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:

\* In the introduction of an initial report for a web application development module, you should provide an overview of the topic, including its objectives, scope, and significance. This section sets the stage for the reader, outlining what the web application aims to achieve and why it is important. Additionally, you can introduce the technologies and methodologies used in the development process and provide a brief overview of the structure of the report.

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.

Preliminary report

● Repeating what is stated in the specification as if it was your decision rather than what you have been told to do.

● Telling the story of how a function will be implemented rather than how the implementation will work.

● Failing to break down tasks into client and server components or attributing all of a multi-stage task to one of these; for example “the student logs into the site” which actually requires multiple steps and cannot be purely done on the client.

● Describing what is stored on the client or server, but not what functionality they have.

● Over-emphasising the choice of technologies, rather than how they are used.

● Writing many-to-one relationships with the foreign key on the wrong side.

● Trying to store foreign keys to two different tables in the same field without distinguishing them.

● Not showing data table descriptions, only sample data that does not formally indicate structure.

● Omitting field types from specifications.

● Stating that the “user will be logged in” without indicating what method is used to signal this.

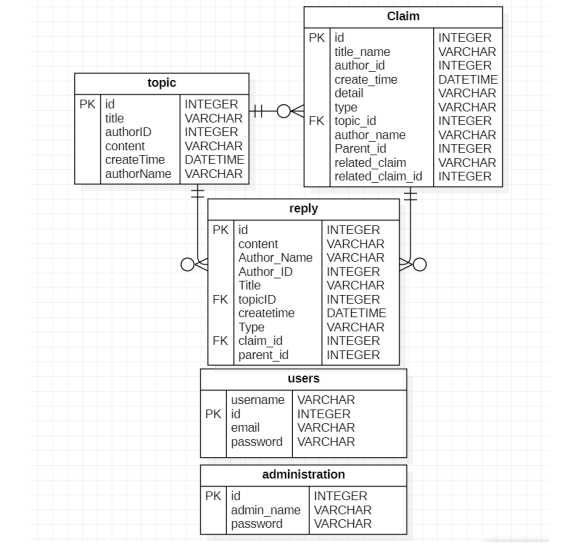
● Large amount of dead space on UI designs.

● Showing UI prototypes instead of wireframes.

● Where search is implemented, not providing a wireframe for the search results page.

● Where AJAX is used, not providing a structure for the JSON messages that will be used.

# Database design:



\* Cover all tables intended for use and provide the correct fields and relationships between them.

首页:在访问网站的第一页时，如果用户没有登录，用户应该被重定向到登录页面，

学生的课程列表，教师的上传列表。-管理员应该有权访问用户，课程，讲座管理(创建，编辑，删除)。-教师应该看到他们的课程，学生名单和讲座管理选项。-学生应该看到他们注册的课程、成绩和即将完成的作业。

●由管理员创建/编辑/删除和分配一个课程，只有一个讲师。

当讲师想要创建一个讲座(名称，描述)时，有一个“添加讲座”按钮，另一个“添加文件”按钮可以上传添加一个资源(word/pdf文件)。

●学生可以查看所提供的课程，并申请参加他们想参加的课程

教师和学生可以通过消息传递相互沟通

●通知:接收新公告、截止日期的提醒。

●搜索功能:搜索系统内的课程、讲座和其他用户

管理:

·课程注册:将一门课程分配给一名(或多名)讲师，为学生注册课程

教练:

·课程管理:创建课程内容，包括讲座和作业，讲师只能编辑自己的课程。

·作业管理:创建和管理作业(名称，文件，截止日期)，所有注册同一课程的学生将自动分配

·评分和反馈:访问学生的提交，评分并提供反馈

学生:

←Course Enrollment: Look over the offered courses and request to be enrolled in those they would like to take (only administrators can decide whether to accept or enroll a student).

课程注册:查看所提供的课程，并申请参加他们想参加的课程(只有管理员可以决定是否接受或注册学生)。

lContent Access: Access enrolled courses, lectures, assignments.

内容访问:访问已注册的课程、讲座、作业。

Submission: Submit assignments, within specified deadlines.

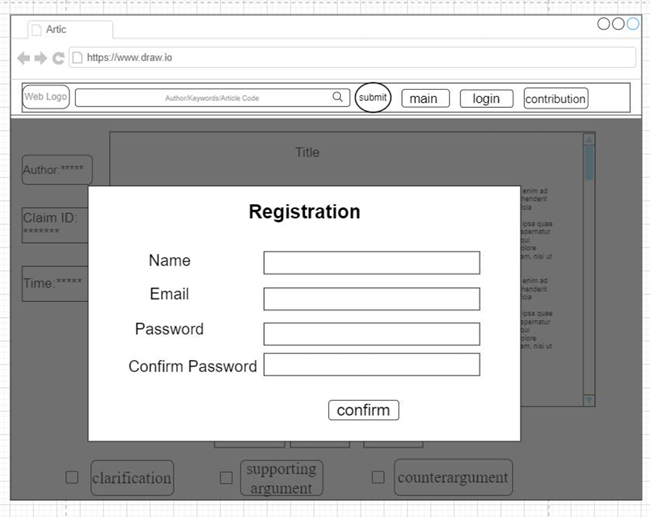
提交:在规定的期限内提交作业。

Feedback and Grades: Receive feedback and grades on assignments and assignments.

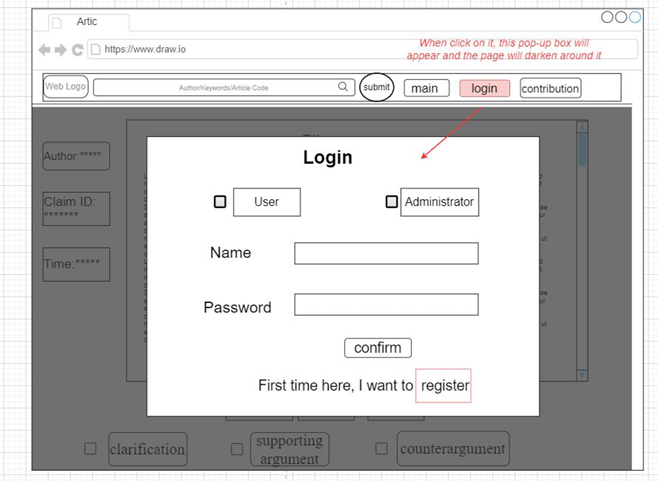
反馈和评分:接收作业和作业的反馈和评分。

# Wireframes:

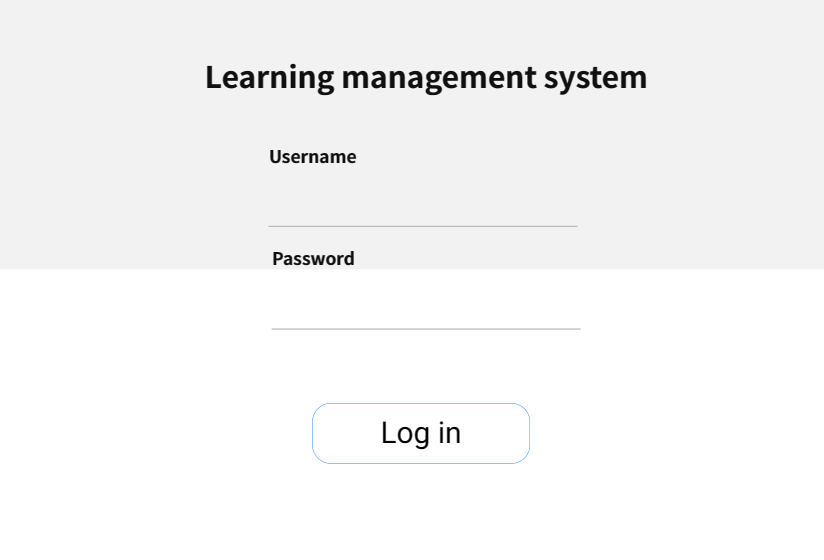
* 1. Register



* 1. Login



\* That does not mean that you will stop here, you will have to cover as much possible wireframes.



# Functionality of client and server

* 1. Register

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

As same as log in ；registration page requires username, password, password2, email

<Using Method1 in 4.1.1 to transfer the data>

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

"POST" request to the backend "register" route. Write code in the view function to determine the type of request. if request.method=="POST", then capture the user input and determine if the user input is complete by using "all([username,password,password2,email])", followed by the user name is duplicated (by comparing the user's input with the data stored in the database), and the password is the same for both entries. Feedback on the user's input is given via "flash". If this is correct, To ensure the security of the user's password, a hash transcoding ("generate\_password\_hash") is used so that the user's password is not visible in the database，and then add the user information to the database (newUser = users(username = username,password=password,email=email) db.session.add(newUser)

* 1. Login

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

use jQuery to make pop-up boxes, write the login and registration code in html, and use jQuery to manipulate DOM to show and hide. Clicking on the "login" button will bring up the interface, avoiding page jumps. The user enters information on the login/registration page (login page requires username and password; and then clicks the "submit" button, which sends "POST" to the backend. <Using Method1 in 4.1.1 to transfer the data>

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

* "POST" request to the backend "login" route. Write code in the view function to determine the type of request
* determine if the username and password entered by the user and the data stored in the database match (users = users.query.filter(users.username== username, users.password==password).first()), if they do, the login is successful. If it does not match, the user is given feedback via "flash": ""Wrong password"" etc.

To modify the code as per your requirements, you can make the following changes:

* c) Course List for Students:
* Client: HTML+ CSS+ JavaScript (jQuery+ DOM)
* 1. Create an HTML template to display the course list for students. Use CSS to style the template as desired.
* 2. Use JavaScript, jQuery, and DOM manipulation to retrieve the course data from the server asynchronously.
* 3. Make an AJAX request to the server, specifying the route "/courses" to fetch the course data.
* 4. Handle the response from the server and dynamically generate the course list on the client-side using JavaScript and DOM manipulation.
* Server: Flask + MySQL + JSON API
* 1. Define a route in Flask ("/courses") that corresponds to fetching the course list for students.
* 2. Implement the route handler function for "/courses" in Flask.
* 3. In the route handler, query the database (e.g., MySQL) to retrieve the relevant course information for the logged-in student.
* 4. Serialize the retrieved data into JSON format.
* 5. Send the serialized data as a response to the client.
* d) Upload List for Instructors:
* Client: HTML+ CSS+ JavaScript (jQuery+ DOM)
* 1. Create an HTML template to display the upload list for instructors. Use CSS to style the template as desired.
* 2. Use JavaScript, jQuery, and DOM manipulation to retrieve the upload list data from the server asynchronously.
* 3. Make an AJAX request to the server, specifying the route "/uploads" to fetch the upload list data.
* 4. Handle the response from the server and dynamically generate the list on the client-side using JavaScript and DOM manipulation.
* Server: Flask + MySQL + JSON API
* 1. Define a route in Flask ("/uploads") that corresponds to fetching the upload list for instructors.
* 2. Implement the route handler function for "/uploads" in Flask.
* 3. In the route handler, query the database (e.g., MySQL) to retrieve the relevant upload information for the logged-in instructor.
* 4. Serialize the retrieved data into JSON format.
* 5. Send the serialized data as a response to the client.
* e) User Management for Admins:
* Client: HTML+ CSS+ JavaScript (jQuery+ DOM)
* 1. Create an HTML template to display the user management interface for admins. Use CSS to style the template as desired.
* 2. Use JavaScript, jQuery, and DOM manipulation to handle user interactions, such as creating, editing, and deleting users.
* 3. Send appropriate requests to the server based on user actions (e.g., creating a user, editing a user, deleting a user).
* Server: Flask + MySQL + Render Template
* 1. Define routes in Flask ("/users/create", "/users/edit", "/users/delete") that correspond to user management functionalities for admins.
* 2. Implement route handler functions for each route in Flask.
* 3. In the route handler functions, perform the necessary actions, such as creating, editing, or deleting user records in the database (e.g., MySQL).
* 4. Render a template with appropriate feedback or confirmation messages to be displayed on the client-side.
* f) Course Management for Instructors:
* Client: HTML+ CSS+ JavaScript (jQuery+ DOM)
* 1. Create an HTML template to display the course management interface for instructors. Use CSS to style the template as desired.
* 2. Use JavaScript, jQuery, and DOM manipulation to handle user interactions, such as creating, editing, and deleting courses.
* 3. Send appropriate requests to the server based on instructor actions (e.g., creating a course, editing a course, deleting a course).
* Server: Flask + MySQL + Render Template
* 1. Define routes in Flask ("/courses/create", "/courses/edit", "/courses/delete") that correspond to course management functionalities for instructors.
* 2. Implement route handler functions for each route in Flask.
* 3. In the route handler functions, perform the necessary actions, such as creating, editing, or deleting course records in the database (e.g., MySQL).
* 4. Render a template with appropriate feedback or confirmation messages to be displayed on the client-side.
* g) Lecture Management for Instructors:
* Client: HTML+ CSS+ JavaScript (jQuery+ DOM)
* 1. Create an HTML template to display the lecture management interface for instructors. Use CSS to style the template as desired.
* 2. Use JavaScript, jQuery, and DOM manipulation to handle user interactions, such as creating, editing, and deleting lectures.
* 3. Send appropriate requests to the server based on instructor actions (e.g., creating a lecture, editing a lecture, deleting a lecture).
* Server: Flask + MySQL + Render Template
* 1. Define routes in Flask ("/lectures/create", "/lectures/edit", "/lectures/delete") that correspond to lecture management functionalities for instructors.
* 2. Implement route handler functions for each route in Flask.
* 3. In the route handler functions, perform the necessary actions, such as creating, editing, or deleting lecture records in the database (e.g., MySQL).
* 4. Render a template with appropriate feedback or confirmation messages to be displayed on the client-side.
* Remember to import the required modules and dependencies in your serverc) Course List for Students:
* Client: HTML+ CSS+ JavaScript (jQuery+ DOM)
* The client-side code will include HTML templates and CSS styles to display the course list for students. JavaScript (using jQuery and DOM manipulation) will be used to retrieve the course data from the server and dynamically generate the course list on the client-side. The client will make an asynchronous request (e.g., AJAX) to the server to fetch the course data.
* Server: Node.js + Express + MongoDB + REST API
* The server-side code will handle the request from the client to fetch the course list for students. In the Node.js framework with Express, there will be a route defined (e.g., "/courses") that corresponds to this functionality. The server will query the database (e.g., MongoDB) to retrieve the relevant course information for the logged-in student. The server will then send the data back as a response in JSON format to the client.
* d) Upload List for Instructors:
* Client: HTML+ CSS+ JavaScript (jQuery+ DOM)
* The client-side code will include HTML templates and CSS styles to display the upload list for instructors. JavaScript (using jQuery and DOM manipulation) will be used to retrieve the upload list data from the server and dynamically generate the list on the client-side. The client will make an asynchronous request (e.g., AJAX) to the server to fetch the upload list data.
* Server: Node.js + Express + MongoDB + REST API
* The server-side code will handle the request from the client to fetch the upload list for instructors. In the Node.js framework with Express, there will be a route defined (e.g., "/uploads") that corresponds to this functionality. The server will query the database (e.g., MongoDB) to retrieve the relevant upload information for the logged-in instructor. The server will then send the data back as a response in JSON format to the client.
* e) User Management for Admins:
* Client: HTML+ CSS+ JavaScript (jQuery+ DOM)
* The client-side code will include HTML templates and CSS styles to display the user management interface for admins. JavaScript (using jQuery and DOM manipulation) will be used to handle user interactions, such as creating, editing, and deleting users. The client will send appropriate requests to the server based on user actions.
* Server: Node.js + Express + MongoDB + Template Engine (e.g., EJS)
* The server-side code will handle the requests from the client related to user management for admins. In the Node.js framework with Express, there will be routes defined (e.g., "/users/create", "/users/edit", "/users/delete") that correspond to these functionalities. The server will perform the necessary actions, such as creating, editing, or deleting user records in the database (e.g., MongoDB). The server may also render a template (using an engine like EJS) with appropriate feedback or confirmation messages to be displayed on the client-side.
* f) Course Management for Instructors:
* Client: HTML+ CSS+ JavaScript (jQuery+ DOM)
* The client-side code will include HTML templates and CSS styles to display the course management interface for instructors. JavaScript (using jQuery and DOM manipulation) will be used to handle user interactions, such as creating, editing, and deleting courses. The client will send appropriate requests to the server based on instructor actions.
* Server: Node.js + Express + MongoDB + Template Engine (e.g., EJS)
* The server-side code will handle the requests from the client related to course management for instructors. In the Node.js framework with Express, there will be routes defined (e.g., "/courses/create", "/courses/edit", "/courses/delete") that correspond to these functionalities. The server will perform the necessary actions, such as creating, editing, or deleting course records in the database (e.g., MongoDB). The server may also render a template (using an engine like EJS) with appropriate feedback or confirmation messages to be displayed on the client-side.
* g) Lecture Management for Instructors:
* Client: HTML+ CSS+ JavaScript (jQuery+ DOM)
* The client-side code will include HTML templates and CSS styles to display the lecture management interface for instructors. JavaScript (using jQuery and DOM manipulation) will be used to handle user interactions, such as creating, editing, and deleting lectures. The client will send appropriate requests to the server based on instructor actions.
* Server: Node.js + Express + MongoDB + Template Engine (e.g., EJS)
* The server-side code will handle the requests from the client related to lecture management for instructors. In the Node.js framework with Express, there will be routes defined (e.g., "/lectures/create", "/lectures/edit", "/lectures/delete") that correspond to these functionalities. The server will perform the necessary actions, such as creating, editing, or deleting lecture records in the database (e.g., MongoDB). The server may also render a template (using an engine like EJS) with appropriate feedback or confirmation messages to be displayed on the client-side.

# Conclusion:

The main points covered in the report are as follows:

1. The development of a web application with different functionalities for students, instructors, and admins.

2. The functionalities include course list for students, upload list for instructors, user management for admins, course management for instructors, and lecture management for instructors.

3. The client-side code involves HTML, CSS, and JavaScript (jQuery and DOM manipulation) to create interactive interfaces.

4. The server-side code is implemented using Node.js, Express, and MongoDB, along with REST APIs and a template engine (e.g., EJS) for rendering dynamic content.

5. The system utilizes a database (e.g., MySQL or MongoDB) to store and retrieve data related to courses, uploads, users, and lectures.

In terms of meeting project objectives, it is important to evaluate if the implemented functionalities align with the initial requirements and if they are working as intended. The success of the implementation can be evaluated based on the following criteria:

1. Functionality: Assess whether the web application successfully provides the specified functionalities for students, instructors, and admins. Verify if users can perform actions such as accessing course lists, managing uploads, managing users, creating/editing/deleting courses, and creating/editing/deleting lectures.

2. User Experience: Evaluate the overall user experience of the web application. Consider factors such as ease of use, responsiveness, and intuitive design. Gather feedback from users to gauge their satisfaction with the interface and functionality.

3. Performance: Assess the performance of the web application in terms of speed and responsiveness. Ensure that the application can handle concurrent requests and data retrieval efficiently. Evaluate if there are any bottlenecks or performance issues during heavy usage.

4. Security: Verify that appropriate security measures are implemented to protect user data and prevent unauthorized access. Ensure that user authentication and authorization mechanisms are in place and that sensitive information is securely stored.

Challenges encountered during the development process may include:

1. Data modeling: Designing the database schema and establishing relationships between different entities can be challenging, especially when dealing with complex data structures.

2. Asynchronous communication: Implementing asynchronous communication between the client and server using AJAX or other techniques can require careful handling to ensure data consistency and error handling.

3. User management: Managing user roles, permissions, and access control can be complex and challenging to implement correctly. Ensuring proper authentication and authorization mechanisms are in place is crucial.

4. Performance optimization: Optimizing the performance of the web application, especially when dealing with large datasets or handling concurrent requests, can be a challenging task.

Lessons learned from the development process can include:

1. Proper planning: Thoroughly defining the requirements and creating a detailed plan before starting development can help in avoiding pitfalls and streamlining the implementation process.

2. Modular code structure: Breaking down the code into modular components and following best practices for code organization and maintainability can make the development and debugging process easier.

3. Continuous testing and debugging: Implementing a robust testing strategy, including unit tests and end-to-end tests, can help identify and fix issues early in the development process.

4. Regular feedback and iteration: Seeking feedback from users and stakeholders throughout the development process can help identify areas for improvement and ensure that the final product meets their expectations.

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

1. Integration with additional systems: Integrating the web application with other systems or platforms, such as a learning management system or external APIs, can enhance its functionality and provide additional features.

2. Enhanced user interface: Improving the user interface design and incorporating modern design trends can enhance the overall user experience and make the application more visually appealing.

3. Advanced analytics and reporting: Adding analytics capabilities to track user engagement, generate insights, and provide detailed reports can help administrators and instructors make data-driven decisions.

4. Real-time collaboration: Implementing real-time collaboration features, such as live chat or collaborative document editing, can facilitate communication and interaction among users within the web application.

5. Mobile responsiveness: Optimizing the web application for mobile devices and ensuring a seamless user experience across different screen sizes can expand its accessibility and reach.

6. 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 how well the project objectives were met, the usability and performance of the web application, and the satisfaction of the end-users. Regular feedback, testing, and continuous improvement are key to ensuring the web application's success and meeting the evolving needs of the users.

\* Summarize the main points covered in the report, discuss whether the project objectives were met and evaluate the success of the implementation. Reflect on any challenges encountered during the development process and discuss lessons learned. Additionally, consider the potential for future improvements or enhancements to the web application.

# References: