Software Engineering Program

Sino-British Collaborative Education

CDUT

Learning Management System Web Site

- Preliminary Report -

Student ID: 202118010418
Date: 28th April 2024
Group: L5C-4

Table of Contents

1. Introduction:

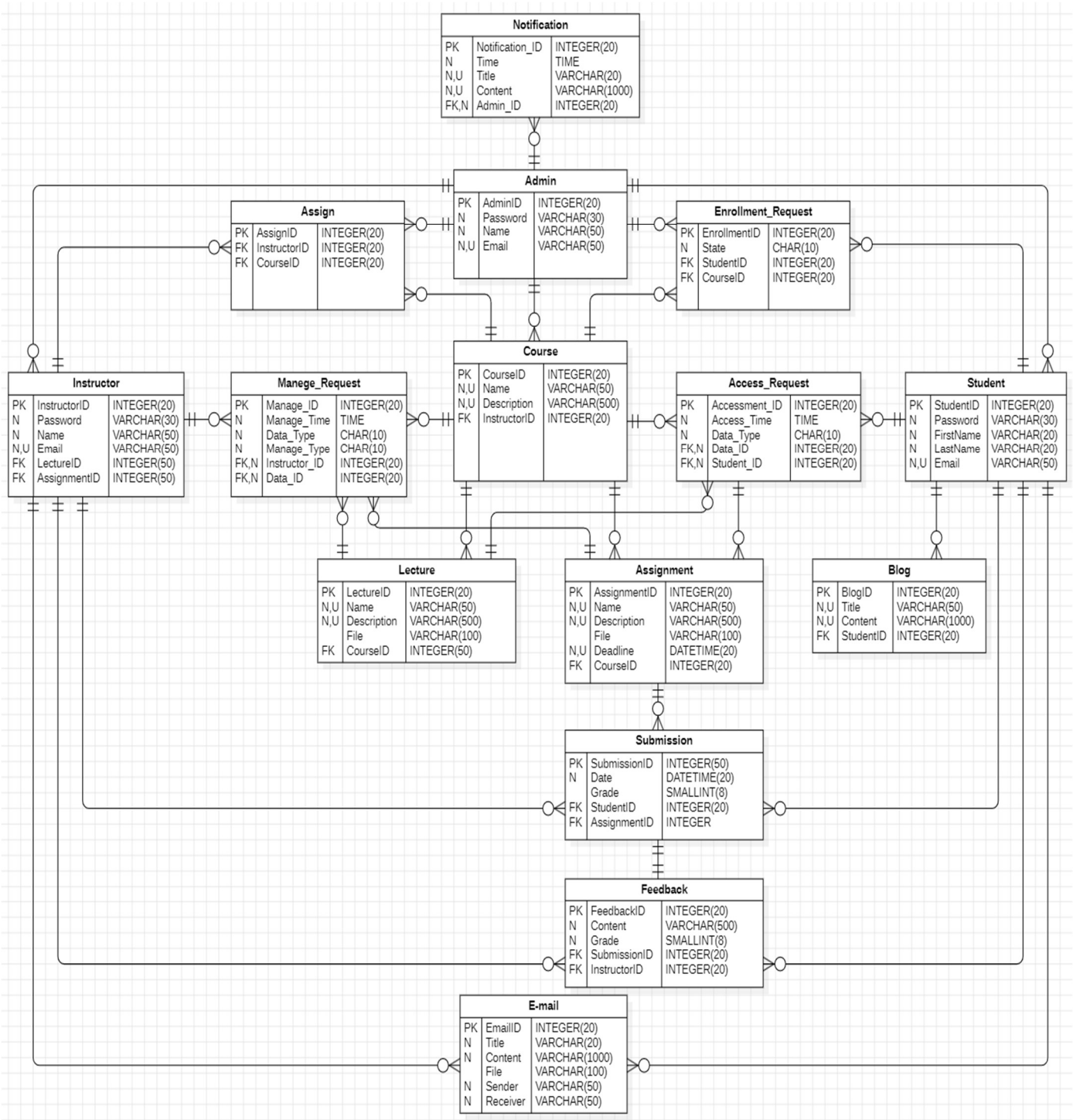
As an initial report for Web application development (learning management system Web application), this report includes an introduction to the report, database ER diagram, UI interface wireframes, and functional division and interaction mode of the front and back end.

The target users of the Learning Management System web application are administrators, teachers and students of educational institutions, providing them with various auxiliary functions required in their daily learning work, with the purpose of facilitating the management, delivery and tracking of educational courses, and ultimately improving the learning efficiency of students. The web application focuses on efficiency, accessibility, and scalability, making it easier for administrators, teachers, and students to interact and access important information. Teachers can upload lectures, submit assessment and grade student work; Students can register, view course content, submit assignments, track progress, and participate in discussion forums; Administrators are responsible for creating and managing accounts for teachers, students, courses, and other administrators.

In today's digital age, traditional methods of managing educational institutions are no longer sufficient. With the increasing number of students and the increasing number of teachers, universities need a strong and effective system to manage their operations. Education management systems provide a comprehensive solution to this problem, providing a series of functions that meet the needs of modern educational institutions [1]. It aims to improve the efficiency and accessibility of educational institutions to meet the growing management needs of modern educational institutions.

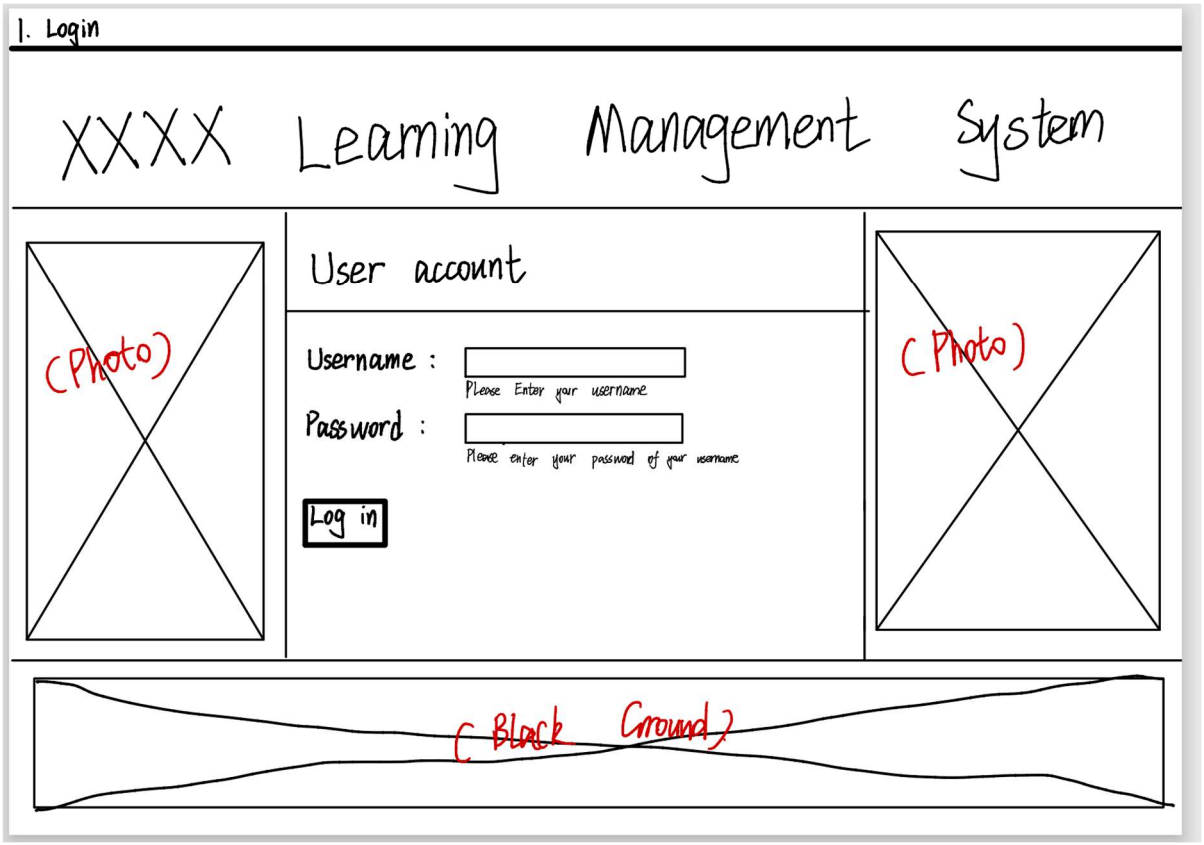
The technology required to realize the construction of learning management system Web application is three front-end technologies, that is, the client: HTML, CSS, JavaScript and the jQuery library inside. These are the three most classic and basic front-end implementation techniques. The back-end, that is, the server side, is constructed with python and flask frameworks, and finally, MySQL database technology is used to build the website database [2]. Relying on these technologies can meet the minimum requirements of web development.

2. Database design:



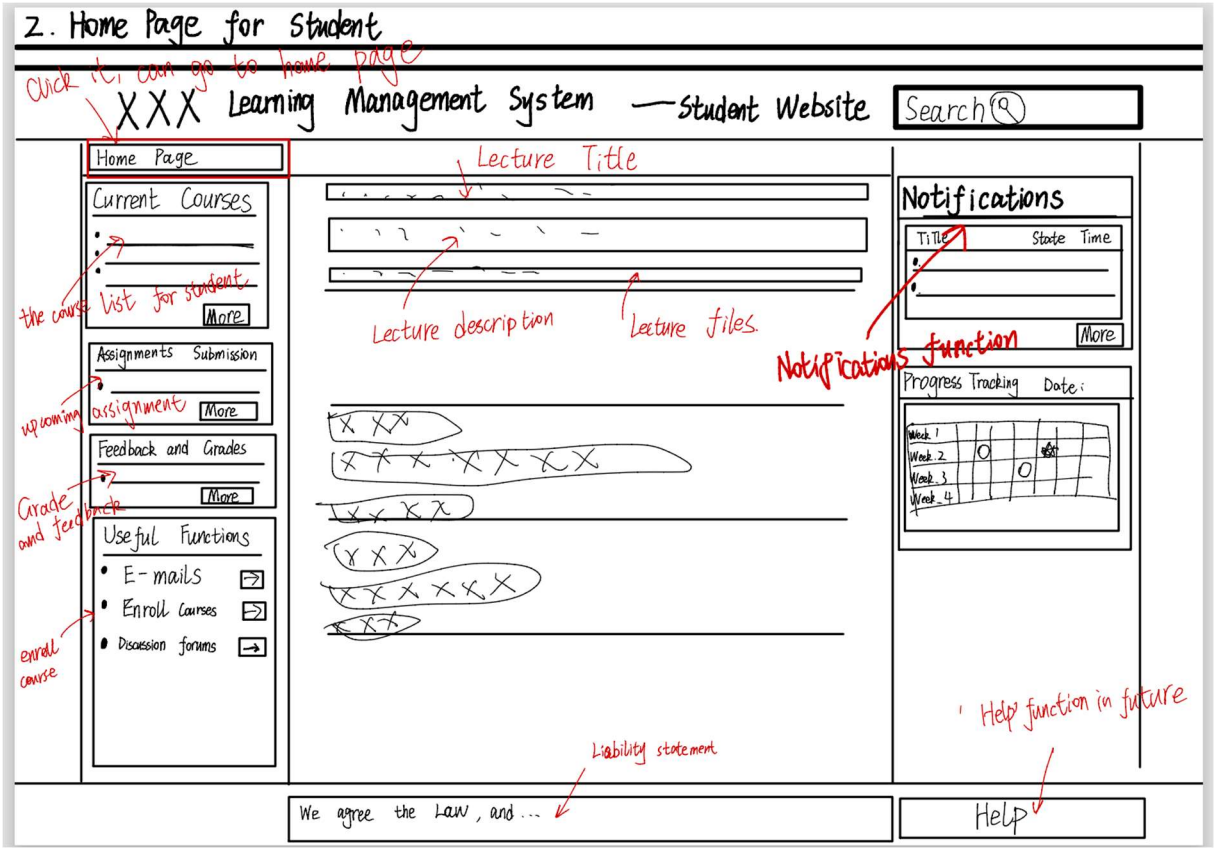
3. Wireframes:

a) Login

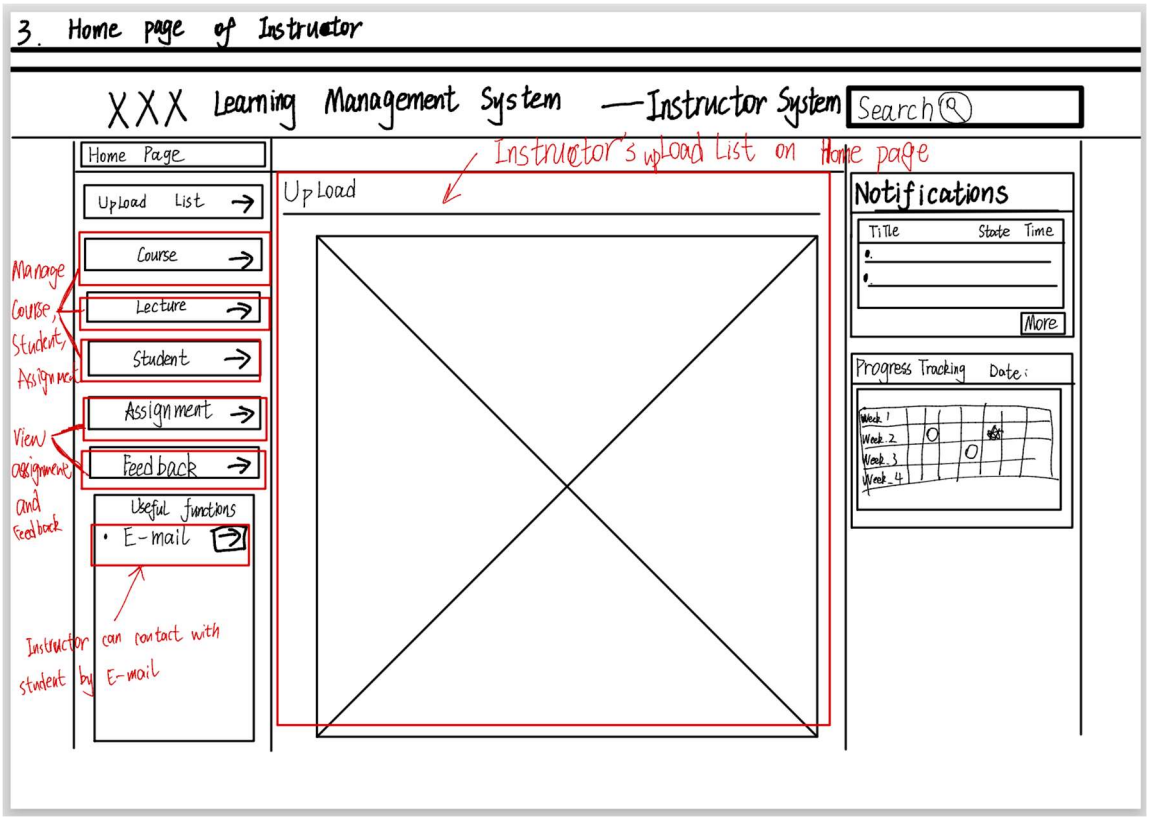


b) Home Page

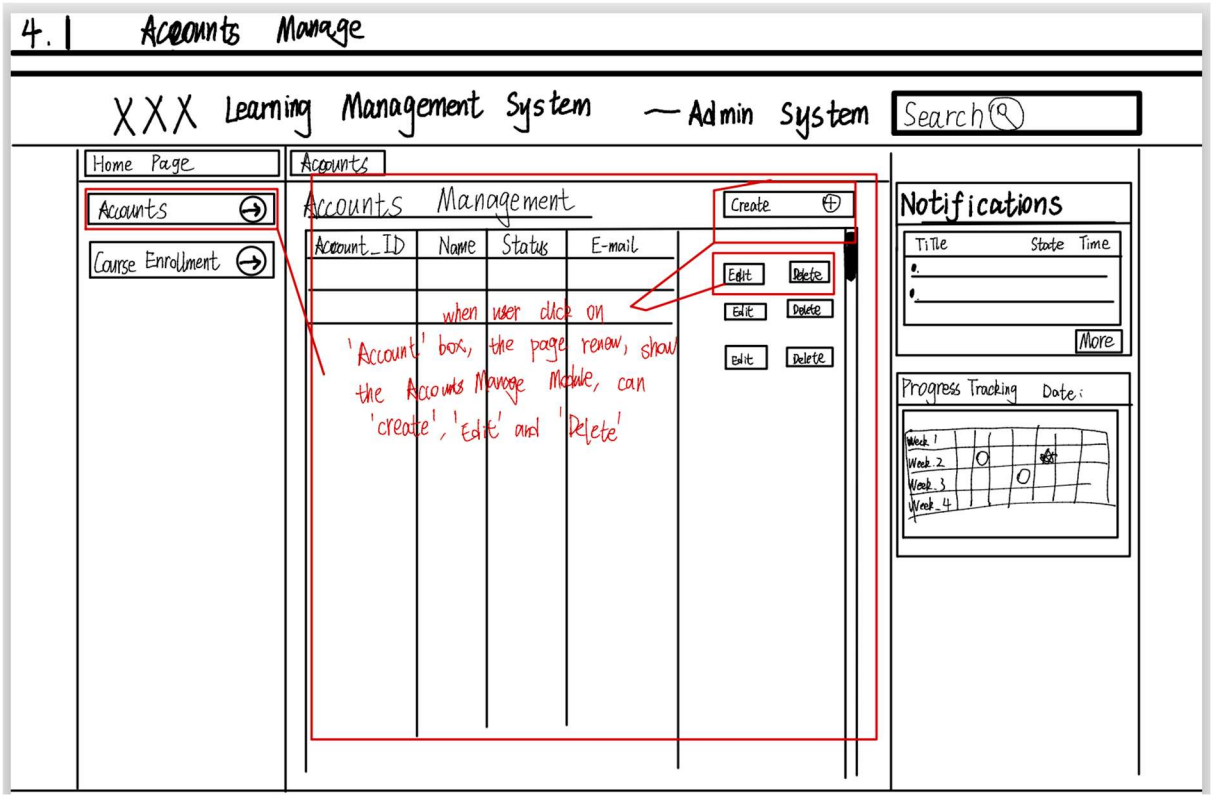
- Student



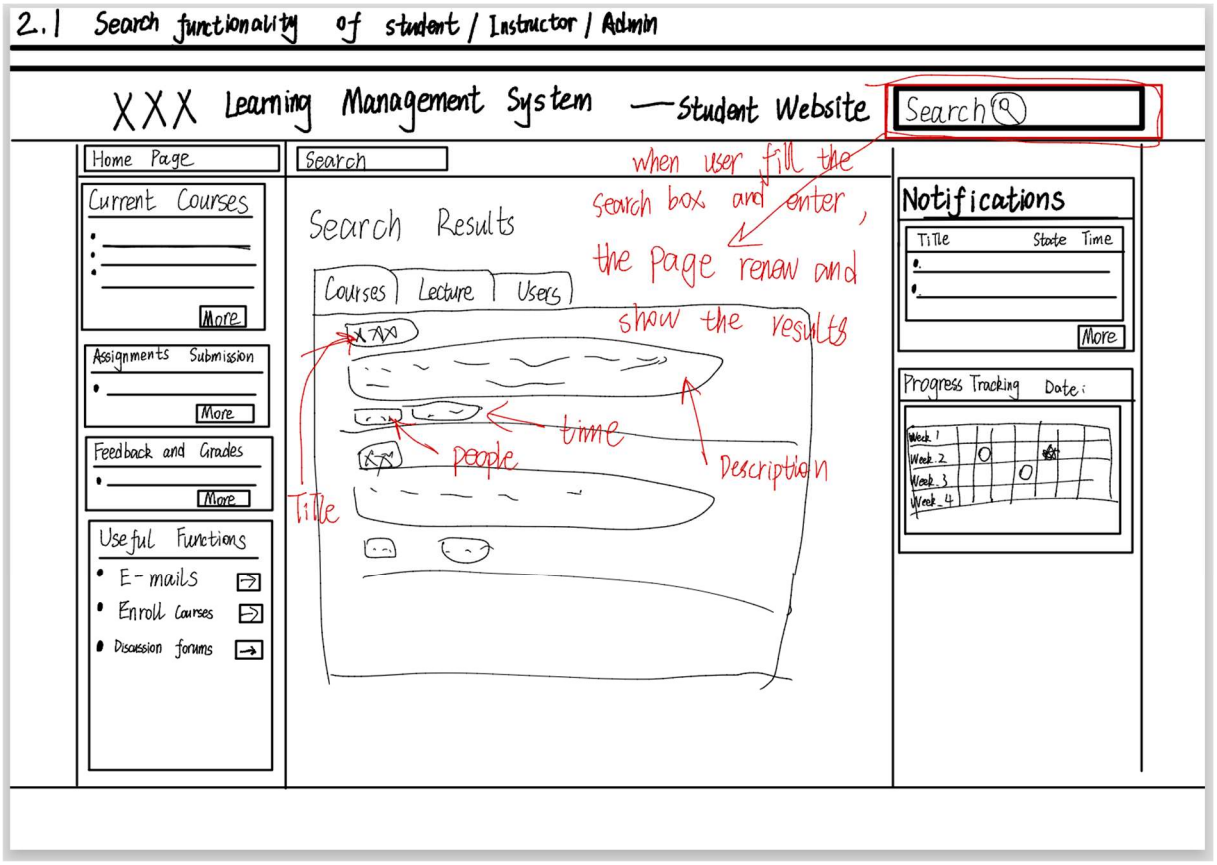
- Instructor



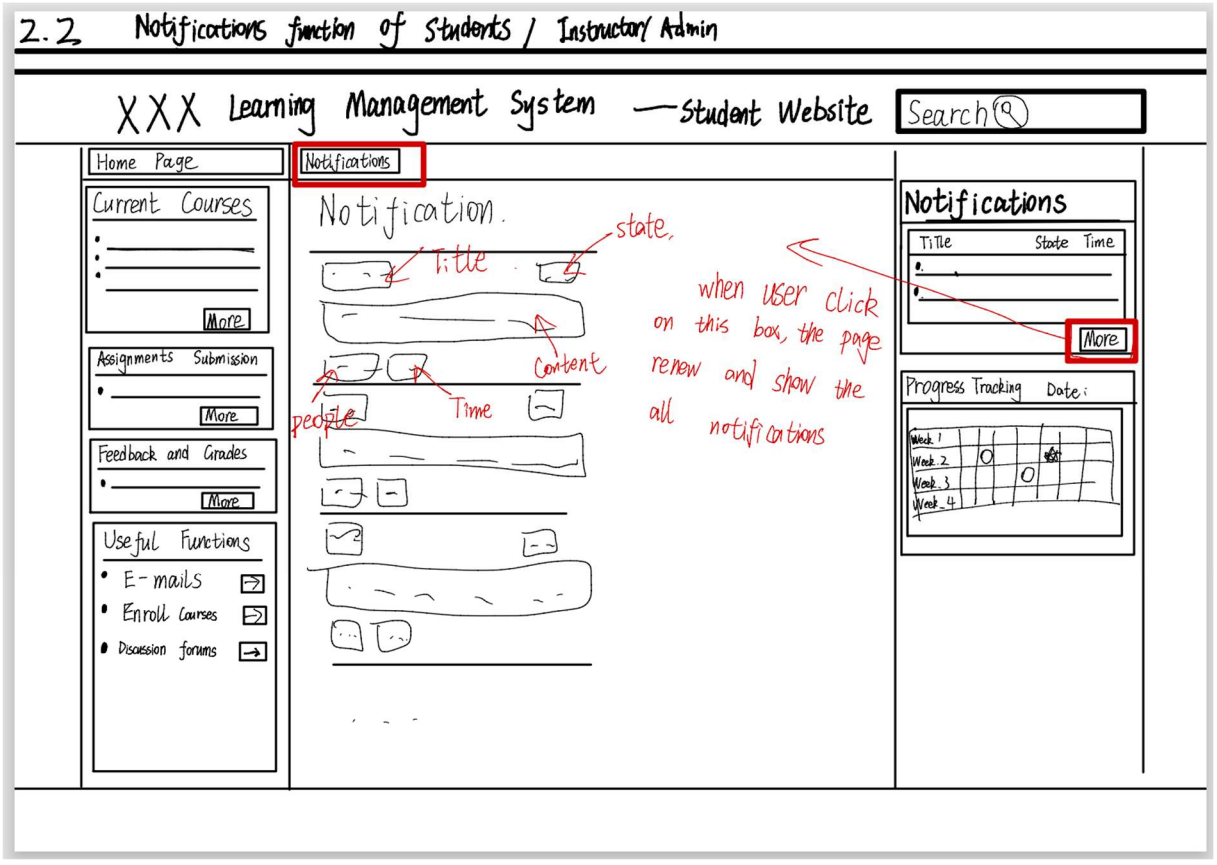
- Admin



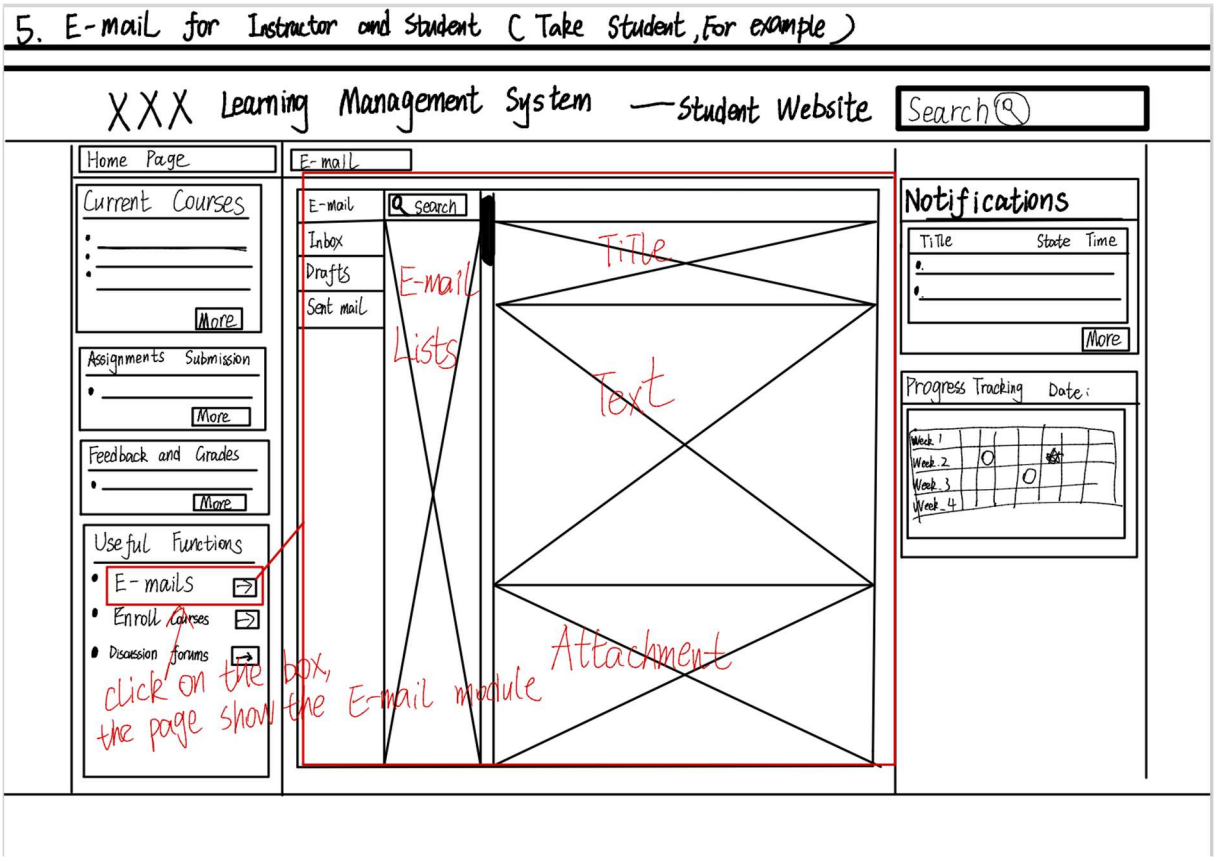
c) Search Function



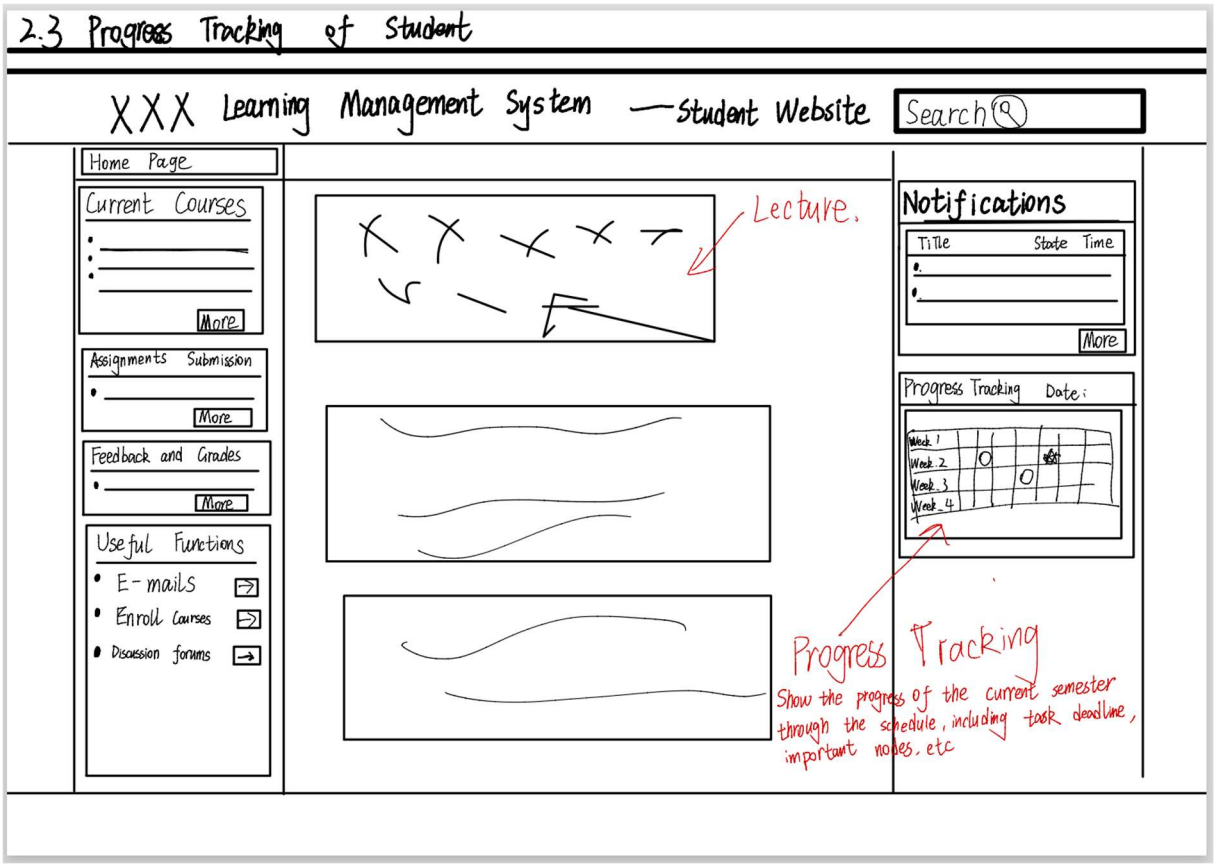
d) Notification Function



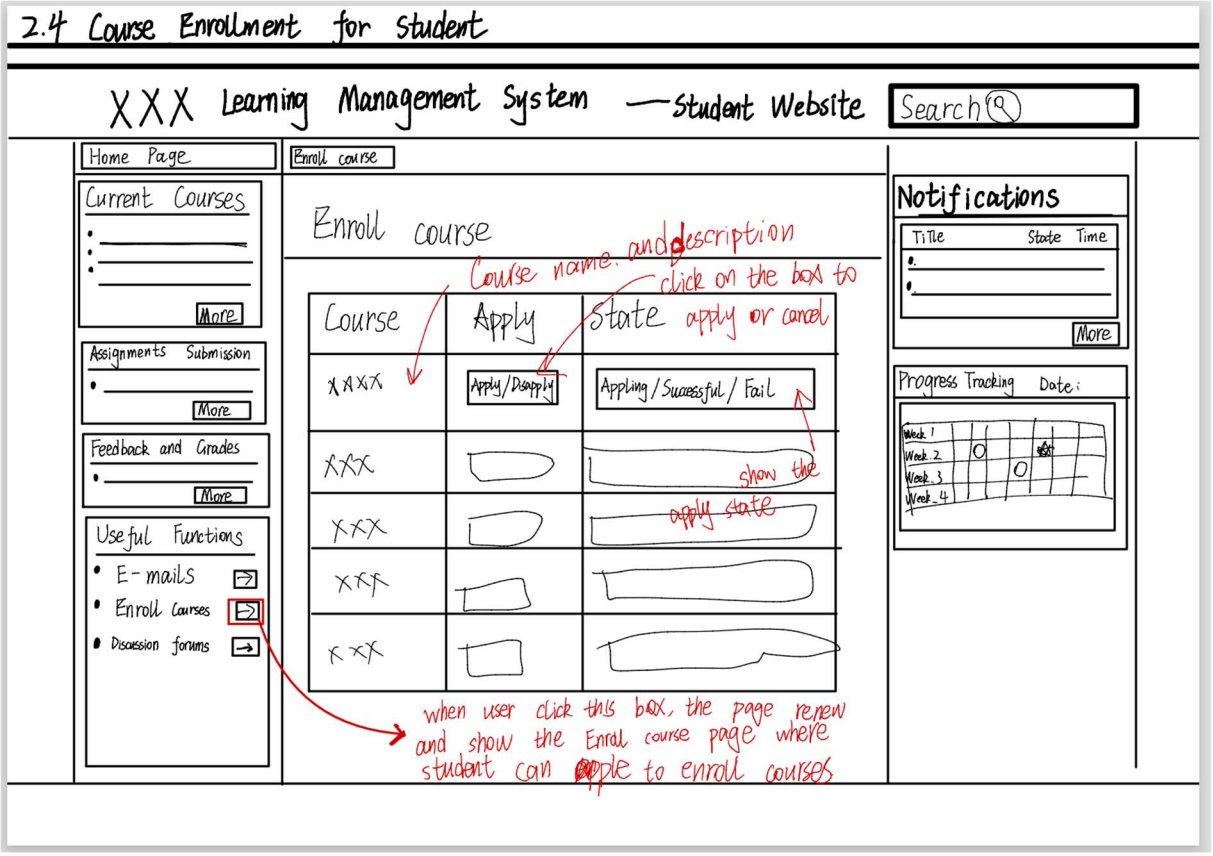
e) Communication Function by E-mail



f) Progress Tracking of Student

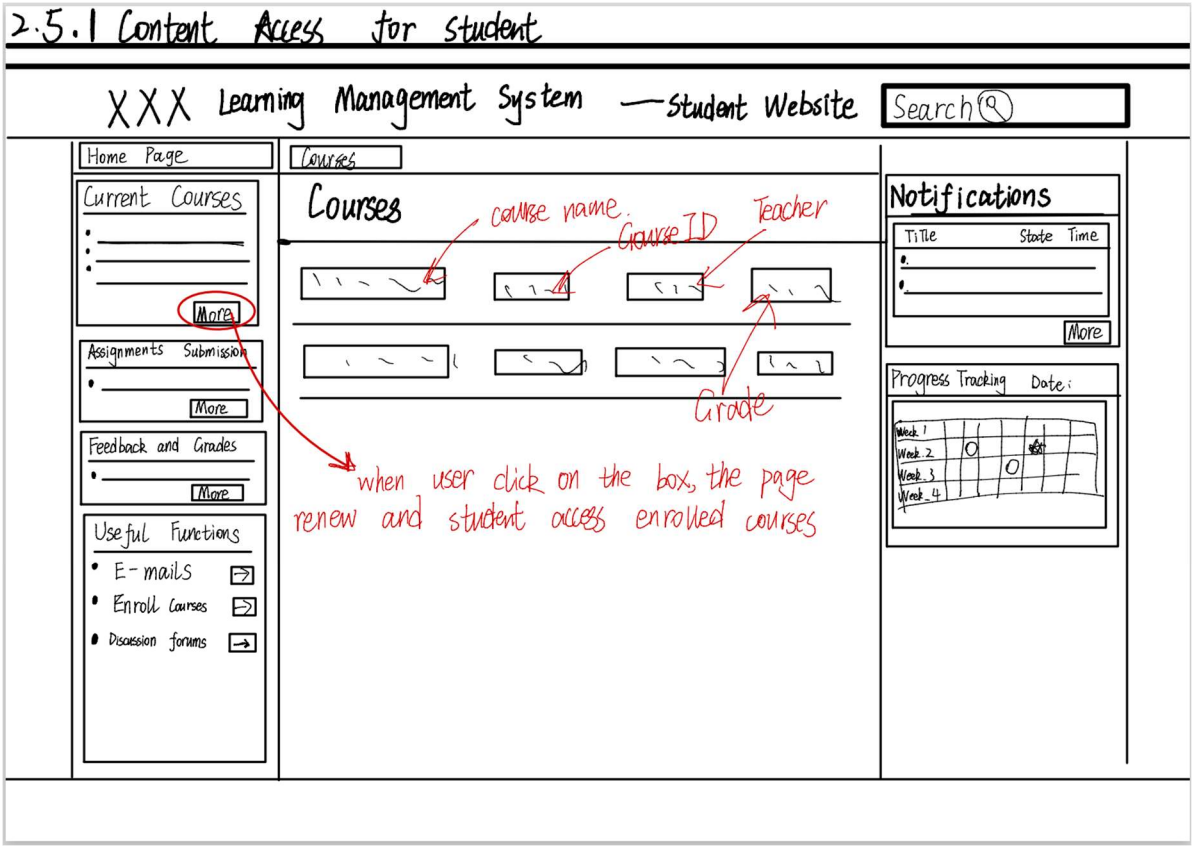


g) Course Enrollment for Student

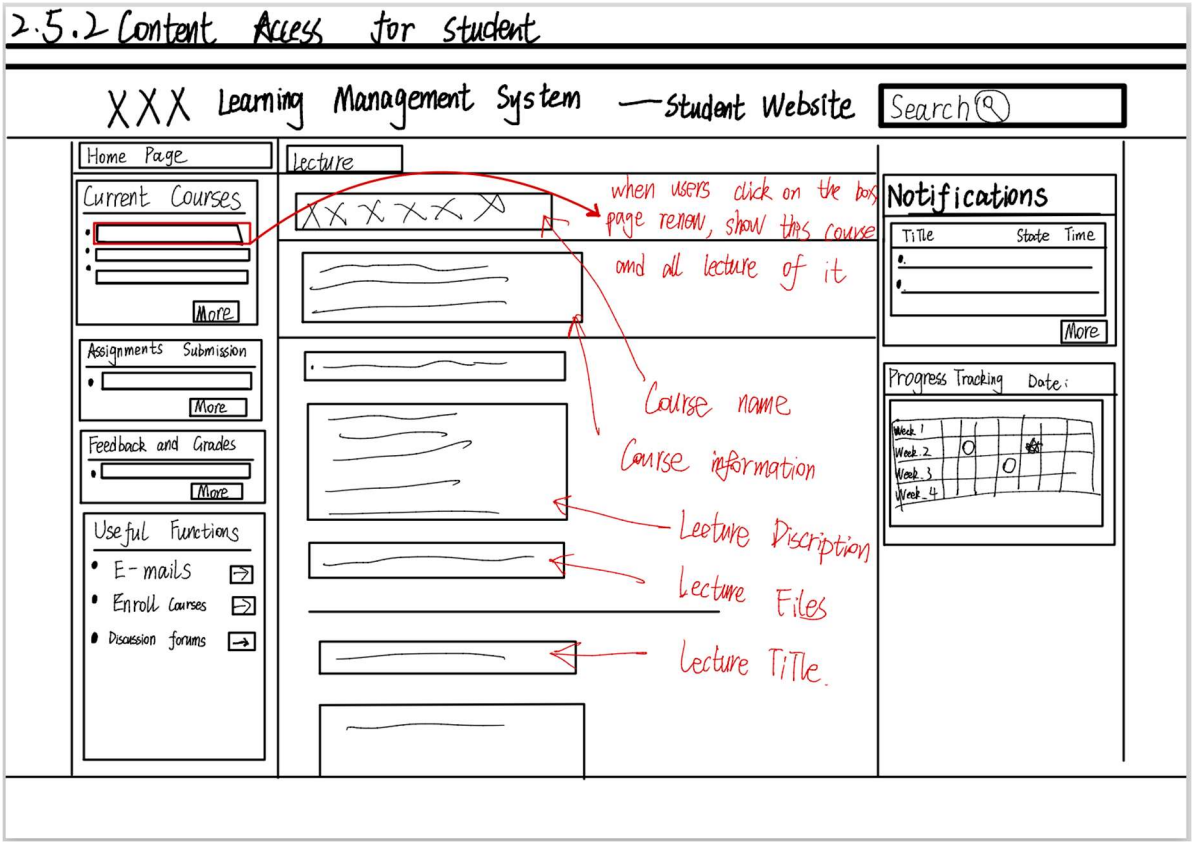


h) Content Access for Student

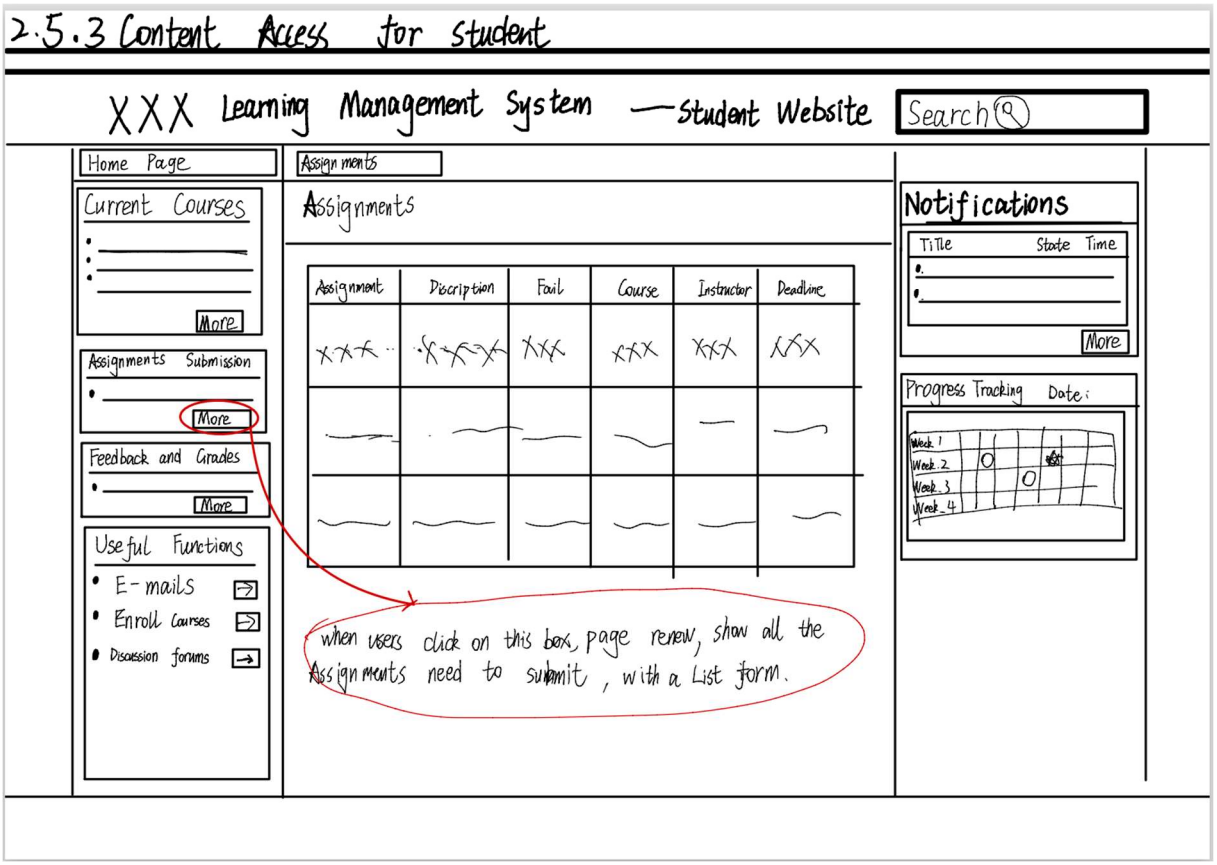
- Course Access



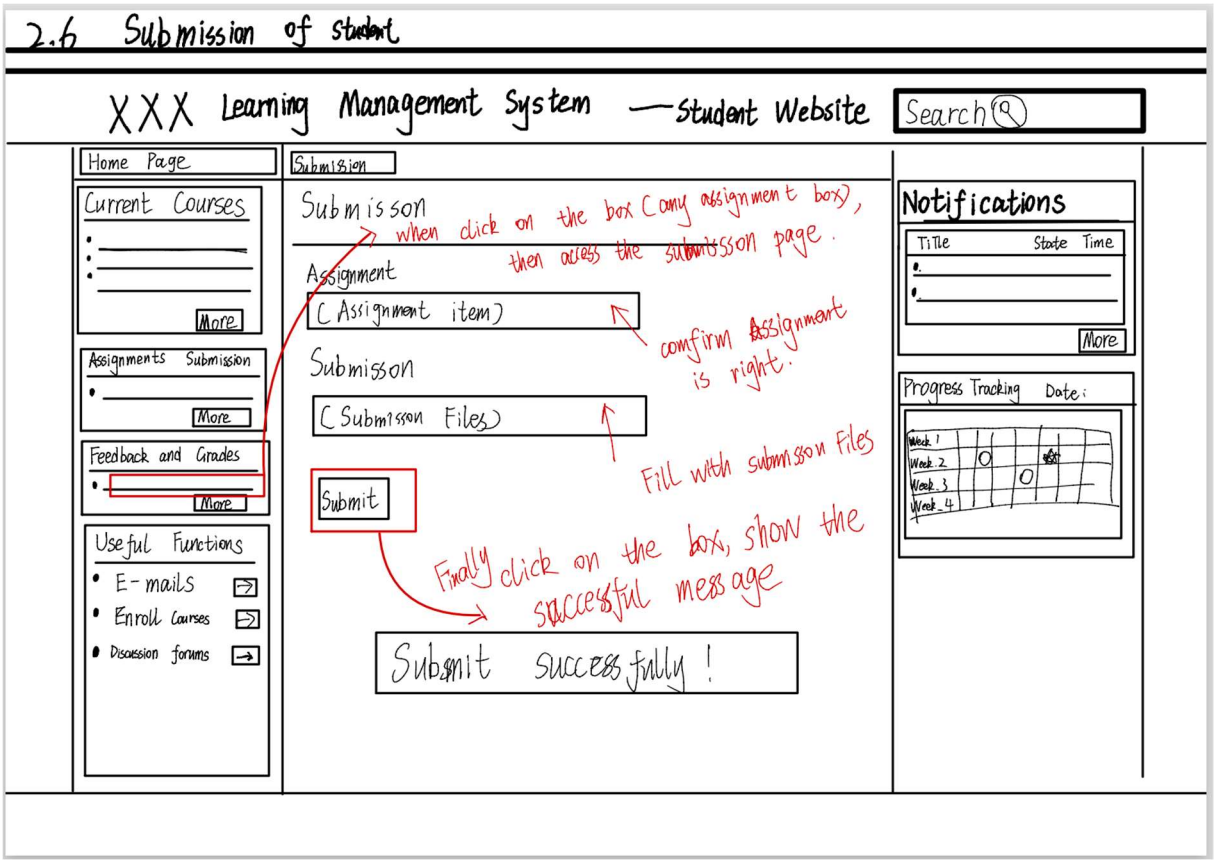
- Lecture Access



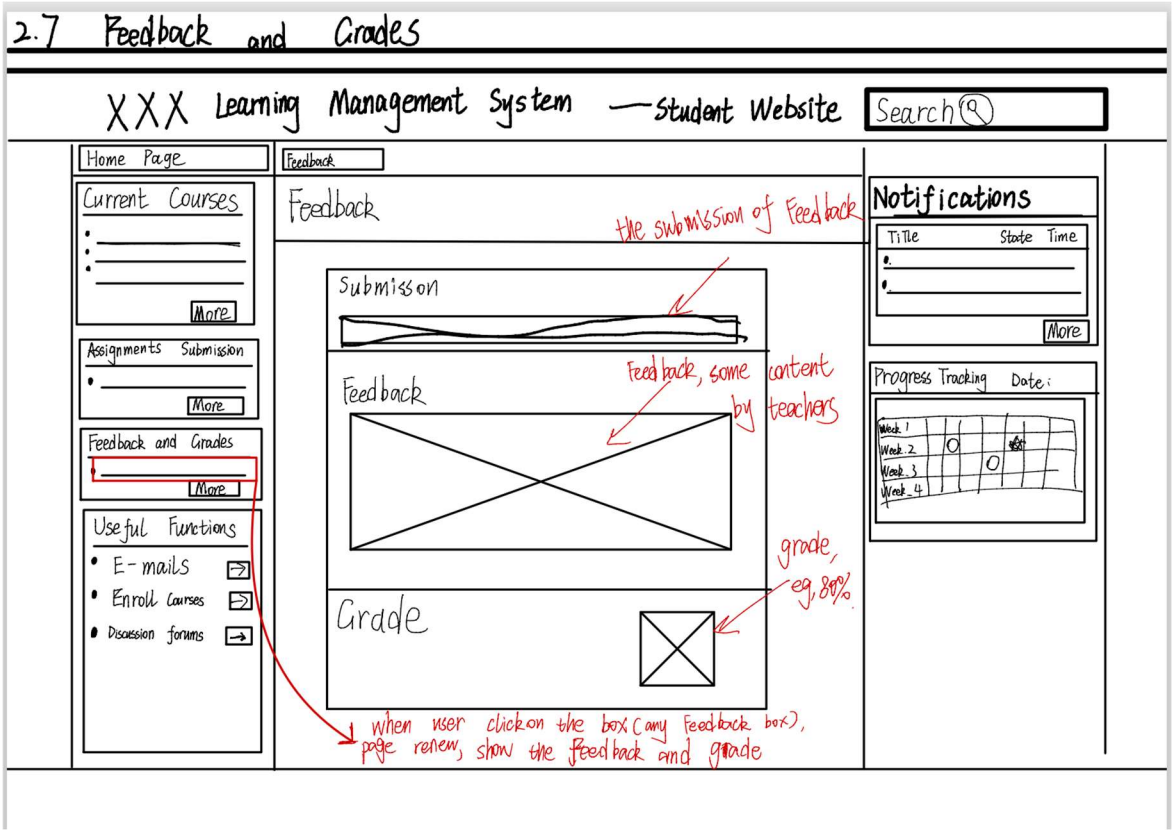
- Assignment Access



i) Submission for Student

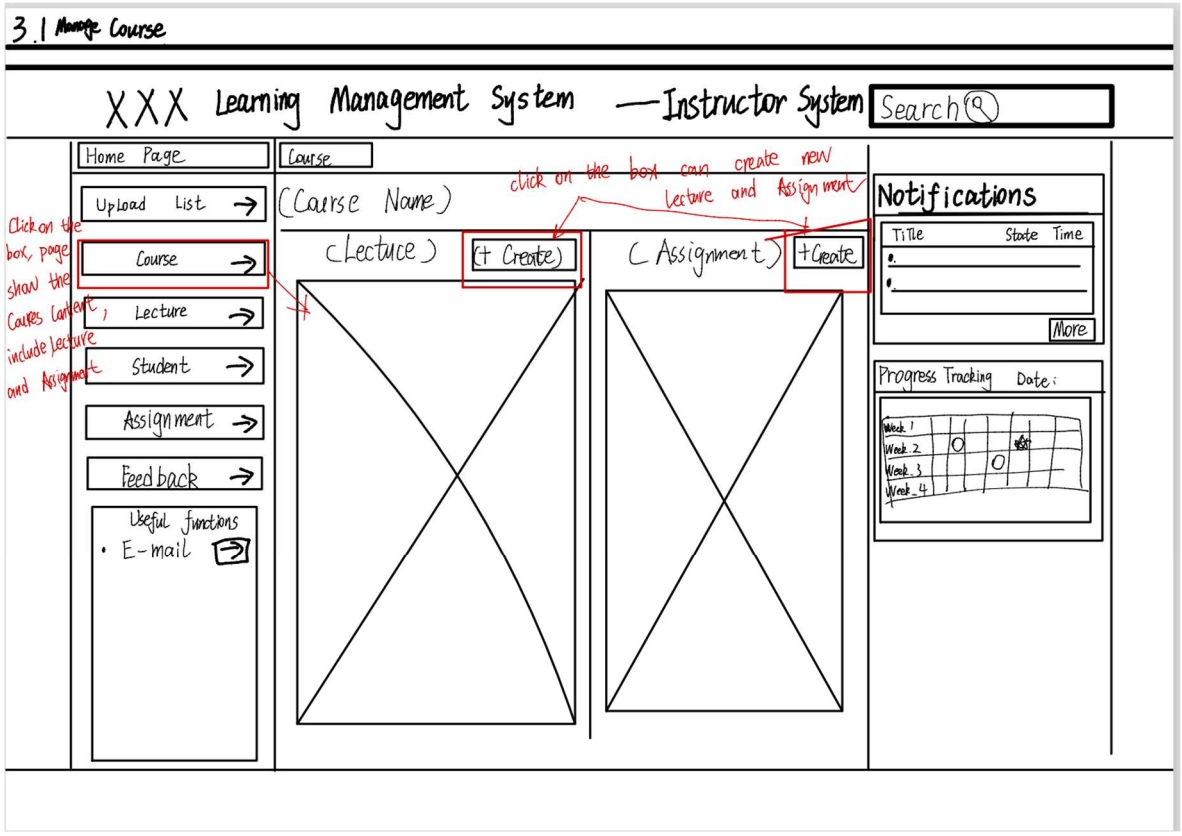


j) Feedback and Grades for Student

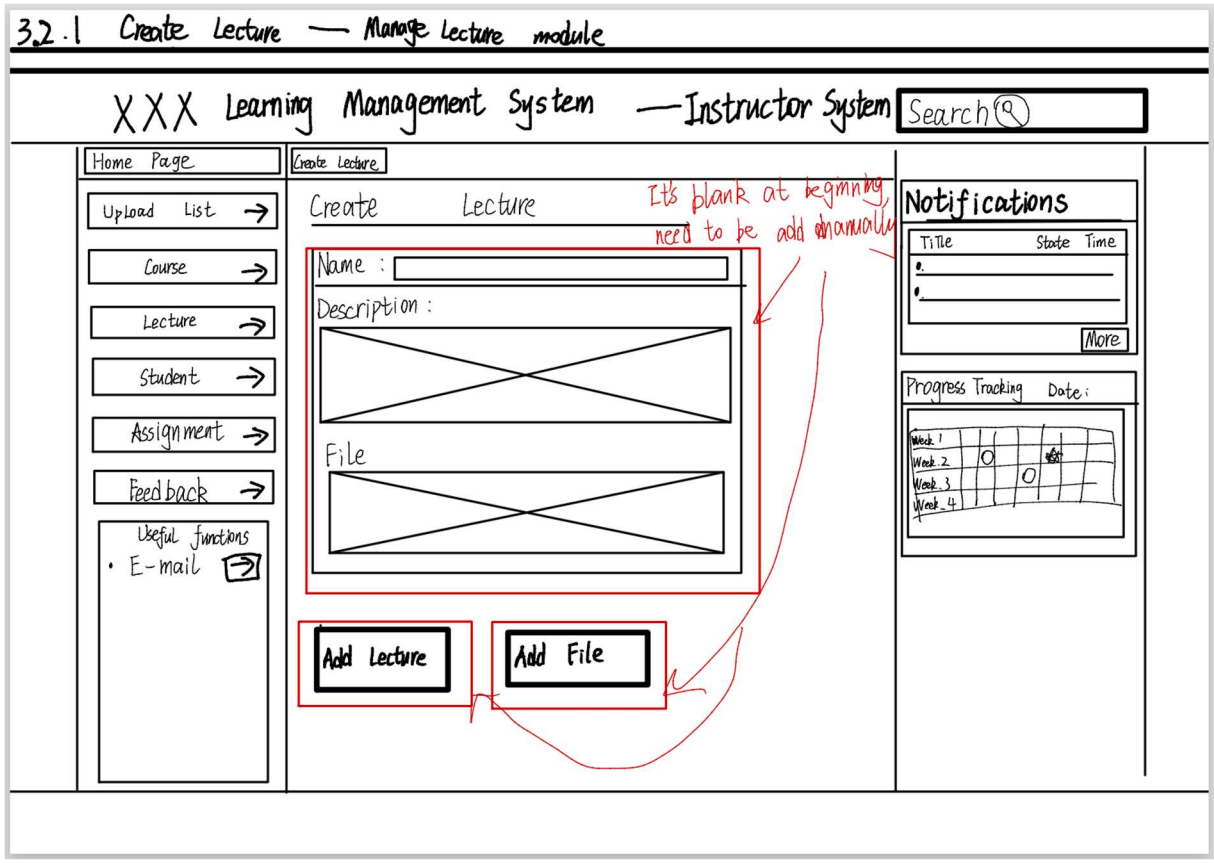


k) Manage Course for Instructor

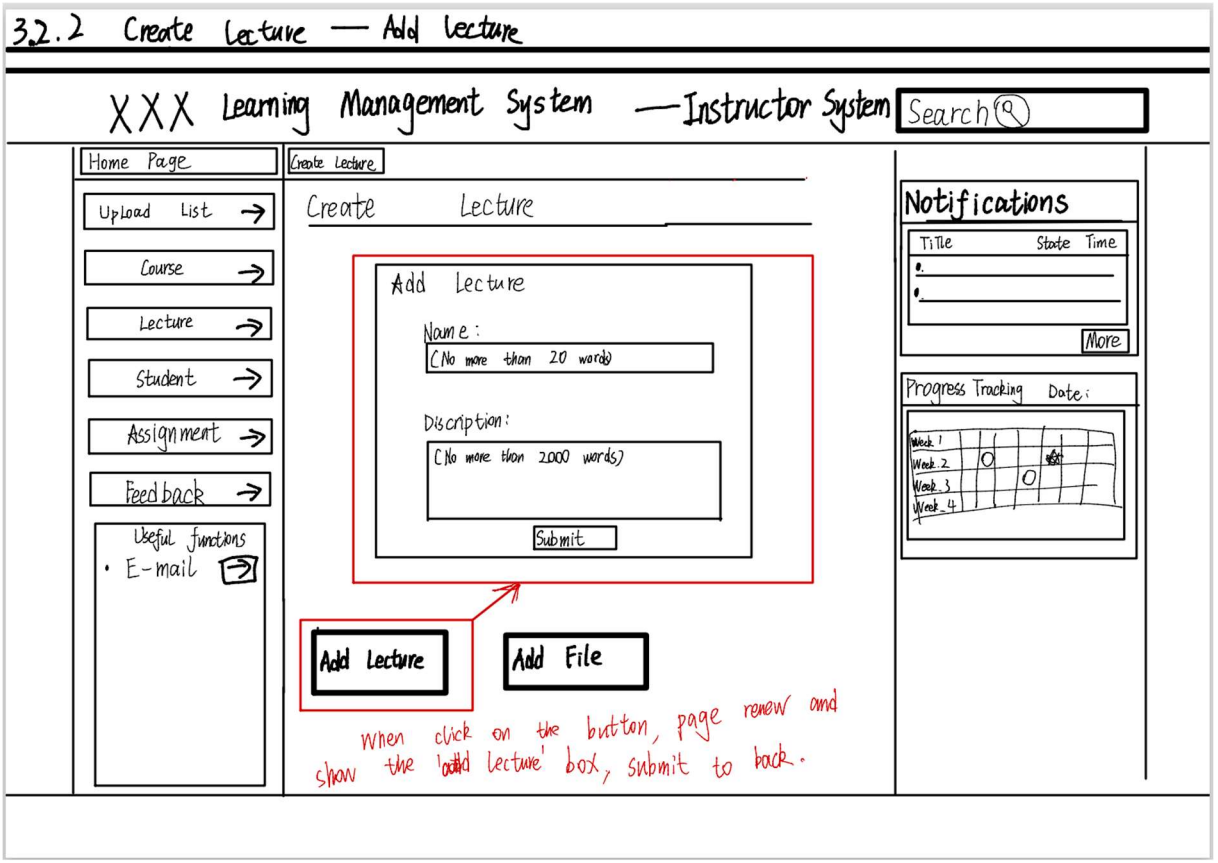
- Access Course Management Module



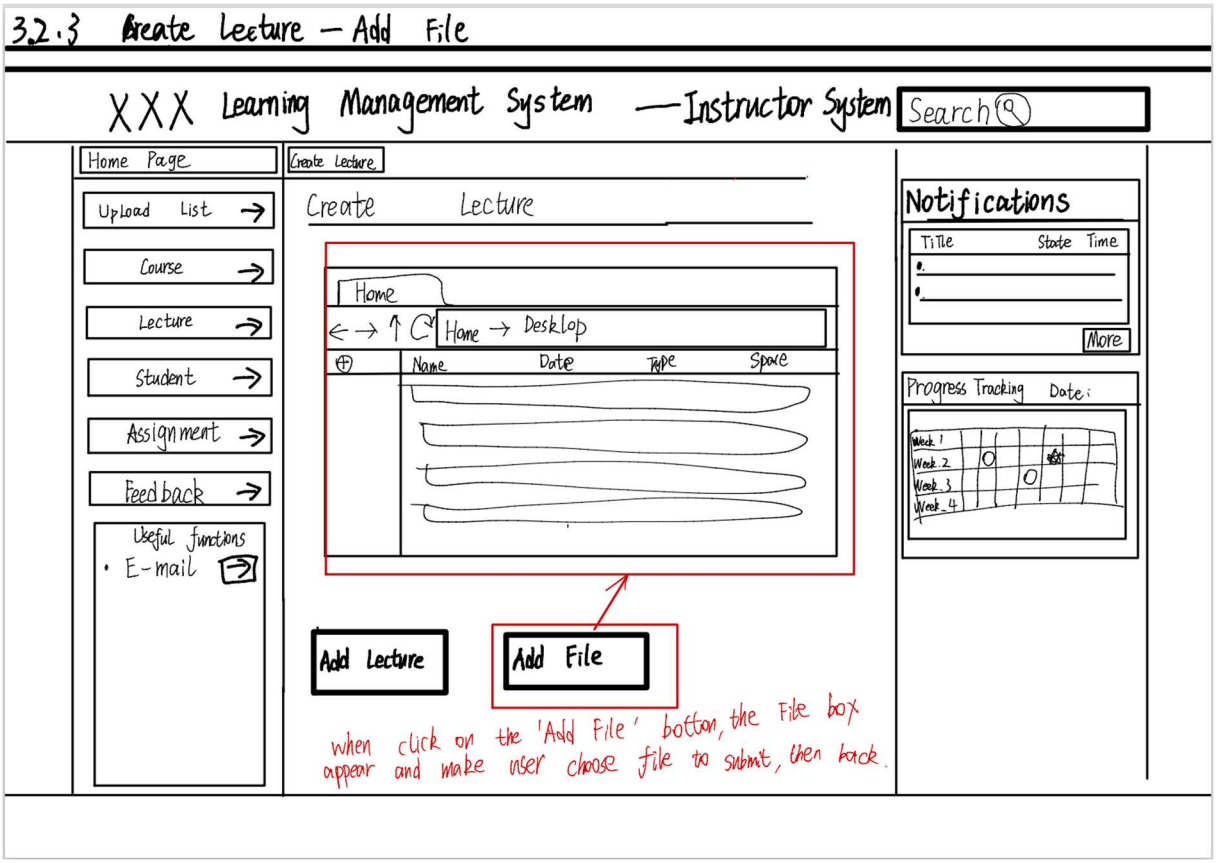
- Create Lecture _Manage Lecture Module



- Create Lecture _Add Lecture

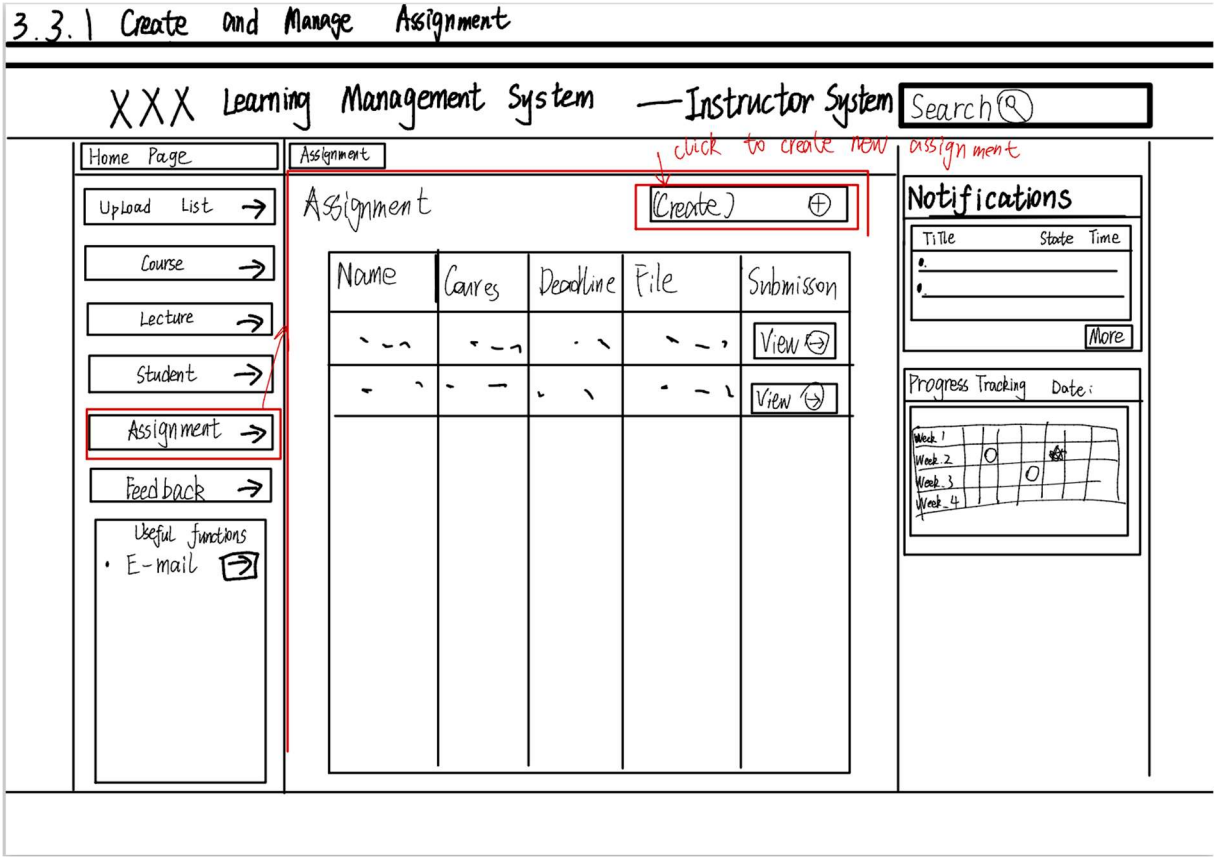


- Create Lecture _Add File

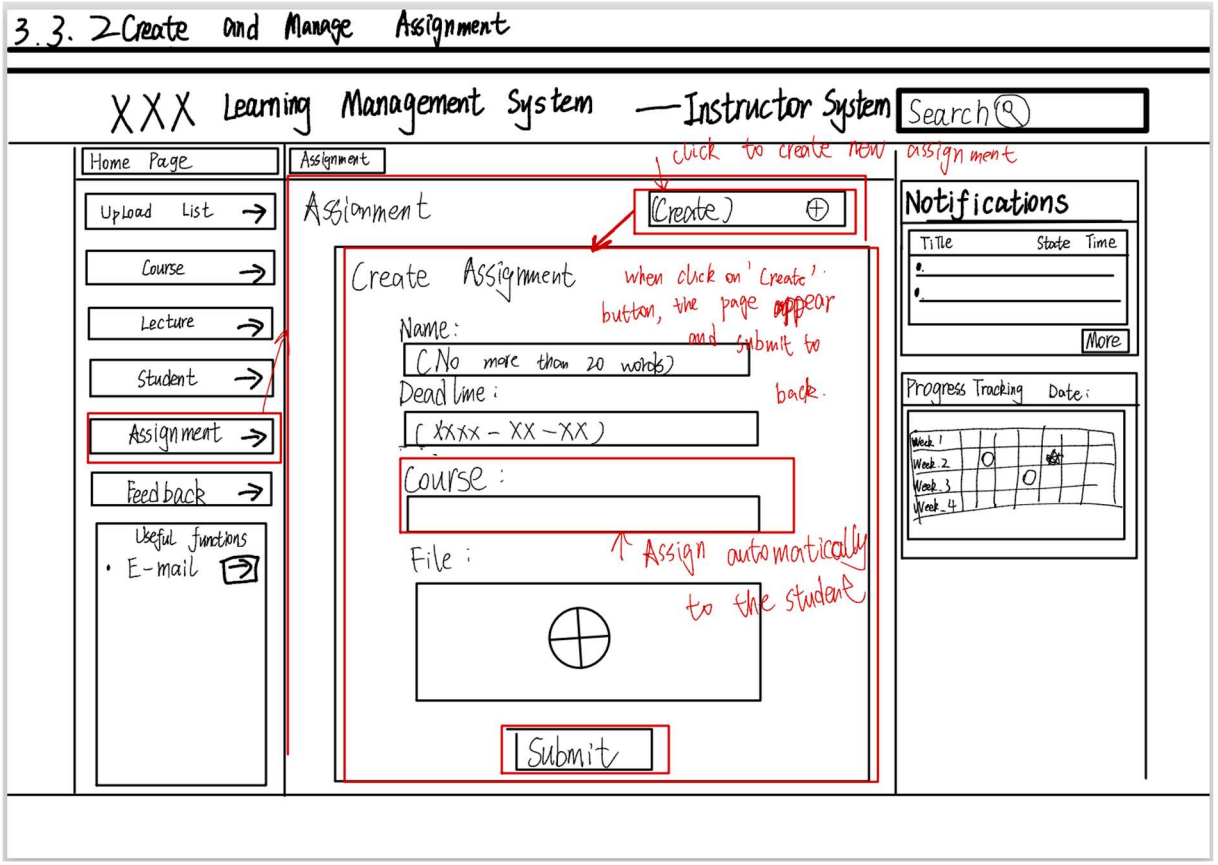


l) Assignment Management for Instructor

- Access Assignment Module

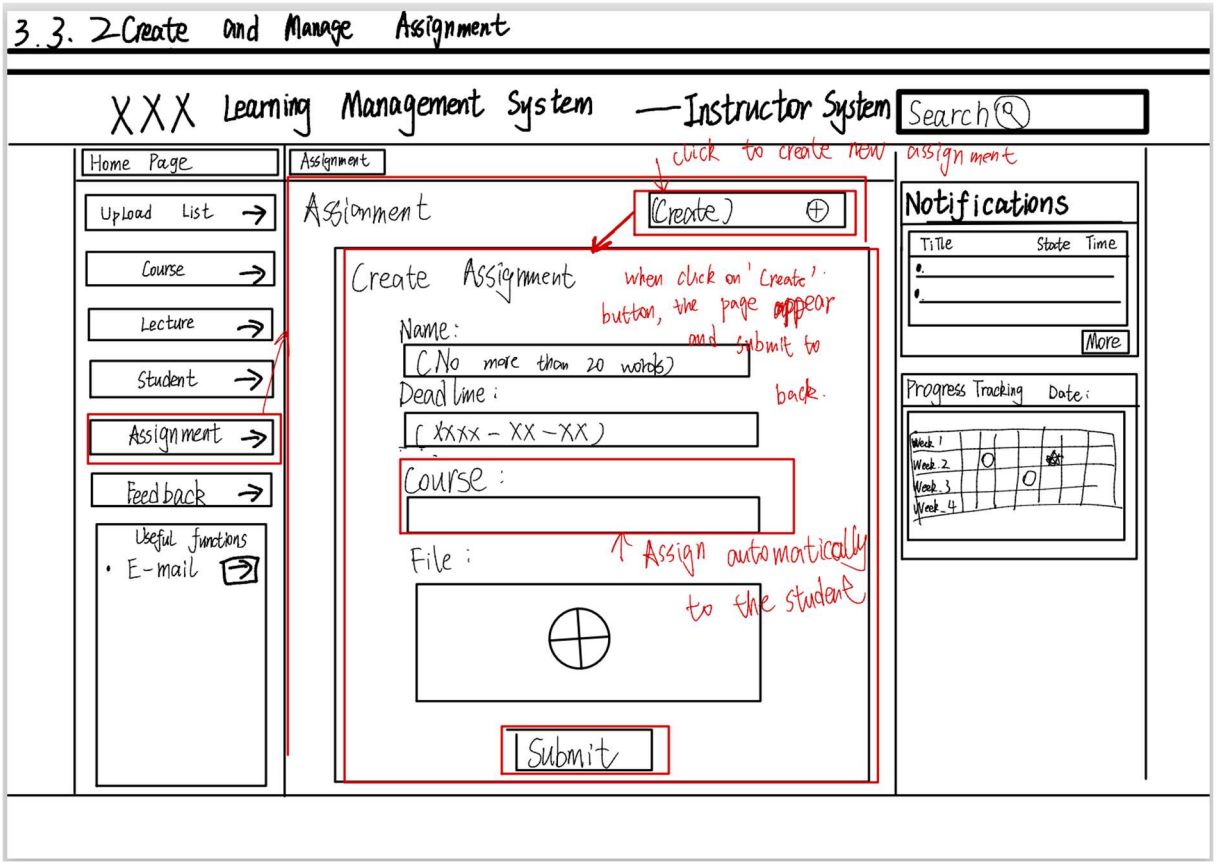


- Create Assignment

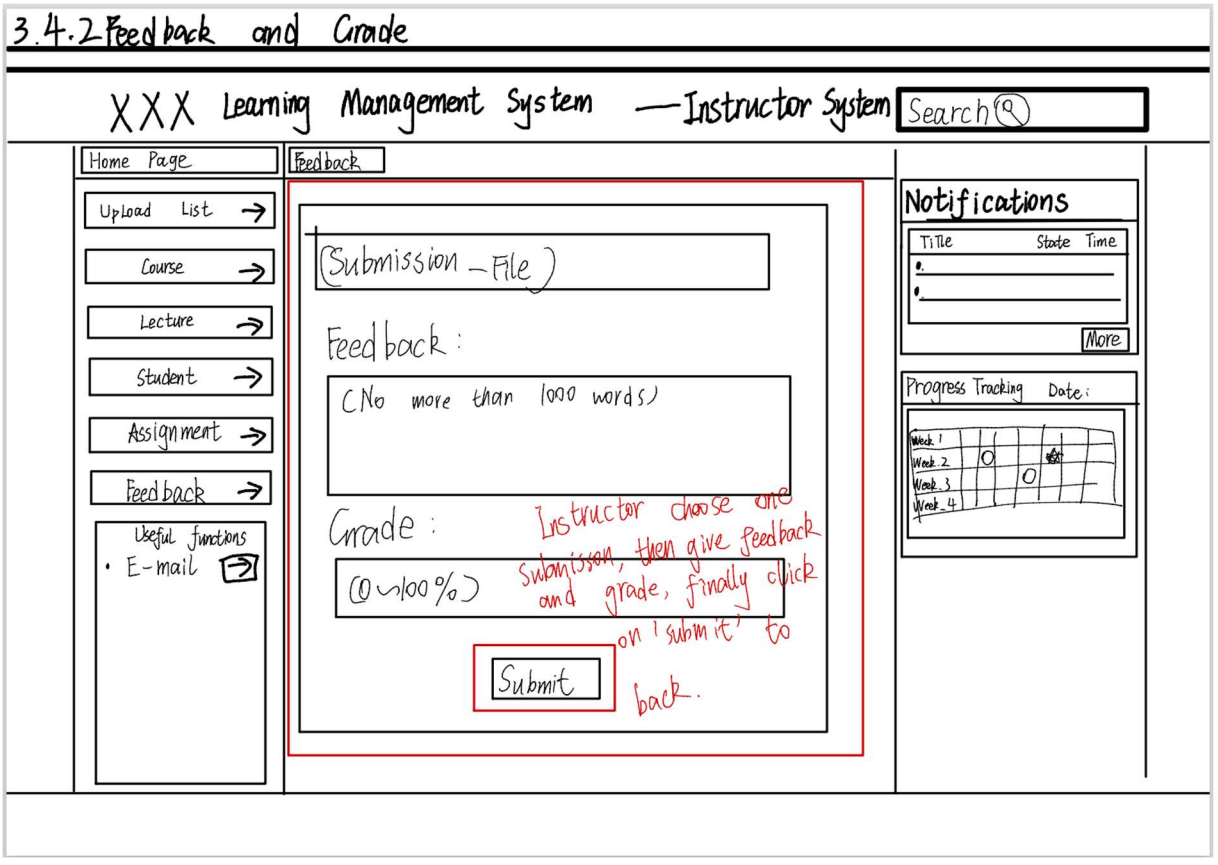


m) Grading and Feedback for Instructor

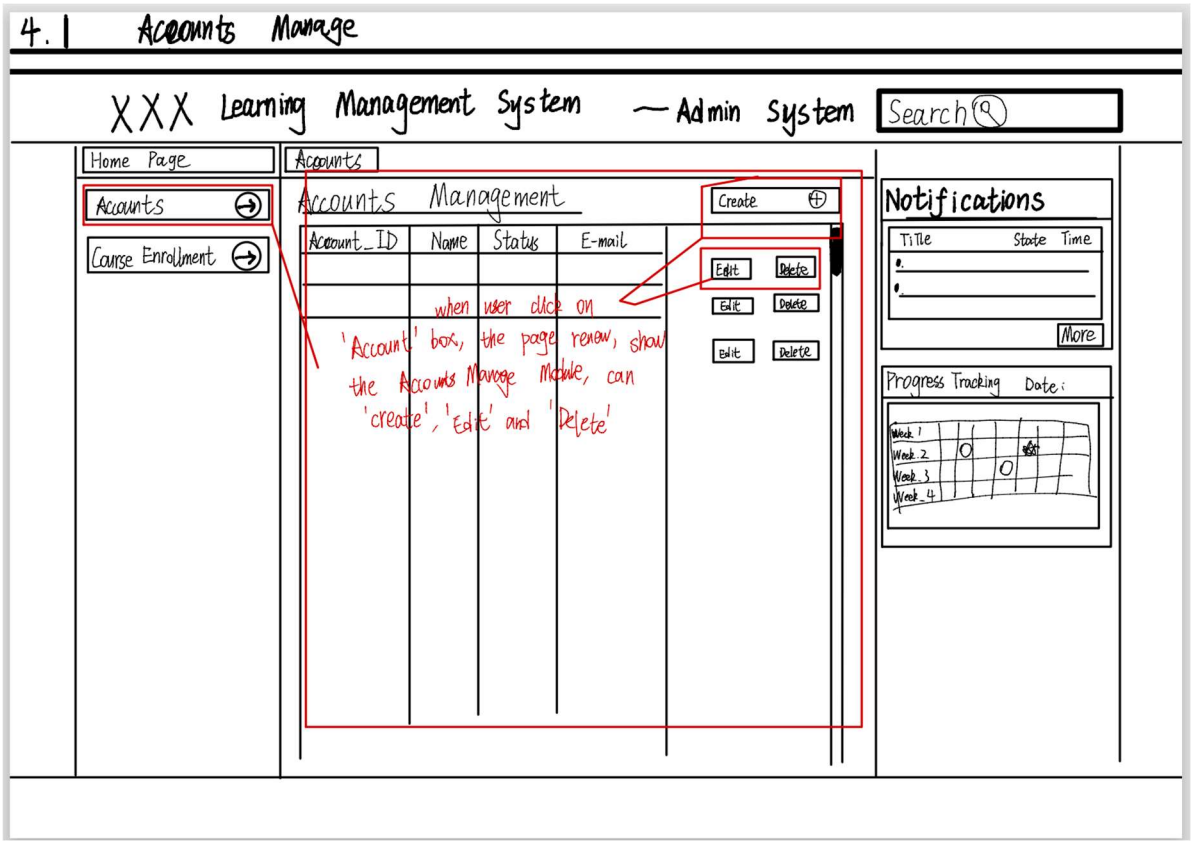
- Access students' submission



- Grade and provide feedback

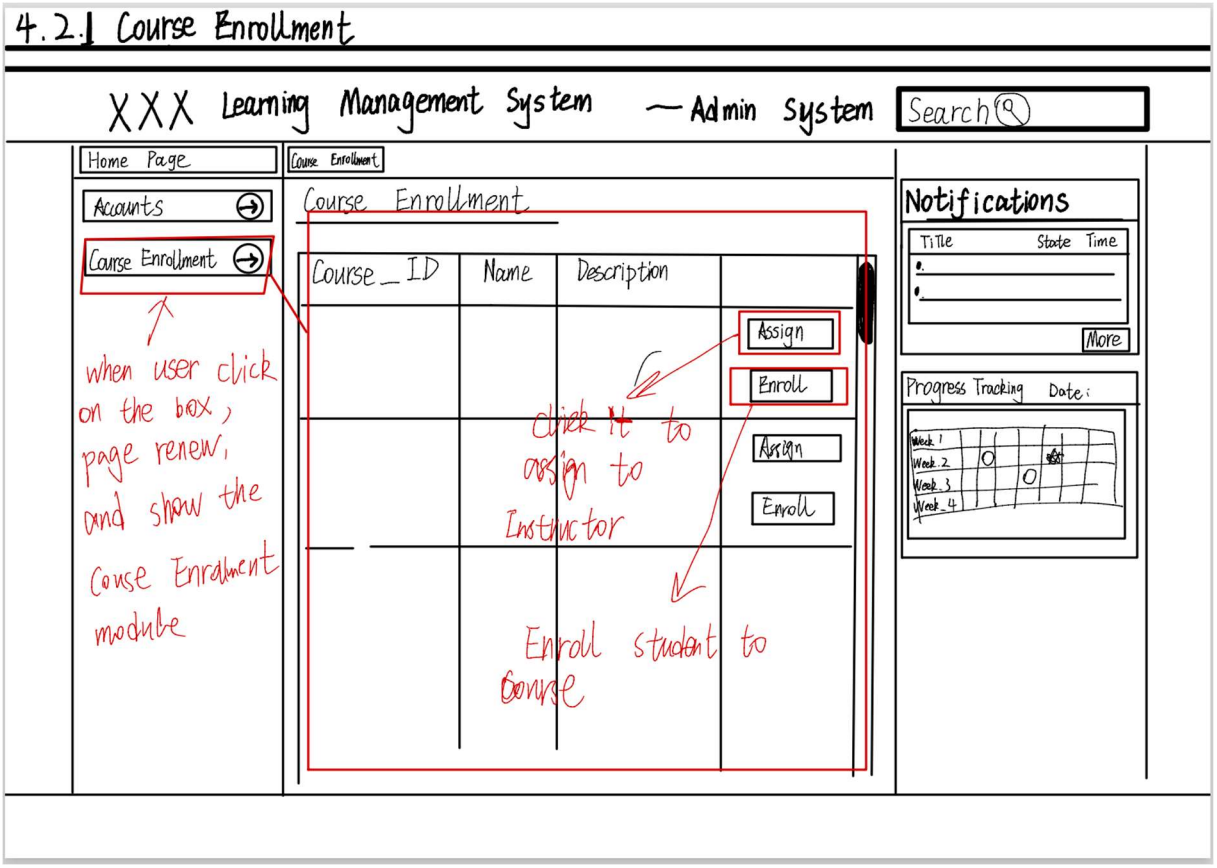


n) Account Management and Creation for Admin

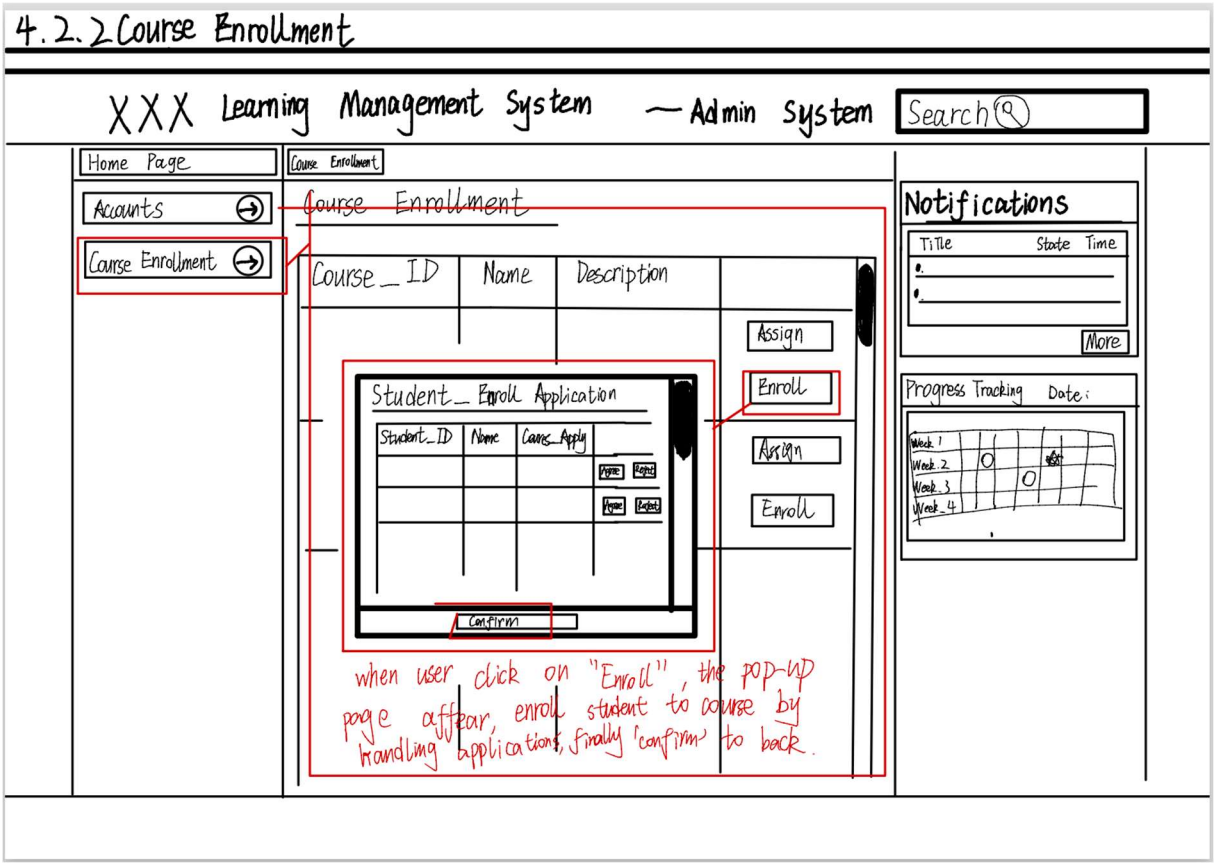


o) Course Enrollment for Admin

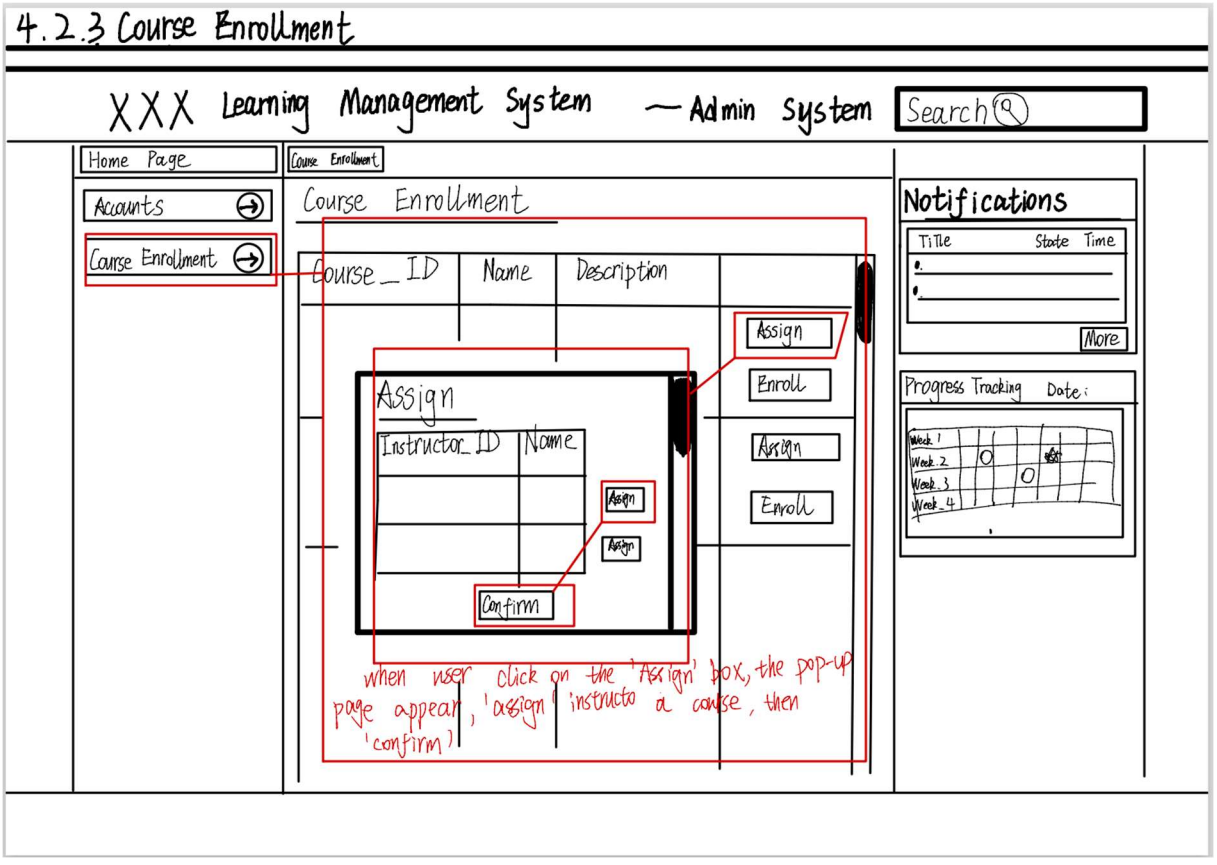
- Access Course Enrollment Module



- Enroll students in courses



- Assign a course to an instructor (or instructors)



4. Functionality of client and server

A): Login

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

The front-end Settings input box provides a field for the user to fill in the user's login information. The Password is protected by type=" Password ". The onclick event is used to monitor the user's click on the login button. The login function uses the jQuery syntax and the binding id to obtain the information filled in by the user, compose Json data, and send asynchronous ajax post requests to the back-end interface. <Using Method1 in 4.1.1 to transfer the data>

< For login function, the Json data that login page requires is “username: 123456”, “password: 0000”>

Server: Python + Flask + request + MySQL + connector

Python uses flask, request@server.route, to set the interface's unique url request.json. Get ("json key") Gets the value transmitted by the front-end. The connector uses mysql. The Connector command sets the user name, password, and database name to connect to the connector. Check whether the user information exists. If so, the connector returns user information and a success identifier.

B): Home

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

For the dynamic content change feature, the web page needs a button element with an onclick event listener to trigger the content change. The front-end setup consists of creating a button element with a unique identifying id. JavaScript (jQuery+ DOM) syntax is used to select the button element by id and add an onclick event listener to it. When the button is clicked, the onclick event triggers a JavaScript function that dynamically modifies the content of the web page. <Using Method1 in 4.1.1 to transfer the data>

Server: This feature does not require server-side processing

C): Search

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

For web search functionality, the front-end setup includes creating text entry fields and button elements for users to enter search keywords. JavaScript (jQuery+ DOM) syntax is used to select the text input field and the button element, and to add an onclick event listener to the button element. When the user enters a search keyword in the text entry field and clicks a button, the onclick event triggers a JavaScript function that retrieves the search keyword entered by the user and sends it to the back-end for processing. <Using Method1 in 4.1.1 to transfer the data>

< For search function, the Json data that search page requires is “Search Keyword: 123456” >

Sever: Python + Flask + MySQL + connector + request

The back end uses the Flask framework in Python to handle search requests. The Flask framework defines a function to handle search requests by setting a route. When a search request is received from the front end, the Flask framework uses the request library to retrieve the Json key sent by the front end and uses the MySQL database connector to perform the query operation. The query operation looks for the content containing the search key in the MySQL database and returns the search results to the front end.

D): More

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

To display more informational functionality, a web page needs a button element with an onclick event listener to trigger content changes. The front-end setup consists of creating a button element with a unique identifying id. JavaScript (jQuery+ DOM) syntax is used to select the button element by id and add an onclick event listener to it. When the button is clicked, the onclick event triggers a JavaScript function that dynamically modifies the content of the web page.

Server: This feature does not require server-side processing

E): Courses Management: click button

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

For this part of the course management functionality, first use a button element with an onclick event listener to trigger content changes. The front-end setup consists of creating a button element with a unique identifying id. JavaScript (jQuery+ DOM) syntax is used to select the button element by id and add an onclick event listener to it. When the button is clicked, the onclick event triggers a JavaScript function that dynamically modifies the content of the web page.

Server: This feature does not require server-side processing

F): Courses Management: add/edit

Client: HTML+ CSS+ JavaScript (jQuery+ DOM) + fetch API

The user clicks the "Add Info" icon on the front page, which is a typical HTML button element. Clicking the icon triggers the front-end JavaScript event handler. In the event handler, the fetch API is used to send GET requests to the Flask back-end route 'add info' or 'edit info'.

The front end receives the returned page and displays it, including a form for entering information. The user fills in the required information on this page. After completing the form, the user clicks the "Confirm" button, which triggers another front-end JavaScript event handler. The front end then

sends a POST request containing the input information, routing 'add info' and 'edit Infor' to the Flask back end.

After receiving a successful response from the back end, the front end can choose to refresh the main page or use JavaScript to dynamically update the page content to display the newly added information.

< For edit and add function, the Json data that edit or add page requires is “Course_ID: 123456” or “Course_Name: abcdefg”, “Instructor_Name: abcdefg”, “Faculty: abcdefg”, “Number_Student: 1234”>

Server: Python + Flask + MySQL + connector + request

The Flask back end receives the GET request and calls the appropriate view function add_info_page or edit_info_page. This function renders an HTML page with a form to enter the information, and then returns the form to the front end as an HTTP response.

The Flask back end receives the POST request, parses the JSON data in the request, and then stores the data in a MySQL database.

G): Courses Management: delete

Client: HTML+ CSS+ JavaScript (jQuery+ DOM) + fetch API

The user clicks the Delete button on the front page, which is typically an HTML button element. Clicking the button triggers the front-end JavaScript event handler. In the event handler, the front end uses the fetch API to send POST and DELETE requests to the Flask backend 'delete_info' route.

The front end receives the response and determines whether the deletion was successful based on the information in the response. If successful, the front end can choose to refresh the page or use JavaScript to dynamically update the page content to remove the deleted information.

< For delete function, the Json data that delete page requires is “Course_ID: 123456” or “Course Name: abcdefg”, “Instructor_Name: abcdefg”, “Faculty: abcdefg”, “Student_ID: 1234”>

Server: Python + Flask + MySQL + connector + request

The Flask back end receives the request and calls the corresponding view function 'delete info', which retrieves the identifier of the information to be deleted from the request and then performs the corresponding deletion operation in the database via MySQL. Connector. After processing the request, the back end sends a JSON response to the front end with a success or failure message.

H): Instructors Management: click button

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

For this part of the instructor management functionality, first use a button element with an onclick event listener to trigger content changes. The front-end setup consists of creating a button element

with a unique identifying id. JavaScript (jQuery+ DOM) syntax is used to select the button element by id and add an onclick event listener to it. When the button is clicked, the onclick event triggers a JavaScript function that dynamically modifies the content of the web page.

Server: This feature does not require server-side processing

I): Instructors Management: add/edit

Client: HTML+ CSS+ JavaScript (jQuery+ DOM) + fetch API

The user clicks the "Add Info" icon on the front page, which is usually an HTML button element. Clicking the icon triggers the front-end JavaScript event handler. In the event handler, the fetch API is used to send GET requests to the Flask back-end route 'add info' or 'edit info'.

The front end receives the returned page and displays it, including a form for entering information. The user fills in the required information on this page. After completing the form, the user clicks the "Confirm" button, which triggers another front-end JavaScript event handler. The front end then sends a POST request containing the input information, routing 'add info' and 'edit Infor' to the Flask back end [3].

After receiving a successful response from the back end, the front end can choose to refresh the main page or use JavaScript to dynamically update the page content to display the newly added information.

< For add and edit function, the Json data that add or edit page requires is “Instructor_ID: 123456” or “Instructor_Name: abcdefg”, “password: 1123456”, “Faculty: abcdefg”, “Course_Name: abcdefg”>

Server: Python + Flask + MySQL + connector + request

The Flask back end receives the GET request and calls the appropriate view function add_info_page or edit_info_page. This function renders an HTML page with a form to enter the information, and then returns the form to the front end as an HTTP response.

The Flask back end receives the POST request, parses the JSON data in the request, and then stores the data in a MySQL database.

J): Instructors Management: delete

Client: HTML+ CSS+ JavaScript (jQuery+ DOM) + fetch API

The user clicks the Delete button on the front page, which is typically an HTML button element. Clicking the button triggers the front-end JavaScript event handler. In the event handler, the front end uses the fetch API to send POST and DELETE requests to the Flask backend 'delete info' route.

The front end receives the response and determines whether the deletion was successful based on the information in the response. If successful, the front end can choose to refresh the page or use JavaScript to dynamically update the page content to remove the deleted information.

< For delete function, the Json data that delete page requires is “Instructor_ID: 123456” or “Instructor_Name: abcdefg”, “password: 1123456”, “Faculty: abcdefg”, “Course_Name: abcdefg”>

Server: Python + Flask + MySQL + connector + request

The Flask back end receives the request and calls the corresponding view function 'delete_info', which retrieves the identifier of the information to be deleted from the request and then performs the corresponding deletion operation in the database via MySQL. Connector. After processing the request, the back end sends a JSON response to the front end with a success or failure message.

K): Students Management: click button

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

For this part of the student management functionality, first use a button element with an onclick event listener to trigger content changes. The front-end setup consists of creating a button element with a unique identifying id. JavaScript (jQuery+ DOM) syntax is used to select the button element by id and add an onclick event listener to it. When the button is clicked, the onclick event triggers a JavaScript function that dynamically modifies the content of the web page.

Server: This feature does not require server-side processing

L): Students Management: add/edit

Client: HTML+ CSS+ JavaScript (jQuery+ DOM) + fetch API

The user clicks the "Add Info" icon on the front page, which is usually an HTML button element. Clicking the icon triggers the front-end JavaScript event handler. In the event handler, the fetch API is used to send GET requests to the Flask back-end route 'add info' or 'edit info'.

The front end receives the returned page and displays it, including a form for entering information. The user fills in the required information on this page. After completing the form, the user clicks the "Confirm" button, which triggers another front-end JavaScript event handler. The front end then sends a POST request containing the input information, routing 'add info' and 'edit Infor' to the Flask back end.

After receiving a successful response from the back end, the front end can choose to refresh the main page or use JavaScript to dynamically update the page content to display the newly added information.

< For add and edit function, the Json data that add or edit page requires is “Student_ID: 123456” or “Student_Name : abcdefg”, “Instructor_Name : abcdefg”, “Faculty : abcdefg”, “Course_Name : abcdefg”>

Server: Python + Flask + MySQL + connector + request

The Flask back end receives the GET request and calls the appropriate view function add_info_page or edit_info_page. This function renders an HTML page with a form to enter the information, and then returns the form to the front end as an HTTP response.

The Flask back end receives the POST request, parses the JSON data in the request, and then stores the data in a MySQL database.

M): Students Management : delete

Client: HTML+ CSS+ JavaScript (jQuery+ DOM) + fetch API

The user clicks the Delete button on the front page, which is typically an HTML button element. Clicking the button triggers the front-end JavaScript event handler. In the event handler, the front end uses the fetch API to send POST and DELETE requests to the Flask backend 'delete info' route.

The front end receives the response and determines whether the deletion was successful based on the information in the response. If successful, the front end can choose to refresh the page or use JavaScript to dynamically update the page content to remove the deleted information.

< For delete function, the Json data that delete page requires is “Student_ID: 123456” or “Student_Name : abcdefg”, “Instructor_Name : abcdefg”, “Faculty : abcdefg”, “Course_Name : abcdefg”>

Server: Python + Flask + MySQL + connector + request

The Flask back end receives the request and calls the corresponding view function 'delete info', which retrieves the identifier of the information to be deleted from the request and then performs the corresponding deletion operation in the database via MySQL. Connector. After processing the request, the back end sends a JSON response to the front end with a success or failure message.

N): Request Management

Client: HTML+ CSS+ JavaScript (jQuery+ DOM) + fetch API

The front end sends a POST request containing the course selection request ID and approval status to the route 'approve_course_request' on the Flask back end.

The administrator reviews course selection requests on the back-end administration page and decides whether to approve them. The front end provides an administrator page for viewing and processing course selection requests.

< For request react function, the Json data that request page requires is “Student_ID: 123456”, “Faculty: abcdefg”, “Course_ID.: 123456”, “Course_Name: abcdefg”>

Server: Python + Flask + MySQL + connector + request

The Flask back end receives the request and calls the corresponding view function 'add_course_request', which retrieves the course selection information from the request and stores the request in a database for administrator approval. The administrator clicks the Approval button on the back-end management page

The Flask back end receives the approval request and calls the corresponding view function 'approve_course_request', which updates the status of the course selection request to approved and stores the course selection information in a database via MySQL.connector.

O): student take courses

Client: HTML+ CSS+ JavaScript (jQuery+ DOM) + fetch API

On the front page, the user selects a course and clicks the Submit button. The front end uses the fetch API to send a POST request containing course selection information to the route 'add_course_request' on the Flask back end.

The student views the course selection results on the front-end page and sends a GET request to the route 'get_course_result' on the Flask back end. The back end retrieves the student's course selection results from the database through MySQL. Connectors and return them to the front end as HTTP responses.

< For take course function, the Json data that take course page requires is “Instructor_Name: abcdefg”, “Course_ID: 123456”, “Course_Name: abcdefg”>

Server: Python + Flask + MySQL + connector + request

The Flask back end receives the request and calls the corresponding view function 'add_course_request', which retrieves the course selection information from the request and stores the request in a database for administrator approval.

Q): students check their taken courses

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

For this part of the review course functionality, first use a button element with an onclick event listener to trigger content changes. When the student clicks View course material, an AJAX GET request is sent backwards to get the course material. After receiving the JSON data from the back end, the data is parsed and the course material is displayed on the page.

< For course checking function, the Json data that this checking page requires is “Instructor_Name : abcdefg”, “Course_Name : abcdefg”>

Server: Python + Flask + MySQL + connector + request

Flask back-end queries the lecture materials and information of the corresponding course from the database through MySQL to process GET requests from students. The connector then returns the query results to the front end in JSON format.

R): look their courses feedback

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

For this function, first use a button element with an onclick event listener to trigger the content change. When the student clicks the button to retrieve their grade, an AJAX GET request is sent back to get the grade data. After receiving the JSON data from the back end, the data is parsed to display the student's grades on the web page.

< For feedback check function, the Json data that feedback page requires is “Student_Name: abcdefg”, “Course_Name: abcdefg”>

Server: Python + Flask + MySQL + connector + request

The Flask back end queries the student's score information from the database through the MySQL connector to process GET requests. The retrieved grade information is then returned to the front end in JSON format.

S): Upload and Submission

Client: HTML+ CSS+ JavaScript (jQuery+ DOM) + fetch API

Students click the "Submit assignment" button on the web page, which triggers a JavaScript event handler on the front end. This event handler creates a Form Data object that contains the selected file to be uploaded. The front end then uses the fetch API to send a POST request to the 'upload homework' route on the Flask back end.

Server: Python + Flask + request

On the Flask back end, call a view function 'upload homework' to handle upload requests. This function uses request to retrieve the uploaded file from the request. file ['homework'], use the save method to save the file to a specified directory on the server and return a JSON response to indicate that the job was uploaded successfully.

T): Forum (student and instructor)

Client: HTML+ CSS+ JavaScript (jQuery+ DOM) + fetch API

For this function, first use a button element with an onclick event listener to trigger the content change. When the user is ready to share, clicking the button to retrieve the word sends an AJAX GET request to the backend to retrieve the content data. After receiving the JSON data from the back end, the data is parsed and the shared content is displayed on the web page. < For forum function, the Json data that forum page requires is “Student_Name: abcdefg” or “Instructor_Name: abcdefg”>

Server: Python + Flask + request + MySQL + connector

The Flask back end queries the shared information from the database through the MySQL connector to process GET requests. The retrieved shared information is then returned to the front end in JSON format.

U): look through student list

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

For this function, first use a button element with an onclick event listener to trigger the content change. The front-end setup consists of creating a button element with a unique identifying id. JavaScript (jQuery+ DOM) syntax is used to select the button element by id and add an onclick event

listener to it. When the button is clicked, the onclick event triggers a JavaScript function that dynamically modifies the content of the web page.

Server: This feature does not require server-side processing.

V): Assessment Evaluation

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

For this function, first use a button element with an onclick event listener to trigger the content change. The teacher interface uses AJAX to send GET requests backwards to retrieve a list of assignments submitted by students. When the teacher clicks the assignment link or button, another AJAX GET request is sent back to retrieve the details of that assignment.

The teacher then enters the score of the student's work on the interface and sends a POST request to the back end using AJAX to upload the score.

< For grading function, the Json data that grading page requires is “Student_Name: abcdefg”, “Course_Name: abcdefg”>

Server:

Flask back-end receives a GET request, obtains the job list from the teacher, queries the corresponding job information in the database, and returns the result to the front-end in JSON format.

The Flask back end receives the POST request from the teacher to upload the score and updates the received score information to the database through the MySQL. Connector.

W): Lecture Management

Client: HTML+ CSS+ JavaScript (jQuery+ DOM) + fetch API

For this function, first use a button element with an onclick event listener to trigger the content change. There will be a button ready. When the teacher uploads the course material, click. It will use Form Data objects to encapsulate the file data, and then use AJAX to send POST requests backwards.

< For lecture manage function, the Json data that lecture manage page requires is “Course_Name: abcdefg”, “Instructor_Name: abcdefg”>

Server: Python + Flask + request + MySQL + connector

The Flask back-end receives the POST request from the teacher to upload course materials, saves the received file to the file system of the server, and saves the course information to the database through the MySQL.connector.

X): Email

Client: HTML+ CSS+ JavaScript (jQuery+ DOM) + fetch API

For this function, first use a button element with an onclick event listener to trigger the content change. When the user finishes typing the content and clicks the button to retrieve the score, an AJAX GET request is sent backwards to retrieve the content data. After receiving the JSON data from the back end, the data is parsed and the communication content is displayed on the web page.

< For email function, the Json data that email communicate page requires is “Student_Name: abcdefg” or “Instructor_Name: abcdefg”>

Server: Python + Flask + request + MySQL + connector

The Flask back end queries the content information from the database through the MySQL.connector to process GET requests. The retrieved content information is then returned to the front-end in JSON format.

5. Conclusion:

In general, the preliminary report includes sections such as project introduction, database ER diagrams, wireframed diagrams, and an overview of front and back-end functionality and expected implementation.

As for the project completion progress, the learning management website has achieved the initial goal, including the most basic functions such as search, forum, etc., and the development of the database has also been completed. For now, it should be possible to complete the basic functions required by the project and be up and running locally by week 10.

However, I still encountered some challenges during the development process, such as technical difficulties and time pressure. The technical problems include the front and back-end interaction technology, the development environment of the overall website is not familiar with the previously unfamiliar development technology and problems; There is also considerable pressure on time, because the development of the website needs to use a lot of new technologies, encounter new problems, so we have to constantly increase the development time and improve the development efficiency. But through these tests, I learned a lot about project management and technical implementation experience, understand the project development process, and really improve the project development technology [4].

In the future, I will consider the possibility of performance optimization, user experience improvement, feature expansion and security enhancement [5]. For example, provide user authentication to prevent security vulnerabilities to ensure user security; Use responsive design to provide users with real-time notifications and other ways to improve user experience. In addition, consider developing an extended feature for online testing for students.

6. References:

[1] Salman R A , Fraiwan M , Khasawneh N .Protecting AJAX Code Using Secure Communication[J]. 2022.

[2] Chen Z .Design of Computer Multimedia Intelligent Platform Using Big Data Analysis[J].Journal of Interconnection Networks, 2022.DOI:10.1142/S0219265921430246.

[3] Couto F .Desenvolvimento de Aplicaes Web baseadas em SOA e AJAX[J]. 2022.

[4] Garrett J J .Ajax: A New Approach to Web Applications[J].http://www.adaptivepath.com/, 2007.DOI:http://dx.doi.org/.

[5] Justin Gehtland,Ben Galbraith,Dion Almaer.Pragmatic Ajax – A Web 2.0 Primer[M].Pragmatic Bookshelf,2006.