Software Engineering Program

Sino-British Collaborative Education

CDUT

Learning Management System Web Site

- Preliminary Report -

|  |
| --- |
| **Student ID:** |
| **Date:** 28th April 2024 |
| **Group:** |

Table of Contents

[1. Introduction: 3](#_Toc160390204)

[2. Database design: 4](#_Toc160390205)

[3. Wireframes: 5](#_Toc160390206)

[4. Functionality of client and server 6](#_Toc160390207)

[5. References: 7](#_Toc160390208)

# Introduction:

The advent of digital technology has changed the educational landscape, introducing innovative solutions to enhance learning experiences and administrative efficiency. This report introduces a web-based Learning Management System (LMS), a comprehensive platform tailored to facilitate the administration, delivering, and monitoring of educational courses.

The scope of this project encompasses the development and testing of a full-stack application that enables: instructors can efficiently manage course content and assessments, students can engage with the material and track their progress, and administrators can oversee the system's operation and user management.

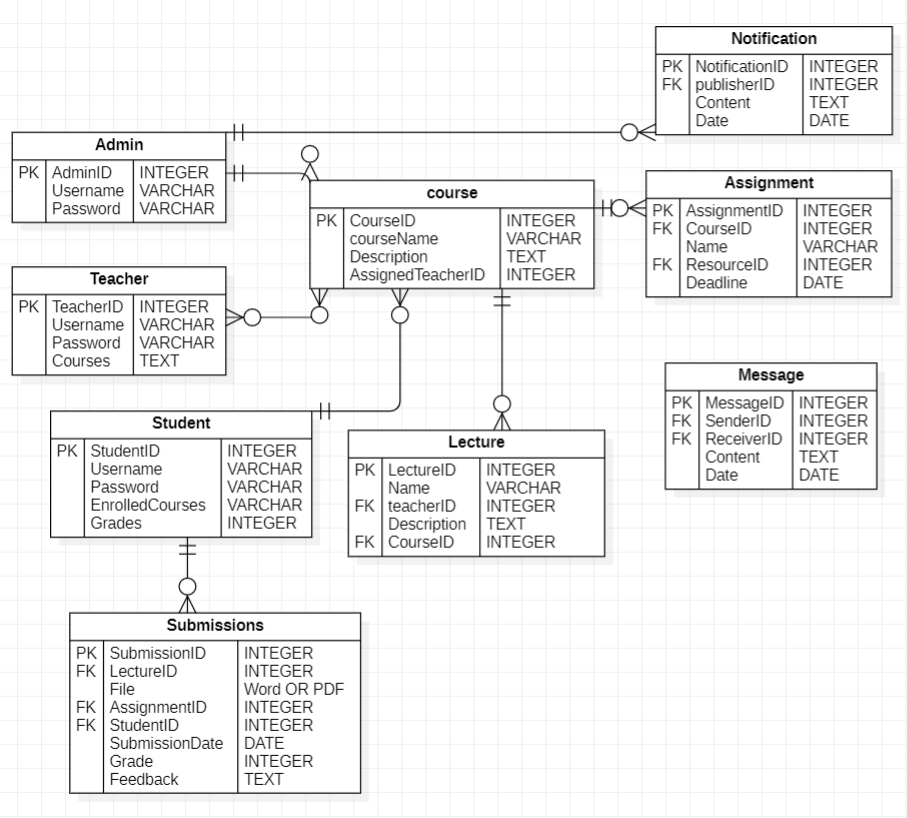
The significance of the LMS lies in its potential to facilitate a seamless educational experience, bridging the gap between traditional and digital learning. It aims to enhance the accessibility of educational resources, improve communication between educators and learners, and provide a centralized platform for academic management.

The LMS is developed using HTML5 and ECMAScript 6 for the client-side, ensuring a responsive and interactive user experience. The server-side is powered by Flask, MySQL, and Python 3, forming a robust backend capable of handling complex data transactions and user interactions. The choice of these technologies is driven by their proven reliability and flexibility in web application development.

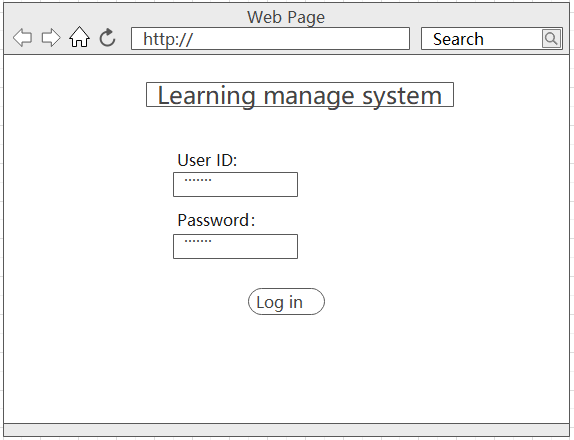
The report is structured to provide an overview of the Learning Management System (LMS) details. It starts with an exploration of the database structure, illustrating table relationships and key data fields. Following this, it presents wireframes depicting the potential visual layout of the website. Lastly, it outlines the client-server interactions and technologies required to implement the LMS functionality.

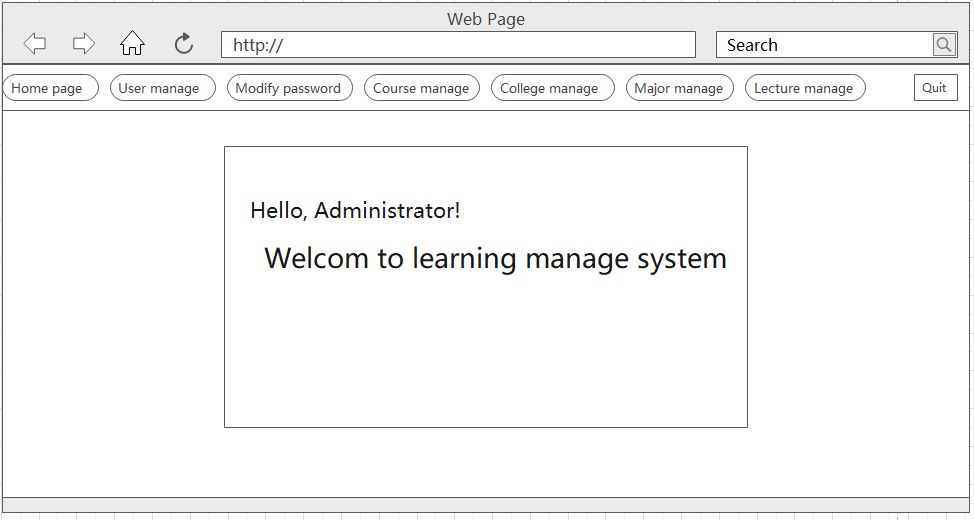
In conclusion, this introduction serves as a precursor to the detailed exploration of the LMS development journey. It emphasizes the system's objectives, scope, and the thoughtful application of technologies to achieve a transformative educational platform.

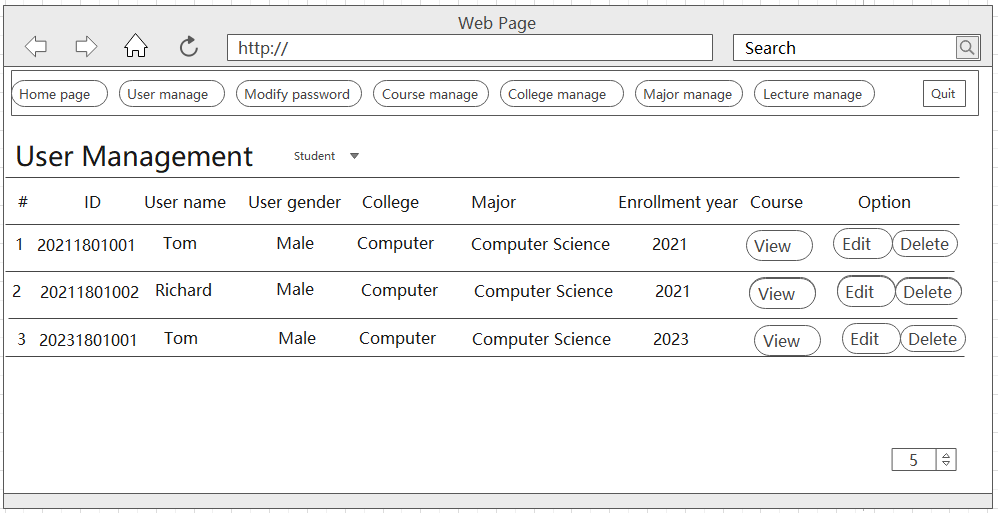
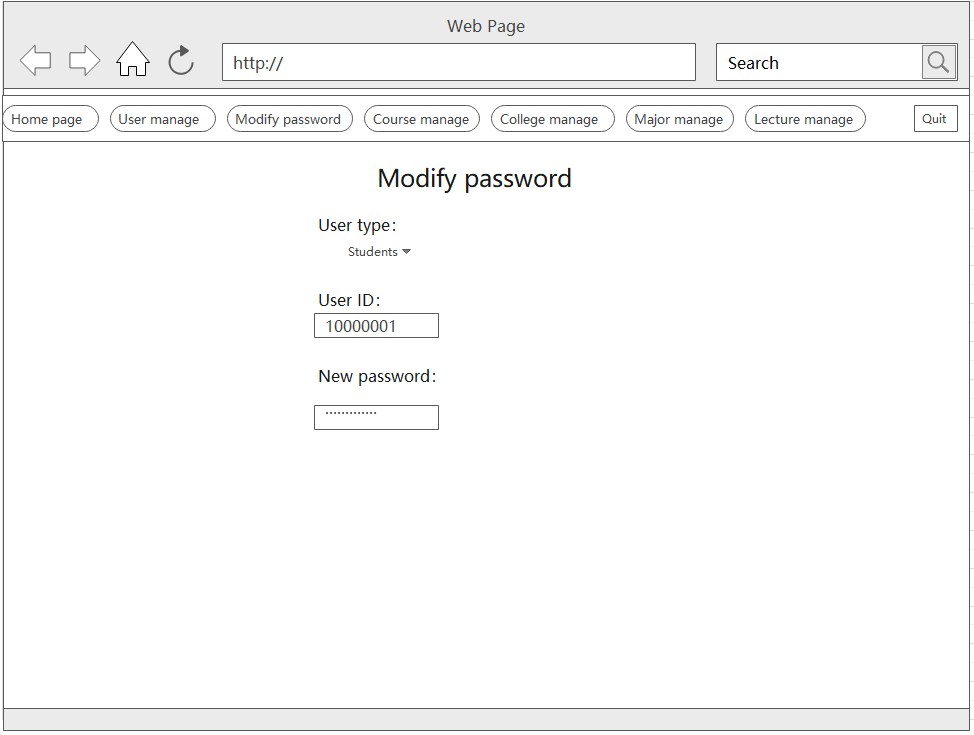
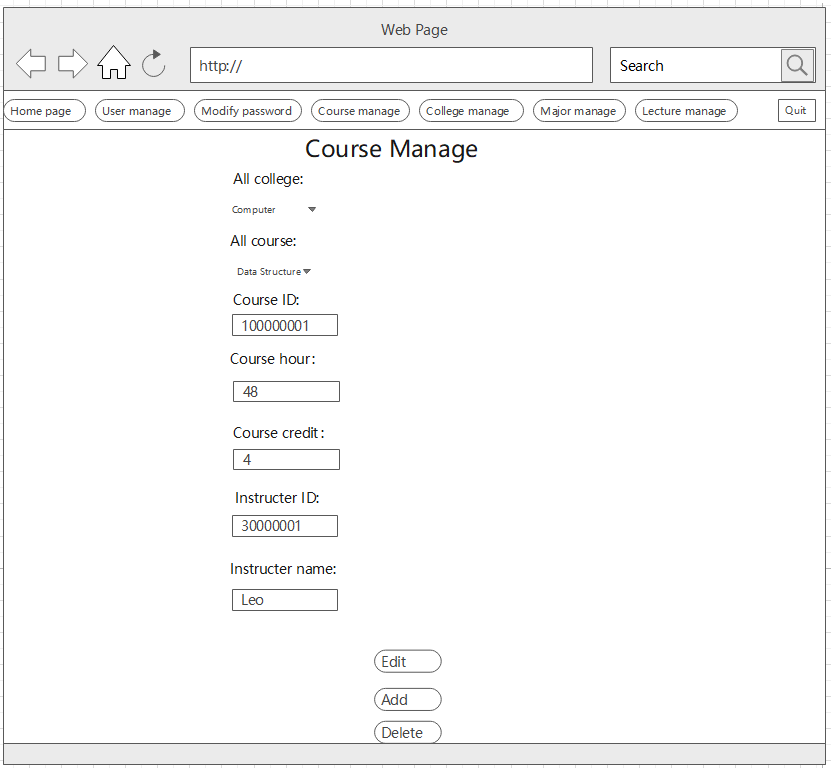
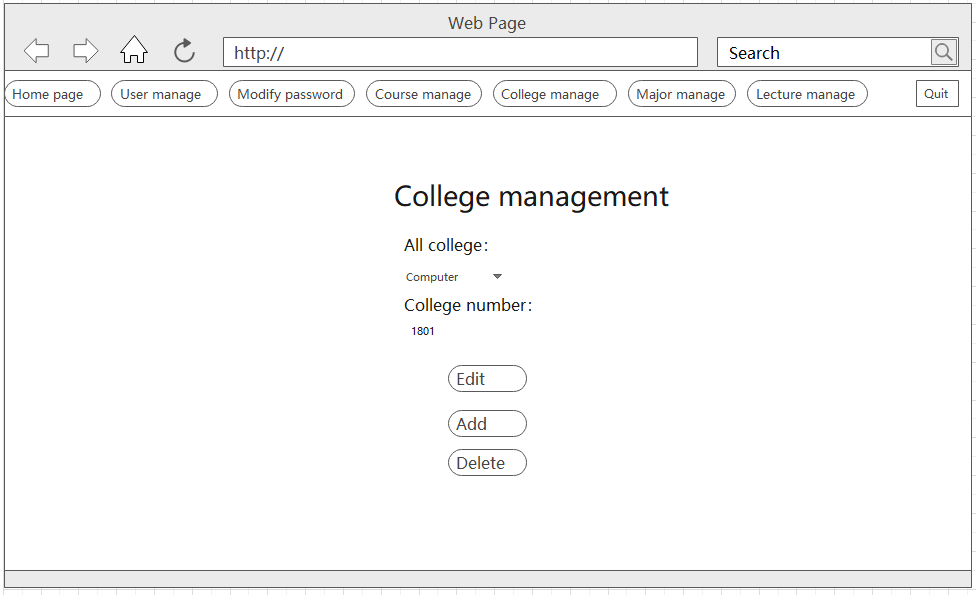
# Database design:

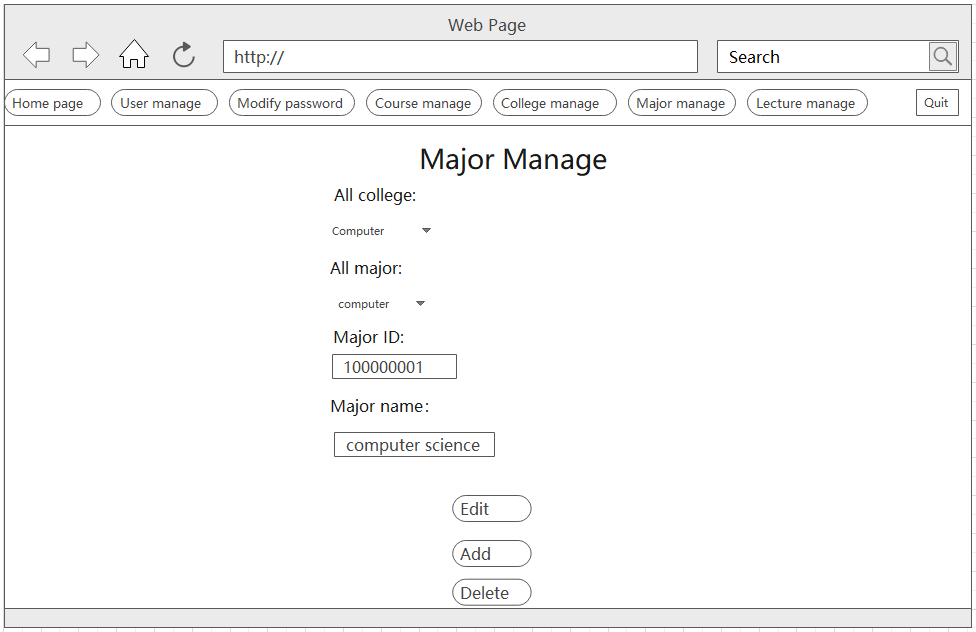


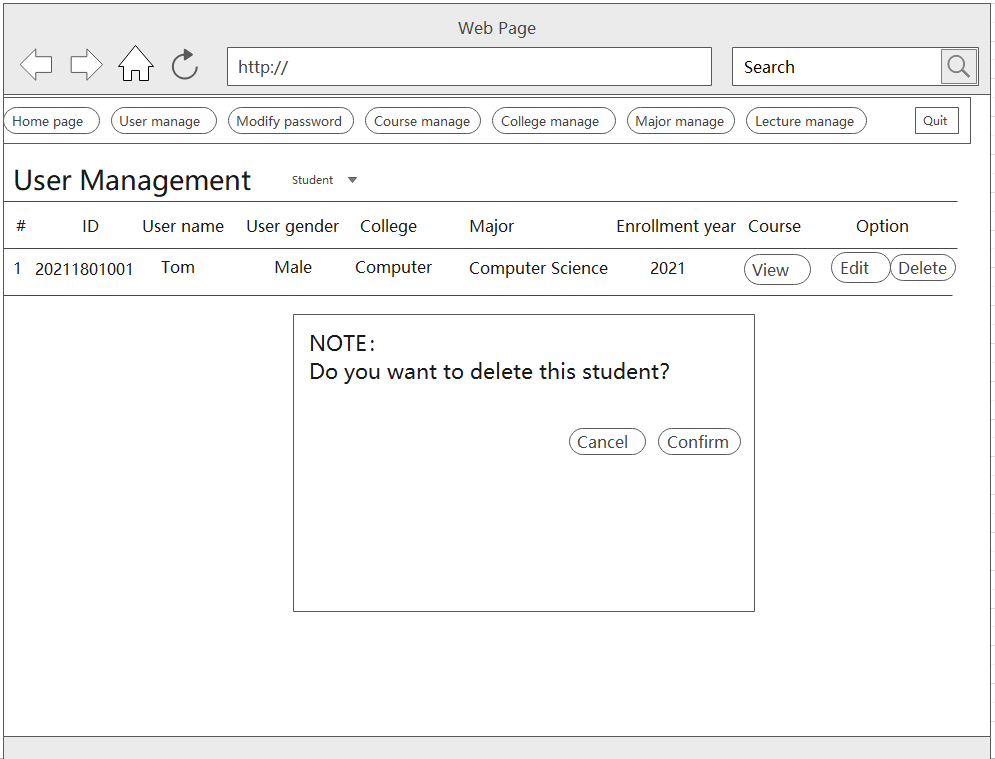
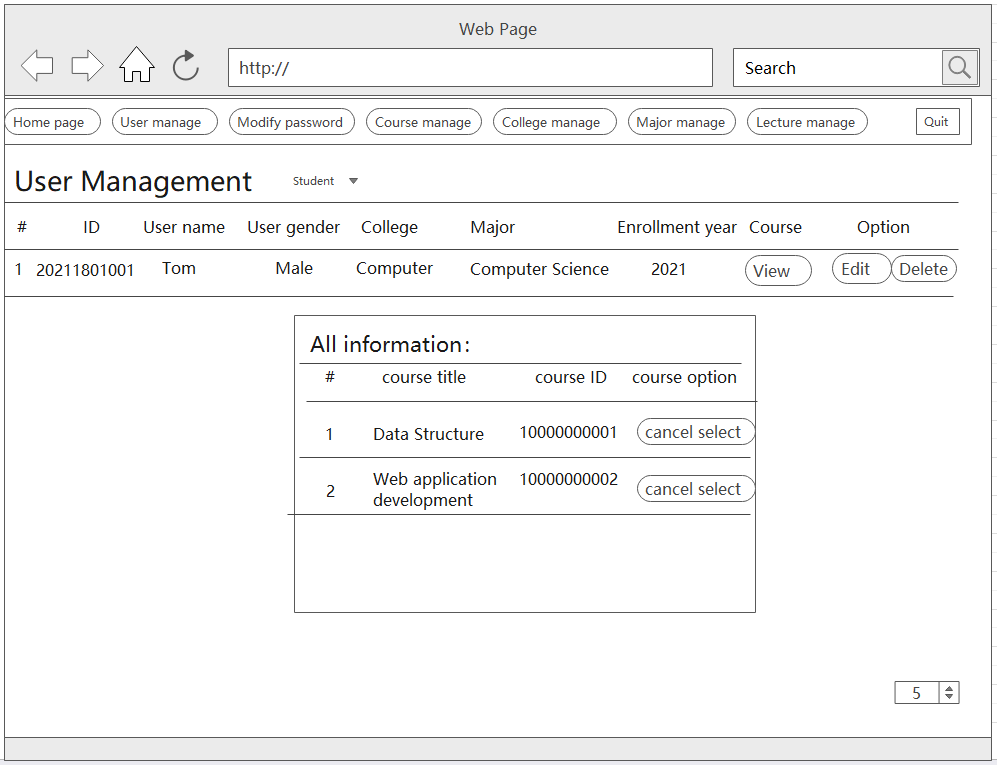
# Wireframes:

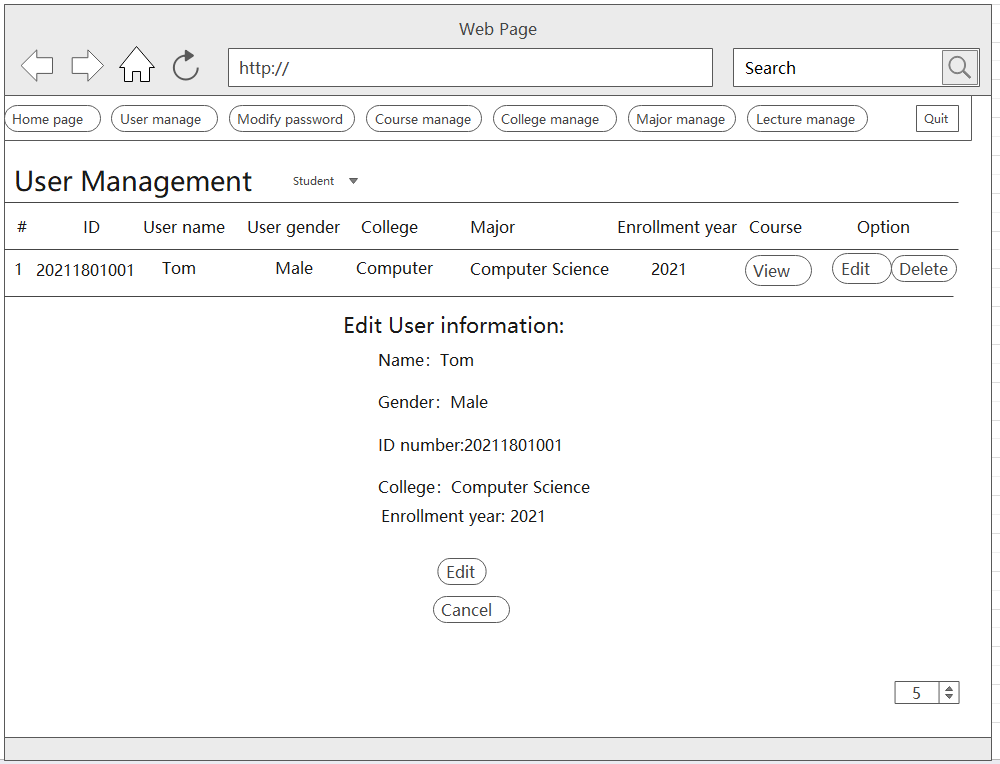
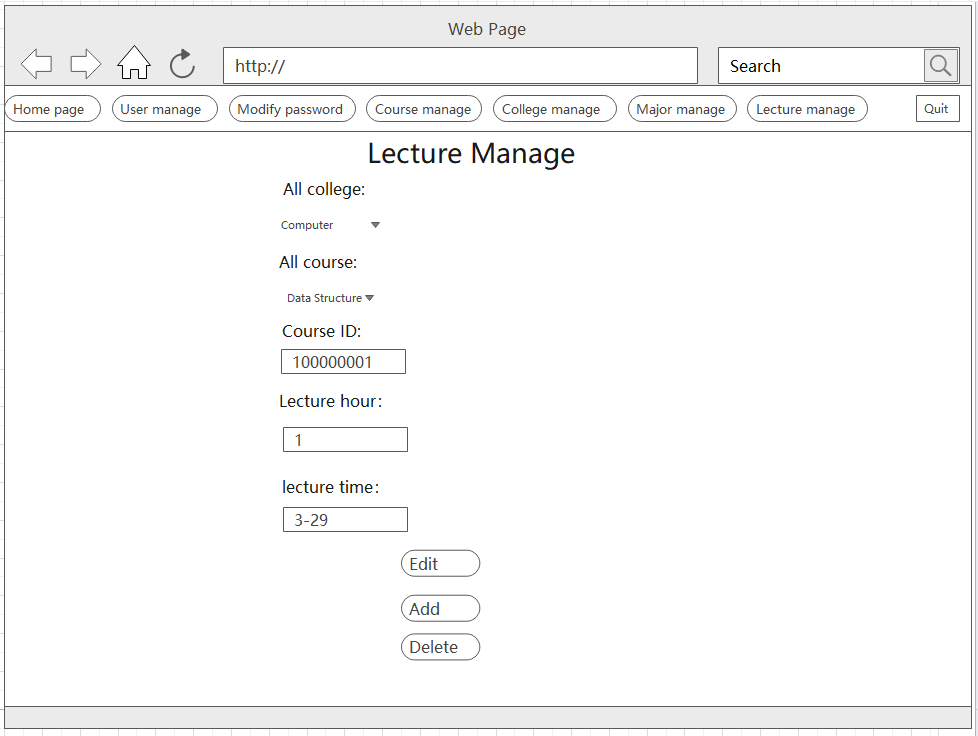
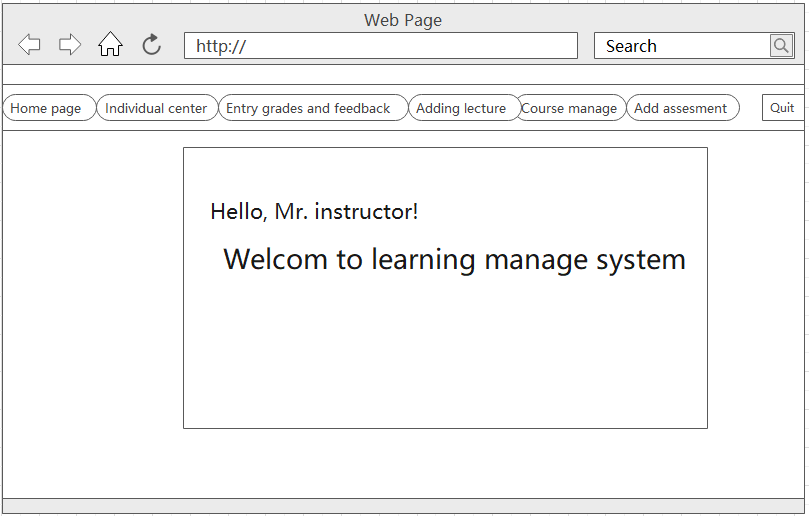
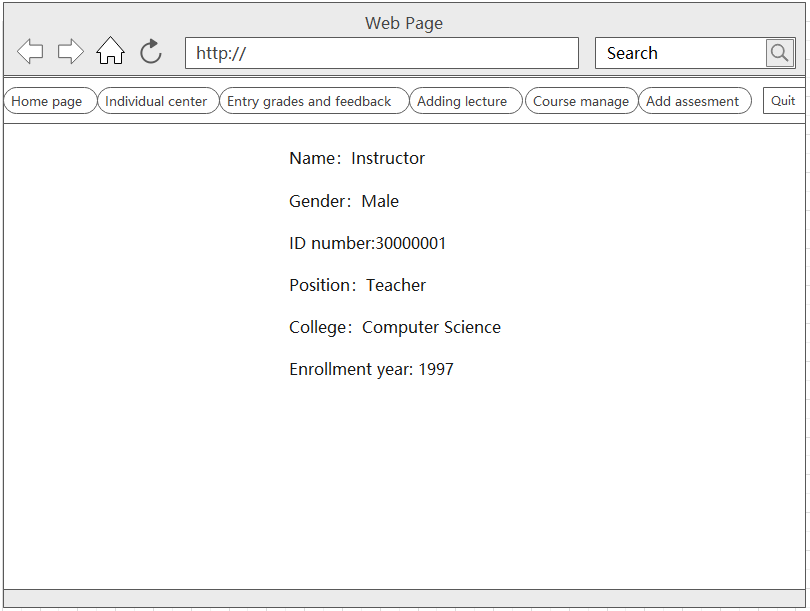
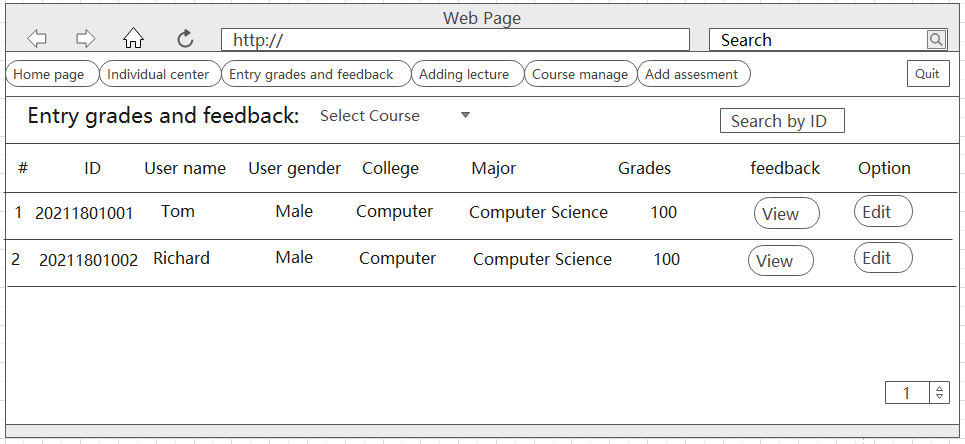
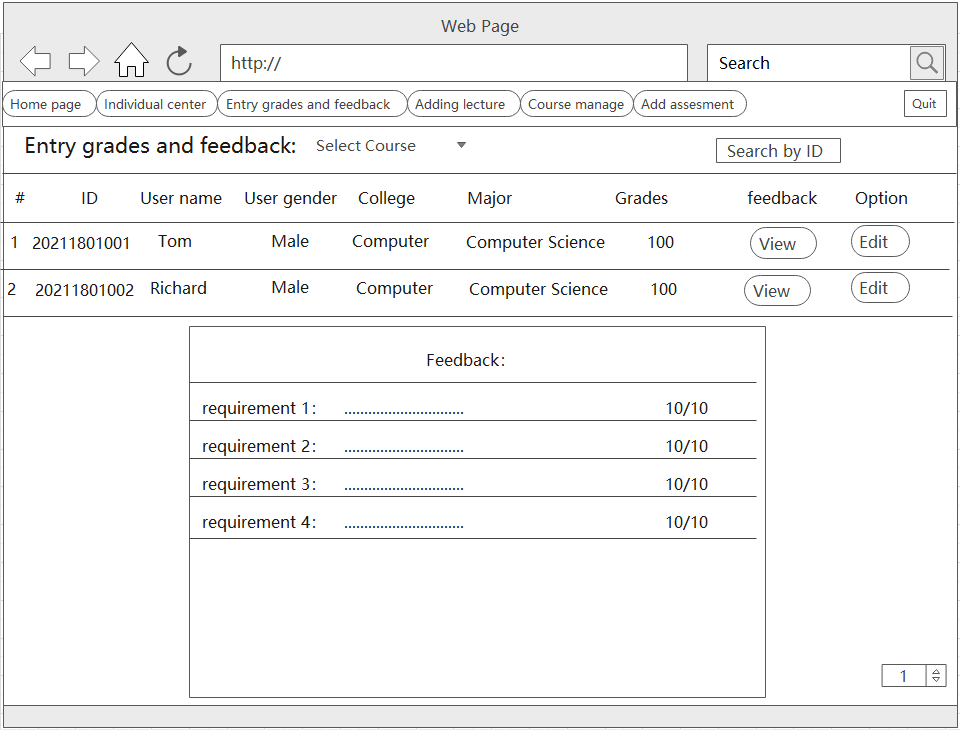
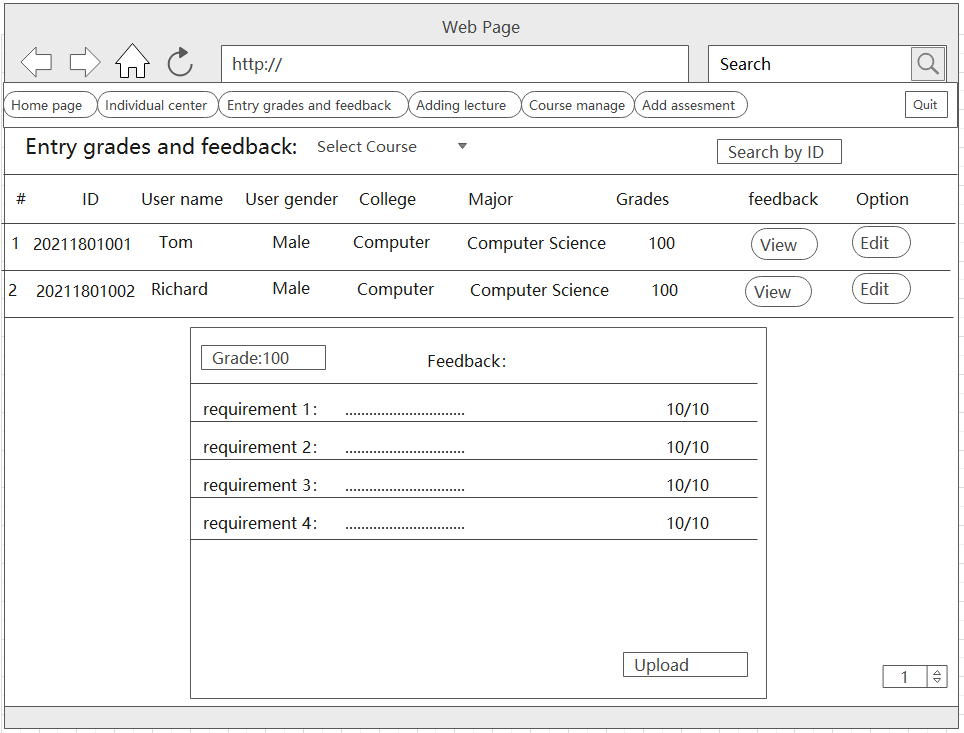
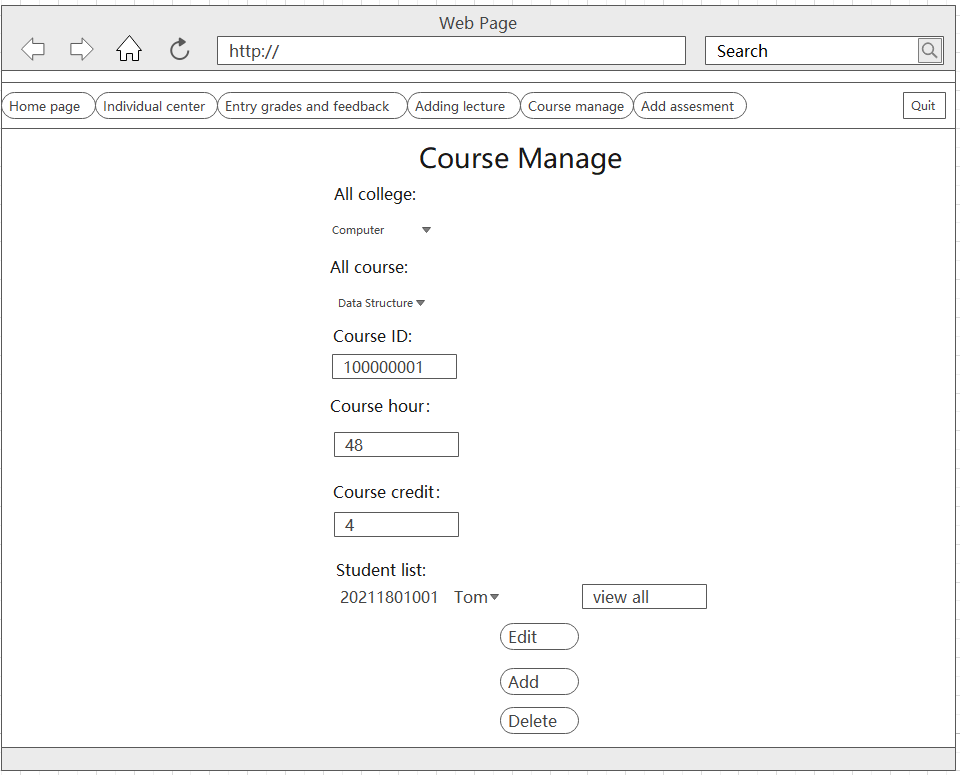
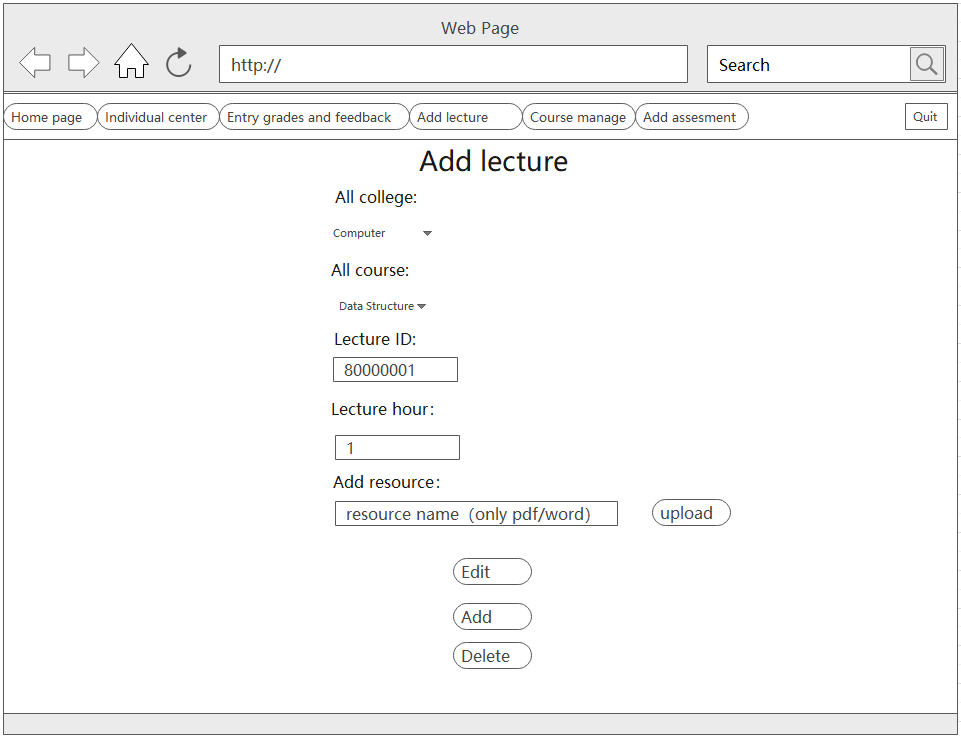
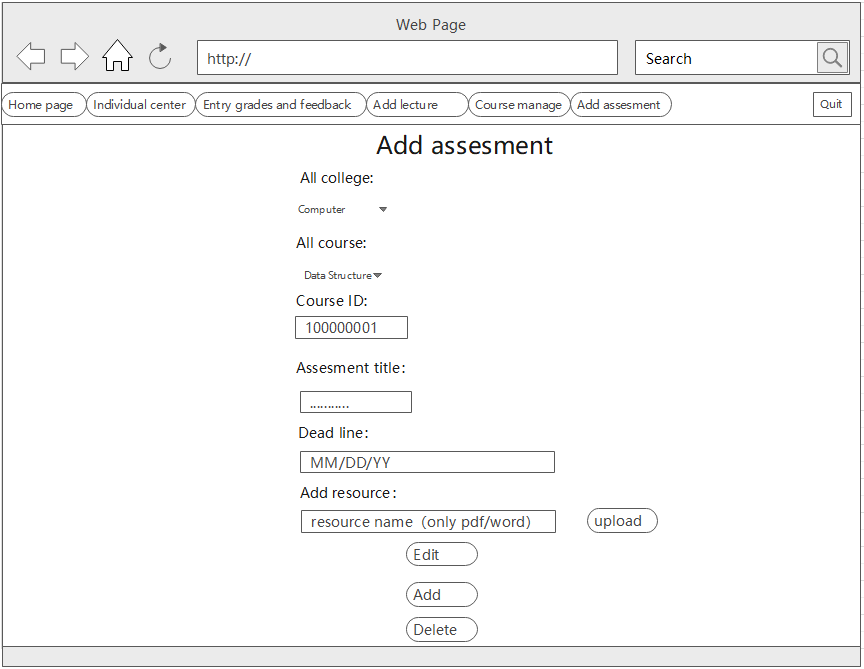
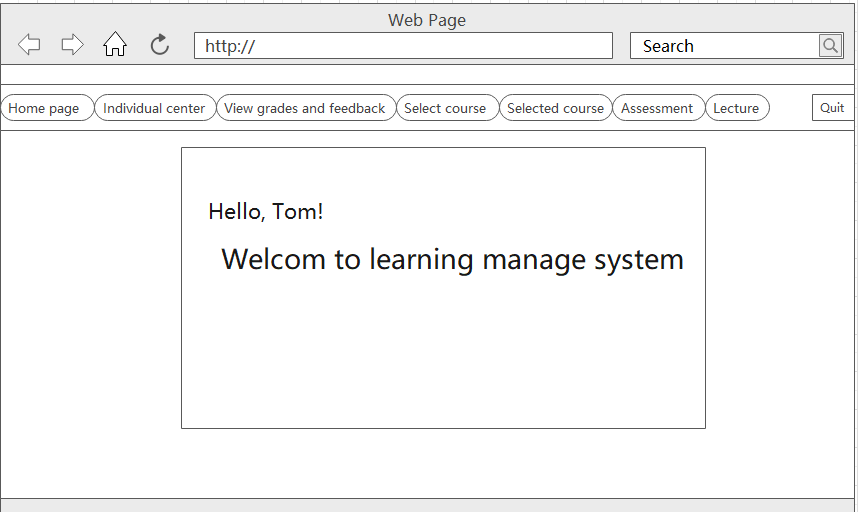
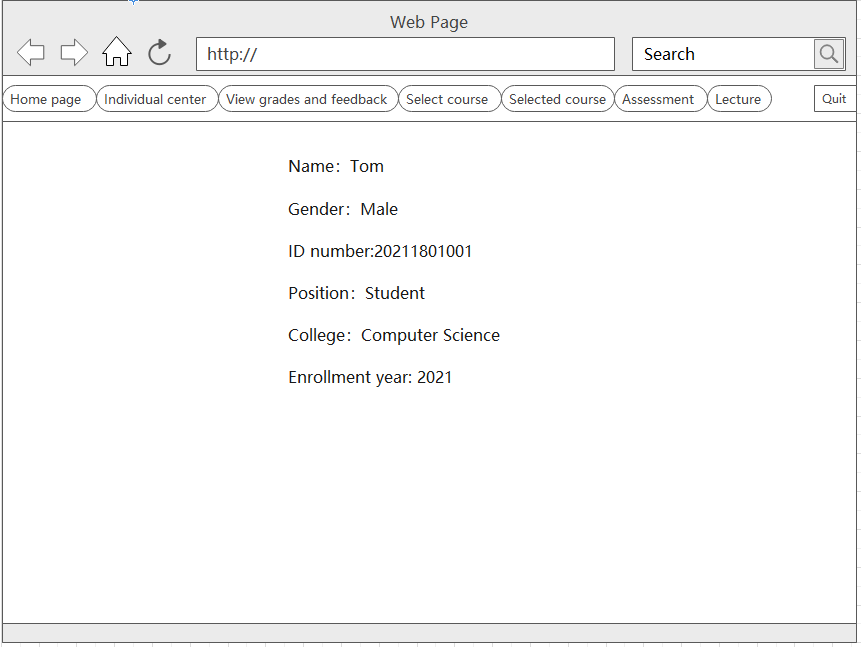
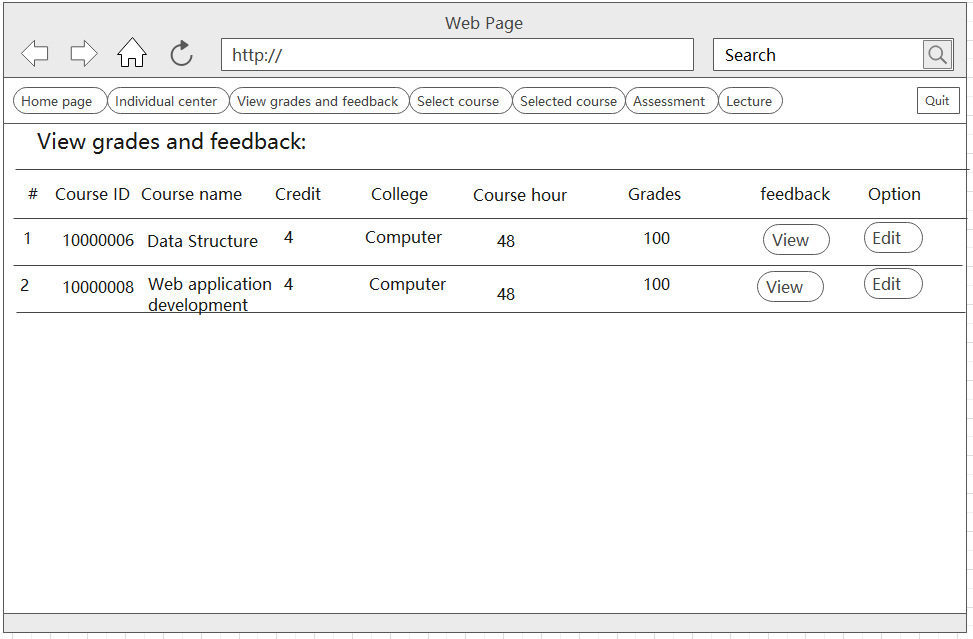
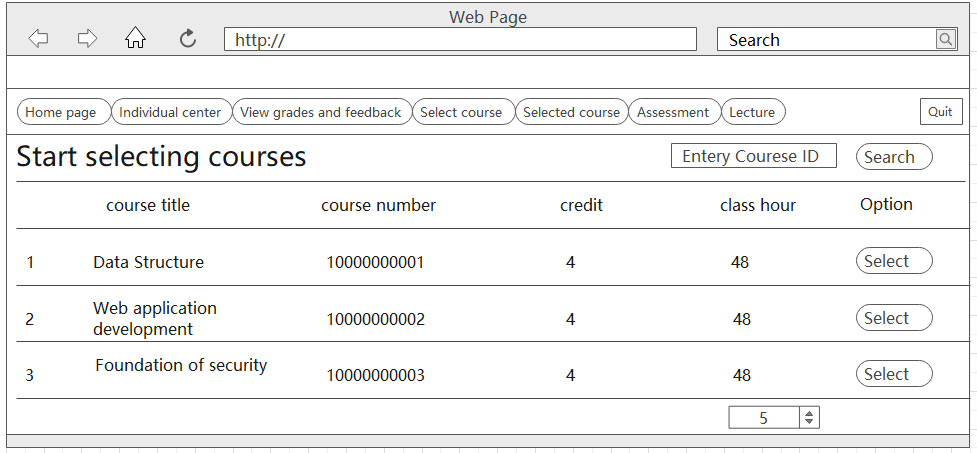
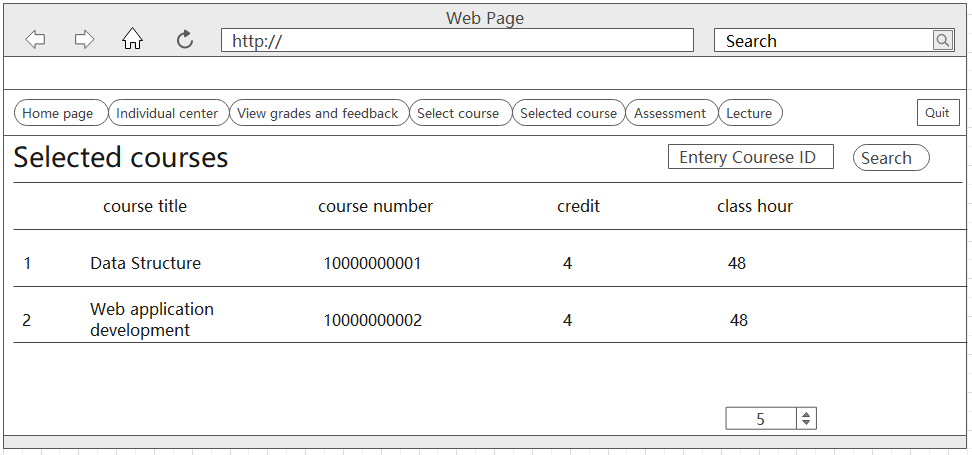
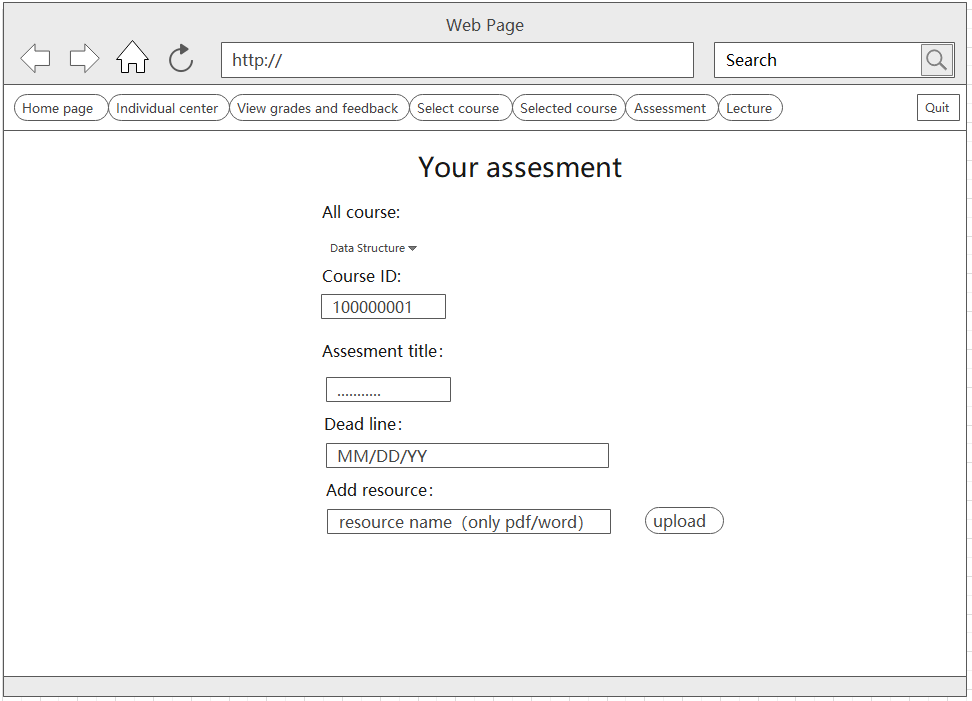
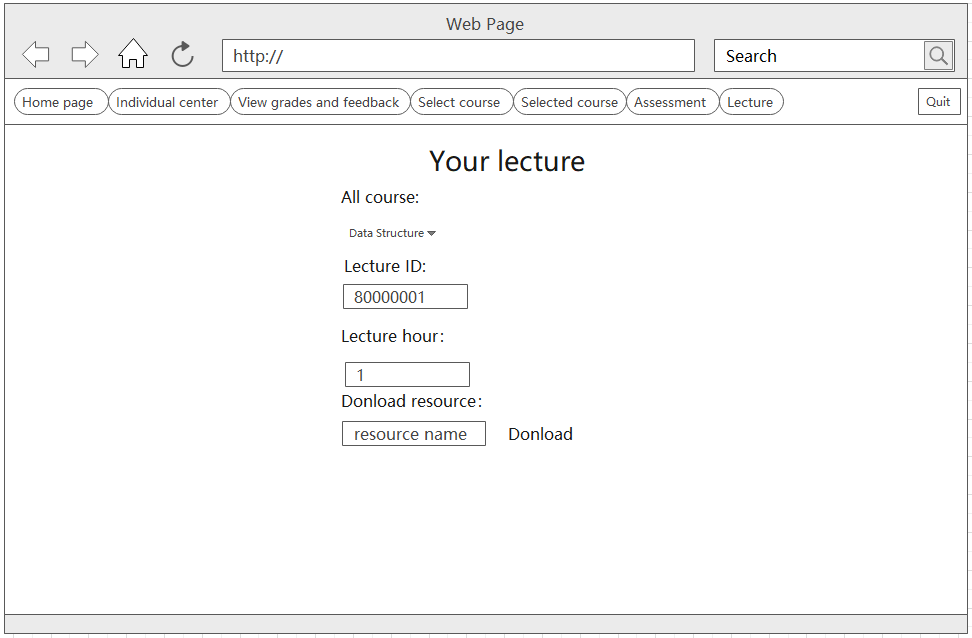
* 1. Log in 
  2. Home page （administer）



* 1. User management（admin）
  2. Modify password（admin）
  3. Course manage（admin）
  4. College management 
  5. Major manage（admin）



* 1. User manage（delete）
  2. User manage （view option）

1. User manage （Edit）
2. Lecture manage（admin）
3. Home page(instructor) 
4. Individual center 
5. Entry grades and feedback 
6. Entry grades and feedback（feedback view）
7. Entry grades and feedback（edit）
8. Course manage（instructor）
9. Add Lecture（instructor）
10. Add assessment 
11. Home page（student）
12. Individual center（student）
13. View grades and feedback 
14. Select course 
15. Selected course 
16. Assessment(student) 
17. Lecture 

# Functionality of client and server

* 1. Login

**Client:** HTML+ CSS+ JavaScript (jQuery+ DOM)

write the login page code in html, and use jQuery to manipulate DOM to show and hide. Clicking on the "login" button, client takes care of the form submission and sends the user name and password to the server through a POST request.

**Server:** Flask +MySQL+render\_template+flash+request

* The server uses the Flask framework and MySQL database. The server defines the route /login that handles the login request. In the view function, first check if the user has passed is\_authenticated authentication. In the case of a POST request, the server verifies the validity of the form data and attempts to query the corresponding user in the database based on the user number.
* If the user exists and the password is correct, use the login\_user function to login to the user and redirect the user to next\_page.
* If next\_page does not exist, the route is redirected to web.home by default.
* If an error occurs during the login (for example, the user does not exist or the password is incorrect), the flash function is used to display an error message to the user.
  1. Course List:

**Client:** HTML+ CSS+ JavaScript +AJAX

Create an HTML template to display the course list for students. Use CSS to style the template as desired. Use JavaScript to retrieve the course data from the server asynchronously. Make an AJAX request to the server, specifying the route "/courses" to fetch the course data. Handle the response from the server and dynamically generate the course list on the client-side using JavaScript.

**Server:** Flask + MySQL + JSON API

* Define a route in Flask ("/courses") that corresponds to fetching the course list for students.
* Implement the route handler function for "/courses" in Flask.
* In the route handler, query the database to retrieve the relevant course information for the logged-in student. (sql = "select \* from COURSE a where not exists(select \* from COURSE\_SELECT\_TABLE b)) Serialize the retrieved data into JSON format.
* Send the serialized data as a response to the client.
  1. Upload:

**Client:** HTML+ CSS+ JavaScript +AJAX

* The instructor interface should allow instructors to create and manage assignments and lecture resource. Implement a form or interface using HTML, CSS, and JavaScript to capture the assignment details such as name, files/resources, and deadline. Make an AJAX request to the server, specifying the route "/uploads" to fetch the assignments and lecture data.

**Server:** Flask + MySQL

* The server-side code will handle the request from the client to fetch the upload lists. In the Node.js framework, there will be a route defined (e.g., "/uploads") that corresponds to this functionality.
* Implement the route handler function for "/uploads" in Flask.
* In the route handler, query the database (e.g., MySQL) to retrieve the relevant upload information for the logged-in instructor.
* Serialize the retrieved data into JSON format.
* Send the serialized data as a response to the client.
  1. Add

**Client:** HTML+ CSS+ JavaScript

The client-side implementation involves HTML, CSS, and JavaScript. The form for adding items (college, major, or course) should include fields for relevant information such as college number, college name, major number, major name, course number, course name, course credit, and course hour. The data from the form needs to be transferred to the server using a POST request.

**Server:** Flask + MySQL + flash+ request.

* There should be a route defined for the "/manage\_add" endpoint with the "POST" method.
* Check if the current user is a manager. If not, return an appropriate response or error message.
* Retrieve the data sent from the client-side. This can be done using the request.values.get() method to access the values corresponding to the form fields.
* Determine the type of item to be added based on the received data. The type can be identified using a parameter such as 'type' sent from the client-side. Based on the type, perform the corresponding actions:
* If the type is '0', it corresponds to adding a college. Check if the college already exists in the database. If it exists, return an appropriate response indicating that the college already exists. Otherwise, create a new College object with the provided college number and name.
* If the type is '1', it corresponds to adding a major. Check if the major already exists in the database. If it exists, return a response indicating that the major already exists. Otherwise, create a new Major object with the provided major number and name. Assign the corresponding college using the college number.
* If the type is '2', it corresponds to adding a course. Check if the course already exists in the database. If it exists, return a response indicating that the course already exists. Otherwise, create a new Course object with the provided course number, name, credit, and hour. Assign the corresponding college using the college number.
* Add the newly created item (college, major, or course) to the database using the db.session.add() method. Then, commit the changes to the database using db.session.commit().
* Return a response indicating the success or failure of the operation. If the item was added successfully, return a success message. Otherwise, return an appropriate “500” HTML.
  1. edit:

**Client:** HTML+ CSS+ JavaScript

* There should be separate pages or forms for editing college, major, and course information. Each form should include fields for relevant information such as college number, college name, major number, major name, course number, course name, course credit, and course hour. The data from the form needs to be transferred to the server using a POST request.

**Server:** Flask + MySQL + request+ json.

* There should be a route defined for the "/manage\_edit" endpoint with the "POST" method.
* Check if the current user is a manager. If not, return an appropriate response or error message.
* Retrieve the data sent from the client-side. This can be done using the request.values.get() method to access the values corresponding to the form fields.
* Determine the type of item to be edited based on the received data. The type can be identified using a parameter such as 'type' sent from the client-side. Based on the type, perform the corresponding actions:
* If the type is '0', it corresponds to editing a college. Update the college information in the database by using the update() method with the appropriate filters and values.
* If the type is '1', it corresponds to editing a major. Update the major information in the database by using the update() method with the appropriate filters and values.
* If the type is '2', it corresponds to editing a course. Update the course information in the database by using the update() method with the appropriate filters and values.
* Commit the changes to the database using db.session.commit().
* Based on the success or failure of the update operation, set the appropriate message indicating the result.
* Use the flash() function to provide feedback to the user about the outcome of the edit operation.
* Render the appropriate template (e.g., 'managing\_college.html', 'managing\_major.html', or 'managing\_course.html') with the updated data, error message, and the type of item being edited.
  1. Delete

**Client:** HTML+ CSS+ JavaScript

There should be a form or button on the page that triggers the deletion process. The form or button should include fields for relevant information such as type (college, major, or course), college number, major number, and course number. The data from the form needs to be transferred to the server using a POST request.

**Server:** Flask + MySQL + request+ json.

* There should be a route defined for the "/manage\_delete" endpoint with the "POST" method.
* Check if the current user is a manager. If not, return an appropriate response or error message.
* Retrieve the data sent from the client-side. This can be done using the request.values.get() method to access the values corresponding to the form fields.
* Determine the type of item to be deleted based on the received data. The type can be identified using a parameter such as 'type' sent from the client-side. Based on the type, perform the corresponding actions:
* If the type is '0', it corresponds to deleting a college. Check if the college has associated teachers or students by accessing the related objects (e.g., TEACHERS) and checking their length. If there are associated teachers or students, return a response indicating the failure to delete and a message stating that there are related records. Otherwise, delete the college object from the database using the delete() method.
* If the type is '1', it corresponds to deleting a major. Check if the major has associated teachers or students by accessing the related objects (e.g., STUDENTS) and checking their length. If there are associated teachers or students, return a response indicating the failure to delete and a message stating that there are related records. Otherwise, delete the major object from the database.
* If the type is '2', it corresponds to deleting a course. Check if the course has associated teachers by accessing the related objects (e.g., TEACHERS) and checking their length. If there are associated teachers, return a response indicating the failure to delete and a message stating that there are related records. Otherwise, delete the course object from the database.
* Commit the changes to the database using db.session.commit(). Based on the success or failure of the deletion operation, set the appropriate message indicating the result.
* Return a JSON response including the result message and a status code.
  1. Grading and Feedback :

**Client:** HTML+ CSS+ JavaScript

The interface should allow instructors or students to create and manage assignments. Implement a form or interface using HTML, CSS, and JavaScript to show the Feedback details and grades.

**Server:** Flask + MySQL + request +flash

* The server should provide routes "/ Grading and Feedback "to handle the grading and feedback process.
* When an instructor or students accesses the grading interface, the server should retrieve the submitted assignments for the specific course and provide an interface for grading and providing feedback.
* Upon grading, the server should update the database with the grades and feedback for the respective assignments.
* Provide appropriate feedback to the instructor using "flash" messages to indicate the success or failure of the grading process.
  1. Download

**Client:** HTML+ CSS+ JavaScript+ AJAX

* Design the button " Download " in lecture manage page (student) to download lecture resource and feedback page (instructor) to download student’s assignments.
* Apply CSS styles for html to make the button " Download " usage in pages.
* Write JavaScript code to handle the download action triggered by clicked. When the button " Download " be clicked, an AJAX request will be sent the server to fetch the file and lecture and feedback.

**Server:** Flask + MySQL + request

* There should be a route defined for the "/download" endpoint with the "POST" method.
* Check if the current user is a student when request to download the file. If not, assignment from database and server. If student, retrieving the lecture use (sql = "select \* from lecture a where not exists(select \* from LECTURE\_resouce)) .
* Return a JSON response including the result message and a status code. Allowing the browser to handle the file download process for the instructor
  1. Notification

**Client:** HTML+ CSS+ JavaScript+ AJAX

* Design the " Notification " button in three user’s homepage to show users notification. The form or button should include fields for relevant information such as type (Name, Description , push time and so on)
* Use Html to design the notification page for displaying the notifications content.
* Use CSS styles to design the notification page, ensuring it is suitable visual.
* Write the JavaScript code to handle the " Notification " button. When the button be clicked by users, triggering the action to the notification page.

**Server:** Flask + MySQL + request+ json.

* Define a route in Flask ("/notification") that corresponds to fetching the course list for students, instructor and administrator.
* Implement the route handler” GET” function for "/courses" in Flask. When the " Notification " button be clicked, render the HTML template for the notification page, displaying the notification data from the server.
  1. Message

**Client:** HTML+ CSS+ JavaScript+ AJAX

* Write JavaScript code to design the page of checking the inbox message and the page of composing the email.
* Use CSS styles to make the pages recognizable.
* Use AJAX requests or fetch the actions, such as send email, receive email and check email.

**Server:** Flask + MySQL + request+ json.

Define an Inbox Messages route in Flask ("/inbox" ). Fetch messages from the database. Return the messages as JSON data to the client.

Define a Compose Email route in Flask ("/send\_email "). Receive data (recipient, subject, message) from the client. Save the email to the database (if valid). Return a success message or handle errors.

# Conclusion:

This report summarizes the development of a web application with different functionalities targeting students, teachers, and administrators. These functionalities include course listings for students, course, assessment and lecture lists for teachers, user management for administrators. The client-side code utilizes HTML, CSS, and JavaScript to create an interactive interface. The server-side code is implemented using Node.js, along with REST APIs and template engines to render dynamic content. The system leverages a database (such as MySQL) for storing and retrieving data related to courses, uploads, users, and lectures.

In evaluating the achievement of project objectives and successful implementation, several aspects need to be considered three fields: First is functionality of web, assess whether the web application effectively provides the specified functionalities for students, teachers, and administrators. Verify if users can perform operations such as accessing course listings, managing uploads, user management, creating/editing/deleting courses, and creating/editing/deleting lectures. Second field is user experience, evaluate the overall user experience of the web application. Consider factors such as usability, responsiveness, and intuitive design. Gather user feedback to measure their satisfaction with the interface and functionalities. Third field is security, verify that appropriate security measures have been implemented to protect user data and prevent unauthorized access. Ensure that user authentication and authorization mechanisms are in place, and sensitive information is securely stored.

Challenges encountered during the development process may include:

First of first, Asynchronous communication is the most difficult field, handling asynchronous communication between the client and server using technologies like AJAX requires careful handling to ensure data consistency and error handling. Then the user management field, managing user roles, permissions, and access control can be a complex and challenging implementation task. Ensuring proper implementation of authentication and authorization mechanisms is crucial. Finally, Optimizing the performance of the web application, particularly when dealing with large datasets or handling concurrent requests, can be a challenging task.

I get some valuable experience learned from the development process. First is a proper plan, comprehensive definition of requirements and creating detailed plans before starting development can help avoid issues and streamline the implementation process. Second is Modular code structure which means breaking down the code into modular components and following best practices for code organization and maintainability can make development and debugging processes easier.

Potential future improvements or enhancements to the web application could include:

1. Improved user interface: Enhancing the user interface design and incorporating modern design trends can enhance the overall user experience and make the application visually appealing.
2. Real-time collaboration: Implementing real-time collaboration features such as live chat or collaborative document editing can facilitate communication and interaction among users.
3. Performance optimization: Continuously monitoring and optimizing the performance of the web application to handle increased traffic and improve response times can enhance user satisfaction.

Overall, the success of the implementation depends on the final achievement of project objectives, the usability and performance of the web application, and the satisfaction of end users. Regular testing and continuous improvement are key to ensuring the successful operation of the web application and meeting the evolving needs of users.

# References: