

Contents

1 Introduction.....	1
2 Project	1
2.1 Requirements	1
2.1.1 Requirements breakdown	2
2.2 Database.....	4
2.2.1 Relational Schema	4
2.3 Functional requirement Requirements.....	4
2.4 Project operations	6
2.4.1 Home	6
2.4.2 Admin	6
2.4.3 Teacher	9
2.4.4 Student	11
2.4.5 Prerequisites.....	11
2.4.6 How to Run.....	12
3 Technology Overview	12
3.1 HTML	12
3.2 CSS	12
3.2.1 Bootstrap	12
3.2.2 Materialize CSS	13
3.3 PHP	13
3.4 Javascript	13
3.4.1 Node.js.....	13
4 Conclusion	14
References	15
API Documentatiom	16

Figures

Figure-1: Structure	1
Figure-2: Relational Schema	4
Figure-3: Login	6
Figure-4: Admin Home and Editing data of user	7
Figure-5: Class overview	8
Figure-6: Subject overview of a class	8
Figure-7: Assign/de-assign of teacher/pupil	9
Figure-8: Teacher home and overview of student performance a subject.....	9
Figure-9: Creating test and test list of a subject	10
Figure-10: Upload test grade and overview of grades	10
Figure-11: Student view of subject and test overview of a subject	11

Tables

Table-1: Requirements breakdown	2
Table-2: Functional requirements	4
Table-3: Required Package for Installation.	11

1 Introduction

The project's purpose is to develop a management system for school in order to meet the "Datenbanken und Web-Techniken" final standard requirements. The project is divided into three different user perspective: Admin, Teacher, and Student.

It is developed with the intention of providing all three users with certain rights and an easy-to-maintain system. It also improves teacher-student communication.

Each user has specific permissions, and the administrator can add and remove users, classes, and subjects, assign and de-assign students for classes, and update the attributes of these features.

Teachers can also update their own information and organize subject-specific assessments. Eventually, students can see the attended subjects and the performance of the tests. [1]

2 Project

2.1 Requirements

The primary goal of this project is to construct and utilizing APIs (Application Programming Interfaces) which serve the purpose of the store and fetch data for the application. Only APIs can communicate with the backend. The frontend here is used for user interaction with the system. The diagram below illustrates the communications between these components.

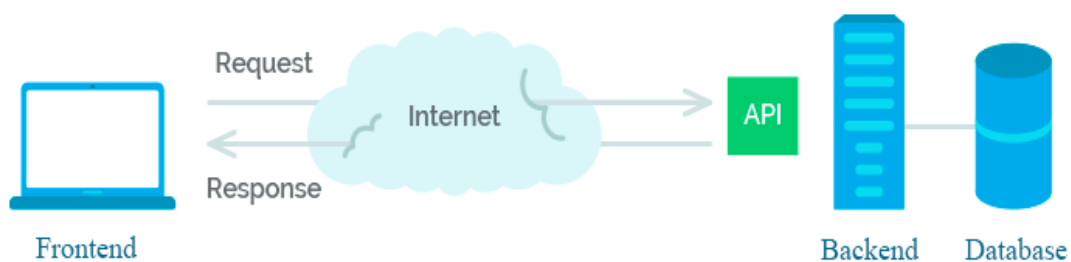


Figure-1: Structure of workflow diagram [2]

2.1.1 Requirements breakdown

DigitalSchoolManagement		
Application STACK	Tasks ToDo	Remarks
Frontend	UI for Login	Default for all users
	UI for Admin	List of all users
	UI for Create/Modification of User	Basic information can be added or change by admin.
	UI for delete user	Teacher cannot be removed while they are assigned any subject. Delete user leads remove all associate data.
	UI for add, edit, delete a class	Each class has unique name. Assigned students and subjects will be removed if a class is deleted.
	UI for add, edit, delete, archive a subject	Each subject has unique name. Archive a subject but no non-archive state. Delete subject results remove all associate data.
	UI for Teacher	Show assigned subjects. Show list of the associate students with mean grades.
	UI for add, edit, delete a test	Upload grades through CSV Edit for individual student grade. Delete test means remove all the given grades.

	UI for student	Only show the attended subjects. Show each test result.
REST Interface Backend	Send HTTP request from frontend	
	Validate URL	Validation of each URL with proper status code.
	APIs request correspond controller	Data will be stored or fetched through API.
	Utilizations of all functional REST APIs	Implement functionalities including associate architecture of REST.
Database	Utilizations of a structural database	Build proper relation establishment, since it is an organized project.

Table-1: Main requirement breakdown

2.2 Database

2.2.1 Relational Schema

The relational schema presents both a high-level and a technical perspective of the database design. In terms of logic and business rules, it's utilized to create a relational database.

The relational schema plays a very important part in the project's development. Before we began to write the script, developers first completed the relational schema. Because the relational structure explains how the database's tables are linked together via foreign keys and primary keys.

Often by using primary keys from a table to other table join operation can be done easily. [3]

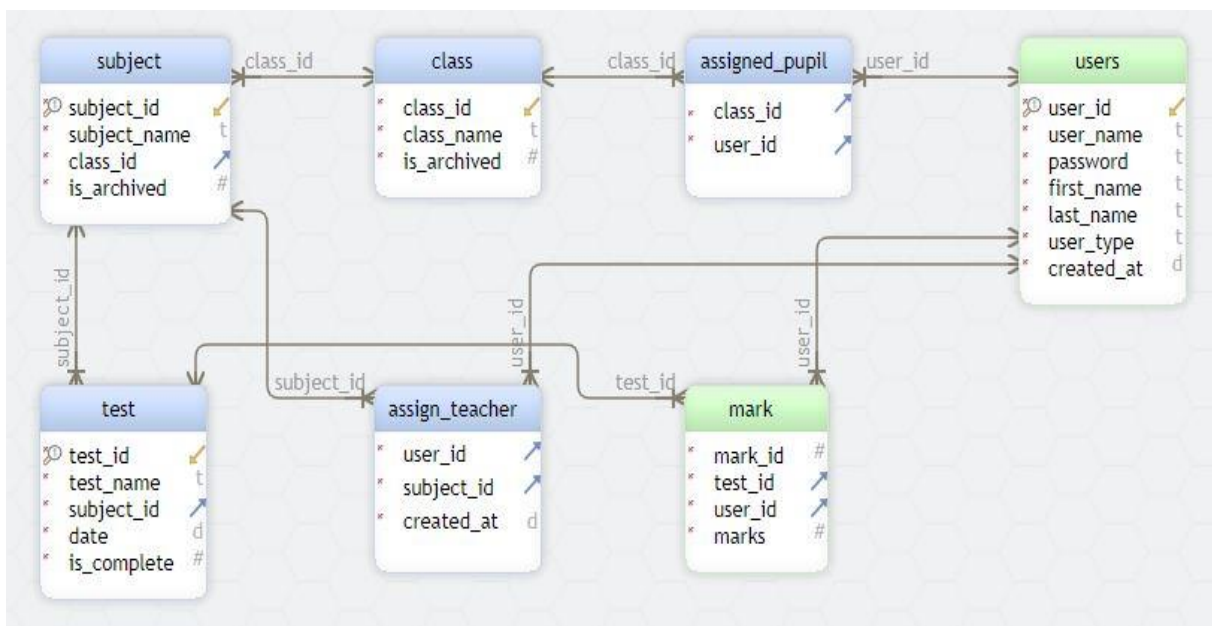


Figure-2: Relational Schema

2.3 Functional requirement Requirements

The system mainly has two users. Both student and admin panel have their separate tasks to handle. They perform different tasks according to their role and requirements.

Application	Task
Admin	<ul style="list-style-type: none">• Login• Create user• See all users• Edit/modify/delete user

	<ul style="list-style-type: none"> • Add class • Assign/De-assign pupil class • Edit/modify/delete class • Add subject • Edit/modify/delete subject • Archive subject • Assign/change teacher subject
Teacher	<ul style="list-style-type: none"> • Login • Update own information • List all assigned subject • Create subject test • Edit/modify/delete test • Upload test grade • Change individual grade
Student	<ul style="list-style-type: none"> • Login • Update own information • View attended subjects • View achieved test grades

Table-2: Functional requirements of the requirements

2.4 Project operations

This system consists three user interfaces. The main interface is for the admin and the other two is for teachers and students. All the major operation is conducted through to the database using REST APIs. Admin can create, edit, and delete users, classes, subjects and add or remove pupil from a class. Teacher can see assigned subjects and management of tests. And students only have the rights for seeing their attended subjects with grades

2.4.1 Home

A single login panel is used for all users. The user is dynamically identified and re-directed to their intended views using the login credentials. For three specific users, there has three different views. Those are:

- Admin view
- Teacher view
- Student View

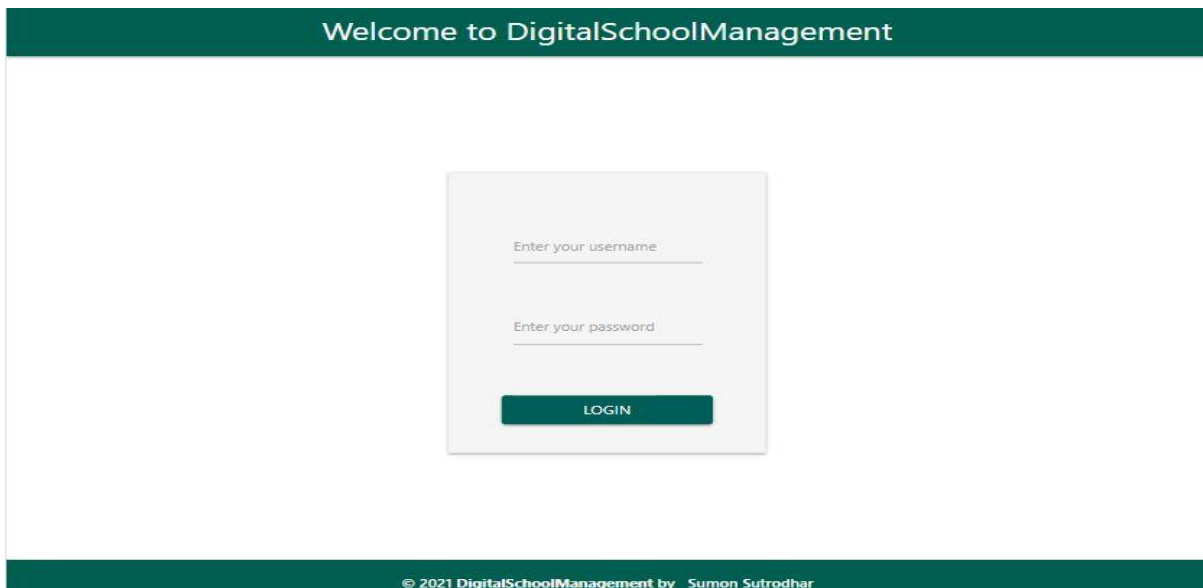


Figure-3: login

2.4.2 Admin view

The admin view will be accessed using the admin's login data. Admins can see the list of all available users. They can also see all of the classes, and the subjects of each class.

Login:

Admins have to access the system by using their username and password.

Add user:

New users can be added to the system by the admins. Users are identifiable by a unique user ID generated by the system. There are three categories of users: Admin, Teachers, and Students. Initially, the admin provides data to the system, which the user can be changed afterwards.

Delete user:

An existing user can be deleted at any time by the admin. In certain cases, the system can prevent deletion. If a teacher is assigned to at least one non-archived subject, he or she cannot be removed. When a student is deleted, all associated data is also removed from the system.

Modifying user attributes:

Admins can make changes to user data such as first and last names, as well as other attributes by clicking on profile. After a user has been created the user role cannot be updated. Altering the data results synchronization with other views.

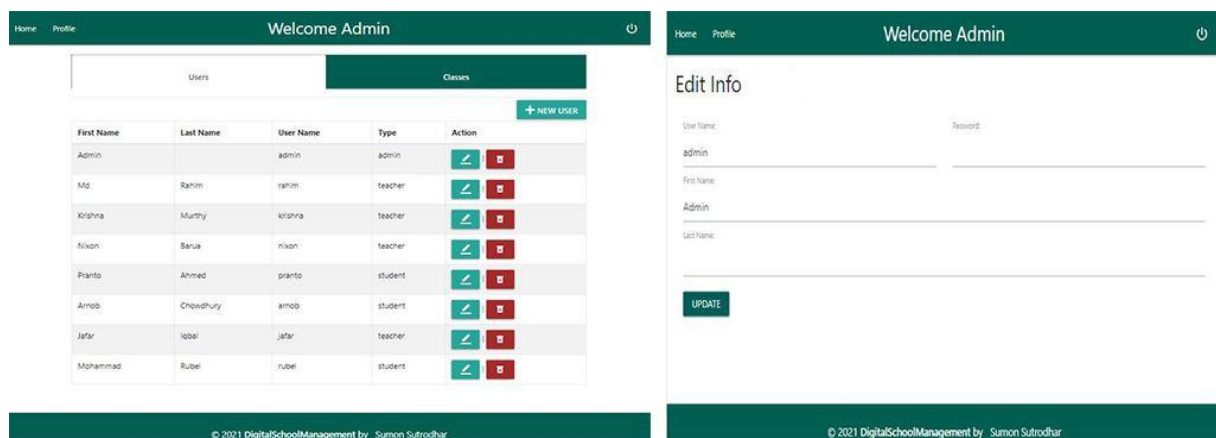


Figure-4: Admin Home and Editing data of user

Add and modifying of class attributes:

Only admin has the rights to create, update and delete a class. Class name is unique.

Delete class:

Deletion of a class results de-assigns all the assigned students from that class and depending on the criteria the subjects of the class either removed or archived.

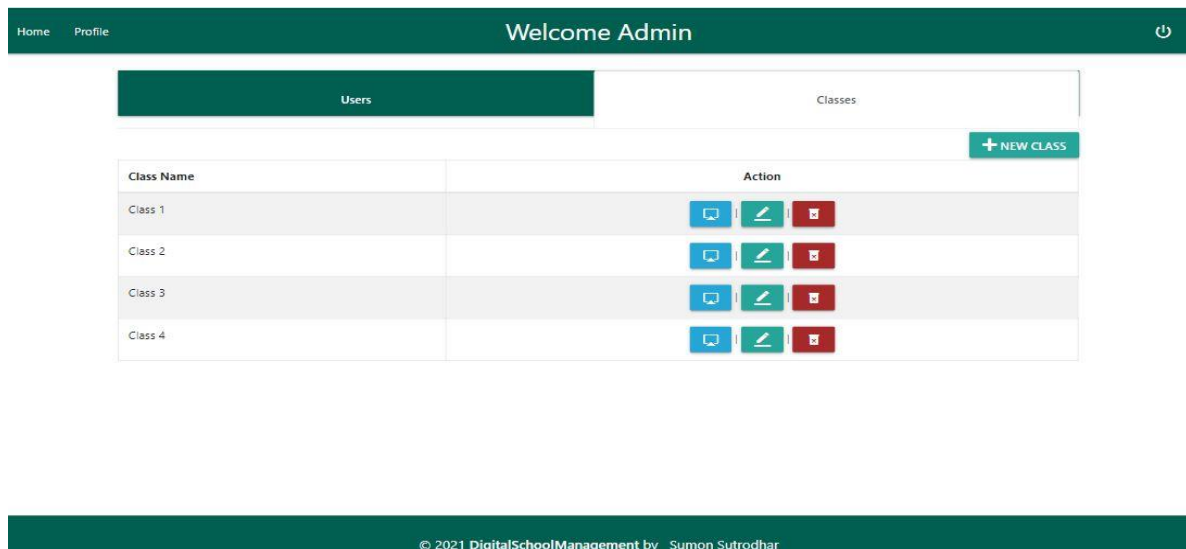


Figure-5: Class overview

Add or modifying a subject including archiving:

Different subjects can be added into a class. Each subject is identified by a unique name and ID. An already existing teacher is assigned while the subject is created. Only admin can change the subject attributes but retrieving an archiving a subject into the non-archiving state is not possible according to the requirements. No change can be done with an archived subject.

Delete subject:

Deleting a subject delete associate test and test results. And it is synchronized with corresponding user views.

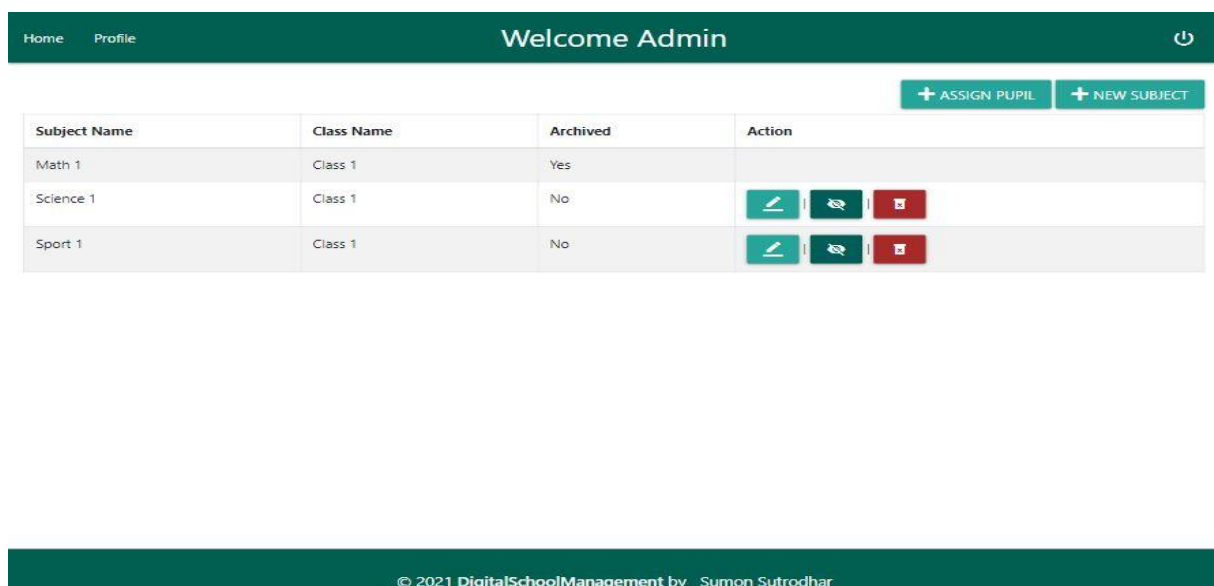


Figure-6: Subject overview of a class

Assign teacher and assign/de-assign of pupil:

Admin assign a teacher while creating a subject and it can be changed later by admin. Students can be assign or de-assign from a class. Assigning student to a new class will automatically de-assign them from the previous one.

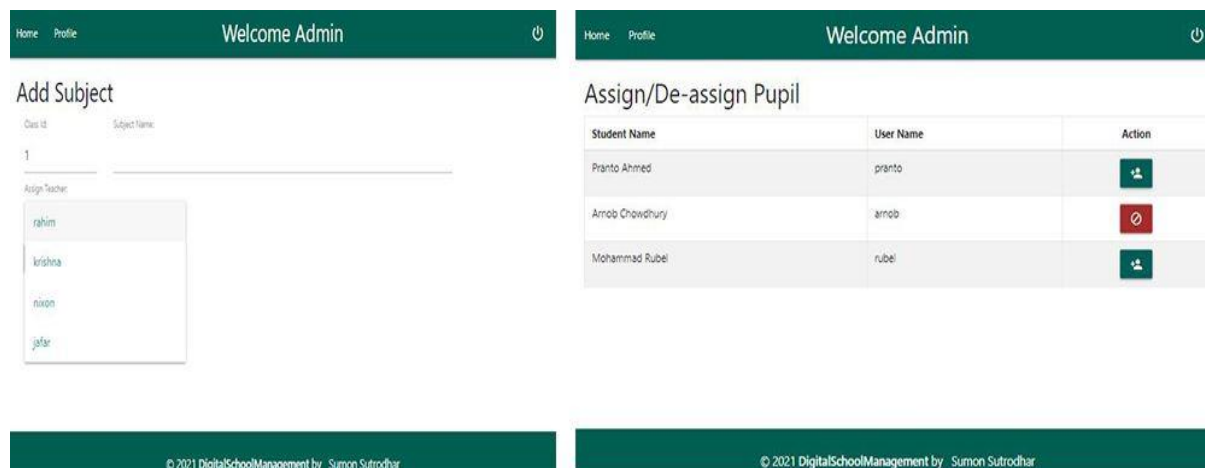


Figure-7: Assign/de-assign of teacher/pupil

2.4.3 Teacher

Teacher view provides the list of all the subject that he/she is assigned to. They can also see all the students studying the subjects along with their respective average grades.

Login:

Teacher have to access the system by using their username and password.

Change own data:

Teacher can update basic attributes like first name, last name and password by clicking on profile.

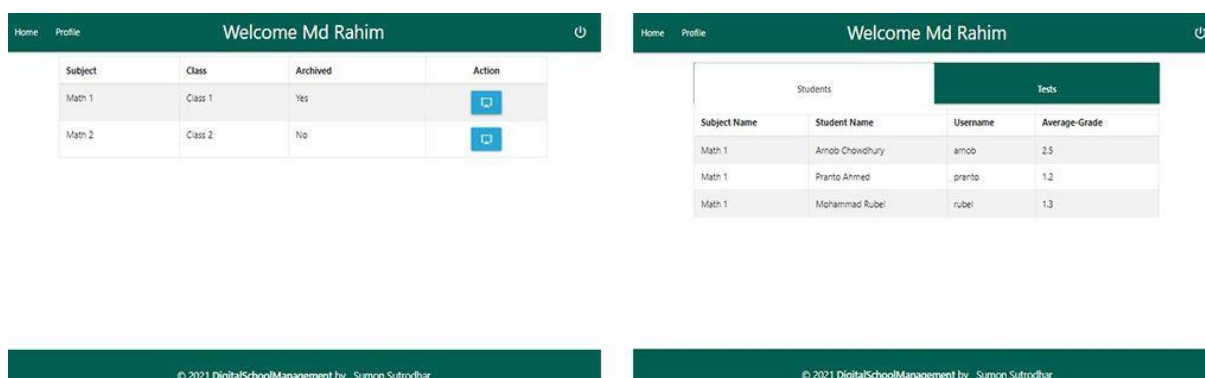


Figure-8: Teacher home and overview of student performance a subject

Add/modify test attributes:

Teacher can add test for each subject, there is no limitation in test. A test name and date should be given while creating a test. Teacher can also update the test attributes like as name and date.

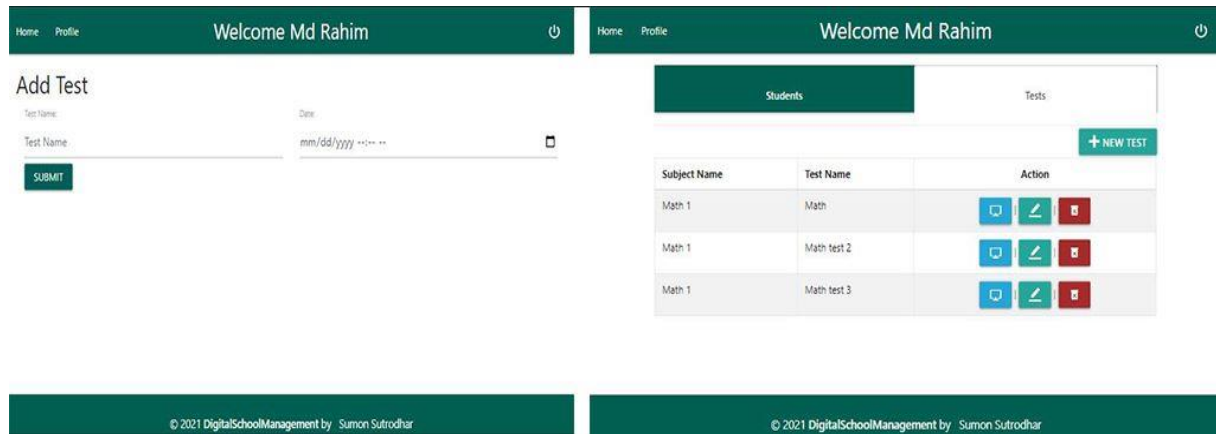


Figure-9: Creating test and test list of a subject

Add/change grades:

Teacher can easily provide grades of each test by uploading CSV files to the system. The grades are visible to teacher and teacher has the right to update any student's grade. The change is also automatically updated in student view.

Delete test:

Only teacher can delete a test. If test is deleted, all related information for example given grade of each student will also be deleted.

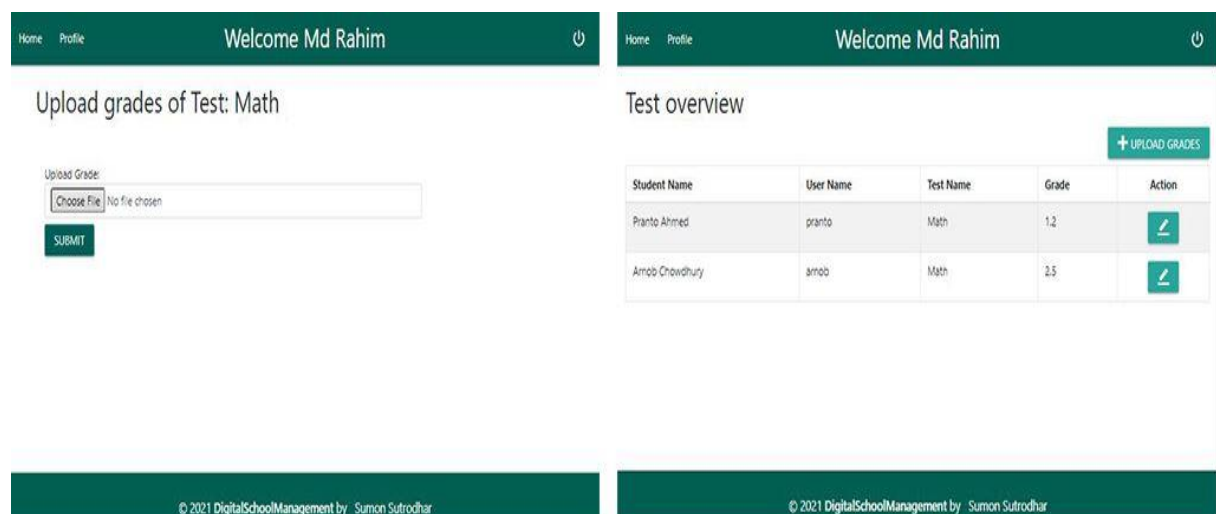


Figure-10: Upload test grade and overview of grades

2.4.4 Student

Student view starts with an overview of attended subjects with an average result of the subjects. The average is calculated from the given marks of every test of a single subject. Student can also see each test performance of every subjects.

Login:

Student get access into the system by using their username and password.

Change personal attributes:

Student can update basic attributes like first name, last name and password by clicking on profile.

Subject and Test:

Student can only see their attended subjects and the test performances of each subjects.

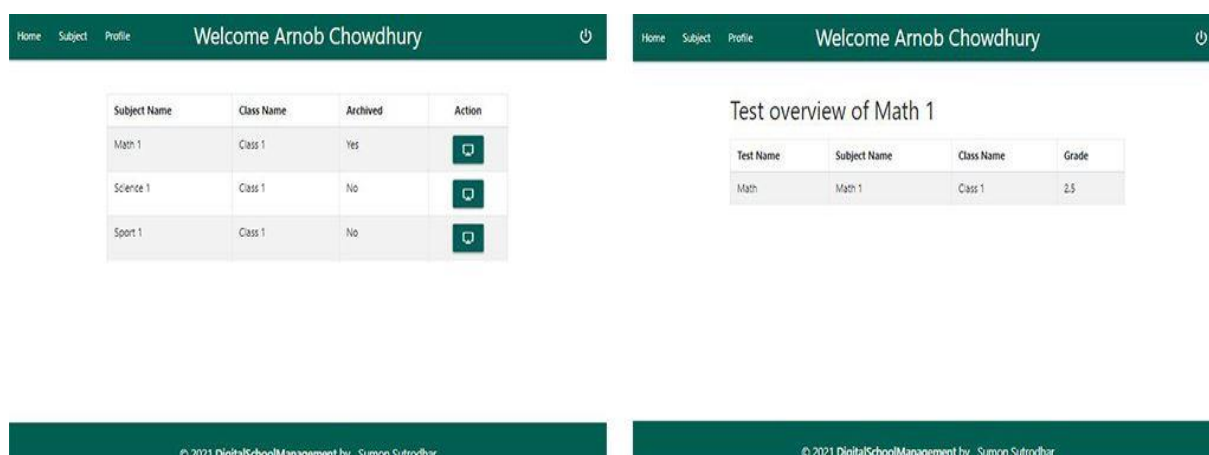


Figure-11: Student view of subject and test overview of a subject

2.4.5 Prerequisites

The zip file needs to extract at the server's htdocs directory (for local server) or public_html directory (for remote server).

APPS	REQUIRED PACKAGES
App	PHP==7 or above [3] Node.js == 10.15.2 or above [4] Local Server==XAMPP (Windows) or MAMP (Mac)
Database	Mysql = 5.7.33 or higher [1]

Table-3: Required Package for Installation.

2.4.6 How to Run

After starting the Apache server (XAMPP/MAMP), need to run server for Node.js where the web interfaces are implemented. The command for running the server the project is given below.

```
npm run dev
```

Then we just need to hit the URL (directory of our app at the server) at the browser. In our environment the URL was like:

```
http://localhost/dsm
```

Now the application will run and is ready for showing the functionalities of DigitalSchoolManagement (DSM)

3 Technology Overview

3.1 HTML

HTML is an acronym which stands for Hyper Text Markup Language which is used for creating web pages and web applications. Let's see what is meant by Hypertext Markup Language, and Web page. Hence, HTML is a markup language which is used for creating attractive web pages with the help of styling, and which looks in a nice format on a web browser. An HTML document is made of many HTML tags and each HTML tag contains different content. [4]

3.2 CSS

Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language such as HTML. CSS is a cornerstone technology of the World Wide Web, alongside HTML and JavaScript. [5]

3.2.1 Bootstrap

Bootstrap is a free front-end - CSS framework. It contains HTML and CSS- based design templates for typography, forms, buttons, tables, grid systems, navigation and other interface design elements as well as additional, optional JavaScript extensions.

Initially we did not use this language because it is mainly used to stylize the project. We almost entirely coded our project interface on PHP. After implementing all the logics and running a number of tests to verify all the required objectives and tasks, we finally used CSS to give the project a visual aesthetic look. [6]

3.2.2 Materialize CSS

A modern CSS framework based on Material Design. Materialize CSS is a UI component library which is created with CSS, JavaScript and HTML. It is created and designed by Google. Materialize CSS is also known as Material Design. It is used to construct attractive, consistent, and functional web pages and web apps while adhering to modern web design principles such as browser portability, device independence, and graceful degradation. [7]

3.3 PHP

PHP is a general-purpose scripting language especially suited to web development. PHP code is usually processed on a web server by a PHP interpreter implemented as a module, a daemon or as a Common Gateway Interface (CGI) executable. On a web server, the result of the interpreted and executed PHP code – which may be any type of data, such as generated HTML or binary image data – would form the whole or part of an HTTP response. [8]

3.4 JavaScript

JavaScript often abbreviated as JS, is a programming language that conforms to the ECMAScript specification. JavaScript is high-level, often just-in-time compiled, and multi-paradigm. It has curly-bracket syntax, dynamic typing, prototype-based object-orientation, and first-class functions. [5]

We used JavaScript specially ajax, for some important functionalities like error handling warnings, page transformation functions and debugging through. [9]

3.4.1 Node.js

The Node.js run-time environment includes everything that need to execute a program written in JavaScript. Many people use the JavaScript programming language extensively for programming the interfaces of websites. Node.js allows this popular programming language to be applied in many more contexts, in particular on web servers. JavaScript and Node.js run on the V8 JavaScript runtime engine. This engine takes your JavaScript code and converts it into a faster machine code. It has numerous framework to build web application interfaces (APIs). For this project Express module is used. [10][11]

4 Conclusion

The project DigitalSchoolManagement met all the provided requirements from the perspective of admin, teacher, and student view. The experience of working on the project was well worth of time. The most important goal of this project was the ability to work with API where I got the clear idea about the web application programming interface work flow. Specially a popular API concept RESTful API architecture is very clear to me now.

By doing this project I also came to learn the control of asynchronous JavaScript. It was initially hard to implement logics on another server and store/fetch them to render on the interface. But as I learned more each day, things became easier. Finally, the project requirements were well-defined, easy to understand, and helpful in avoiding misunderstandings during the project's development.

References

- [1] Daniel Richter; Datenbanken und Web-Techniken, Project Task, Summer Semester 2021; tu-chemnitz.de; [Online]. Available: <https://www.tu-chemnitz.de/informatik/DVS/lehre/DBW/Projekt/> [Accessed:31-May-2021].
- [2] REST APIs, The glue between the frontend and backend; github.com; [Online]. Available: <https://github.com/UFWWebApps/REST-API-Tutorial> [Accessed:3-Jul-2021].
- [3] Modeling Tutorials, mysql.com; [Online]. Available: <https://dev.mysql.com/doc/workbench/en/wb-tutorials.html> [Accessed:3-Jul-2021].
- [4] HTML, javapoint.com; [Online]. Available: <https://www.javatpoint.com/what-ishtml>. [Accessed:3-Jul-2021].
- [5] CSS, tutorialspoint.com; [Online]. Available: https://www.tutorialspoint.com/css/what_is_css.htm [Accessed:3-Jul-2021].
- [6] Bootstrap 4, w3schools.com; [Online]. Available: https://www.w3schools.com/whatis/whatis_bootstrap.asp. [Accessed:3-Jul-2021].
- [7] Materialize CSS, materializecss.com; [Online]. Available: <https://materializecss.com> [Accessed:3-Jul-2021].
- [8] PHP, php.net [Online]. Available: <https://www.php.net/> [Accessed:3-Jul-2021].
- [9] JavaScript; tutorialspoint.com; [Online]. Available: https://www.tutorialspoint.com/javascript/javascript_overview.htm [Accessed:3-Jul-2021].
- [10] Node.js - runtime built, nodejs.com; [Online]. Available: <https://nodejs.org/> [Accessed:3-Jul-2021].
- [11] Fast, unopinionated, minimalist web framework for node; npmjs.com; [Online]. Available: <https://www.npmjs.com/package/express> [Accessed:3-Jul-2021].

Appendix

API Documentation

List of Endpoints

The following is a full overview of all endpoints. Please see the following pages for details.

Resources	Method	Endpoints
User	POST POST GET PUT DELETE GET GET GET GET GET GET GET GET	/users/register /users/login /users/show/{Id} /users/{Id} /users/{Id} /users /users/student/{Id} /users/teacher /users/student /users/subjects/{Id} /users/student/meanResult/{Id} /users/teacher/subjects/{Id} /users/subject/students/{Id}/{subject_id} /users/student/grades/{Id}
Class	POST GET GET PUT DELETE GET GET GET GET GET	/classes /classes /classes/{Id} /classes/{class_Id}/{user_Id} /classes/{Id} /classes/studentAssign/{Id}/{class_id} /classes/studentDeassign/{Id}/{class_id} /classes/subjects/{class_id} /classes/assignedStudent/{class_id} /classes/notAssignedStudent/{class_id}
Subject	GET GET GET DELETE POST POST POST GET GET	/subjects/{user_Id} /subjects/show/{subject_id} /subjects/details/{userId/{subjectId} /subjects/{subject_id} /subjects/teacherAssign/{subject_id} /subjects/addEditSubject/{operation_type} /subjects/archive/{subject_id} /subjects/show/{Id/{subject_id} /subjects/tests/{Id/{subject_id}
Test	POST GET PUT DELETE GET POST GET PUT	/tests/{subject_id} /tests/{test_id} /tests/{Id} /tests/{test_id} /tests/details/{Id/{test_id} /tests/importGrades/{Id} /tests/student_grades/{Id}/{test_id} /tests/grades/{Id}

Details of Endpoints

POST /users/register	
Request	
Headers	Content-Type: application/x-www-form-urlencoded
Params	
Sample Body	<pre>{ "user_name": "rahim", "password": "123", "first_name": "Md", "last_name": "Rahim", "user_type": "student", "created_at": "2021-07-03 17:26:38" }</pre>
Response	
Status	201 (Success), 409 (Conflict), 500 (Error)
Headers	location: {redirect to user list url}
Sample Body	"Created successfully"

POST /users/login	
Request	
Headers	Content-Type: application/x-www-form-urlencoded
Params	
Sample Body	<pre>{ "username": "rahim", "password": "123" }</pre>
Response	
Status	202 (Success), 500 (Error)
Headers	location: {redirect to home url}
Sample Body	"Logged successfully"

GET /users/show/{Id}	
Request	
Headers	Content-Type: application/json

Params	id: int (required)
Sample Body	
Response	
Status	200 (Success), 500 (Error)
Sample Body	<pre>{ "user_id": 2, "user_name": "rahim", "password": "\$2b\$10\$2lQbut7sT9xAJLpM2U9.b.iTSxCXDIIYJ9oBPGsqbOXG53obQuJk6", "first_name": "Md", "last_name": "Rahim", "user_type": "student", "created_at": "2021-07-03T15:29:52.000Z" }</pre>

PUT	/users/{Id}
Request	
Headers	Content-Type: application/x-www-form-urlencoded
Params	id: int (required)
Sample Body	<pre>{ "first_name": "Md", "last_name": "Rahim", "password": "123" }</pre>
Response	
Status	202 (Success), 404 (Error), 500 (Error)
Headers	location: {redirect to home url}
Sample Body	"Updated successfully"

DELETE	/users/{Id}
Request	
Headers	Content-Type: text/html
Params	id: int (required)
Sample Body	
Response	
Status	200 (Ok), 404 (Error), 500 (Error)
Headers	location: {redirect to home url}
Sample Body	"Deleted successfully"

GET	/users
Request	
Headers	Content-Type: application/json
Params	
Sample Body	
Response	
Status	200 (Success), 500 (Error)
Sample Body	<pre>[{ "user_id": 1, "user_name": "admin", "password": "\$2b\$10\$caHdrcuytvqnHfT/6hB0AunX5Vnc4WzRMiKfVU2Ju6fWwgT9gMioO", "first_name": "Admin", "last_name": "Admin", "user_type": "admin", "created_at": "2021-07-03T15:26:38.000Z" }, { "user_id": 2, "user_name": "rahim", "password": "\$2b\$10\$2lQbut7sT9xAJLpM2U9.b.iTSxCXDIIYJ9oBPGsqbOXG53obQujk6", "first_name": "Md", "last_name": "Rahim", "user_type": "student", "created_at": "2021-07-03T15:29:52.000Z" }, { "user_id": 3, "user_name": "krishna", "password": "\$2b\$10\$0tDIlcDJ6ly.B1WABEhcO.hfMmupDTmC50LjnZ6VLewKIPJe87X9m", "first_name": "Krishna", "last_name": "Murthy", "user_type": "teacher", "created_at": "2021-07-03T15:30:19.000Z" }]</pre>

GET	/users/teacher
Request	
Headers	Content-Type: application/json
Params	
Sample Body	
Response	
Status	200 (Success), 500 (Error)

Sample Body	<pre>[{ "user_id": 3, "user_name": "krishna", "password": "\$2b\$10\$0tDIlcDJ6ly.B1WABEhcO.hfMmupDTmC50LjnZ6VLewKIPJe87X9m", "first_name": "Krishna", "last_name": "Murthy", "user_type": "teacher", "created_at": "2021-07-03T15:30:19.000Z" }, { "user_id": 4, "user_name": "nixon", "password": "\$2b\$10\$6/b24puRrleSy3G11EjxEeCCgcjfqMrhUvMiW2K8lwHCCqLyAQtlK", "first_name": "Nixon", "last_name": "Barua", "user_type": "teacher", "created_at": "2021-07-03T15:30:57.000Z" }]</pre>
-------------	--

GET /users/student	
Request	
Headers	Content-Type: application/json
Params	
Sample Body	
Response	
Status	200 (Success), 500 (Error)
Sample Body	<pre>[{ "user_id": 5, "user_name": "pranto", "password": "\$2b\$10\$0tDIlcDJ6ly.B1WABEhcO.hfMmupDTmC50LjnZ6VLewKIPJe87X9m", "first_name": "Pranto", "last_name": "Ahmed", "user_type": "student", "created_at": "2021-07-03T15:30:19.000Z" }, { "user_id": 6, "user_name": "arnob", "password": "\$2b\$10\$6/b24puRrleSy3G11EjxEeCCgcjfqMrhUvMiW2K8lwHCCqLyAQtlK", "first_name": "Arnob", "last_name": "Chowdhury", "user_type": "student", "created_at": "2021-07-03T15:30:57.000Z" }]</pre>

GET	/users/subjects/{Id}
-----	----------------------

Request	
Headers	Content-Type: application/json
Params	id: int (required)
Sample Body	
Response	
Status	200 (Success), 500 (Error)
Sample Body	<pre>{ "subject_id": 1, "subject_name": "Math 1", "class_id": 1, "is_archived": 0, "class_name": "Class 1", "test_id": 1, "test_name": "Math", "is_complete": 1, "user_id": 6, "date": "2021-07-03T15:29:52.000Z" }</pre>

GET	/users/student/meanResult/{Id}
Request	
Headers	Content-Type: application/json
Params	id: int (required)
Sample Body	
Response	
Status	200 (Success), 500 (Error)
Sample Body	<pre>{ "AVG(m.marks)": 1.2 }</pre>

GET	/users/teacher/subjects/{Id}
Request	
Headers	Content-Type: application/json
Params	id: int (required)
Sample Body	
Response	

Status	200 (Success), 500 (Error)
Sample Body	<pre>{ "subject_id": 2, "subject_name": "Math 2", "class_id": 2, "is_archived": 0, "class_name": "Class 2" }</pre>

GET	/users/subject/students/{id}/{subject_id}
Request	
Headers	Content-Type: application/json
Params	id: int (required); subject_id: int (required);
Sample Body	
Response	
Status	200 (Success), 500 (Error)
Sample Body	<pre>{ "subject_id": 2, "subject_name": "Math 2", "class_id": 2, "is_archived": 0, "class_name": "Class 2", "is_archived": 0, "user_id": 6, "AverageGrade": 2.5 }</pre>

GET	/users/student/grades/{id}
Request	
Headers	Content-Type: application/json
Params	id: int (required)
Sample Body	
Response	
Status	200 (Success), 500 (Error)
Sample Body	<pre>{ "subject_id": 2, "subject_name": "Math 2", "class_id": 2, "is_archived": 0, "class_name": "Class 2", "AverageGrade": 2.3 }</pre>

POST	/classes
Request	
Headers	Content-Type: application/x-www-form-urlencoded
Params	
Sample Body	<pre>{ " Class 4": "Class 3" }</pre>
Response	
Status	202 (Success), 409 (Conflict), 500 (Error)
Headers	location: {redirect to class list url}
Sample Body	"Logged successfully"

GET	/classes
Request	
Headers	Content-Type: application/json
Params	
Sample Body	
Response	
Status	200 (Success), 500 (Error)
Sample Body	<pre>[{ "class_id": 1, "class_name": "Class 1", "is_archived": 0 }, { "class_id": 2, "class_name": "Class 2", "is_archived": 1 }]</pre>

GET	/classes/{id}
Request	
Headers	Content-Type: application/json
Params	id: int (required)
Sample Body	
Response	

Status	200 (Success), 500 (Error)
Sample Body	<pre>[{ "class_id": 0, "class_name": "Class 1", "is_archived": 0 }]</pre>

PUT	/classes/{class_Id}/{user_Id}
Request	
Headers	Content-Type: application/x-www-form-urlencoded
Params	class_Id: int (required); user_Id (required)
Sample Body	<pre>{ "class_name ": "Class 6" }</pre>
Response	
Status	202 (Success), 404 (Error), 500 (Error)
Headers	location: {redirect to class list url}
Sample Body	"Updated successfully"

DELETE	/classes/{Id}
Request	
Headers	Content-Type: text/html
Params	id: int (required)
Sample Body	
Response	
Status	200 (Ok), 404 (Error), 500 (Error)
Headers	location: {redirect to class list url}
Sample Body	"Deleted successfully"

GET	/classes/studentAssign/{Id}/{class_id}
Request	
Headers	Content-Type: application/json
Params	Id: int (required); class_id (required)

Sample Body	
Response	
Status	200 (Success), 500 (Error)
Sample Body	"Assigned successfully"

GET	/classes/studentDeassign/{id}/{class_id}
Request	
Headers	Content-Type: application/json
Params	Id: int (required); class_id (required)
Sample Body	
Response	
Status	200 (Success), 500 (Error)
Sample Body	"Deassigned successfully"

GET	/classes/subjects/{class_id}
Request	
Headers	Content-Type: application/json
Params	class_id: int (required)
Sample Body	
Response	
Status	200 (Success), 500 (Error)
Sample Body	<pre>[{ "subject_id": 2, "subject_name": "Math 2", "class_id": 2, "class_name": "Class 2", "is_archived": 0 }, { "subject_id": 4, "subject_name": "English 2", "class_id": 2, "class_name": "Class 2", "is_archived": 0 }]</pre>

GET	/classes/assignedStudent/{class_id}
Request	
Headers	Content-Type: application/json
Params	class_id: int (required)
Sample Body	
Response	
Status	200 (Success), 500 (Error)
Sample Body	<pre>[{ "user_id": 5 }, { "user_id": 6 }]</pre>

GET	/classes/notAssignedStudent/{class_id}
Request	
Headers	Content-Type: application/json
Params	class_id: int (required)
Sample Body	
Response	
Status	200 (Success), 500 (Error)
Sample Body	<pre>[{ "user_id": 7 }, { "user_id": 8 }, { "user_id": 9 }]</pre>

GET	/subjects/{user_id}
Request	

Headers	Content-Type: application/json
Params	user_id: int (required)
Sample Body	
Response	
Status	200 (Success), 401(Unauthorized), 403(Forbidden), 500 (Error)
Sample Body	<pre>[{ "subject_id": 8, "subject_name": "Math 1", "class_id": 1, "is_archived": 0, "class_name": "Class 1", "user_id": 2, "user_name": "rahim" }, { "subject_id": 5, "subject_name": "Science 1", "class_id": 1, "is_archived": 0, "class_name": "Class 1", "user_id": 9, "user_name": "jafar" }, { "subject_id": 2, "subject_name": "Math 1", "class_id": 2, "is_archived": 0, "class_name": "Class 2", "user_id": 2, "user_name": "rahim" }]</pre>

GET	/subjects/show/{subject_id}
Request	
Headers	Content-Type: application/json
Params	subjects_id: int (required)
Sample Body	
Response	

Status	200 (Success), 401(Unauthorized), 403(Forbidden), 500 (Error)
Sample Body	<pre>[{ "subject_id": 1, "subject_name": "Math 1", "class_id": 1, "is_archived": 0 }]</pre>

GET /subjects/details/{userId}/{subjectId}	
Request	
Headers	Content-Type: application/json
Params	user_id: int (required) subjects_id: int (required)
Sample Body	
Response	
Status	200 (Ok), 204(No Content), 404(Error), 500 (Error)
Sample Body	<pre>[{ "test_name": "Math", "marks": 1.2 }]</pre>

DELETE /subjects/{subject_id}	
Request	
Headers	Content-Type: text/html
Params	subject_id: int (required)
Sample Body	
Response	
Status	200 (Ok), 404 (Error), 500 (Error)
Headers	location: {redirect to subject list of class url}
Sample Body	"Deleted successfully"

POST /subjects/teacherAssign/{subject_id} Optional	
Request	
Headers	application/x-www-form-urlencoded
Params	subject_id: int (required)
Sample Body	<pre>[{ "id": 5 }]</pre>
Response	
Status	201 (Success), 500 (Error)
Headers	location: {redirect to subject list of class url}
Sample Body	"Deleted successfully"

POST /subjects/addEditSubject/{operation_type}	
Request	
Headers	application/x-www-form-urlencoded
Params	operation_type: string (required)
Sample Body	<pre>{ "user_id": 2, "subject_name": "Math 4", "class_id": 1, "subject_id": 1 }</pre>
Response	
Status	200 (Ok), 201 (Success), 404 (Error), 500 (Error)
Headers	location:
Sample Body	"Deleted successfully"

POST /subjects/archive/{subject_id}	
Request	
Headers	application/x-www-form-urlencoded
Params	subject_id: int (required)
Sample Body	
Response	
Status	200 (Ok), 404 (Error), 500 (Error)

Headers	location: {redirect to subject list of class url}
Sample Body	"Archived Successfully"

GET /subjects/show/{Id}/{subject_id}	
Request	
Headers	Content-Type: application/json
Params	Id: int (required) subjects_Id: int (required)
Sample Body	
Response	
Status	200 (Ok), 401 (Unauthorized), 404 (Error), 500 (Error)
Sample Body	<pre>{ "subject_id": 1, "subject_name": "Math 4", "is_archived": 1, "class_id": 1, "class_name": "Class 1", "user_id": 1, "user_name": "rahim" }</pre>

GET /subjects/tests/{Id}/{subject_id}	
Request	
Headers	Content-Type: application/json
Params	Id: int (required) subjects_Id: int (required)
Sample Body	
Response	
Status	200 (Ok), 204(No Content), 404 (Error), 500 (Error)

Sample Body	<pre>[{ "test_id": 1, "test_name": "Math", "subject_id": 1, "date": "2021-07-03T15:43:12.000Z", "is_complete": 1 }, { "test_id": 1, "test_name": " Math test 2", "subject_id": 2, "date": "2021-07-03T15:43:12.000Z", "is_complete": 0 }, { "test_id": 3, "test_name": " Math test 3", "subject_id": 1, "date": "2021-07-03T15:43:12.000Z", "is_complete": 0 }]</pre>
-------------	---

POST /tests/{subject_id}	
Request	
Headers	application/x-www-form-urlencoded
Params	subject_id: int (required)
Sample Body	
Response	
Status	200 (Ok), 404 (Error), 500 (Error)
Headers	location: {redirect to test list of subject url}
Sample Body	<pre>[{ "test_name": "Math test 4", "subject_id": 1, "date": "2021-07-03T15:43:12.000Z", "is_complete": 0 }]</pre>

GET /tests/{test_id}	
Request	
Headers	Content-Type: application/json
Params	test_id: int (required)
Sample Body	
Response	
Status	200 (Ok), 404 (Error), 500 (Error)
Sample Body	<pre>[{ "test_id": 3, "test_name": " Math test 3", "subject_id": 1, "date": "2021-07-03T15:43:12.000Z", "is_complete": 0 }]</pre>

PUT /tests/{Id}	
Request	
Headers	Content-Type: application/x-www-form-urlencoded
Params	Id: int (required)
Sample Body	<pre>[{ "test_id": 2, "test_name": " Math test 2", "subject_id": 2, "date": "2021-07-03T15:43:12.000Z", "is_complete": 0 }]</pre>
Response	
Status	200 (Success), 404 (Error), 500 (Error)
Headers	location: {redirect to test list of subject url}
Sample Body	"Updated successfully"

DELETE /tests/{test_id}	
---------------------------------------	--

Request	
Headers	Content-Type: text/html
Params	test_id: int (required)
Sample Body	
Response	
Status	200 (Ok), 404 (Error), 500 (Error)
Headers	location: {redirect to test list of subject url}
Sample Body	"Deleted successfully"

GET	/tests/details/{Id}/{test_id}
Request	
Headers	Content-Type: application/json
Params	test_id: int (required)
Sample Body	
Response	
Status	200 (Ok), 404 (Error), 500 (Error)
Sample Body	<pre>[{ "user_id": 5, "user_name": "pranto", "first_name": "Pranto", "last_name": "Ahmed", "test_id": 1, "test_name": "Math", "marks": 1.2 }, { "user_id": 6, "user_name": "arnob", "first_name": "Arnob", "last_name": "Chowdhury", "test_id": 1, "test_name": "Math", "marks": 2.5 }]</pre>

POST	/tests/importGrades/{Id}
-------------	---------------------------------

Request	
Headers	application/x-www-form-urlencoded
Params	Id: int (required)
Sample Body	<pre>{ "test_id": 1, "file": "Mark.csv", }</pre>
Response	
Status	200 (Ok), 404 (Error), 500 (Error)
Headers	location: {redirect to test list of subject url}
Sample Body	"CSV file uploaded successfully"

GET	/tests/student_grades/{Id}/{test_id}
Request	
Headers	Content-Type: application/json
Params	Id: int (required) test_id: int (required)
Sample Body	
Response	
Status	200 (Ok), 204 (No Content), 404 (Error), 500 (Error)
Sample Body	<pre>[{ "mark_id": 2, "test_id": 1, "user_id": 6, "marks": 2.5 }]</pre>

PUT	/tests/grades/{Id}
Request	
Headers	Content-Type: application/x-www-form-urlencoded
Params	Id: int (required)

Sample Body	<pre>[{ "marks": 2.0, "test_id": 1 }]</pre>
Response	
Status	200 (Success), 404 (Error), 500 (Error)
Headers	location: {redirect to test list of subject url}
Sample Body	"Updated successfully"