ENSF-381: Assignment 5

Winter 2025

Department of Electrical and Software Engineering Schulich

School of Engineering

Due: April 11, 2025, at 11:59 PM

In this assignment, you are required to implement the backend functionality of your full stack web application using Flask. Most of the frontend components have already been developed in Assignment04.

Administrative Details

- The submission should be a ***single compressed file (.zip)*** that includes the complete source code and resources (e.g., images), ready for execution without modifications.
- Name the submitted file as: Assignment5_ENSF381_Section#_Group# Example: `Assignment5_ENSF381_L02_Group01.zip`
- Submit ONLY ONE copy.
- Include group members' names and UCIDs at the top of the `app.py` file as comments.

Specifications:

Implement the following features across the LMS pages:

Instructions:

- If you did not complete Assignment 04, a reference solution will be available on April 1st, 2025, in D2L to help you proceed with this assignment.

---

1. Signup Page Requirements - 20 Marks

Components:

A. `Header` *Component:*

a. Display LMS logo (left) and navigation links (Home, Courses, Login) using React Router.
b. Style links with equal spacing.

B. `RegForm` *Component:*

a. Include a form with fields for username, password, confirm password, and email.
- Add a “Signup” button.
- Style the "Signup" button with a contrasting color of #4CAF50 (green), padding of 10px, border radius of 5px, margin of 10px, and opacity of 0.5.
- Add a hover effect by changing the background color to #45A049 (a slightly darker shade of green) and opacity of 1.0.

- The page should be in the same theme and layout as the Login page that was developed as part of Assignment 02.

b. *Form Validation*:

i) Username:

- Must be between 3 and 20 characters long.
- Allowed characters: alphanumeric characters (letters A-Z, numbers 0-9), hyphens (-), and underscores (_).
- Must start with a letter.
- Cannot contain spaces or special characters other than hyphens and underscores.

ii) Password:

- Must be at least 8 characters long.
- Must contain at least one uppercase letter, one lowercase letter, one number, and one special character.
- Allowed special characters: !@#$%^&*()-_=+[]{}|;:'",.<>?/`~.
- Cannot contain spaces.

iii) Confirm Password:

- Must match the password entered in the password field.

iv) Email:

- Must be a valid email address format (e.g., username@example.com).
- Cannot contain spaces.
- Must contain an "@" symbol followed by a domain name (.com, .net, .io).

c. When the user clicks on the "Sign up" button, perform the validation and:

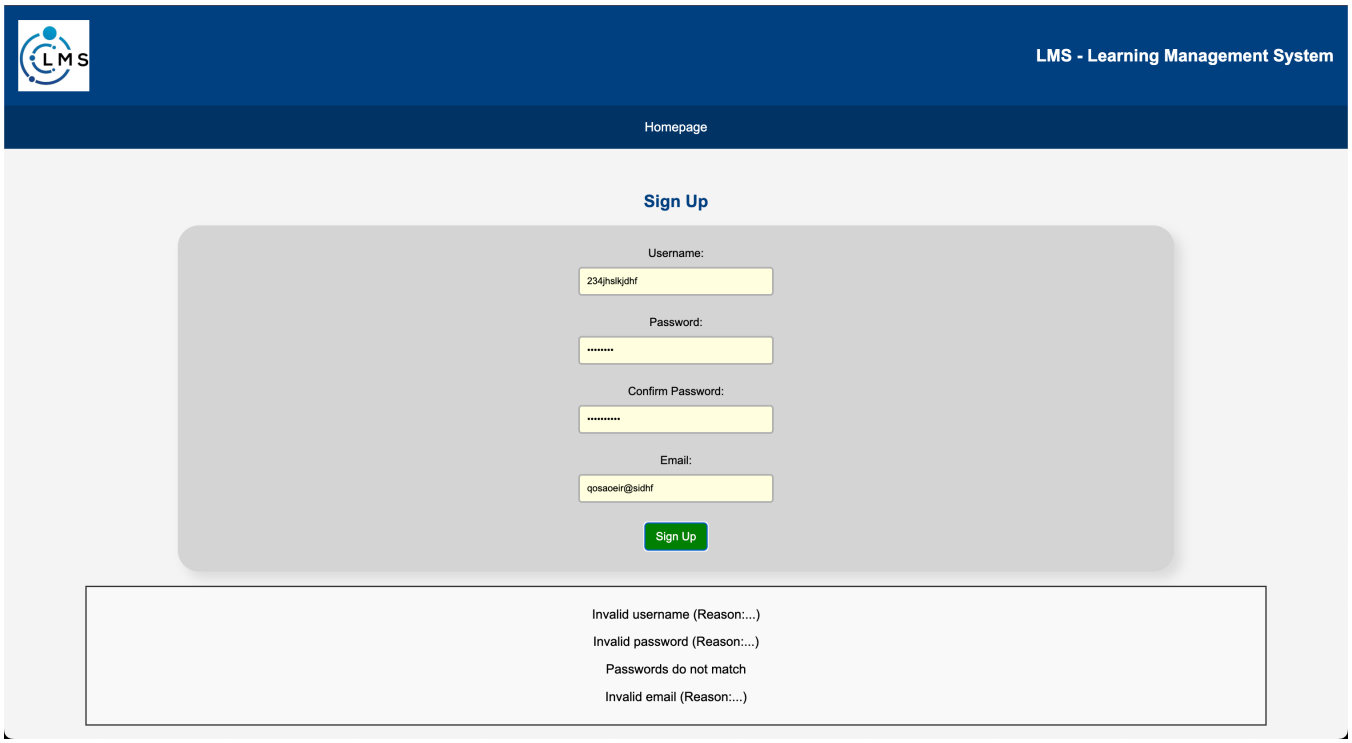
- Transfer the information to the `register` API in your backend (discussed later).
- *Display error messages* in a dynamic box for invalid inputs.
- *On successful signup*, redirect the user to the Login Page.

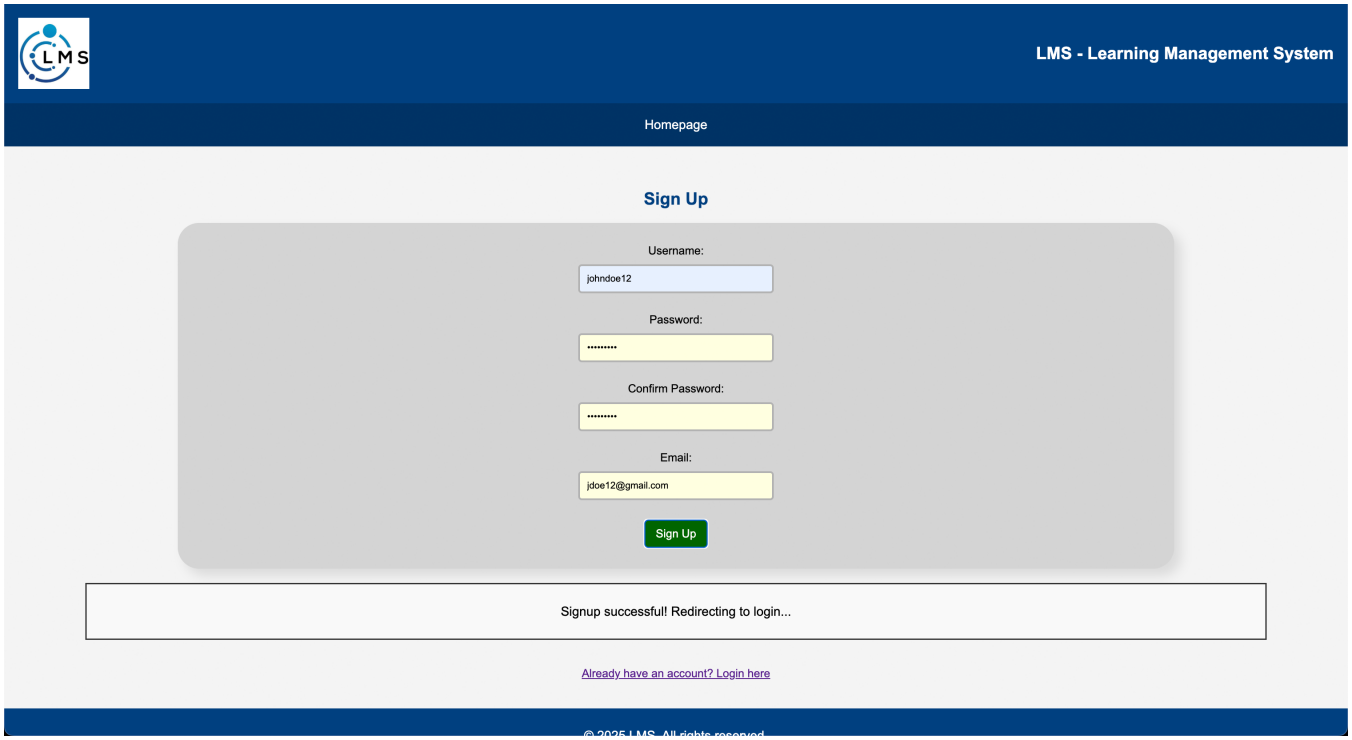
C. `Footer` *Component:*

a. Copyright notice at the bottom.

Structure of SignupPage.js

```
<div>
  <Header />
  <RegForm />
  <Footer />
</div>
```


LMS - Learning Management System
Homepage
Sign Up
Username:
234jhslkjdhf
Password:
Confirm Password:
Email:
qosaoeir@sidhf
Sign Up
Invalid username (Reason:...)
Invalid password (Reason:...)
Passwords do not match
Invalid email (Reason:...)


LMS - Learning Management System
Homepage
Sign Up
Username:
johndoe12
Password:
Confirm Password:
Email:
jdoe12@gmail.com
Sign Up
Signup successful! Redirecting to login...
Already have an account? Login here
© 2025 LMS. All rights reserved

2. Backend Setup and Data Storage: 10 marks

Create and initialize Flask App:

- Create a new directory named "Backend".
- Navigate to the "Backend" directory and initialize a Flask app in the directory by create a Python file named "app.py".
- Move the data files (`courses.js` and `testimonials.js`) defined in Assignment04 to the backend and **change their file extensions to json**. **You must access and serve the data from the backend ONLY**.
- Define and maintain a `students` list on the backend (e.g., a Python list or dictionary) that stores information about each student, including:
    - ID
    - Username
    - Password
    - Email
    - Enrolled_courses (list of courses)

3. Backend-frontend Integration (API Endpoints): 70 marks

1. Student Registration API

Endpoint: `POST /register`
Requirements:

- Receives new user information (e.g., `username`, `password`, etc.) from the Signup Page.
- Checks if the username already exists in the `students` list.
- If it exists, return a message saying the username is already taken.
- Otherwise, add the user to the `students` list and return a success message.

2. Login API

Endpoint: `POST /login`
Requirements:

- Accepts `username` and `password` in the request body.
- Validates the credentials against the `students` list from the Login Page.
- If correct, respond with success and redirect the user from the Login Page to Course Enrolment Page.
- Otherwise, respond with an error message to be displayed on the frontend.

3. Testimonials API

Endpoint: `GET /testimonials`
Requirements:

- Returns **two random testimonials** from `testimonials.json`.
- This API should be called whenever the Home Page is loaded or refreshed.

4. Enroll Courses API

Endpoint: `POST /enroll/<student_id>`
Requirements:

- This API is called when the user clicks on "Enroll Now" button in the Course Enrolment Page.
- Accepts the student ID in the URL (i.e., dynamic routing).
- Receives course information in the request body.
- Adds the course to the corresponding student's `enrolled_courses` in the `students` list.
- If successful, send confirmation to the frontend.
- Otherwise, return an error message to be displayed as an alert to the user.

5. Delete Courses API

Endpoint: `DELETE /drop/<student_id>`
Requirements:

- This API is called when the user clicks on "Drop Course" button in the Course Enrolment Page.
- Accepts the student ID in the URL (i.e., dynamic routing).
- Receives course information in the request body.
- Removes the course from the student's `enrolled_courses` in the `students` list.
- If successful, send confirmation to the frontend.
- Otherwise, return an error message to show an alert to the user.

6. Get All Courses API

Endpoint: `GET /courses`
Requirements:

- Returns all courses available in `courses.json`.
- This API is called when the Course Enrolment Page loads or refreshed.

7. Get Student Courses API

Endpoint: `GET /student_courses/<student_id>`
Requirements:

- This API is called when the Course Enrolment Page loads or refreshed.
- Accepts student ID in the URL (i.e., dynamic routing).
- Returns a list of courses that the student is currently enrolled in.
- If successful, display the enrolled courses in the Course Enrolment Page.
- Otherwise, return an empty list if the student did not register for any courses.

Note: You are required to **update your implementation from Assignment 4** to integrate it with the backend APIs developed in this assignment, following the specifications provided above.