

**DEPLOYED WEB APPLICATION:** <https://nichole.alburo.ph/>

**GITHUB REPOSITORY:** <https://github.com/nicvhine/teacher-web>

**University of San Carlos**  
**Department of Computer and Information Sciences and Mathematics**  
**Talamban Campus, Cebu City, Philippines**

**IT 3202 - Software Quality Assurance**

**Final Project**  
**CRUD Web Application Utilizing JSON Web Token**

**Submitted by:**  
**Nichole Vine Alburo**

**Submitted to:**  
**Mr. Keenan Paul Mendiola**

**May 2024**

## TABLE OF CONTENTS

<b>UNIT TESTING: USER.....</b>	<b>6</b>
Test Title: Adding User Functionality.....	6
1. Setup:.....	6
2. Test Case 1: Adding a new user with unique email.....	6
3. Test Case 2: Adding a user with a duplicate email.....	6
Expected Outcomes:.....	6
Notes:.....	6
Test Title: Fetching Users' Details Functionality.....	7
Test Steps:.....	7
1. Setup:.....	7
2. Test Case 1: Fetching users successfully.....	7
3. Test Case 2: Handling errors when fetching users.....	7
Expected Outcomes:.....	7
Notes:.....	7
<b>UNIT TESTING: STUDENTS.....</b>	<b>8</b>
Test Title: Adding Student Functionality.....	8
1. Setup:.....	8
2. Test Case 1: Adding a new student successfully.....	8
3. Test Case 2: Handling missing name or email.....	8
4. Test Case 3: Handling duplicate email error.....	8
5. Test Case 4: Handling database error.....	8
Expected Outcomes:.....	8
Notes:.....	9
Test Title: Fetching Students' Details Functionality.....	9
Test Steps:.....	9
1. Setup:.....	9
2. Test Case 1: Adding a new student successfully.....	9
3. Test Case 2: Handling missing name or email.....	9
4. Test Case 3: Handling duplicate email error.....	9
5. Test Case 4: Handling database error.....	9
Expected Outcomes:.....	10
Notes:.....	10
Test Title: Updating Student Details Functionality.....	10
Test Steps:.....	10
1. Setup:.....	10
2. Test Case 1: Updating student status successfully.....	10
3. Test Case 2: Handling database error while updating student status.....	10
4. Test Case 3: Handling missing status.....	10
Expected Outcomes:.....	11

Notes:.....	11
<b>UNIT TESTING: TASKS.....</b>	<b>11</b>
Test Title: Adding Task Functionality.....	11
Test Steps:.....	11
1. Setup:.....	11
2. Test Case 1: Adding a task successfully.....	11
3. Test Case 2: Handling database error while adding a task.....	11
4. Test Case 3: Handling missing required fields.....	12
Expected Outcomes:.....	12
Notes:.....	12
Test Title: Fetching Tasks' Details Functionality.....	12
Test Steps:.....	12
1. Setup:.....	12
2. Test Case 1: Fetching tasks for the given class successfully.....	12
3. Test Case 2: Handling database error while fetching tasks.....	13
Expected Outcomes:.....	13
Notes:.....	13
Test Title: Updating Task Details Functionality.....	13
Test Steps:.....	13
1. Setup:.....	13
2. Test Case 1: Updating task status successfully.....	13
3. Test Case 2: Handling database error while updating task status.....	13
4. Test Case 3: Handling missing status.....	14
Expected Outcomes:.....	14
Notes:.....	14
<b>AUTOMATION TESTING: REGISTER PAGE.....</b>	<b>15</b>
Test 1: 'displays form inputs and registers a user'.....	15
Test 2: 'displays error message when form is submitted with empty fields'.....	15
Test 3: 'displays error message when registration fails'.....	15
Test 4: 'navigates to login page when "Login Here" link is clicked'.....	15
<b>AUTOMATION TESTING: LOGIN.....</b>	<b>16</b>
Test 1: 'displays form inputs and logs in a user'.....	16
Test 2: 'displays error message when form is submitted with empty fields'.....	16
Test 3: 'displays error message when login fails'.....	16
Test 4: 'navigates to registration page when "Register Here" link is clicked'.....	16
<b>AUTOMATION TESTING: CLASS PAGE.....</b>	<b>17</b>
Test 1: 'displays form inputs and adds a class'.....	17
Test 2: 'displays error message when form is submitted with empty fields'.....	17
Test 3: 'navigates to class details page when class item is clicked'.....	17
<b>AUTOMATION TESTING: DASHBOARD PAGE.....</b>	<b>18</b>
Test 1: 'displays class navigation links'.....	18

Test 2: 'fetches class information when classId is provided'.....	18
Test 3: 'navigates to class list when "Class List" link is clicked'.....	18
<b>AUTOMATION TESTING: STUDENT PAGE.....</b>	<b>18</b>
Test 1: 'displays form inputs and adds a new student'.....	18
Test 2: 'displays error message when form is submitted with empty fields'.....	19
Test 3: 'filters students based on search query'.....	19
Test 4: 'toggles student status when clicked'.....	19
<b>AUTOMATION TESTING: TASK PAGE.....</b>	<b>19</b>
Test 1: 'displays form inputs and adds a new task'.....	19
Test 2: 'displays error message when form is submitted with empty fields'.....	20
Test 3: 'filters tasks based on search query'.....	20
Test 4: 'toggles task status when clicked'.....	20
<b>AUTOMATION TESTING: CLASS MANAGEMENT PAGE.....</b>	<b>20</b>
Test 1: 'displays class details and updates them'.....	20
Test 2: 'displays error message when form is submitted with empty fields'.....	21
Test 3: 'deletes class when "Delete Class" button is clicked'.....	21
<b>SCREEN CAPTURES FROM THE DEPLOYED APP.....</b>	<b>22</b>
<b>REGISTER PAGE.....</b>	<b>22</b>
Placed email is already registered.....	22
Successfully registered.....	23
<b>LOGIN PAGE.....</b>	<b>23</b>
User is not found/registered.....	24
Placed password is incorrect.....	24
<b>CLASS PAGE.....</b>	<b>25</b>
Field is empty.....	25
Class added successfully.....	26
Duplicate details.....	26
<b>DASHBOARD PAGE.....</b>	<b>27</b>
<b>STUDENT PAGE.....</b>	<b>27</b>
Empty fields.....	28
Student added successfully.....	28
Switching of student's status.....	29
<b>TASK PAGE.....</b>	<b>29</b>
Empty Fields.....	30
Task added successfully.....	30
Switching of task's status.....	31
<b>CLASS DETAILS PAGE.....</b>	<b>31</b>
Class details updated.....	32

## UNIT TESTING: USER

**Test Title:** *Adding User Functionality*

**Test Steps:**

**1. Setup:**

- Import required modules and mock dependencies.
- Initialize the database connection pool.

**2. Test Case 1: Adding a new user with unique email**

- Test Step 1: Mock the database query to return an empty result set (indicating no duplicate emails).
- Test Step 2: Define a Jest mock callback function.
- Test Step 3: Call addUser function with a unique email and password, passing the callback.
- Test Step 4: Verify that the database was queried to check for existing emails.
- Test Step 5: Verify that the database was queried to insert the new user.
- Test Step 6: Verify that the callback was called without any error, and with an object argument.

**3. Test Case 2: Adding a user with a duplicate email**

- Test Step 1: Mock the database query to return a result set containing a user with the same email.
- Test Step 2: Define a Jest mock callback function.
- Test Step 3: Call addUser function with an email that already exists and a password, passing the callback.
- Test Step 4: Verify that the database was queried to check for existing emails.
- Test Step 5: Verify that the callback was called with an error.
- Test Step 6: Verify that the error object has a status code of 409 (Conflict).

**Expected Outcomes:**

- Test Case 1 should pass, indicating successful addition of a new user.
- Test Case 2 should pass, indicating proper handling of duplicate email errors.

**Notes:**

- Ensure that the mock implementation of the database query behaves as expected in each test case.
- Verify that the callback functions are invoked correctly with the expected arguments and error handling

## **Test Title: *Fetching Users' Details Functionality***

### **Test Steps:**

#### **1. Setup:**

- Import required modules and mock dependencies.
- Initialize the database connection pool.

#### **2. Test Case 1: Fetching users successfully**

- Test Step 1: Mock the database query to return a list of mock users.
- Test Step 2: Define a Jest mock callback function.
- Test Step 3: Call getUsers function, passing the callback.
- Test Step 4: Verify that the database was queried to fetch users.
- Test Step 5: Verify that the callback was called without any error, and with the expected list of users.

#### **3. Test Case 2: Handling errors when fetching users**

- Test Step 1: Mock the database query to simulate a database connection error.
- Test Step 2: Define a Jest mock callback function.
- Test Step 3: Call getUsers function, passing the callback.
- Test Step 4: Verify that the database was queried to fetch users.
- Test Step 5: Verify that the callback was called with an error object indicating the database connection error.

### **Expected Outcomes:**

- Test Case 1 should pass, indicating successful retrieval of users.
- Test Case 2 should pass, indicating proper error handling when there's an issue with the database connection.

### **Notes:**

- Ensure that the mock implementation of the database query behaves as expected in each test case.
- Verify that the callback functions are invoked correctly with the expected arguments and error handling.

## UNIT TESTING: STUDENTS

**Test Title:** *Adding Student Functionality*

**Test Steps:**

**1. Setup:**

- Import required modules and mock dependencies.
- Initialize Express app and mount the router.
- Mock the database query and repository functions.

**2. Test Case 1: Adding a new student successfully**

- Test Step 1: Mock the repository function to simulate successful addition of a student.
- Test Step 2: Send a POST request to the endpoint with valid student data.
- Test Step 3: Verify that the response status is 201 (Created).
- Test Step 4: Verify that the response body contains a success message.

**3. Test Case 2: Handling missing name or email**

- Test Step 1: Send a POST request to the endpoint with missing name or email.
- Test Step 2: Verify that the response status is 400 (Bad Request).
- Test Step 3: Verify that the response body contains an error message indicating missing name and email.

**4. Test Case 3: Handling duplicate email error**

- Test Step 1: Mock the repository function to simulate a duplicate email error.
- Test Step 2: Send a POST request to the endpoint with a duplicate email.
- Test Step 3: Verify that the response status is 409 (Conflict).
- Test Step 4: Verify that the response body contains an error message indicating the duplicate email error.

**5. Test Case 4: Handling database error**

- Test Step 1: Mock the repository function to simulate a database error.
- Test Step 2: Send a POST request to the endpoint.
- Test Step 3: Verify that the response status is 500 (Internal Server Error).
- Test Step 4: Verify that the response body contains an error message indicating the database error.

**Expected Outcomes:**

- Test Case 1 should pass, indicating successful addition of a new student.
- Test Case 2 should pass, indicating proper handling of missing name or email.
- Test Case 3 should pass, indicating proper handling of duplicate email error.
- Test Case 4 should pass, indicating proper handling of database errors.

**Notes:**

- Ensure that the mock implementations of repository functions behave as expected in each test case.
- Verify that the responses are correctly structured with the appropriate status codes and error messages.

**Test Title: *Fetching Students' Details Functionality***

**Test Steps:**

**1. Setup:**

- Import required modules and mock dependencies.
- Initialize Express app and mount the router.
- Mock the database query and repository functions.

**2. Test Case 1: Adding a new student successfully**

- Test Step 1: Mock the repository function to simulate successful addition of a student.
- Test Step 2: Send a POST request to the endpoint with valid student data.
- Test Step 3: Verify that the response status is 201 (Created).
- Test Step 4: Verify that the response body contains a success message.

**3. Test Case 2: Handling missing name or email**

- Test Step 1: Send a POST request to the endpoint with a missing name or email.
- Test Step 2: Verify that the response status is 400 (Bad Request).
- Test Step 3: Verify that the response body contains an error message indicating missing name and email.

**4. Test Case 3: Handling duplicate email error**

- Test Step 1: Mock the repository function to simulate a duplicate email error.
- Test Step 2: Send a POST request to the endpoint with a duplicate email.
- Test Step 3: Verify that the response status is 409 (Conflict).
- Test Step 4: Verify that the response body contains an error message indicating the duplicate email error.

**5. Test Case 4: Handling database error**

- Test Step 1: Mock the repository function to simulate a database error.
- Test Step 2: Send a POST request to the endpoint.
- Test Step 3: Verify that the response status is 500 (Internal Server Error).
- Test Step 4: Verify that the response body contains an error message indicating the database error.

## **Expected Outcomes:**

- Test Case 1 should pass, indicating successful addition of a new student.
- Test Case 2 should pass, indicating proper handling of missing name or email.
- Test Case 3 should pass, indicating proper handling of duplicate email error.
- Test Case 4 should pass, indicating proper handling of database errors.

## **Notes:**

- Ensure that the mock implementations of repository functions behave as expected in each test case.
- Verify that the responses are correctly structured with the appropriate status codes and error messages.

## **Test Title: *Updating Student Details Functionality***

### **Test Steps:**

- 1. Setup:**
  - Import required modules and mock dependencies.
  - Initialize Express app and mount the router.
  - Mock the database query and repository functions.
- 2. Test Case 1: Updating student status successfully**
  - Test Step 1: Clear all mocks to ensure a clean state.
  - Test Step 2: Mock the repository function to simulate successful update of student status.
  - Test Step 3: Send a PUT request to the endpoint with valid student status data.
  - Test Step 4: Verify that the response status is 200 (OK).
  - Test Step 5: Verify that the response body contains a success message.
- 3. Test Case 2: Handling database error while updating student status**
  - Test Step 1: Clear all mocks to ensure a clean state.
  - Test Step 2: Mock the repository function to simulate a database error while updating student status.
  - Test Step 3: Send a PUT request to the endpoint.
  - Test Step 4: Verify that the response status is 500 (Internal Server Error).
  - Test Step 5: Verify that the response body contains an error message indicating the failure to update student status.
- 4. Test Case 3: Handling missing status**
  - Test Step 1: Send a PUT request to the endpoint with missing status.
  - Test Step 2: Verify that the response status is 400 (Bad Request).

- Test Step 3: Verify that the response body contains an error message indicating missing status.

#### **Expected Outcomes:**

- Test Case 1 should pass, indicating successful update of student status.
- Test Case 2 should pass, indicating proper handling of database errors while updating student status.
- Test Case 3 should pass, indicating proper handling of missing status.

#### **Notes:**

- Ensure that the mock implementations of repository functions behave as expected in each test case.
- Verify that the responses are correctly structured with the appropriate status codes and error messages.

## **UNIT TESTING: TASKS**

### **Test Title: *Adding Task Functionality***

#### **Test Steps:**

##### **1. Setup:**

- Import required modules and mock dependencies.
- Initialize Express app and mount the router.
- Mock the database query and repository functions.

##### **2. Test Case 1: Adding a task successfully**

- Test Step 1: Clear all mocks to ensure a clean state.
- Test Step 2: Mock the repository function to simulate successful addition of a task.
- Test Step 3: Send a POST request to the endpoint with valid task data.
- Test Step 4: Verify that the response status is 201 (Created).
- Test Step 5: Verify that the response body contains a success message.

##### **3. Test Case 2: Handling database error while adding a task**

- Test Step 1: Clear all mocks to ensure a clean state.
- Test Step 2: Mock the repository function to simulate a database error while adding a task.
- Test Step 3: Send a POST request to the endpoint.
- Test Step 4: Verify that the response status is 500 (Internal Server Error).

- Test Step 5: Verify that the response body contains an error message indicating the failure to add a task.
- 4. Test Case 3: Handling missing required fields**
- Test Step 1: Send a POST request to the endpoint with missing required fields.
  - Test Step 2: Verify that the response status is 400 (Bad Request).
  - Test Step 3: Verify that the response body contains an error message indicating missing required fields.

#### **Expected Outcomes:**

- Test Case 1 should pass, indicating successful addition of a task.
- Test Case 2 should pass, indicating proper handling of database errors while adding a task.
- Test Case 3 should pass, indicating proper handling of missing required fields.

#### **Notes:**

- Ensure that the mock implementations of repository functions behave as expected in each test case.
- Verify that the responses are correctly structured with the appropriate status codes and error messages.

### **Test Title: *Fetching Tasks' Details Functionality***

#### **Test Steps:**

- 1. Setup:**
  - Import required modules and mock dependencies.
  - Initialize Express app and mount the router.
  - Mock the database query and repository functions.
- 2. Test Case 1: Fetching tasks for the given class successfully**
  - Test Step 1: Mock the repository function to simulate successful fetching of tasks for the given class.
  - Test Step 2: Send a GET request to the endpoint with a valid class ID.
  - Test Step 3: Verify that the response status is 200 (OK).
  - Test Step 4: Verify that the response body contains the expected tasks.

### **3. Test Case 2: Handling database error while fetching tasks**

- Test Step 1: Mock the repository function to simulate a database error while fetching tasks.
- Test Step 2: Send a GET request to the endpoint.
- Test Step 3: Verify that the response status is 500 (Internal Server Error).
- Test Step 4: Verify that the response body contains an error message indicating the failure to fetch tasks.

#### **Expected Outcomes:**

- Test Case 1 should pass, indicating successful fetching of tasks for the given class.
- Test Case 2 should pass, indicating proper handling of database errors while fetching tasks.

#### **Notes:**

- Ensure that the mock implementations of repository functions behave as expected in each test case.
- Verify that the responses are correctly structured with the appropriate status codes and error messages.

## **Test Title: *Updating Task Details Functionality***

#### **Test Steps:**

##### **1. Setup:**

- Import required modules and mock dependencies.
- Initialize Express app and mount the router.
- Mock the database query and repository functions.

##### **2. Test Case 1: Updating task status successfully**

- Test Step 1: Mock the repository function to simulate successful update of task status.
- Test Step 2: Send a PUT request to the endpoint with valid task status data.
- Test Step 3: Verify that the response status is 200 (OK).
- Test Step 4: Verify that the response body contains a success message.

##### **3. Test Case 2: Handling database error while updating task status**

- Test Step 1: Mock the repository function to simulate a database error while updating task status.
- Test Step 2: Send a PUT request to the endpoint.
- Test Step 3: Verify that the response status is 500 (Internal Server Error).

- Test Step 4: Verify that the response body contains an error message indicating the failure to update task status.

#### **4. Test Case 3: Handling missing status**

- Test Step 1: Send a PUT request to the endpoint with missing status.
- Test Step 2: Verify that the response status is 400 (Bad Request).
- Test Step 3: Verify that the response body contains an error message indicating missing status.

#### **Expected Outcomes:**

- Test Case 1 should pass, indicating successful update of task status.
- Test Case 2 should pass, indicating proper handling of database errors while updating task status.
- Test Case 3 should pass, indicating proper handling of missing status.

#### **Notes:**

- Ensure that the mock implementations of repository functions behave as expected in each test case.
- Verify that the responses are correctly structured with the appropriate status codes and error messages.

## AUTOMATION TESTING: REGISTER PAGE

### **Test 1: '*displays form inputs and registers a user*'**

- This test checks if the form inputs are displayed properly and if a user is successfully registered.
- Test Steps:
  - Type 'test@example.com' into the input field with type "email".
  - Type 'password123' into the input field with type "password".
  - Click on the submit button.
  - Check if a success message with the text 'User registered successfully!' is displayed.

### **Test 2: '*displays error message when form is submitted with empty fields*'**

- This test verifies if an error message is displayed when the form is submitted with empty fields.
- Test Steps:
  - Click on the submit button without filling any fields.
  - Check if an error message with the text 'Email and password cannot be empty.' is displayed.

### **Test 3: '*displays error message when registration fails*'**

- This test simulates a scenario where registration fails (e.g., due to an email already being taken).
- Test Steps:
  - Intercept the registration request and mock a response with a 409 status code (conflict) and an error message indicating that the email is already taken.
  - Type 'existinguser@example.com' into the input field with type "email".
  - Type 'password123' into the input field with type "password".
  - Click on the submit button.
  - Wait for the registration request to complete.
  - Check if an error message with the text 'This email is already registered. Please use a different email.' is displayed.

### **Test 4: '*navigates to login page when "Login Here" link is clicked*'**

- This test checks if clicking on the "Login Here" link navigates to the login page.
- Test Steps:
  - Click on the link containing the text 'Already registered?'.
  - Check if the URL includes '/' indicating navigation to the login page.

## AUTOMATION TESTING: LOGIN

### **Test 1: '*displays form inputs and logs in a user*'**

- This test verifies that the form inputs are displayed correctly and a user can successfully log in.
- Test Steps:
  - Visit the login page.
  - Type 'test@example.com' into the email input field.
  - Type 'password123' into the password input field.
  - Click on the submit button.
  - Check if the URL includes '/class', indicating successful login and navigation to the class page.

### **Test 2: '*displays error message when form is submitted with empty fields*'**

- This test ensures that an error message is displayed when the form is submitted with empty fields.
- Test Steps:
  - Visit the login page.
  - Click on the submit button without filling any fields.
  - Check if an error message with the text 'Email and password cannot be empty.' is displayed.

### **Test 3: '*displays error message when login fails*'**

- This test simulates a scenario where login fails due to incorrect credentials.
- Test Steps:
  - Intercept the login request and mock a response with a 401 status code (unauthorized) and an error message indicating invalid credentials.
  - Visit the login page.
  - Type 'invalid@example.com' into the email input field.
  - Type 'incorrectpassword' into the password input field.
  - Click on the submit button.
  - Wait for the login request to complete.
  - Check if an error message with the text 'Invalid credentials' is displayed.

### **Test 4: '*navigates to registration page when "Register Here" link is clicked*'**

- This test confirms that clicking on the "Register Here" link navigates to the registration page.
- Test Steps:

- Visit the login page.
- Click on the link containing the text 'Haven't registered yet?'.
- Check if the URL includes '/register', indicating navigation to the registration page.

## AUTOMATION TESTING: CLASS PAGE

### **Test 1: 'displays form inputs and adds a class'**

- This test checks if the form inputs are displayed properly and if a class is successfully added.
- Test Steps:
  - Type 'Test Class' into the input field with placeholder "Class Name".
  - Type 'This is a test class' into the textarea with placeholder "Description".
  - Type 'Test Group' into the textarea with placeholder "Group Number".
  - Type '2024' into the input field with placeholder "Start Year".
  - Type '2025' into the input field with placeholder "End Year".
  - Click on the submit button.
  - Check if a success message container with the text 'Class added successfully' is displayed.

### **Test 2: 'displays error message when form is submitted with empty fields'**

- This test verifies if an error message is displayed when the form is submitted with empty fields.
- Test Steps:
  - Click on the submit button without filling any fields.
  - Check if an error message container with the text 'All fields are required' is displayed.

### **Test 3: 'navigates to class details page when class item is clicked'**

- This test ensures that clicking on a class item navigates to the class details page.
- Test Steps:
  - Click on the first class item.
  - Check if the URL contains '/dashboard/' indicating navigation to the class details page.

## AUTOMATION TESTING: DASHBOARD PAGE

### **Test 1: '*displays class navigation links*'**

- This test verifies that the class navigation links are displayed correctly.
- Test Steps:
  - Visit the dashboard page for class ID 1.
  - Check if the 'Dashboard', 'Student List', 'Task Management', 'Class Management', 'Class List', and 'Logout' links exist.

### **Test 2: '*fetches class information when classId is provided*'**

- This test ensures that class information is fetched and displayed when the class ID is provided.
- Test Steps:
  - Intercept the GET request to fetch class information and mock a response with class information from the server.
  - Visit the dashboard page for class ID 1.
  - Wait for the request to complete.
  - Verify if class information such as 'Class Name' and 'Teacher' are displayed in the dashboard container.

### **Test 3: '*navigates to class list when "Class List" link is clicked*'**

- This test checks if clicking on the "Class List" link navigates to the class list page.
- Test Steps:
  - Visit the dashboard page for class ID 1.
  - Click on the 'Class List' link.
  - Verify if the URL includes '/class', indicating navigation to the class list page.

## AUTOMATION TESTING: STUDENT PAGE

### **Test 1: '*displays form inputs and adds a new student*'**

- This test verifies that the form inputs are displayed correctly and a new student can be added.
- Test Steps:
  - Visit the student list page.
  - Type 'John Doe' into the name input field.
  - Type 'john@example.com' into the email input field.
  - Submit the form.

- Check if a success message with the text 'Student added successfully!' is displayed.

**Test 2: '*displays error message when form is submitted with empty fields*'**

- This test ensures that an error message is displayed when the form is submitted with empty fields.
- Test Steps:
  - Visit the student list page.
  - Submit the form without filling out any inputs.
  - Check if an error message with the text 'Name and email cannot be empty.' is displayed.

**Test 3: '*filters students based on search query*'**

- This test checks if students are filtered based on the search query.
- Test Steps:
  - Visit the student list page.
  - Type 'john' into the search input.
  - Verify if only matching students are displayed in the table body.

**Test 4: '*toggles student status when clicked*'**

- This test verifies that the student status toggles when clicked.
- Test Steps:
  - Visit the student list page.
  - Assuming there's at least one student rendered with active status, find the status span of the first student.
  - Click on the status span.
  - Verify if the status toggles successfully by checking the color of the status span.

## **AUTOMATION TESTING: TASK PAGE**

**Test 1: '*displays form inputs and adds a new task*'**

- This test verifies that the form inputs are displayed correctly and a new task can be added.
- Test Steps:
  - Visit the task management page.
  - Type 'New Task' into the title input field.
  - Type 'This is a new task' into the description input field.

- Type '2024-06-01' into the due date input field.
- Click on the 'Add Task' button.
- Check if a success message with the text 'Task added successfully' is displayed.

**Test 2: '*displays error message when form is submitted with empty fields*'**

- This test ensures that an error message is displayed when the form is submitted with empty fields.
- Test Steps:
  - Visit the task management page.
  - Click on the 'Add Task' button without filling out any inputs.
  - Check if an error message with the text 'Fields cannot be empty.' is displayed.

**Test 3: '*filters tasks based on search query*'**

- This test checks if tasks are filtered based on the search query.
- Test Steps:
  - Visit the task management page.
  - Type 'new' into the search input.
  - Verify if only matching tasks are displayed in the table body.

**Test 4: '*toggles task status when clicked*'**

- This test verifies that the task status toggles when clicked.
- Test Steps:
  - Visit the task management page.
  - Assuming there's at least one task rendered with pending status, find the status span of the first task.
  - Click on the status span.
  - Verify if the status toggles successfully by checking the color of the status span.

## **AUTOMATION TESTING: CLASS MANAGEMENT PAGE**

**Test 1: '*displays class details and updates them*'**

- This test verifies that the class details are displayed correctly and can be updated.
- Test Steps:
  - Visit the settings page.
  - Clear and type 'New Class Name' into the name input field.
  - Clear and type 'New Class Description' into the description input field.
  - Clear and type '2024' into the start year input field.

- Clear and type '2025' into the end year input field.
- Click on the 'Update Class' button.
- Check if a success message with the text 'Class updated successfully' is displayed.

**Test 2: '*displays error message when form is submitted with empty fields*'**

- This test ensures that an error message is displayed when the form is submitted with empty fields.
- Test Steps:
  - Visit the settings page.
  - Click on the 'Update Class' button without filling out any inputs.
  - Check if an error message with the text 'Fields cannot be empty.' is displayed.

**Test 3: '*deletes class when "Delete Class" button is clicked*'**

- This test checks if the class can be deleted when the "Delete Class" button is clicked.
- Test Steps:
  - Visit the settings page.
  - Click on the "Delete Class" button.
  - Assuming you have a confirmation dialog, handle it accordingly.
  - Verify if the class is deleted by checking if the URL changes to '/class'.

# SCREEN CAPTURES FROM THE DEPLOYED APP

## REGISTER PAGE

The screenshot shows the EduTech register page. The page has a header "EduTech" and a sub-header "Technology for Education". It features two input fields for "Email address" and "Password", and a "Register" button. Below the inputs is a paragraph about EduTech's purpose. A link "Already registered? Login Here." is present. The developer tools Network tab shows a list of requests:

Name	Status	Type	Initiator	Size	Time
register	200	document	Other	783 B	209 ms
main.885d2c01.js	200	script	register_0	(memory cache)	0 ms
plugin.js	200	script	loadi18n	3.0 MB	45 ms
watemark-k1.png	200	image	oblateI18n	4.1 kB	4 ms
watemark-k2.png	200	image	oblateI18n	4.5 kB	5 ms
favicon.ico	200	x-icon	Other	(disk cache)	8 ms
manifest.json	200	manifest	Other	759 B	288 ms
dota-application/a...	200	script	Other	0 B	0 ms
logo192.png	200	png	Other	(disk cache)	1 ms

At the bottom of the developer tools, there is a message: "The Performance panel can now show you CSS selector statistics for long-running Recalculate" with a "new" badge.

*Placed email is already registered*

The screenshot shows the EduTech register page again. The "Email address" field contains "ace@gmail.com" and has a red validation message "A user with the same email already exists.". The "Register" button is highlighted in blue. The developer tools Network tab shows a list of requests, including one for "users" which has a status of 409:

Name	Status	Type	Initiator	Size	Time
register	200	document	Other	783 B	209 ms
main.885d2c01.js	200	script	register_0	(memory cache)	0 ms
plugin.js	200	script	loadi18n	3.0 MB	45 ms
watemark-k1.png	200	image	oblateI18n	4.1 kB	4 ms
watemark-k2.png	200	image	oblateI18n	4.5 kB	5 ms
favicon.ico	200	x-icon	Other	(disk cache)	8 ms
manifest.json	200	manifest	Other	759 B	288 ms
dota-application/a...	200	script	Other	0 B	0 ms
logo192.png	200	png	Other	(disk cache)	1 ms
users	409	xhr	RegisterForm.onSubmit	554 B	110 ms

At the bottom of the developer tools, there is a message: "The Console now shows you chains of error causes in the stack trace." and "CSS selector statistics in Performance".

*Successfully registered*

The screenshot shows the EduTech registration page at [nichole.alburu.ph/register](http://nichole.alburu.ph/register). The page displays a success message: "User registered successfully!" followed by the email address "taken@gmail". Below the message is a password input field with the placeholder "....." and a "Register" button. To the right of the form, the Chrome DevTools Network tab is open, showing the following resource list:

Name	Status	Type	Initiator	Size	Time
register	200	document	Other	788 B	208 ms
main.88d2c0f.js	200	script	register@	(memory cache)	0 ms
plugin.js	200	script	load.js!	3.0 MB	48 ms
watermark-ai.png	200	fetch	plugin.js:2	4.5 kB	4 ms
logo192.png	200	ping	Other	4.5 kB	3 ms
favicon.ico	200	image	Other	(disk cache)	9 ms
manifest.json	200	manifest	Other	759 B	288 ms
data/application/a...	200	script	Other	0 B	0 ms
logo192.png	200	ping	Other	(disk cache)	1 ms
users	409	xhr	RegisterForm.e:30	554 B	110 ms
users	201	xhr	RegisterForm.e:30	540 B	263 ms

At the bottom of the Network tab, it says "11 requests 3.0 MB transferred 3.3 MB resources | Finish: 3.5 min | DOMContentLoaded: 249 ms | Load: 433 ms". The Console tab is also visible, showing "Highlights from the Chrome 125 update".

## LOGIN PAGE

The screenshot shows the EduTech login page at [nichole.alburu.ph](http://nichole.alburu.ph). The page features a "Technology for Education" banner with a "Login" button. Below the banner is a form with fields for "Email address" and "Password", and a "Forgot password?" link. To the right of the form, the Chrome DevTools Network tab is open, showing the following resource list:

Name	Status	Type	Initiator	Size	Time
nichole.alburu.ph	200	document	Other	788 B	90 ms
main.88d2c0f.js	200	script	(index):9	(memory cache)	0 ms
plugin.js	200	script	load.js!	3.0 MB	44 ms
watermark-ai.png	200	fetch	plugin.js:2	4.5 kB	4 ms
logo192.png	200	ping	Other	4.5 kB	3 ms
favicon.ico	200	image	Other	(disk cache)	3 ms
manifest.json	200	manifest	Other	759 B	283 ms
data/application/a...	200	script	Other	0 B	0 ms
logo192.png	200	ping	Other	(disk cache)	1 ms

At the bottom of the Network tab, it says "9 requests 3.0 MB transferred 3.3 MB resources | Finish: 598 ms | DOMContentLoaded: 128 ms | Load: 357 ms". The Console tab is also visible, showing "Highlights from the Chrome 125 update".

User is not found/registered

The screenshot shows a browser window with the URL [nichole.alburo.ph](http://nichole.alburo.ph). The page title is "React App". The main content area displays the EduTech logo and a "User not found" message with an email input field containing "email@gmail". Below it is a password input field with four dots. A blue "Login" button is centered. To the right, a "Register Here" link is visible. The left sidebar contains a brief description of EduTech's features. The developer tools Network tab is open, showing a list of resources. One entry for "LoginForm.js" at 404 status is highlighted, indicating the source of the error.

Name	Status	Type	Initiator	Size	Time
nichole.alburo.ph	200	document	Other	788 B	90 ms
main.885e2c03.js	200	script	(index):0	(memory cache)	0 ms
plugin.js	200	script	load.js:2	3.0 MB	44 ms
watermark-ai.png	200	fetch	blob:0x1:2	4.5 kB	4 ms
watermark-ai.png	200	png	Other	4.5 kB	5 ms
favicon.ico	200	x-icon	Other	(disk cache)	3 ms
manifest.json	200	manifest	Other	759 B	283 ms
data-application.js	200	script	Other	0 B	0 ms
logo192.png	200	png	Other	(disk cache)	1 ms
login	404	xhr	LoginForm.js:21	529 B	276 ms

Placed password is incorrect

The screenshot shows a browser window with the URL [nichole.alburo.ph](http://nichole.alburo.ph). The main content area displays the EduTech logo and an "Invalid password" message with an email input field containing "ace@gmail". Below it is a password input field with four dots. A blue "Login" button is centered. To the right, a "Register Here" link is visible. The left sidebar contains a brief description of EduTech's features. The developer tools Network tab is open, showing a list of resources. One entry for "LoginForm.js" at 401 status is highlighted, indicating the source of the error.

Name	Status	Type	Initiator	Size	Time
nichole.alburo.ph	200	document	Other	788 B	90 ms
main.885e2c03.js	200	script	(index):0	(memory cache)	0 ms
plugin.js	200	script	load.js:2	3.0 MB	44 ms
watermark-ai.png	200	fetch	blob:0x1:2	4.5 kB	4 ms
watermark-ai.png	200	png	Other	4.5 kB	5 ms
favicon.ico	200	x-icon	Other	(disk cache)	3 ms
manifest.json	200	manifest	Other	759 B	283 ms
data-application.js	200	script	Other	0 B	0 ms
logo192.png	200	png	Other	(disk cache)	1 ms
login	401	xhr	LoginForm.js:21	529 B	276 ms
login	401	xhr	LoginForm.js:21	525 B	298 ms

## CLASS PAGE

The screenshot shows a browser window with the URL [nichole.alburo.ph/class](http://nichole.alburo.ph/class). The main content area displays a form for adding a class, with fields for 'Class Name', 'Description', 'Group Number', 'Start Year', and 'End Year'. A 'Logout' button is visible above the form. Below the form is a large 'Add Class' button. The browser's developer tools are open, specifically the Network tab, which shows a list of requests made by the page. The requests include files like 'class', 'main-985d2c01.js', 'plugin.js', 'class.css', 'watermark-ai.png', 'favicon.ico', 'manifest.json', and 'logo192.png'. The Network tab also includes a timeline at the top and various filter options.

Field is empty

This screenshot is similar to the one above, but it includes a validation message 'All fields are required' displayed prominently above the form fields. The rest of the interface, including the form fields, developer tools, and network requests, is identical to the first screenshot.

*Class added successfully*

React App

nichole.alburo.ph/class

Logout

Class added successfully

Class Name

Description

Group Number

Start Year

End Year

Add Class

IT 1101

Programming

2

2022 - 2023

Network

Name	Status	Type	Initiator	Size	Time
class	200	document	Other	750 B	84 ms
main.985dc2c01.js	200	script	class:0	(memory cache)	0 ms
plugin.js	200	script	load:js:1	3.0 MB	30 ms
class	304	xhr	AddClassForm:is:19	428 B	213 ms
- watermark-ai.png	200	fetch	plugin:js:4	4.5 kB	4 ms
- watermark-ai.png	200	png	Other	4.5 kB	4 ms
favicon.ico	200	x-icon	Other	(disk cache)	2 ms
manifest.json	200	manifest	Other	726 B	93 ms
e data:application/la...	200	script	Other	0 B	0 ms
logo192.png	200	png	Other	(disk cache)	0 ms
class	201	xhr	AddClassForm:is:19	642 B	332 ms
class	200	xhr	AddClassForm:is:19	546 B	66 ms

12 requests | 3.0 MB transferred | 3.3 MB resources | Finish: 2.0 min | DOMContentLoaded: 101 ms | Load: 293 ms

Console What's new

Error causes in the Console

CSS selector statistics in Performance

The Performance panel can now show you CSS selector statistics for long-running Recalculate

*Duplicate details*

React App

nichole.alburo.ph/class

Logout

A class with the same name and group already exists.

IT 1101

Programming

2

2022 - 2023

Add Class

Network

Name	Status	Type	Initiator	Size	Time
class	200	document	Other	751 B	96 ms
main.985dc2c01.js	200	script	class:0	(memory cache)	0 ms
plugin.js	200	script	load:js:1	3.0 MB	30 ms
class	304	xhr	AddClassForm:is:19	428 B	213 ms
- watermark-ai.png	200	fetch	plugin:js:2	4.5 kB	4 ms
- watermark-ai.png	200	png	Other	4.5 kB	4 ms
favicon.ico	200	x-icon	Other	(disk cache)	2 ms
manifest.json	200	manifest	Other	726 B	93 ms
e data:application/la...	200	script	Other	0 B	0 ms
logo192.png	200	png	Other	(disk cache)	1 ms
class	409	xhr	AddClassForm:is:19	532 B	103 ms

11 requests | 3.0 MB transferred | 3.3 MB resources | Finish: 12.25 s | DOMContentLoaded: 123 ms | Load: 304 ms

Console What's new

Error causes in the Console

CSS selector statistics in Performance

The Performance panel can now show you CSS selector statistics for long-running Recalculate

## DASHBOARD PAGE

The screenshot shows the EduTech dashboard for IT 1101 Programming. The main content area displays two summary boxes: 'Number of Students' (0) and 'Number of Tasks' (0). The left sidebar contains navigation links for Dashboard, Student List, Task Management, Class Management, Class List, and Logout. The developer tools Network tab is open, showing a list of resources loaded by the application. The timeline at the bottom indicates a total load time of 495 ms.

Name	Status	Type	Initiator	Size	Time
7	200	document		789 B	281 ms
main.9865d2c01.js	200	script	72	(memory cache)	0 ms
plugin.js	200	script	load a cl	3.0 MB	54 ms
7	200	xhr	Dashboard	144 B	21 ms
7	304	xhr	DashboardHome.js:21	434 B	293 ms
studentCount	200	xhr	DashboardHome.js:22	481 B	211 ms
taskCount	200	xhr	DashboardHome.js:23	484 B	218 ms
- watermark-ai.png	200	fetch	glue.js:2	4.5 kB	9 ms
- watermark-ai.png	200	png	Other	4.5 kB	1 ms
manifest.json	200	manifest	Other	763 B	338 ms
data-application.js...	200	script	Other	0 B	1 ms
login192.png	200	ping	Other	(disk cache)	1 ms

## STUDENT PAGE

The screenshot shows the EduTech student list page. At the top, there is a form to 'Add New Student' with fields for Name and Email, and a 'Add Student' button. Below this is a 'Student List' section with a search bar and a dropdown menu set to 'ALL'. A table lists student information: ID, Name, Email, and Status. The developer tools Network tab is open, showing a list of resources loaded by the application. The timeline at the bottom indicates a total load time of 495 ms.

ID	Name	Email	Status
7			
studentCount	200	xhr	DashboardHome.js:22
taskCount	200	xhr	DashboardHome.js:23
- watermark-ai.png	200	fetch	glue.js:2
- watermark-ai.png	200	png	Other
manifest.json	200	manifest	Other
data-application.js...	200	script	Other
login192.png	200	ping	Other
7	304	xhr	Dashboard.js:27
students	200	xhr	StudentList.js:21

## Empty fields

The screenshot shows the EduTech application's dashboard. On the left, a sidebar lists: Dashboard, Student List, Task Management, Class Management, Class List, and Logout. The main area has two sections: "Add New Student" and "Student List". In "Add New Student", there is an error message: "Name and email cannot be empty." Below it are input fields for Name and Email, and a blue "Add Student" button. In "Student List", there is a search bar and a table with columns: ID, Name, Email, and Status. One row is shown with ID 2, Name Nichole, Email alburonichole2@gmail.com, and Status active. The browser's developer tools Network tab is open, showing a list of requests. The table data is as follows:

Name	Status	Type	Initiator	Size	Time
students	200	document	Other	781 B	97 ms
main.985d2cd1.js	200	script	studentList.js	(memory ...)	0 ms
plugin.js	200	script	load.js1	3.0 MB	23 ms
xhr	304	xhr	Dashboard.list.eZ2	411 B	223 ms
studentList.js	200	script	studentList.js2	438 B	206 ms
watermark-ai.png	200	fetch	plugin.js2	4.5 kB	6 ms
watermark-ai.png	200	png	Other	4.5 kB	4 ms
favicon.ico	200	x-icon	Other	(disk cache)	3 ms
manifest.json	200	manifest	Other	764 B	127 ms
# data-application.js...	200	script	Other	0 B	0 ms
logo92.png	200	png	Other	0 B	0 ms

*Student added successfully*

The screenshot shows the EduTech application's dashboard after a successful addition. The sidebar and "Add New Student" section are identical to the previous screenshot. In "Student List", the table now includes a new row with ID 3, Name Mark, Email markgelmadadaro@gmail.com, and Status active. The browser's developer tools Network tab is open, showing a list of requests. The table data is as follows:

Name	Status	Type	Initiator	Size	Time
students	200	document	studentList.js	781 B	97 ms
main.985d2cd1.js	200	script	studentList.js	(memory ...)	0 ms
plugin.js	200	script	load.js1	3.0 MB	23 ms
xhr	304	xhr	Dashboard.list.eZ2	411 B	223 ms
studentList.js	200	script	studentList.js2	438 B	206 ms
watermark-ai.png	200	fetch	plugin.js2	4.5 kB	6 ms
watermark-ai.png	200	png	Other	4.5 kB	4 ms
favicon.ico	200	x-icon	Other	(disk ...)	3 ms
manifest.json	200	manifest	Other	764 B	127 ms
# data-application.js...	200	script	Other	0 B	0 ms
logo92.png	200	png	Other	0 B	0 ms
students	201	xhr	studentList.js2	534 B	136 ms
students	200	xhr	studentList.js2	608 B	220 ms

## Switching of student's status

The screenshot shows the EduTech application's Student List page. On the left, a sidebar menu includes Dashboard, Student List, Task Management, Class Management, Class List, and Logout. The main area has a title "Add New Student" and a search bar. Below it is a table titled "Student List" with columns ID, Name, Email, and Status. Two students are listed: Nichole (inactive) and Mark (active). The developer tools Network tab is open, showing a list of requests including scripts, images, and fetches for the page.

## TASK PAGE

The screenshot shows the EduTech application's Task List page. The sidebar menu is identical to the previous screenshot. The main area has a title "Add New Task" with input fields for Title, Description, and Due Date. Below it is a table titled "Task List" with columns Due Date, Title, Description, and Status. One task is listed: "Tasks" (200 ms). The developer tools Network tab is open, showing a list of requests including scripts, images, and fetches for the page.

## Empty Fields

The screenshot shows a browser window for a React application titled "EduTech". On the left is a sidebar with links: Dashboard, Student List, Task Management, Class Management, Class List, and Logout. The main content area has a title "Add New Task" and a message "Fields cannot be empty." Below this are input fields for "Title" and "Description", and a date picker for "mm/dd/yyyy". A blue "Add Task" button is at the bottom. To the right is a "Task List" table with columns: Due Date, Title, Description, and Status. At the bottom of the table is a search bar and a dropdown menu set to "ALL". The developer tools Network tab is open, showing a list of requests. The table has 16 rows, mostly from "students" and "tasks" components, with various status codes like 200, 304, and 201. Request details include file names like "main.895d2c01.js", "plugin.js", and "Tasks.js". The developer tools also show a "Console" tab with error causes and a "Performance" tab.

Task added successfully

This screenshot is similar to the previous one but shows a success message "Task added successfully" above the "Add Task" button after the user has submitted the form. The rest of the interface and developer tools are identical to the first screenshot.

## Switching of task's status

The screenshot shows the EduTech application interface. On the left, a sidebar menu includes Dashboard, Student List, Task Management, Class Management, Class List, and Logout. The main area displays the 'Add New Task' page with a success message 'Task added successfully'. It has fields for Title, Description, and Due Date (mm/dd/yyyy). A blue 'Add Task' button is present. Below this is the 'Task List' section with a search bar and a table. The table has columns: Due Date, Title, Description, and Status. One task listed is '5/15/2024 Final This is so hard. Be ready to fail.' with the status 'completed' in red. The right side of the screen shows the Network tab of the developer tools, listing various requests with their status, type, initiator, size, and time.

## CLASS DETAILS PAGE

The screenshot shows the EduTech application interface. The sidebar menu is identical to the previous screenshot. The main area displays the 'Edit Class Details' page for a class named 'IT 1101'. It has fields for Class Name, Class Description, and Class Year (2022/2023). Buttons for 'Update Class' and 'Delete Class' are at the bottom. The right side of the screen shows the Network tab of the developer tools, listing various requests with their status, type, initiator, size, and time.

## Class details updated

The screenshot shows a web application interface for 'EduTech'. On the left, a sidebar menu includes 'Dashboard', 'Student List', 'Task Management', 'Class Management', 'Class List', and 'Logout'. The main content area is titled 'Edit Class Details' and displays a success message: 'Class updated successfully'. It contains fields for 'Class Name' (IT 1102), 'Class Description' (Programming), and 'Class Year' (2022-2023). Below these are two buttons: 'Update Class' (blue) and 'Delete Class' (red).

To the right of the main content is the Chrome DevTools Network tab. It lists network requests with the following table:

Name	Status	Type	Initiator	Size	Time
7	200	docu...		788 B	91 ms
main.985d2c01.js	200	script	7.0	(inmem...	0 ms
plugin.js	200	script	load.js!	3.0 MB	34 ms
7	304	xhr	Dashboard.js?2	477 B	257 ms
7	304	xhr	Dashboard.js?2	435 B	327 ms
studentCount	200	xhr	Dashboard.js?2	471 B	206 ms
taskCount	200	xhr	Dashboard.js?2	513 B	230 ms
-watermark-k-al.png	200	fetch	styleIt.js?2	4.5 kB	6 ms
-watermark-k-al.png	200	png		4.5 kB	2 ms
manifest.json	200	manifest	Other	754 B	284 ms
data:application/ja...	200	script	Other	0 B	0 ms
logo192.png	200	png	Other	(disk ...)	2 ms
7	304	xhr	Dashboard.js?2	432 B	75 ms
tasks	200	xhr	Settings.js?24	591 B	77 ms
7	304	xhr	Settings.js?39	428 B	158 ms
7	200	xhr	Settings.js?65	369 B	203 ms

At the bottom of the Network tab, there are status indicators: '16 requests | 3.0 MB transferred | 3.3 MB resources | Finish: 10 min | DOMContent'. Below this, a 'Console' tab is open, showing 'What's new' and 'Highlights from the Chrome 125 update'. A small video thumbnail for 'CSS selector statistics in Performance' is also visible.