HCMC UNIVERSITY OF TECHNOLOGY AND EDUCATION

FACULTY FOR HIGH QUALITY TRAINING

INFORMATION TECHNOLOGY

THE FIRST PROJECT REPORT

STUDENT MANAGEMENT SYSTEM

LECTURER NAME : Dr. Nguyen Dang Quang

STUDENT NAME : Nguyen Hong Thai

STUDENT ID 21AD11040

Ho Chi Minh City, December 2019

Acknowledgment

The course project is a significant assignment that requires students to apply all the knowledge acquired during their studies and practical experiences to complete. It marks a milestone in the development of generations of students at the University of Technical Education in Ho Chi Minh City. For this course, I would like to express my profound gratitude to Mr. Nguyen Dang Quang, the supervisor of my research topic. He is a dedicated lecturer who is highly committed to his students, demonstrating the ability to broaden students' perspectives during the learning and research process. He consistently provides favorable conditions to help students overcome obstacles throughout the project implementation.

Sincerely thank you!

Preface

The purpose and objective of this training and mainly the content is time-being, and with this training, I have gained some confidence regarding introducing the application. I also believe that way I gained some sorts of IT knowledge, and if I practice much and having some expertise in the field, then I will be able to survive smartly in today's competitive environment. The effort to write the report is a partial fulfilment to complete the course. In the report, I try my best to represent all the content that I learned in a great deal in the program in a systematic and presentable order. I divided each of the topics as an individual chapter to reflect the entire topic more prominently and clearly. In reference, I have used the citation method in the entire report. Finally, I am very hopeful that the structure and topic of the report will be a useful material for all the reader, especially to the user.

Content

I. Project Description

1. Objectives

The main goals of student information management are multifaceted. First and foremost, the accurate, comprehensive, and secure collection and storage of student information is absolutely essential. This not only helps track the learning progress of each student, but also ensures the privacy and security of the data.

Additionally, student information management aims to monitor and manage the learning process of students. This includes tracking academic progress, learning outcomes, managing grades, records, and evaluations. From this, administrators can develop appropriate learning plans, as well as manage student schedules and attendance.

Another key objective is to support decision-making and planning. The student information management system will provide timely and accurate information to help administrators and teachers make effective decisions about academics, discipline, and counseling. This information also supports planning, forecasting, and resource allocation to meet student needs.

By focusing on these goals, the site hopes to provide users who manage students more easily.

2. User Benefits

The student management provides with the abilities to add, edit, sort, filter (gender), delete. Essential managing functions are:

- Read: Users can view information for each object.

- Add: The user adds the object by filling in the blank information and the object is added to the table.
- Edit: The user can change the information of the object in the table.
- Sort: Sort objects in order from (A-Z) or (small to large) if numeric.
- Filter (gender): Can filter out only male or female.
- Delete: Users can select 1 object or multiple objects to delete from the system.

3. Use Case Diagram

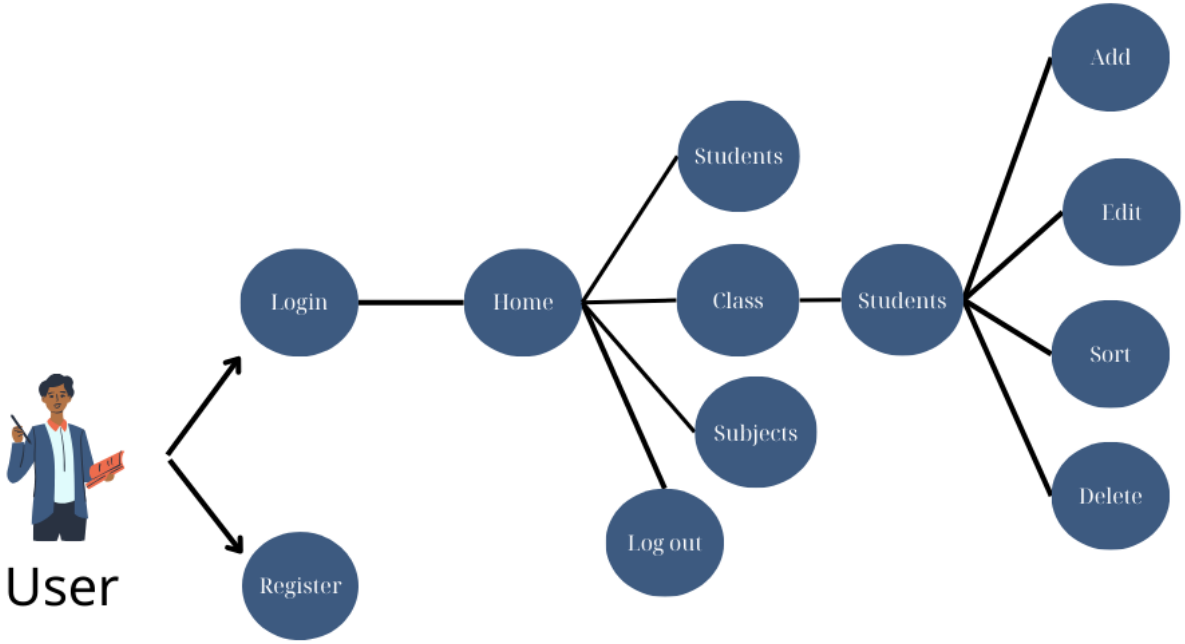


Image 1-Use case diagram

4. Use Case Description Tables

Use case name	Register
Actor	User
Goal	Gain an account to access to website
Preconditions	User has email and password

Main flow of events	User accesses the system's website
	User clicks on `Don't have an account? Sign Up`
	User input required elements (email, password, etc.)
Alternative flows	
Postconditions	User receives notification of incorrect password and the option to retry

Table 1-Use case register description

Use case name	Login
Actor	User
Goal	Gain access to the student management system
Preconditions	User has a valid account
Main flow of events	User accesses the system's website
	User enters username and password
	System authenticates login credentials
	System grants user access if login credentials are valid
Alternative flows	Incorrect email and password
Postconditions	User receives notification of incorrect password and the option to retry

Table 2-Use case login description

Use case name	Add
Actor	User
Goal	Enroll a new student into the student management system
Preconditions	User is logged in and new student information is prepared

Main flow of events	User clicks on the “more detail” of total student
	User clicks on add button
	User input the required student information
	The system validates the entered information for accuracy and completeness.
	Upon successful validation, the system creates a new student record and assigns a unique student identification number
	The system displays the newly created student record for confirmation
Alternative flows	
Postconditions	A new student record is created in the system, and the student's unique ID is generated

Table 3-Use case add description

Use case name	Edit
Actor	User
Goal	Update student information
Preconditions	User is logged in
Main flow of events	User clicks on the “more detail” of total student
	User clicks on edit button
	User modifies student information
	User clicks on edit button to save change
	System stores the updated student information
Alternative flows	
Postconditions	User can update student information including name, email, gender, course, class

Table 4-Use case edit description

Use case name	Sort
Actor	User
Goal	Arrange student (A-Z) or (Z-A)

Preconditions	User is logged in
Main flow of events	User clicks on the “more detail” of total student
	User clicks on header name (ex: ID, name,… etc.)
	System will arrange by selected header name
Alternative flows	
Postconditions	User can arrange header name including ID, name, GPA

Table 5-Use case sort description

Use case name	Delete
Actor	User
Goal	Delete student
Preconditions	User is logged in
Main flow of events	User clicks on the “more detail” of total student
	User clicks 1 or more students
	User clicks on dustbin icon
	System deletes student
Alternative flows	
Postconditions	Student has deleted successful

Table 6-Use case delete description

II. Design

1. Process Description

Build a student management system using ReactJS and Node.JS. The system should allow users to add, edit, delete, filter, and sort student profiles. You will be provided with an array of student objects, each object containing the following properties: ID, name, gender, email, class, course, and GPA. In this architecture, Vitejs is responsible for rendering the user interface (UI) of the application. When a user interacts with the UI, Vitejs sends requests to the Node.js backend. The Node.js backend then retrieves data from the MySQL database and sends it back to Vitejs. Vitejs then updates the UI based on the data that it has received from the backend.

2. API Endpoints

ID	URL	Method	Description	Params	Return
1	http://localhost:3000/register	Post	User register	Firstname, lastname, email, password	User_id, fullname, password
2	http://localhost:3000/login	Post	User logins	Email, password	User_id
3	http://localhost:3000/students	Get	User views student list		Student list
4	http://localhost:3000/students/create	Post	User adds student	Name, gender, email, course,	Stu_id, name, gender, email, course, class

				class	
5	http://localhost:3000/students/:id	Put	User modifies student's information	name, gender, email, course, class, grade	Updated student information
6	http://localhost:3000/students/:id	Delete	User deletes student		Deleted student

Table 7-API

3. ERD Diagram

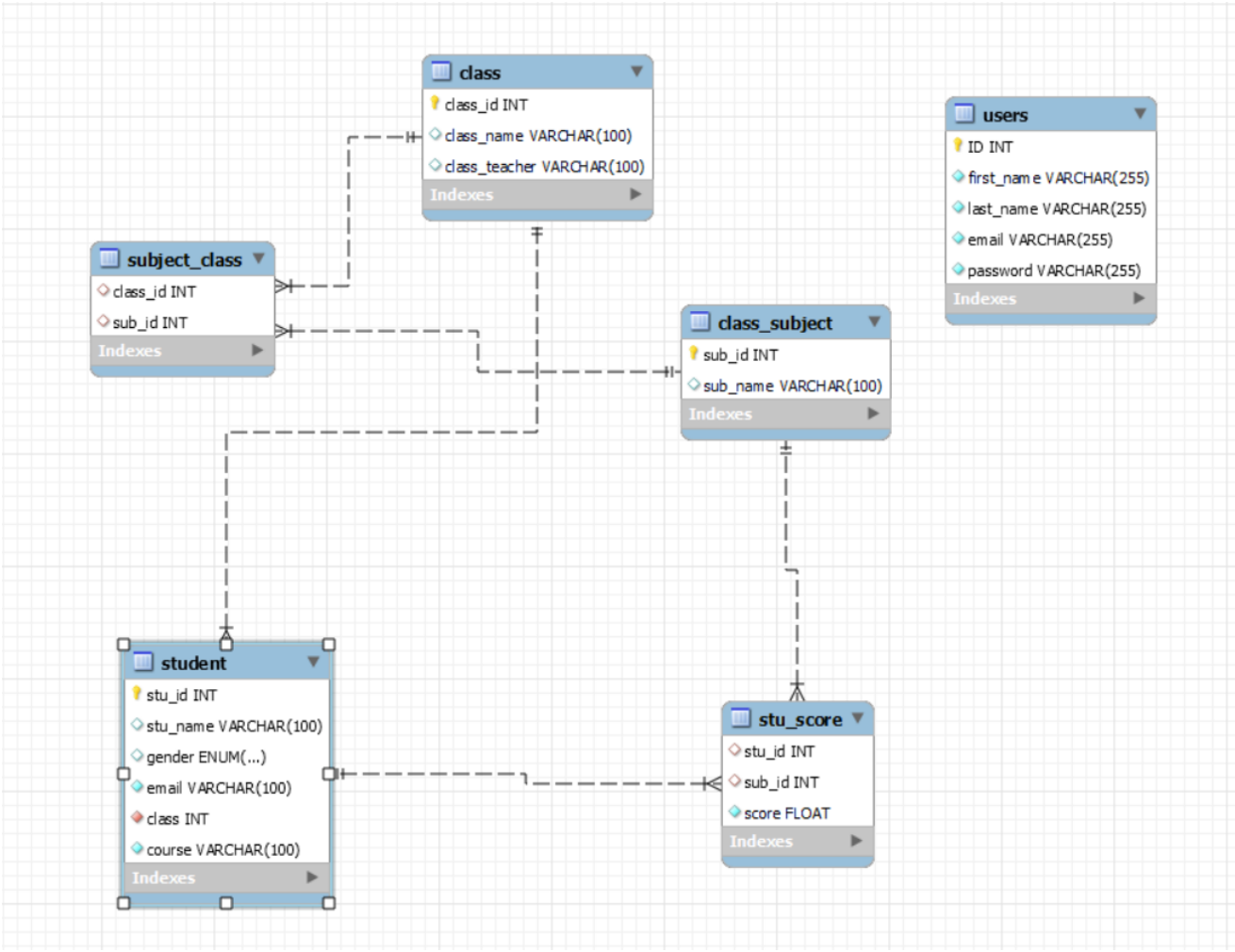


Image 2

The image shows a database schema, representing the structure of a relational database. It includes tables like student, class, users, and

relationships between them. The lines between tables indicate foreign key constraints that link related data across different tables, essential for maintaining referential integrity within the database. Each table has a list of columns with data types and constraints specified.

4. Data tables

a. Users

ID	first_name	last_name	email	password
1	John	Doe	john.doe@example.com	1
2	Jane	Smith	jane.smith@example.com	1
4	Emily	Brown	emily.brown@example.com	1
5	David	Lam	Lam@lam.com	1
6	Thai	Hong	ht@ht.com	1
7	Tu	Smith	tutu@example.com	1

Image 3-Data table of users

b. Class

	class_id	class_name	class_teacher
▶	2	10A2	Sam
	3	10A3	Jayden
	4	10A4	Smith
	5	10A5	George
	6	10A6	Peter
	7	11A1	Sammuel
	8	11A2	Deva
	9	11A3	Sasha
	10	11A4	Mike
	11	11A5	Candace
	12	12A1	Tiffany
	13	12A2	Bob
	14	12A3	Katie
	15	12A4	Lory

Image 4-Data table of class

c. Subject

sub_id	sub_name
1	Math
2	English
3	Physics
4	Litetures
5	Chemistry

Image 5-Data table of subject

d. Student

stu_id	stu_name	gender	email	class	course
2	Jane Smith	female	jane.smith@example.com	6	2021-2022
4	Emily Brown	female	emily.brown@example.com	3	2021-2022
5	William Taylor	male	william.taylor@example.com	4	2021-2022
6	Sophia Martinez	female	sophia.martinez@example.com	4	2021-2022
7	Ethan Anderson	male	ethan.anderson@example.com	5	2021-2022
8	Olivia Wilson	female	olivia.wilson@example.com	5	2021-2022
10	Alexander Thomas	male	alexander.thomas@example.com	6	2021-2022
11	Mia Garcia	female	mia.garcia@example.com	2	2021-2022
28	Tran Tu	male	trantu@gmail.com	3	2020-2021
29	Binh An	male	binhan@gmail.com	3	2021-2022

Image 6-Data table of student

e. Student's score

stu_id	sub_id	score
2	1	7
2	2	8
2	3	6.75
2	4	7.25
2	5	9
4	4	7.5
4	5	8
4	1	7
4	2	6
4	3	6
5	3	7

Image 7-Data table of student's score

III. Implementation

1. Back End

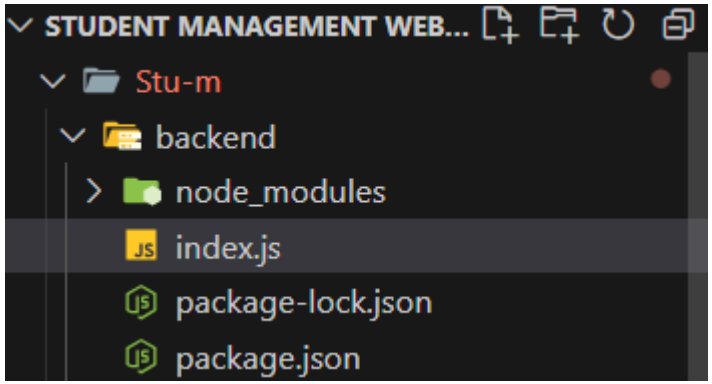


Image 8-Back-end folder structure

Source code samples

```
const express = require("express");
const app = express();
const port = 3000;
const cors = require("cors");
const mysql = require("mysql2");

const connection = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "080203",
  database: "students",
});

connection.connect((error) => {
  if (error) {
    console.error("Error connecting: " + error.stack);
```

```
    return;
  }

  console.log("Connected as id " + connection.threadId);
});

app.use(
  cors({
    origin: "http://localhost:5173",
  })
);

app.use(express.urlencoded({ extended: true }));

app.use(express.json());

app.use(express.static(__dirname + "/public"));
```

This code creates an Express.js application, connects to a MySQL database, configures CORS, and sets up middleware to handle HTTP requests. This will establish a foundation for developing a RESTful API.

Connect to mySQL database:

```
const mysql = require("mysql2");

const connection = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "080203",
```

```
  database: "students",
});

connection.connect((error) => {
  if (error) {
    console.error("Error connecting: " + error.stack);
    return;
  }

  console.log("Connected as id " + connection.threadId);
});
```

Show student

```
app.get("/students", (req, res) => {
  // Example query
  connection.query(
    "SELECT * FROM student st INNER JOIN class c WHERE st.class = c.class_id",
    (error, results, fields) => {
      if (error)
        return res.status(400).send({
          error: error.message,
        });

      const re = results.map((r) => ({
        stu_id: r.stu_id,
        stu_name: r.stu_name,
        email: r.email,
        gender: r.gender,
        class: r.class_name,
        course: r.course,
      }));

      return res.status(200).send(re);
    }
  );
```

```
});
```

Add student

```
app.post("/students/create", (req, res) => {
  const student = req.body;

  if (!student)
    return res.status(400).send({
      error: "student is required",
    });
  connection.query(
    `INSERT INTO student (stu_name,gender, email, course, class) VALUES ('${student.name}', '${student.gender}',
'${student.email}', '${student.course}', '${student.class}')`,
    (error, results, fields) => {
      if (error)
        return res.status(400).send({
          error: error.message,
        });

      return res.status(200).send({
        message: "student created",
        data: {
          studentId: results.insertId,
        },
      });
    }
  );
});
```

Edit student

```
app.put("/students/:id", (req, res) => {
  const id = req.params.id;
  if (!id)
```

```
    return res.status(400).send({
      error: "ID is required",
    });
  const student = req.body;
  if (!student)
    return res.status(400).send({
      error: "student is required",
    });
  // Initialize an array to store the update clauses
  const updateClauses = [];

  // Check each field and add it to the update clauses if it exists in the request body
  console.log(student.class);
  if (student.stu_name) updateClauses.push(`stu_name = '${student.stu_name}'`);
  if (student.class) updateClauses.push(`class = '${student.class}'`);
  if (student.gender) updateClauses.push(`gender = '${student.gender}'`);
  if (student.email) updateClauses.push(`email = '${student.email}'`);
  if (student.course) updateClauses.push(`course = '${student.course}'`);
  // Construct the SQL query with optional fields
  let query = `UPDATE student SET ${updateClauses.join(
    ","
  )} WHERE stu_id = ${id}`;
  connection.query(query, (error, results, fields) => {
    if (error)
      return res.status(400).send({
        error: error.message,
      });
    return res.status(200).send(results);
  });
});
```

Delete student

```
app.delete("/students/:id", (req, res) => {
  const id = req.params.id;
  if (!id)
```

```
    return res.status(400).send({
      error: "ID is required",
    });
  connection.query(
    `DELETE FROM student WHERE stu_id = ${id}`,
    (error, results, fields) => {
      if (error)
        return res.status(400).send({
          error: error.message,
        });
      return res.status(200).send(results);
    }
  );
});
```

2. Front End

a. Front end folder structure

Image 9-Front end folder structure

The images display the directory structure of a web application, likely designed with a component-based architecture such as React. This setup follows an MVC (Model-View-Controller) pattern, which separates the application logic into three interconnected components to differentiate internal data management from the methods by which the user is presented with and interacts with the

information.

Frontend: Contains various JavaScript (JSX) files representing different user-facing components. These include files like Navbar.jsx for navigation, AddScoreForm.jsx, and ClassTable.jsx for specific functionalities related to forms and tables. These primary JSX files are likely used or included by other files to render complete pages.

Components: The components directory contains subdirectories for various reusable components, such as add-form, class-table, edit-form, and protected-route. Each of these directories includes JSX files like AddForm.jsx, ClassTable.jsx, and EditForm.jsx, indicating specific functionalities for adding forms, displaying tables, and editing forms, respectively. The presence of a ScoreTable.jsx and StuTable.jsx suggests functionality related to managing scores and student information.

Views: The views directory is organized into several subdirectories such as addstu, classes, dashboard, editstu, home, log-in, root, sign-up, and subjects. Each subdirectory contains JSX files specific to that view, like AddStudent.jsx, Class.jsx, Dashboard.jsx, Edit.jsx, Home.jsx, Login.jsx, Root.jsx, Signup.jsx, and Subject.jsx. This structure suggests dedicated views for adding students, managing classes, displaying dashboards, editing student information, home pages, login forms, root components, sign-up forms, and subject details.

Contexts: Contains context-related files like userContext.jsx, which are likely used to manage and provide global state across the application.

Assets: The assets directory includes static files such as images (e.g., book.png, graduated.png, lecture.png, react.svg), which are used across various components and views to maintain a consistent visual style.

The structure of this web application facilitates separation of concerns and component reuse, making it easier to manage and maintain. The use of JSX files for components and views indicates a dynamic, client-side rendering approach typical of modern JavaScript frameworks like React.

Source code samples

Components

Add form:

```
import React, { useState } from "react";
import { TextField, Button, Stack } from "@mui/material";
import { Link, useNavigate } from "react-router-dom";
import ClassSelect from "../ClassSelect/ClassSelect";

const AddForm = () => {
  const [name, setName] = useState("");
  const [email, setEmail] = useState("");
  const [gender, setGender] = useState("");
  const [course, setCourse] = useState("");
  const [classes, setClasses] = useState("");
  const navigate = useNavigate();

  async function handleSubmit(event) {
    event.preventDefault();
    console.log(name, email, gender, course, classes);
    const requestOptions = {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({
```

```
      name: name,
      email: email,
      gender: gender,
      course: course,
      class: classes,
    }),
  };
  const response = await fetch(
    "http://localhost:3000/students/create",
    requestOptions
  );

  const data = await response.json();
  console.log({ data });
  const subjects = await fetch("http://localhost:3000/subjects");

  const subjectsData = await subjects.json();

  if (subjectsData) {
    await Promise.all(
      subjectsData.map(async (c) => {
        const scoreInit = await fetch(
          `http://localhost:3000/score/student/${data.data.studentId}/subject/${c.sub_id}`,
          {
            method: "POST",
            headers: {
              "Content-Type": "application/json",
            },
            body: JSON.stringify({
              score: 0,
            }),
          }
        );

        return scoreInit;
      })
    );
```

```
  }

  navigate("/home");
}

const handleClassSelectionChange = (classId) => {
  setClasses(classId);
};

return (
  <React.Fragment>
    <h2>New Student</h2>
    <form onSubmit={handleSubmit} action={<Link to="/login" />}>
      <Stack spacing={2} direction="row" sx={{ marginBottom: 4 }}>
        <TextField
          type="text"
          variant="outlined"
          color="secondary"
          label="Name"
          onChange={(e) => setName(e.target.value)}
          value={name}
          fullWidth
          required
        />
      </Stack>

      <TextField
        type="email"
        variant="outlined"
        color="secondary"
        label="Email"
        onChange={(e) => setEmail(e.target.value)}
        value={email}
        fullWidth
        required
        sx={{ mb: 4 }}
      />
```

```
        <TextField
          type="text"
          variant="outlined"
          color="secondary"
          label="gender"
          onChange={(e) => setGender(e.target.value)}
          value={gender}
          fullWidth
          required
          sx={{ mb: 4 }}
        />
        <TextField
          type="text"
          variant="outlined"
          color="secondary"
          label="course"
          onChange={(e) => setCourse(e.target.value)}
          value={course}
          fullWidth
          required
          sx={{ mb: 4 }}
        />
        <ClassSelect onClassSelectionChange={handleClassSelectionChange} />

        <Button variant="outlined" color="secondary" type="submit">
          Add
        </Button>
      </form>
    </React.Fragment>
  );
};

export default AddForm;
```

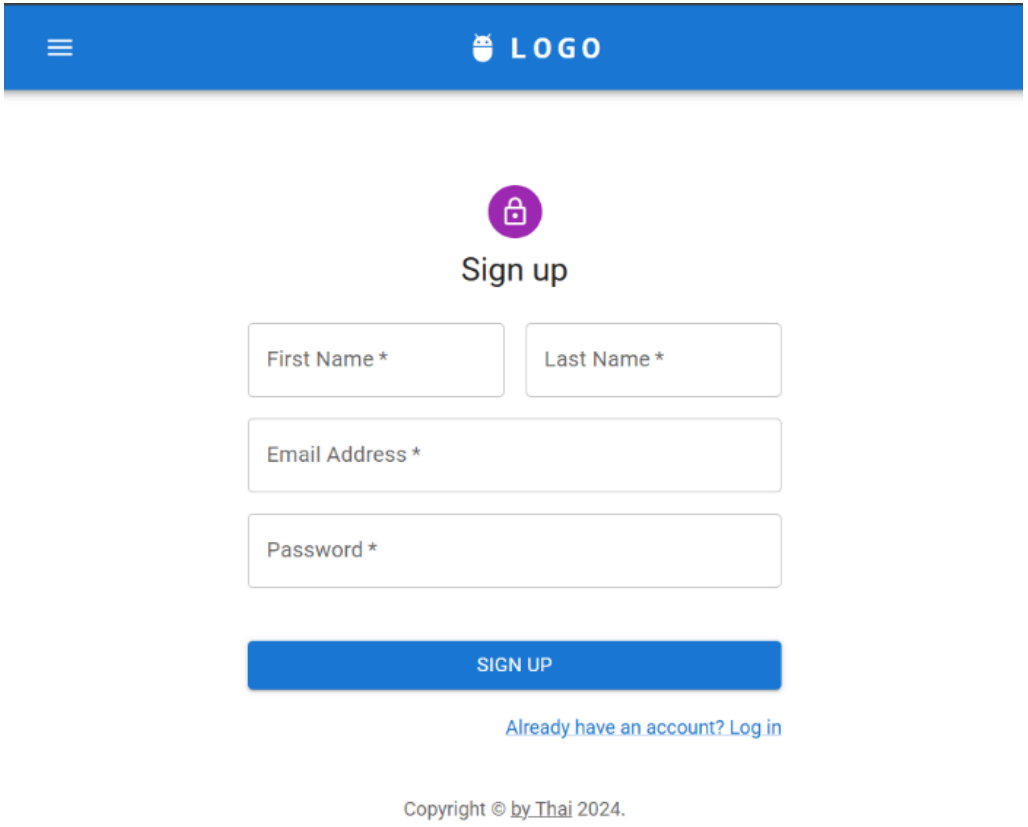
Views

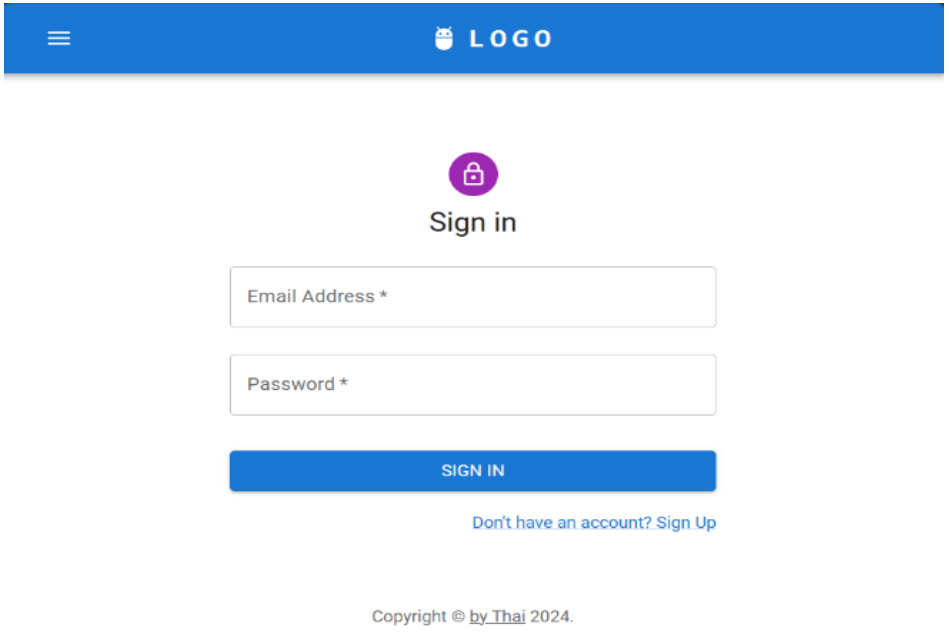
Add student:

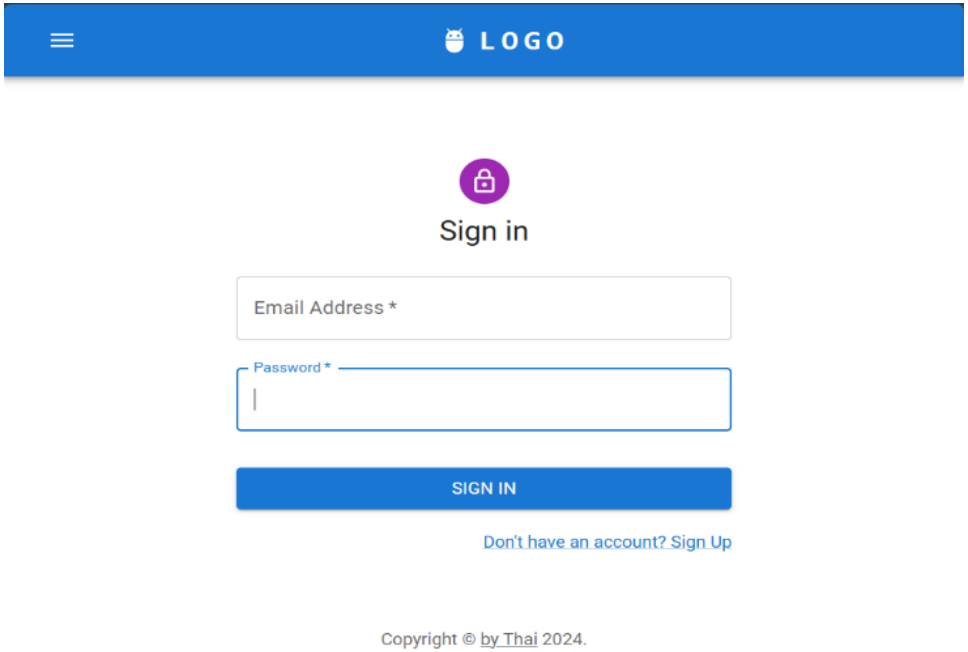
```
import AddForm from "../../components/add-form/AddForm";

const AddStudent = () => {
  return <AddForm />;
};

export default AddStudent;
```

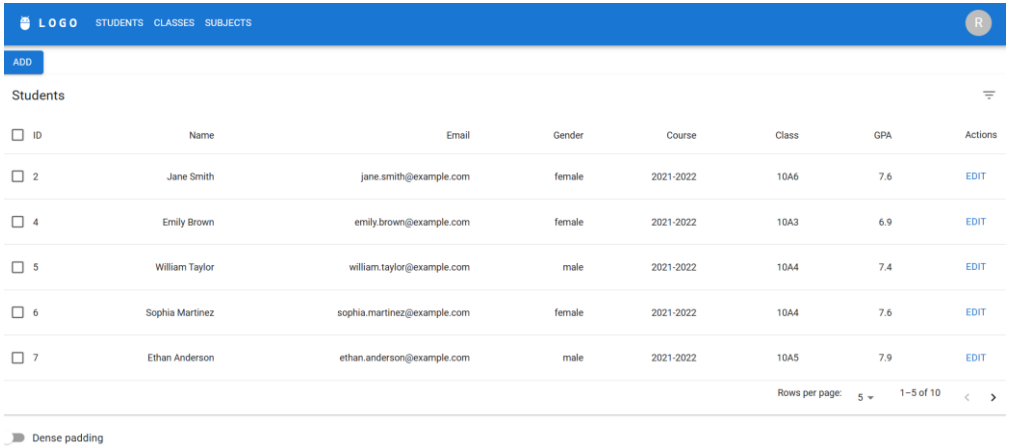
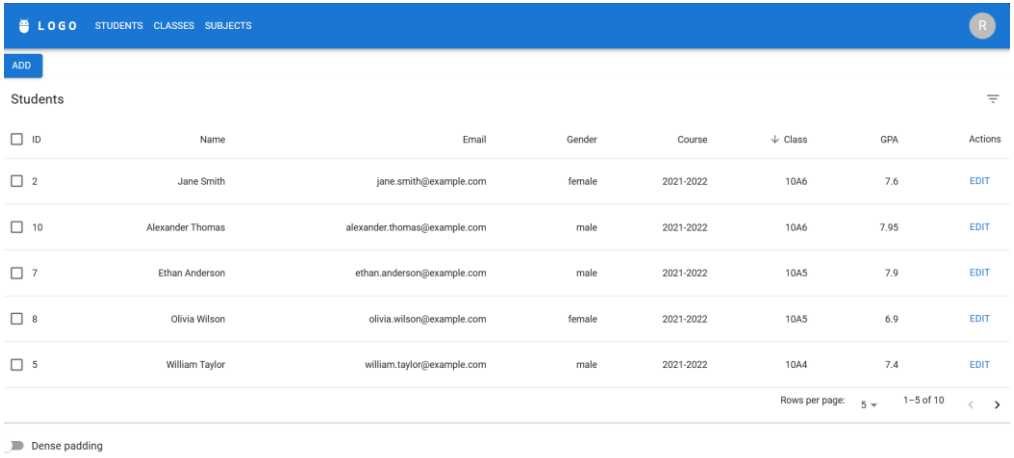
IV. Testing

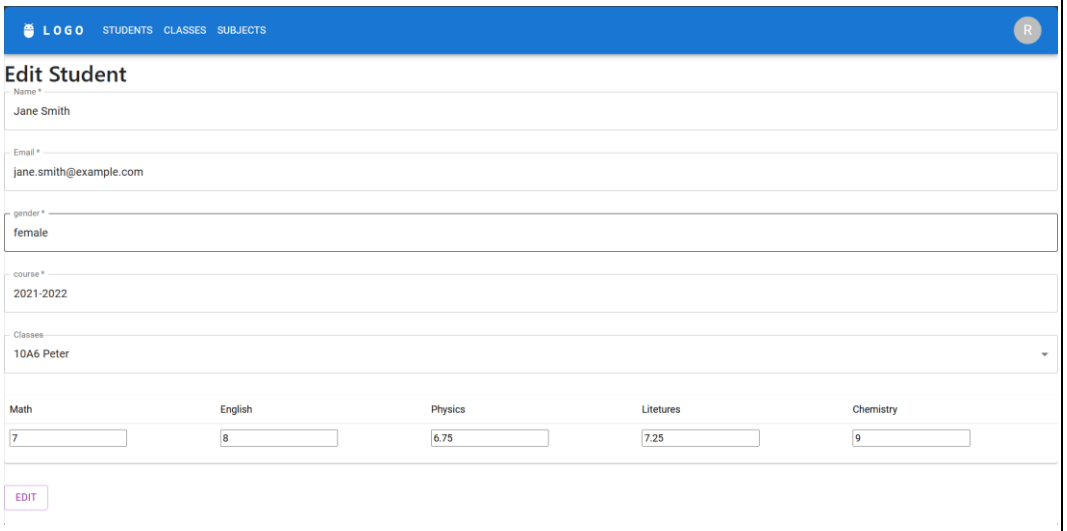
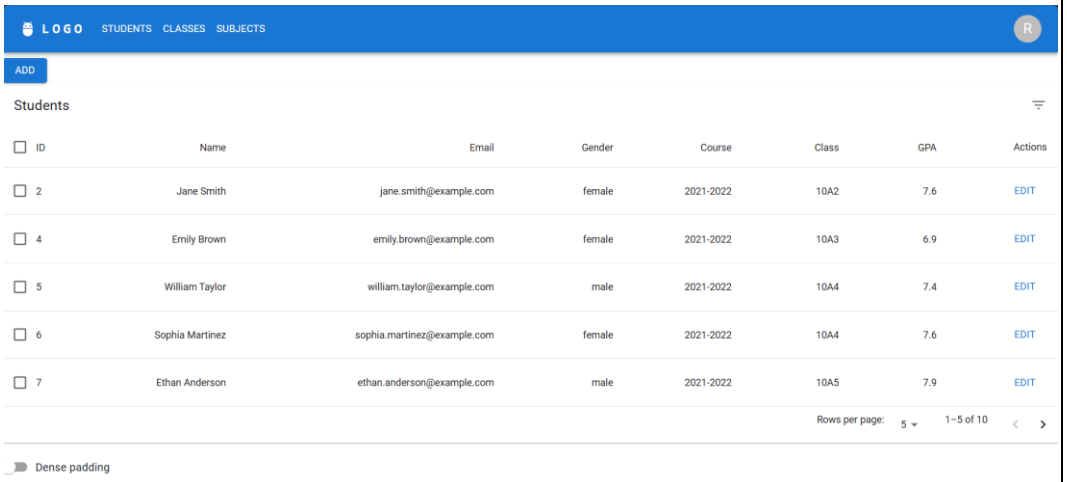
ID	Test scenario	Test step	Expected results	Evidence	Actual result	Status

1	Register	If you would like to access the website but do not currently have an account, you will need to register by filling out the necessary information, which includes your first name, last name, email address and password. After that, the user clicks the "Sign Up"	The user will be taken to the login page		The user has been taken to the home page	Pass

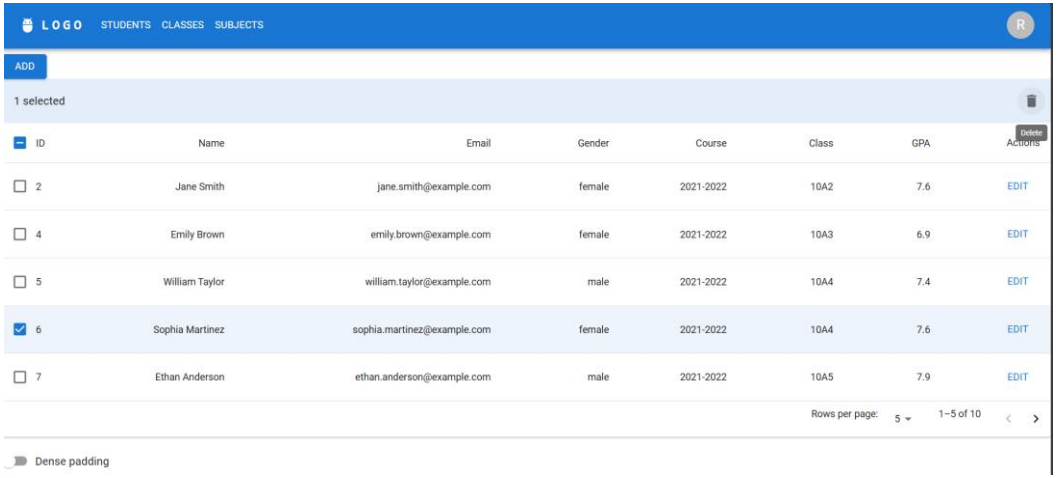
		button.				
2	Login	The user must log in in order to access the website.The user must input the right account and password here before clicking the login button.	The user will be takento the home page.		The homepage is displayed tothe user.	Pass

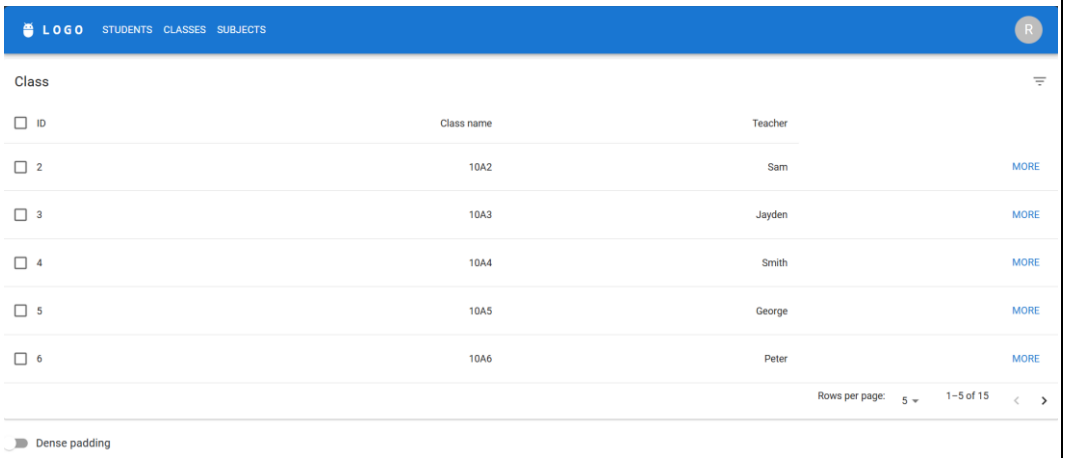
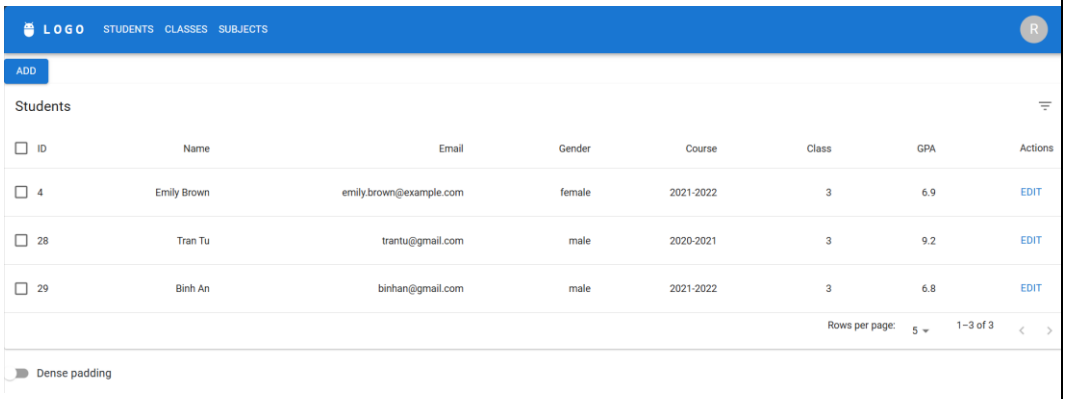
3	Logout	When utilizing the program. Pressing the logout buttonwill allow the user to exit the application.	The user will be taken back to the login page		The user has been taken back to the login page	Pass

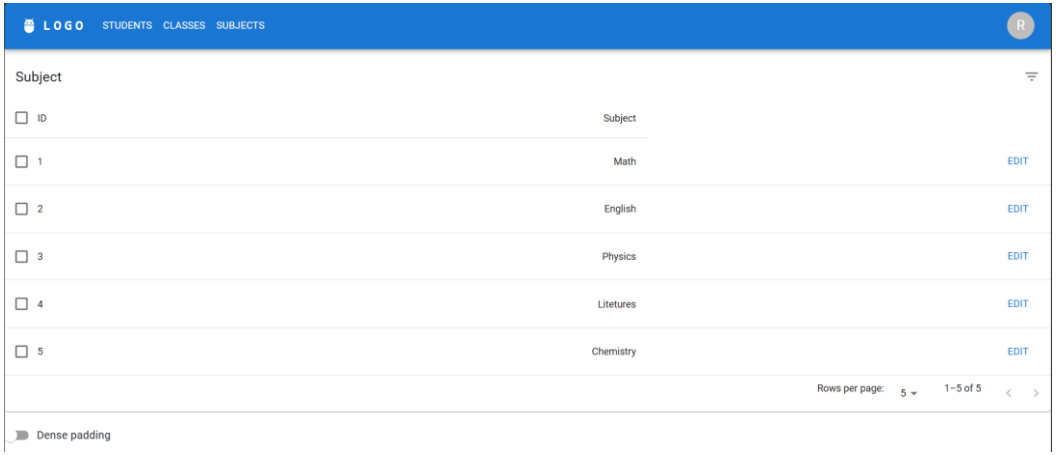
4	Student management	When user taps to access the student managemen t page, the student table appears on the toolbar	The user will be taken to the student managemen t page, which will show all the student information		After being taken to the student managemen t page, the user was allowed to view all student information	Pass
5	Student sorting function	User only needs to click on title they want	The website will arrange the title user choose in order from small to large or a-z		The website arranged the title user choose	Pass

6	Student detail	Click on edit button of the student user want to view the detail	The website will show the student form with more detail		The website has shown the student form with more detail that user want	Pass
7	Edit student	Click on edit button of the student that user want to edit. User just need to modify the information then click on edit button	The user will be taken to edit page. After modifying student information success, the user will be taken back to the student managemen		After modifying success, the user has been taken to the student managemen t page with the corrected information	Pass

			t page			
8	Add student	User clicks on add button to access to add student form, then user fills the information of the new student. When finish click on add button	The user will be taken to the add student form after clicking on the add button. After completely adding student, the user will be taken back to the student managemen		The user has been taken to the add form. After completely adding new student, the user has been taken to the student managemen t page with the updated information	Pass

			t with information just updated			
9	Delete student	User chooses the student they wish then the delete icon will appear. The user clicks on delete icon to delete the student	The website will delete the chosen student with no ask (so be careful when delete)		The website has deleted the chosen student	Pass

10	Management class	User taps to access the classes on toolbar, the list of classes appears	The user will be taken to the classes page, which will show list of classes when user hit classes on the website toolbar		After being taken to the classes page, the user was allowed to view and access to every class that was shown	Pass
11	Manage students by class	User clicks on more button to access class; the students of that class appear	The list of students of that class will show that user can view, add, edit and delete student		The website has shown the student of chosen class, the user was allowed to view, add, edit and delete	Pass

					student	
12	Management subject	User taps to access the subjects on toolbar, the list of subjects appears	The user will be taken to the subjects page, which will show list of subjects when user hit subjects on the website toolbar		After being taken to the subjects page, the user was allowed to view every subject that was shown	Pass

V. Conclusion

1. Student evaluation

The Student Management System (SMS) is a web application developed using ReactJS and NodeJS. It offers a streamlined platform for effectively managing many parts of student information, such as enrollment, grading, and attendance. The system's intuitive interface guarantees that both administrators and students can effortlessly traverse the platform. The system provides extensive functionality through components such as AddStudent.jsx, ClassTable.jsx, and Dashboard.jsx. This allows educators to effectively track and assess student performance, facilitating thorough student evaluations.

2. Difficulties

The development of the SMS posed numerous obstacles. To achieve seamless communication, it was necessary to carefully plan and execute the integration of the front-end ReactJS components with the back-end NodeJS server. Another major challenge was ensuring consistent and efficient management of state and data flow throughout the program. In addition, the establishment of robust authentication and authorization systems was necessary to safeguard sensitive student data, requiring a comprehensive comprehension of security protocols.

3. Advantages

The main benefit of utilizing ReactJS and NodeJS for the SMS comes in the clear division of responsibilities and the modular structure of the components. ReactJS enables the construction of reusable user interface components, streamlining the development process and improving maintainability. NodeJS, with its event-driven architecture and non-blocking I/O approach, guarantees efficient management of simultaneous requests, resulting in a highly scalable solution. The integration of these technologies yields a highly responsive and efficient application capable of meeting the requirements of a contemporary educational setting.

4. Disadvantages

Although the system offers numerous benefits, it also has certain limitations. Setting up and configuring the ReactJS and NodeJS environment can be intricate and time-consuming, particularly for developers who are unfamiliar with these technologies. In addition, achieving interoperability and seamless communication between the front-end and back-end can be difficult, especially when handling asynchronous operations. Keeping up with the newest developments and best practices of both frameworks is challenging due to their rapid evolution.

5. Development ideas

In order to optimize the system, various development concepts can be taken into account:

- Mobile App Integration: Creating a supplementary mobile application using React Native to offer students and administrators

convenient access to the SMS while they are on the move.

- Advanced Analytics: Incorporating sophisticated analytics and reporting functionalities to offer more profound insights into student performance and trends.

- Improved Security: Consistently enhancing security protocols, such as implementing two-factor authentication (2FA) and end-to-end encryption, to safeguard confidential student information.

Insurance is a rather complex subject, and I have done my best to build a program that is capable of student management system. In the process of implementing the topic, I have learned a lot of useful knowledge, found good references, and also know how to overcome gaps and shortcomings. I have absorbed and had very interesting experiences in interface design. I will try to develop deeper and wider if there is a chance in the future.

References

9+ Free React Templates - Material UI. (n.d.-b).

https://mui.com/material-ui/getting-started/templates/

React Button component - Material UI. (n.d.).

https://mui.com/material-ui/react-button/