USC Course Marketplace Project

Benjamin Lu, Lucas Rendina, Jason Zhao, Lucas Rendina, Alex Chen, Charles Meeler

Table of Contents

Proposal

Team Members: Adam Khalil (afkhalil@usc.edu), Felix Chen (felixche@usc.edu), Charles Meeler (cmeeler@usc.edu), Alex Chen (achen715@usc.edu), Zhanhao Zhao (zhanhaoz@usc.edu), Lucas Rendina (rendina@usc.edu), Benjamin Lu (btlu@usc.edu).

Project Proposal: A web application marketplace for USC schedules with class time, subject, professor.

Weekly Meeting Time: Friday's at 5 PM with TA Eliabeth O.

High Level Requirements

1. High-Level Requirements

Project Overview:

The goal of this project is to develop a web-based application where students at USC can buy and sell their courses. The platform allows students to input classes they are interested in selling and search for classes they are interested in buying. The application must also feature secure accounts where students can save the classes they are interested in, and there must be a method to communicate between buyers and sellers provided on the website.

Functional Requirements:

- **User Schedule Input**: The system must provide users with an input form where they can:
    - Enter course names.
    - Select the days of the week the course occurs (Monday to Friday).
    - Enter start and end times for each class
    - Enter professor of the course
- **Course Marketplace:**
    - Allow for “marketplace search” of certain courses. Include filters by time, professor, days, type of course
    - Provide contact info for communication between buyer and seller
- **User Account Management**: The application must allow users to create accounts and log in to save their schedules. Each user's schedule should be unique to their account. Data such as courses held and transactions should be stored to their account as well.

Non-Functional Requirements:

- **Web-Based Interface**: The application must be accessible via a standard web browser, such as Chrome, Firefox, or Safari.
- **Responsiveness**: The interface should be responsive and work on desktop devices.
- **Usability**: The user interface must be intuitive for non-technical users, minimizing the learning curve and allowing for easy purchase and communication
- **Performance**: The application must handle multiple users entering and modifying their schedules simultaneously without performance degradation/bugs.

Technical Specifications

Technical Specifications (due October 27)
With regards to the feedback on your high-level requirements document, write up the technical specifications for the project. If you need to modify your high-level requirement based on feedback, please also make the appropriate changes to your high-level requirements.

Total Grade	2.0%
1.0%	Full outline of tasks that need to be completed
0.5%	Tasks contains descriptions that are sufficient for design phase even if designer was unfamiliar with application
0.25%	Professionally formatted
0.25%	No grammatical mistakes

System Architecture: The web application will involve a 3-tier(frontend, backend, database) architecture
Frontend: HTML, CSS, and Javascript will be used for user interaction
Backend: Server-side application in Java will manage users and data processing
Database: Users, courses, and their data will be stored in an SQL relational database on MySQL

User Course Input:
Input Features:

- Web interface needs to have an “add courses” page that allow users to add classes they are selling
    - Professor Name(String)
    - Days(String)
    - Class Time (Int)
    - Course Name(String)
- Upon clicking the add course button, the courses that people are selling will be added to their account, given that they are logged in

Course Marketplace:

The course marketplace will include a catalog of the available courses on sale. Typing in the search bar will be able to filter the listed courses list by the listed search. Users can click “favorite course” to favorite the course, which will show up in their profile section.

User Account Management:

Login and Authentication: The login process should be fast and easy to ensure good user experience. A form for username and password should be provided, alerting the user if their username or password is not valid, checking against the database.

Registration: The registration process should be fast and easy to use to minimize barriers to entry. A form for username, password, and confirm password should be provided, alerting the user if their passwords do not match or if their chosen username already exists in the database.

Detailed Design

Design of Tasks mentioned in Technical Specifications

- User Course Input (Adding courses to sell)
    - Features a form collecting all relevant fields: Course Code, Professor, start time (Days of class, time)
    - The form will send the information in JSON format to the backend servlet, which will populate the schedule database in MySQL
- Course Marketplace
    - Features every course that sellers have added into the database from the User Course Input
    - Pulls all courses from from the schedule database via making a request via a backend servlet, which returns the data that populates the frontend
    - Allow for “favorite” functionality: Users can click courses to favorite them, adding the course to their profile
- User Account Management
    - Registration: Features a form collecting username, password, and confirm password. Calls to the backend servlet will be made to verify that the username

isn't already taken in the database. Checks will be made to ensure passwords match
    - Login: Features a form collecting username and password. Calls to the backend servlet will be made to verify that the login information is correct. Errors will be raised if not.
- Profile
    - Only appears if the user is logged in. Use localstorage to store usedID of the user logged in
    - Profile is populated with favorited courses, utilizing a servlet to pull from the database

Software Requirements:

Software

- Java Servlets
- MySQL
- Apache Tomcat
- Eclipse

Hardware:

- Processor (Dual-Core or higher)
- RAM (16 GB+ recommended)
- Storage (50-100 GB+)
- Functional OS and WiFi Network

Database Schema:

Database: COURSEMARKET

Table 1: users

Column Name	Data Type	Description
userID	**int**	**Primary key for each user**

username	**String**	**Username**
Pw	**String**	**Password for sign in**
email	**string**	**Email for contact**

Table 2: schedule

Column Name	Data Type	Description
schedID	**int**	**Primary key for scheduleID**
userID	**int**	**Foreign key referencing users**
courseCode	**char**	**Course Code referencing the USC course title (I.E. CSCI 201)**
Professor	**char**	**Professor Name**
stime	**char**	**Start time of course (MW 10 AM)**
contact	**char**	**Email for method of contact between buyers and sellers**

Class Diagram:

AccountServlet

- Methods: `doGet()`, `doPost()`

DBConnection

- `URl: String`
- `USER: String`
- `PASSWORD: String`

- Methods: `getConnection()`

LoginServlet

- Methods: `doPost(), doGet()`

RegisterServlet

- `+ layoutOptions: List<LayoutTemplate>`
- Methods: `+ selectLayout(LayoutTemplate)`

MarketplaceServlet

- Methods: `doPost(), doGet()`

ScheduleServlet

- Methods: `doPost(), doGet()`

RegisterServlet

- Methods: `doPost(), doGet()`

Testing Plan

White Box Testing

Focuses on testing the internal logic of the code, especially servlets, database connections, and JSON handling.

Test Case	Input	Expected Output
Servlet handles valid input	JSON: `{ "courseCode": "CSCI 201", "professor": "Dr. Smith", "stime": "MWF 10:00 AM" }`	Database is updated with new course data, and servlet returns success response (e.g., HTTP 200 with message "Course Added").

Invalid JSON structure	JSON: `{ "course": "CSCI 201" }`	Servlet responds with an error (e.g., HTTP 400 with message "Invalid input").
Database connection failure	Valid JSON: `{ "courseCode": "CSCI 201", "professor": "Dr. Smith", "stime": "MWF 10:00 AM" }`	Servlet returns error response (e.g., HTTP 500 with message "Database connection error").
Login validation logic	Username: "testuser", Password: "testpass"	If credentials match, returns success response with `userID`. If not, returns error (e.g., HTTP 401 with "Invalid login").

Black Box Testing

Focuses on testing application functionality without knowledge of internal implementation.

Test Case	Input	Expected Output
Course addition form	Form fields: `CSCI 201`, `Dr. Smith`, `MWF 10:00 AM`. Click "Add Course".	Confirmation message: "Course successfully added." Course visible in the marketplace.
Empty course form submission	Leave form fields blank and click "Add Course".	Error message: "All fields are required."
Marketplace search	Search term: "CSCI".	Displays all courses with "CSCI" in the course code or title.
Login functionality	Enter correct username/password.	User is redirected to the marketplace.
Login error handling	Enter incorrect username/password.	Error message: "Invalid username or password."

Unit Testing

Tests individual components (servlets, database methods) in isolation.

Component	Test Case	Input	Expected Output

`AccountServlet`	Valid login	Username: "testuser", Password: "testpass"	Returns user ID of "testuser".
	Invalid login	Username: "testuser", Password: "wrongpass"	Returns HTTP 401 with "Invalid login".
`MarketplaceServlet`	Retrieve all courses	N/A	JSON array of all courses in the database.
	Add course	`{ "courseCode": "CSCI 101", "professor": "Dr. Lee", "stime": "TTh 1:00 PM" }`	HTTP 200 with success message.
`DBConnection`	Valid database connection	Database credentials	Returns active database connection object.
	Invalid database connection	Incorrect database credentials	Throws exception with "Connection failed".

Regression Testing

Ensures that new updates or fixes don’t break existing functionality.

Test Case	Description	Expected Outcome
Adding new servlet logic	Test login, registration, and course addition after adding a new servlet to handle marketplace search.	No functionality in login, registration, or course addition is broken.
UI update for profile	Test that favorite courses are still displayed correctly after updating the UI styling for the profile page.	Favorites functionality works without any errors.
Database schema update	After modifying the schema to include "course description," ensure all CRUD operations (add, update, delete courses) still function.	No crashes or incorrect behavior when adding, updating, or deleting courses.

Search performance	After optimizing search logic in the backend, ensure the marketplace still retrieves the correct courses based on filters.	Search results remain accurate and retrieve the expected courses.

Deployment Steps

1. Set Up MySQL Database

1. **Install MySQL**:
    - Download and install MySQL Community Edition from MySQL official website.
2. **Create the Database**:
    - Open MySQL Workbench or a command-line interface.

Execute the following SQL commands to create the database and tables:
sql
Copy code

```
CREATE DATABASE COURSEMARKET;

USE COURSEMARKET;

CREATE TABLE users (
    userID INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) NOT NULL UNIQUE,
    pw VARCHAR(50) NOT NULL,
    email VARCHAR(100)
);

CREATE TABLE schedule (
    schedID INT AUTO_INCREMENT PRIMARY KEY,
    userID INT NOT NULL,
    courseCode VARCHAR(10),
    professor VARCHAR(50),
    stime VARCHAR(50),
    contact VARCHAR(100),
    FOREIGN KEY (userID) REFERENCES users(userID)
```

```
);
```

-

---

2. Configure Eclipse Project

1. **Import the Project**:
   - Open Eclipse and go to `File` > `Import...`.
   - Select `Existing Projects into Workspace` and import the project folder.
2. **Add Tomcat Server**:
   - Go to `Window` > `Preferences` > `Server` > `Runtime Environments`.
   - Click `Add`, select `Apache Tomcat v9.0`, and specify the Tomcat installation directory.
   - Once added, configure the server in the `Servers` tab.
3. **Add Libraries**:
   - Right-click the project > `Build Path` > `Configure Build Path`.
   - Add:
     - MySQL Connector/J `.jar` file.
     - JSTL `.jar` file (if using JSP with JSTL).

---

3. Configure Database Connection

1. **Set up Database Credentials**:
   - Open the `DBConnection` class.

Update the following values:
java
Copy code

```
private static final String URL =
"jdbc:mysql://localhost:3306/COURSEMARKET";
private static final String USER = "your_username";
private static final String PASSWORD = "your_password";
```

-

2. **Test the Connection**:

Write a simple test function in Eclipse to verify the connection:
java

Copy code

```
public static void main(String[] args) {
    Connection conn = DBConnection.getConnection();
    if (conn != null) {
        System.out.println("Database connected!");
    } else {
        System.out.println("Database connection failed!");
    }
}
```

-

---

4. Deploy to Tomcat

1. **Build the Project**:
   - Right-click the project > `Run As` > `Run on Server`.
   - Select Apache Tomcat as the server and deploy.
2. **Access the Application**:
   - Open your browser and navigate to `http://localhost:8080/CourseMarketplace`.