Create new test result	/api/manage/result/create	POST		test=[String] subject=[String] student=[String] score=[Number]	200 “Result was created successfully”	500 “Cannot create test result” 400 “Test result for this student is already created”
Update a test result	/api/manage/result/:id/update	POST	Id=[String]	new_score=[Number]	200 “Result was updated successfully”	500 “Cannot update this test result” 404 “Test result not found”
Delete a test result	/api/manage/result/:id/delete	POST	Id=[String]		200 “Test result was deleted successfully”	500 “Cannot delete this test result” 404 “Test result not found”