Final Project

Timur Rakhimov
T00668753
COMP 4621

MongoDB Database

The MongoDB database includes a users collection with the following fields:

- **username**: Stores the user's unique identifier (username).
- **password**: Stores the user's password.
- **courses**: An array of courses associated with the user. Each course in the array has the following fields:
    - **courseId**: A unique identifier for the course.
    - **courseName**: The name of the course.

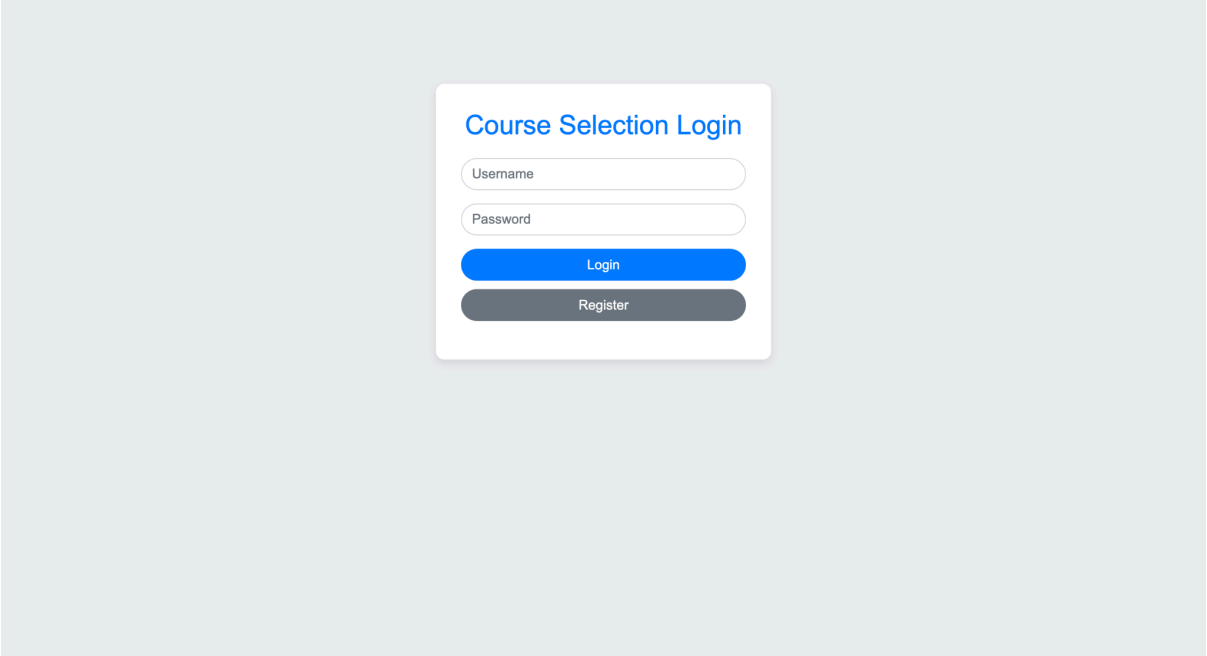
```
  {
    _id: ObjectId('6739000e2139e159a8fae17e'),
    courseId: 'COMP4980',
    courseName: 'Machine Learning'
  },
  {
    _id: ObjectId('6739000e2139e159a8fae17f'),
    courseId: 'MIST3620',
    courseName: 'Web-enabled Business Apps'
  }
]
courseSelectionDB> db.users.find().pretty();
[
  {
    _id: ObjectId('67390e162139e159a8fae180'),
    username: 'timur',
    password: 'timur123',
    courses: [
      {
        _id: ObjectId('673a74f32a9ac9c47efe19f9'),
        courseId: 'COMP4621',
        courseName: 'Web-Based Information Systems'
      },
      {
        _id: ObjectId('673a74f32a9ac9c47efe19fa'),
        courseId: 'COMP4910',
        courseName: 'Computing Science Project'
      },
      {
        _id: ObjectId('673a74f32a9ac9c47efe19fb'),
        courseId: 'COMP4980',
        courseName: 'Machine Learning'
      },
      {
        _id: ObjectId('673a74f32a9ac9c47efe19fc'),
        courseId: 'MIST3620',
        courseName: 'Web-enabled Business Apps'
      }
    ],
    __v: 2
  },
  {
    _id: ObjectId('673910782139e159a8fae182'),
    username: 'tima',
    password: 'tima123',
    courses: []
  },
  {
    _id: ObjectId('67391d1848aff766e3832ad8'),
    username: 'almat',
    password: 'almat123',
    __v: 0,
    courses: []
  },
  {
    _id: ObjectId('6739269fc2024937f3c7bcad'),
    username: 'sanzhar',
    password: 'sanzhar123',
    __v: 0,
    courses: []
  }
]
courseSelectionDB>
```

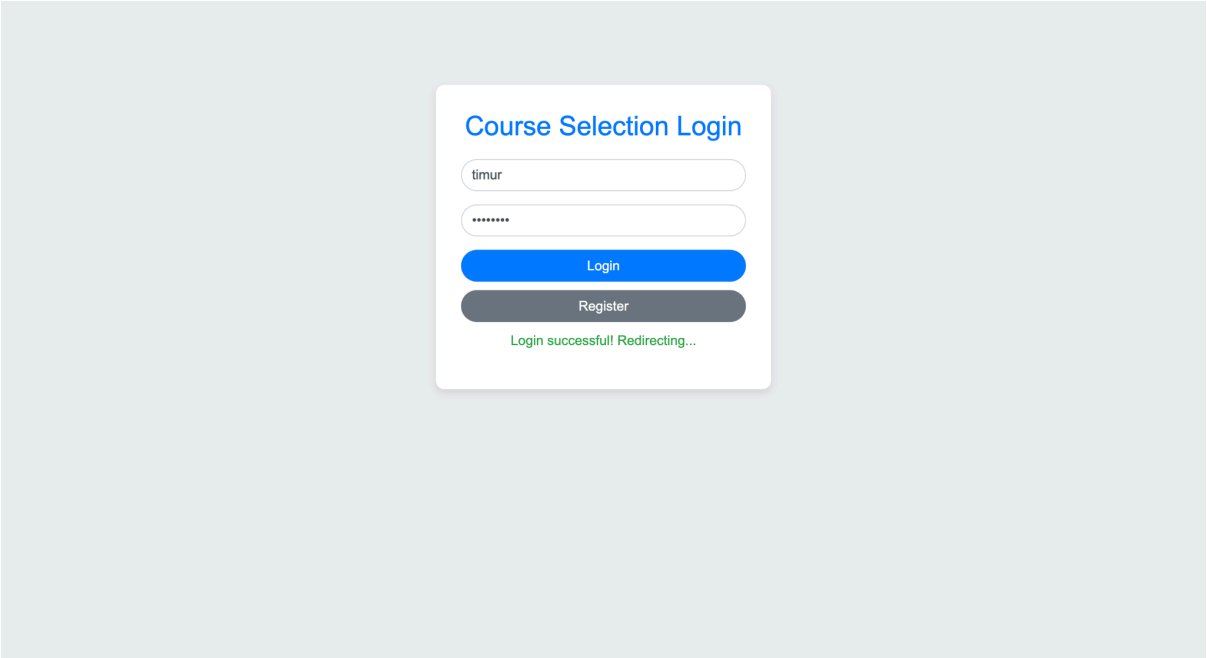
Login Functionality

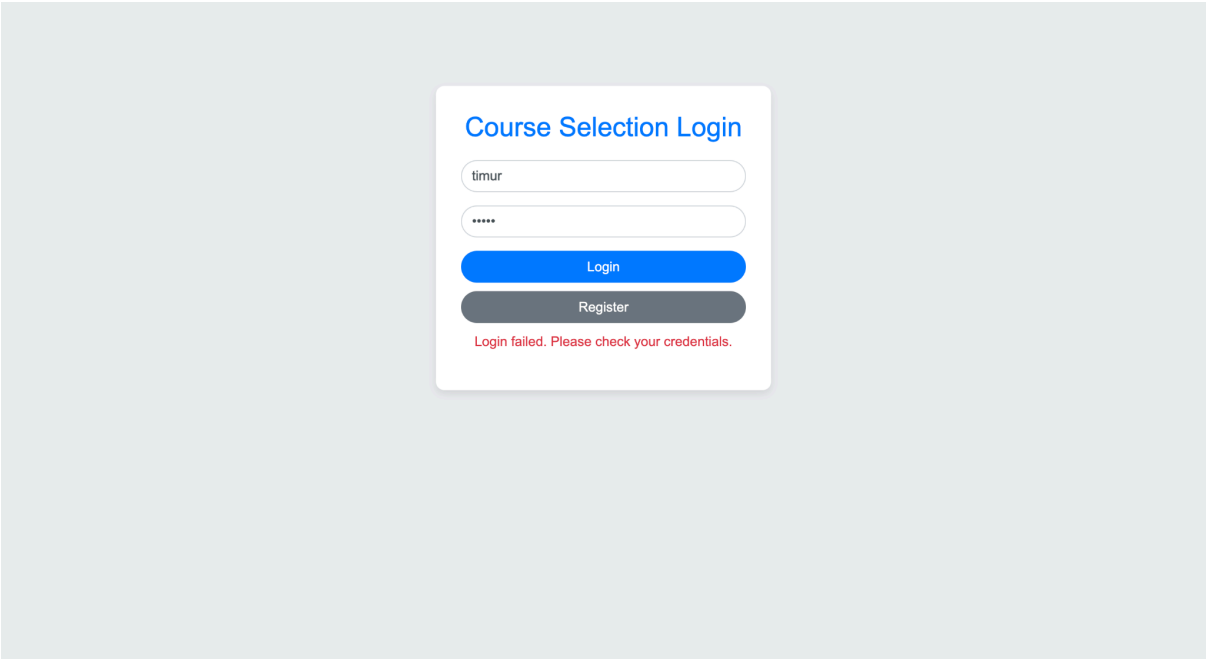
1. **Successful Login**
   - **Given:** The user enters an existing username and a correct password.
   - **When:** The login button is clicked.
   - **Then:** The user should be redirected to the **courses.html** page.
2. **Login Failure (Incorrect Username or Password)**
   - **Given:** The user enters a non-existent username or an incorrect password.
   - **When:** The login button is clicked.
   - **Then:** An error message should be displayed indicating "Login failed. Please check your credentials"

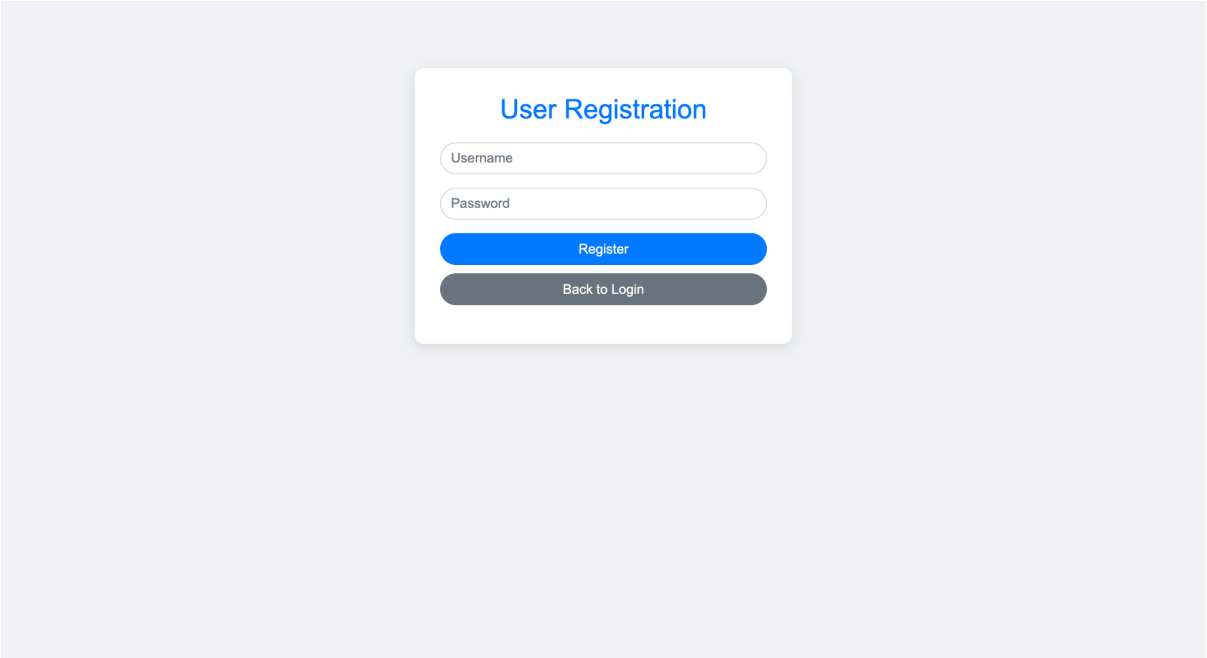
Registration Functionality

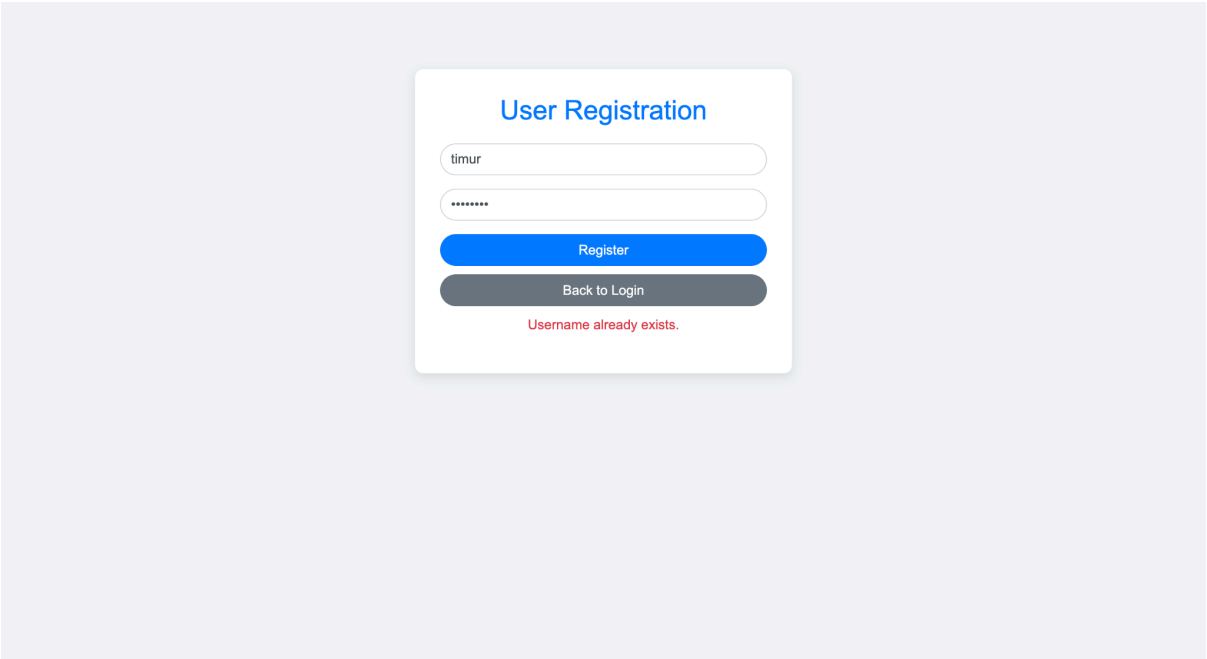
1. **Successful Registration**
   - **Given:** The user enters a new, unique username and a password.
   - **When:** The register button is clicked.
   - **Then:** The user should be successfully registered, and a confirmation message should appear.
2. **Registration Failure (Username Already Exists)**
   - **Given:** The user tries to register with an existing username.
   - **When:** The register button is clicked.
   - **Then:** An error message should be displayed indicating "Username already exists."

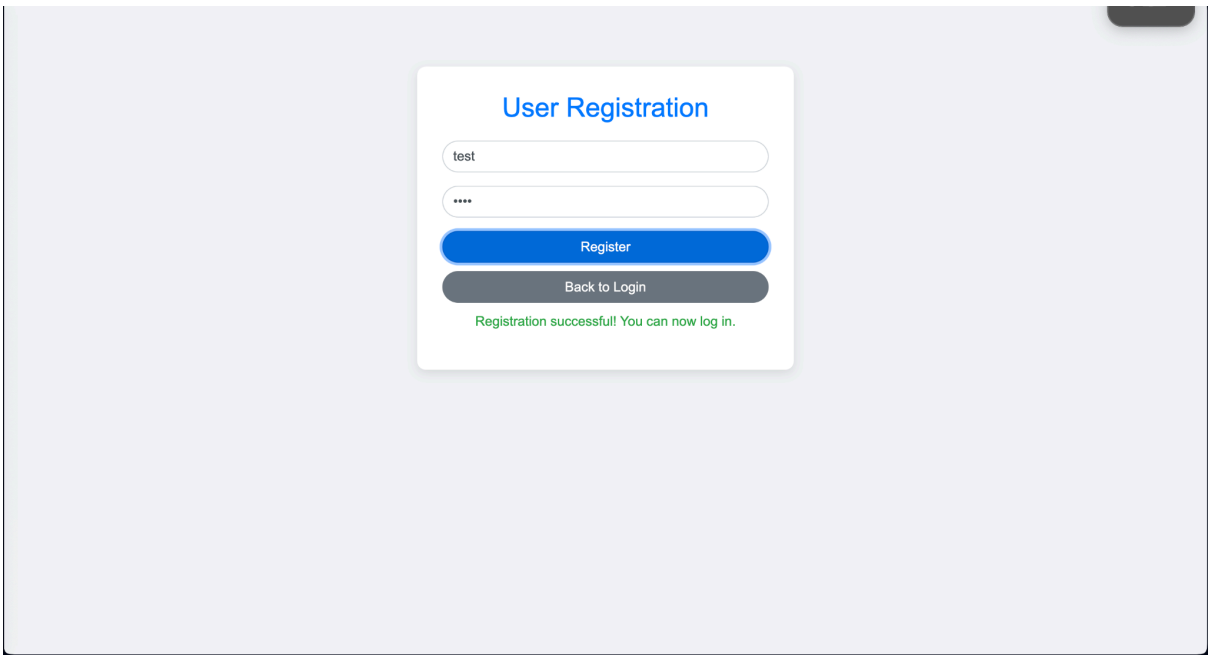



Course Selection Login
timur
••••••••
Login
Register
Login successful! Redirecting...


Course Selection Login
timur
•••••
Login
Register
Login failed. Please check your credentials.


User Registration
Username
Password
Register
Back to Login

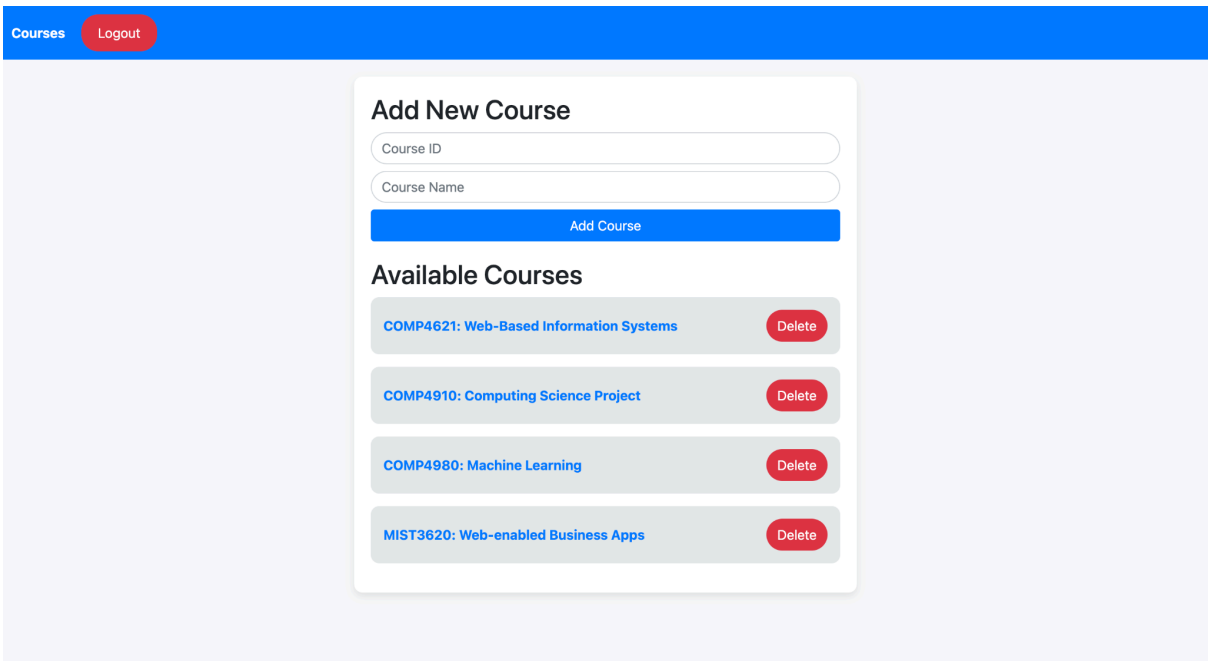

User Registration
timur
••••••••
Register
Back to Login
Username already exists.

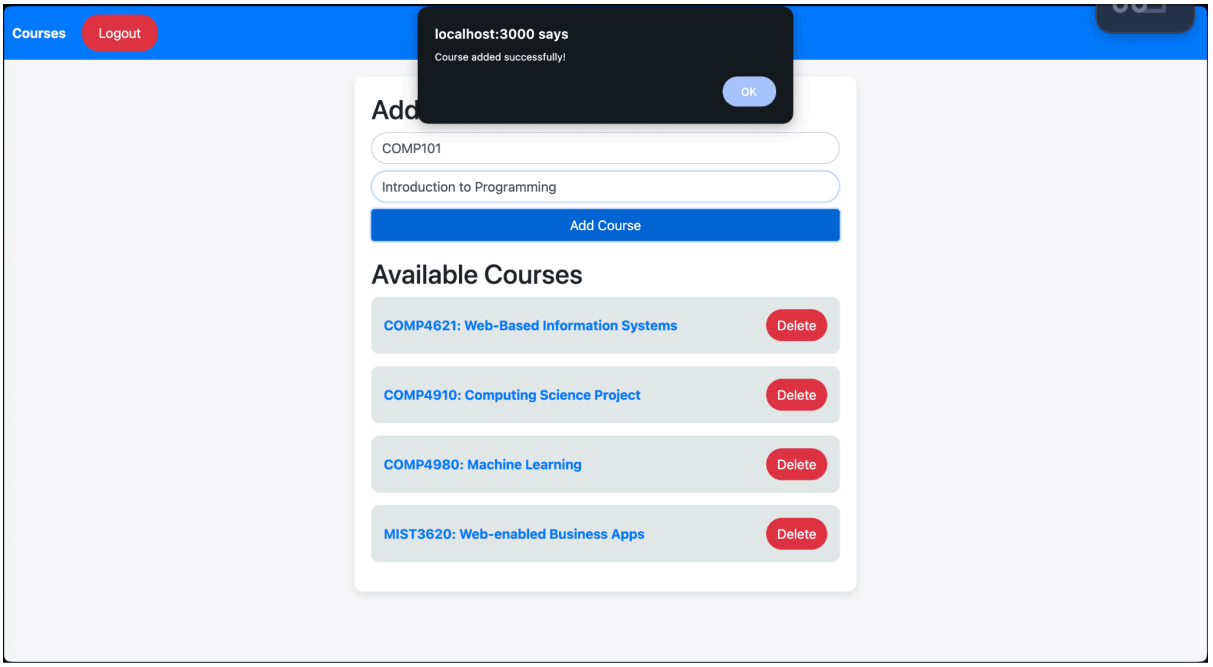


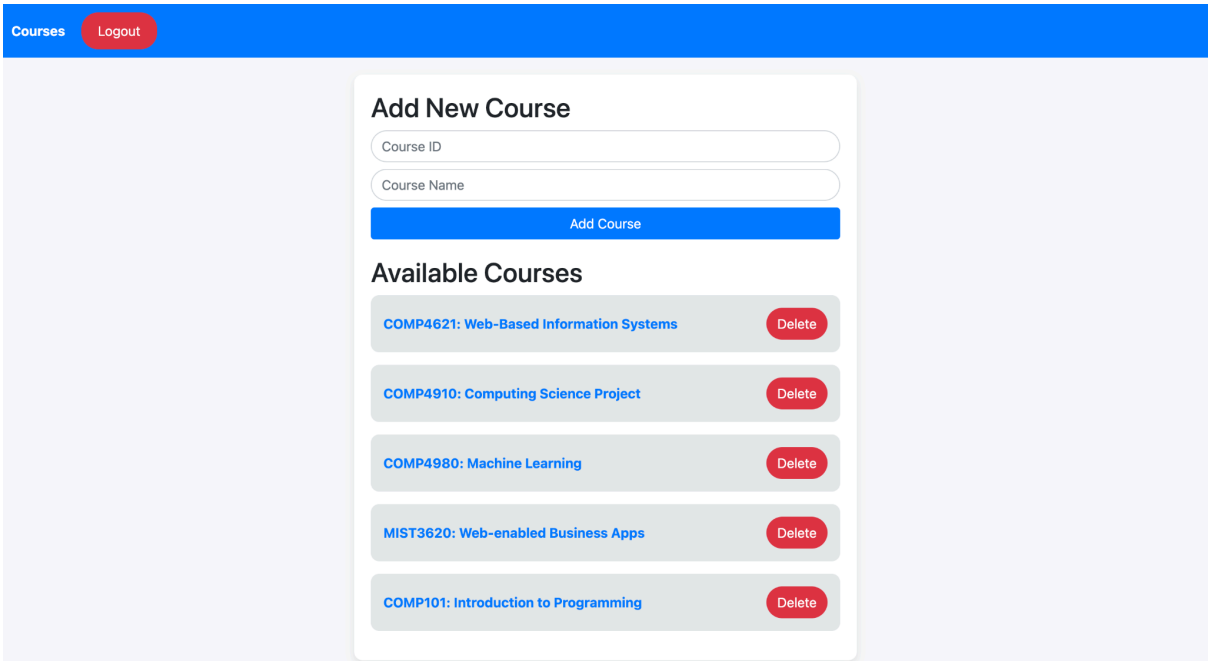
Courses page

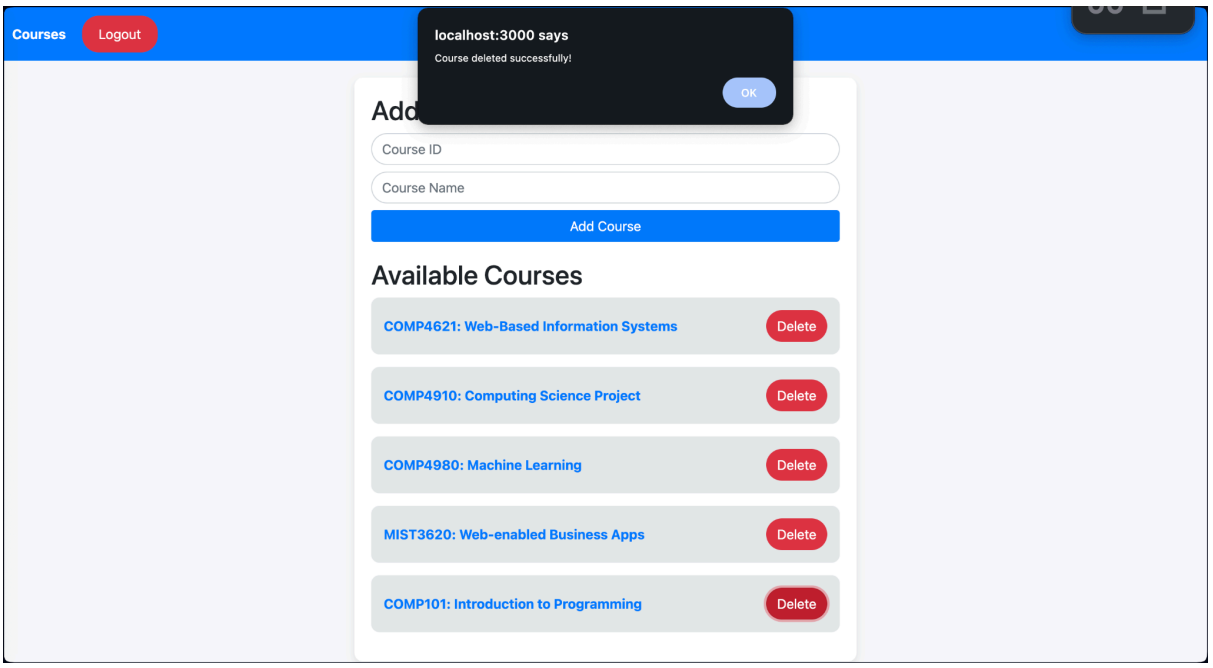
The **Courses** page displays a list of courses linked to each individual user. It provides the following functionalities:

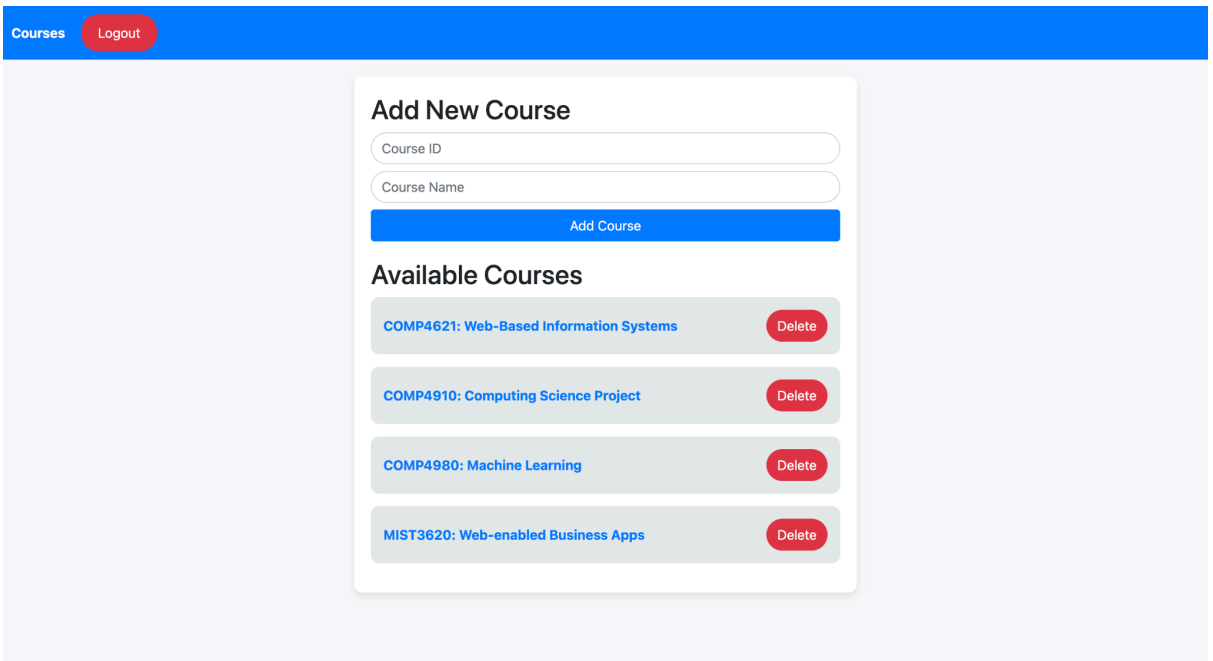
1. **View Courses**: Displays all the courses associated with the logged-in user.
2. **Add Course**: Allows users to add a new course to their list.
3. **Delete Course**: Enables users to remove an existing course from their list.
4. **Logout**: Provides an option for users to securely log out of the application.




Courses
Logout
localhost:3000 says
Course added successfully!
OK
Add
COMP101
Introduction to Programming
Add Course
Available Courses
COMP4621: Web-Based Information Systems
Delete
COMP4910: Computing Science Project
Delete
COMP4980: Machine Learning
Delete
MIST3620: Web-enabled Business Apps
Delete


Courses
Logout
Add New Course
Course ID
Course Name
Add Course
Available Courses
COMP4621: Web-Based Information Systems
Delete
COMP4910: Computing Science Project
Delete
COMP4980: Machine Learning
Delete
MIST3620: Web-enabled Business Apps
Delete
COMP101: Introduction to Programming
Delete


Courses
Logout
localhost:3000 says
Course deleted successfully!
OK
Add
Course ID
Course Name
Add Course
Available Courses
COMP4621: Web-Based Information Systems
Delete
COMP4910: Computing Science Project
Delete
COMP4980: Machine Learning
Delete
MIST3620: Web-enabled Business Apps
Delete
COMP101: Introduction to Programming
Delete


Courses
Logout
Add New Course
Course ID
Course Name
Add Course
Available Courses
COMP4621: Web-Based Information Systems
Delete
COMP4910: Computing Science Project
Delete
COMP4980: Machine Learning
Delete
MIST3620: Web-enabled Business Apps
Delete

Code segments

index.html

```
    <!--
        Author: Timur Rakhimov
        Date: 2024-11-12
        Description: This page provides a simple UI for users to log in
or navigate to the registration page.
    -->

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Course Selection - Login</title>
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.
min.css">
    <style>
        /* Set a light gray background for the entire page */
        body {
            background-color: #e9ecef;
            font-family: Arial, sans-serif;
        }
        /* Center the login form with a maximum width and add padding */
        .login-container {
            max-width: 400px;
            margin: 100px auto;
            padding: 30px;
            background-color: #fff;
            border-radius: 10px;
            box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
        }
        /* Style the header text of the login form */
        .login-header {
            text-align: center;
```

```
            color: #007bff;
            margin-bottom: 20px;
        }
        /* Style input fields with rounded corners */
        .form-control {
            border-radius: 20px;
        }
        /* Style the primary button for login */
        .btn-primary {
            border-radius: 20px;
            width: 100%;
        }
        /* Style the secondary button for registration navigation */
        .btn-secondary {
            border-radius: 20px;
            width: 100%;
            margin-top: 10px;
        }
        /* Error message styling */
        .error-message {
            color: #dc3545;
            text-align: center;
            margin-top: 10px;
        }
        /* Success message styling */
        .success-message {
            color: #28a745;
            text-align: center;
            margin-top: 10px;
        }
    </style>
    <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.
js"></script>
    <!-- Include the AngularJS client-side script -->
    <script src="app.js"></script>
</head>
<body ng-app="courseApp" ng-controller="MainController">
    <!-- Login form container -->
```

```
    <div class="login-container">
        <h2 class="login-header">Course Selection Login</h2>
        <input type="text" ng-model="username" placeholder="Username"
class="form-control mb-3">
        <input type="password" ng-model="password"
placeholder="Password" class="form-control mb-3">
        <button class="btn btn-primary"
ng-click="login()">Login</button>
        <button class="btn btn-secondary"
ng-click="goToRegister()">Register</button>
        <p class="error-message" ng-if="!success">{{ message }}</p>
        <p class="success-message" ng-if="success">{{ message }}</p>
    </div>
</body>
</html>
```

register.html

```
<!--
    Author: Timur Rakhimov
    Date: 2024-11-12
    Description: User registration page for the Course Selection app
using AngularJS and Bootstrap for styling.
-->

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>User Registration</title>
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.
min.css">
    <style>
        /* General page styling */
        body {
            background-color: #e9ecef;
```

```
        font-family: Arial, sans-serif;
    }
    /* Container for the registration form */
    .register-container {
        max-width: 450px;
        margin: 80px auto;
        padding: 30px;
        background-color: #fff;
        border-radius: 10px;
        box-shadow: 0 4px 12px rgba(0, 0, 0, 0.1);
    }
    /* Header styling */
    .register-header {
        text-align: center;
        color: #007bff;
        margin-bottom: 20px;
    }
    /* Form control styling */
    .form-control {
        border-radius: 20px;
    }
    /* Primary button styling */
    .btn-primary {
        border-radius: 20px;
        width: 100%;
    }
    /* Secondary button styling */
    .btn-secondary {
        border-radius: 20px;
        width: 100%;
        margin-top: 10px;
    }
    /* Error message styling */
    .error-message {
        color: #dc3545;
        text-align: center;
        margin-top: 10px;
    }
    /* Success message styling */
```

```
        .success-message {
            color: #28a745;
            text-align: center;
            margin-top: 10px;
        }
    </style>
    <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.
js"></script>
    <!-- Application script -->
    <script src="app.js"></script>
</head>
<body ng-app="courseApp" ng-controller="RegisterController">
    <!-- Registration Form Container -->
    <div class="register-container">
        <h2 class="register-header">User Registration</h2>

        <!-- Username Input -->
        <input type="text" ng-model="username" placeholder="Username"
class="form-control mb-3" required>

        <!-- Password Input -->
        <input type="password" ng-model="password"
placeholder="Password" class="form-control mb-3" required>

        <!-- Register Button -->
        <button class="btn btn-primary"
ng-click="register()">Register</button>

        <!-- Back to Login Button -->
        <button class="btn btn-secondary" ng-click="goToLogin()">Back to
Login</button>

        <!-- Error and Success Messages -->
        <p class="error-message" ng-if="!success">{{ message }}</p>
        <p class="success-message" ng-if="success">{{ message }}</p>
    </div>
</body>
</html>
```

courses.html

```
<!--
   Author: Timur Rakhimov
   Date: 2024-11-12
   Description: HTML page for managing and displaying courses. Users
can view, add, and delete courses.
-->

<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width,
initial-scale=1.0">
   <title>Available Courses</title>
   <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.
min.css">
   <style>
       /* General page styling */
       body {
           background-color: #e9ecef;
       }
       /* Styling for the navigation bar */
       .nav-bar {
           background-color: #007bff;
           padding: 10px;
       }
       /* Links in the navigation bar */
       .nav-bar a {
           color: #fff;
           margin-right: 15px;
           text-decoration: none;
           font-weight: bold;
       }
       /* Main container for courses */
       .courses-container {
           max-width: 600px;
           margin: 20px auto;
```

```
        padding: 20px;
        background-color: #fff;
        border-radius: 10px;
        box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
    }
    /* Styling for individual course cards */
    .course-card {
        background-color: #e2e6ea;
        border-radius: 10px;
        padding: 15px;
        margin-bottom: 15px;
        display: flex;
        justify-content: space-between;
        align-items: center;
    }
    /* Course title styling */
    .course-title {
        color: #007bff;
        font-weight: bold;
    }
    /* Delete button styling */
    .btn-danger {
        border-radius: 20px;
    }
    /* Logout button styling */
    .logout-btn {
        background-color: #dc3545;
        color: #fff;
        border: none;
        border-radius: 20px;
        padding: 10px 20px;
        cursor: pointer;
    }
    /* Hover effect for the logout button */
    .logout-btn:hover {
        background-color: #c82333;
    }
    /* Form styling for adding new courses */
    .add-course-form {
```

```
            margin-bottom: 20px;
        }
        /* Input field styling */
        .add-course-input {
            border-radius: 20px;
        }
    </style>
    <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.8.2/angular.min.
js"></script>
    <!-- Application script -->
    <script src="app.js"></script>
</head>
<body ng-app="courseApp" ng-controller="CoursesController">
    <!-- Navigation Bar -->
    <div class="nav-bar">
        <a href="#" ng-click="loadCourses()">Courses</a>
        <button class="logout-btn" ng-click="logout()">Logout</button>
    </div>

    <!-- Add Course Form -->
    <div class="courses-container">
        <h2>Add New Course</h2>
        <div class="add-course-form">
            <input type="text" ng-model="newCourseId"
placeholder="Course ID" class="form-control mb-2 add-course-input">
            <input type="text" ng-model="newCourseName"
placeholder="Course Name" class="form-control mb-2 add-course-input">
            <button class="btn btn-primary btn-block"
ng-click="addCourse()">Add Course</button>
        </div>

        <!-- Courses List -->
        <h2>Available Courses</h2>
        <div ng-if="courses.length === 0" class="no-courses">
            No courses available.
        </div>
        <div class="course-card" ng-repeat="course in courses">
```

```
            <span class="course-title">{{ course.courseId }}: {{
course.courseName }}</span>
            <button class="btn btn-danger"
ng-click="deleteCourse(course.courseId)">Delete</button>
        </div>
    </div>
</body>
</html>
```

app.js

```
/**
* Author: Timur Rakhimov
* Date: 2024-11-12
* Description: AngularJS client-side script for handling user
registration and navigation.
*/


// Define the AngularJS module for the Course Selection app
const app = angular.module("courseApp", []);

// Main Controller for Login
app.controller("MainController", function ($scope, $http) {
    $scope.message = "";
    $scope.success = false;

    // Function to handle user login
    $scope.login = function () {
        // Prepare login data with trimmed username and password
        const loginData = {
            username: $scope.username.trim(),
            password: $scope.password.trim(),
        };

        $http.post("/login", loginData)
            .then((response) => {
                // If login is successful, store the username and
redirect to courses page
```

```
                if (response.data.success) {
                    localStorage.setItem("username", $scope.username);
                    $scope.message = "Login successful! Redirecting...";
                    $scope.success = true;
                    setTimeout(() => {
                        window.location.href = "/courses.html";
                    }, 1000);
                } else {
                    // Display error message if login fails
                    $scope.message = "Login failed. Please check your
credentials.";
                    $scope.success = false;
                }
            })
            .catch((error) => {
                // Handle server error
                console.error("Server error:", error);
                $scope.message = "Server error. Please try again
later.";
            });
    };

    // Function to navigate to the registration page
    $scope.goToRegister = function () {
        window.location.href = "/register.html";
    };
});

// Register Controller for User Registration
app.controller("RegisterController", function ($scope, $http) {
    $scope.message = "";
    $scope.success = false;
    // Function to handle user registration
    $scope.register = function () {
        // Prepare registration data with trimmed username and password
         const registrationData = {
             username: $scope.username.trim(),
             password: $scope.password.trim(),
         };
```

```
        console.log("Sending registration data:", registrationData);
        $http.post("/register", registrationData)
            .then((response) => {
                console.log("Server response:", response.data);
                // If registration is successful, display success
message and redirect to login page
                if (response.data.success) {
                    $scope.message = "Registration successful! You can
now log in.";
                    $scope.success = true;
                    setTimeout(() => {
                        window.location.href = "/index.html";
                    }, 1000);
                } else {
                   // Display error message if registration fails
                    $scope.message = response.data.message ||
"Registration failed. Please try again.";
                    $scope.success = false;
                }
            })
            .catch((error) => {
               // Handle server error
                console.error("Server error:", error);
                $scope.message = "Server error during registration.
Please try again later.";
            });
    };

    // Function to navigate back to the login page
    $scope.goToLogin = function () {
        window.location.href = "/index.html";
    };
});

// Main Controller for Courses Page
app.controller("CoursesController", function ($scope, $http) {
    $scope.courses = [];
    $scope.newCourseId = "";
    $scope.newCourseName = "";
```

```
    const username = localStorage.getItem("username");
    // Function to load all courses for the logged-in user
    $scope.loadCourses = function () {
       const username = localStorage.getItem("username");

       $http.get("/courses", { params: { username } })
           .then((response) => {
               console.log("Full response from backend:", response);

               // Check if the response data is an array and contains
courses
               if (Array.isArray(response.data)) {
                   $scope.courses = response.data;
                   console.log("Loaded courses:", $scope.courses);
               } else {
                   // If the response format is unexpected, log the
issue
                   console.log("Unexpected response format:",
response.data);
                   $scope.courses = [];
               }
           })
           .catch((error) => {
               // Handle error when fetching courses
               console.error("Error fetching courses:", error);
               alert("Failed to load courses. Please try again
later.");
           });
    };


    // Function to add a new course
    $scope.addCourse = function () {
       // Validate input fields before making the request
        if ($scope.newCourseId && $scope.newCourseName) {
            const newCourse = {
                username,
                courseId: $scope.newCourseId.trim(),
                courseName: $scope.newCourseName.trim(),
```

```
            };
            $http.post("/api/addCourse", newCourse)
                .then((response) => {
                   console.log("Server response:", response.data);
                   // If course is added successfully, reload the
courses list
                    if (response.data.success) {
                        alert("Course added successfully!");
                        $scope.loadCourses();
                        $scope.newCourseId = "";
                        $scope.newCourseName = "";
                    } else {
                        alert("Failed to add course. Please try
again.");
                    }
                })
                .catch((error) => {
                   // Handle error when adding course
                    console.error("Error adding course:", error);
                    alert("Error adding course. Please try again.");
                });
        } else {
           // Alert user if input fields are empty
            alert("Please enter both Course ID and Course Name.");
        }
    };
    // Function to delete a course
    $scope.deleteCourse = function (courseId) {
        $http.delete(`/api/deleteCourse/${courseId}`, { params:{
username }})
            .then((response) => {
               // If course is deleted successfully, reload the courses
list
                if (response.data.success) {
                    alert("Course deleted successfully!");
                    $scope.loadCourses();
                } else {
                    alert("Failed to delete course. Please try
again.");
```

```
                }
            })
            .catch((error) => {
               // Handle error when deleting course
                console.error("Error deleting course:", error);
                alert("Error deleting course. Please try again.");
            });
    };

    // Function to log out the user
    $scope.logout = function () {
         localStorage.removeItem("username");
         window.location.href = "/index.html";
    };
    // Load courses when the controller is initialized
    $scope.loadCourses();
});
```

server.js

```
/**
* Author: Timur Rakhimov
* Date: 2024-11-12
* Description: Node.js server using Express and MongoDB for a course
selection application.
* Provides routes for user registration, login, and course management
(view, add, delete).
*/

const express = require("express");
const mongoose = require("mongoose");
const bodyParser = require("body-parser");
const path = require("path");

const app = express();
const PORT = 3000;

// Connect to MongoDB
mongoose.connect("mongodb://127.0.0.1:27017/courseSelectionDB", {
```

```
    useNewUrlParser: true,
    useUnifiedTopology: true,
});

app.use(bodyParser.json());
app.use(express.static("public"));

// Define Mongoose models
const User = require("./models/userModel.js");
const Course = mongoose.model("Course", { courseId: String, courseName:
String });

// Route to handle user login
app.post("/login", async (req, res) => {
    const { username, password } = req.body;

    try {
        // Find user by username and password
        const user = await User.findOne({ username, password });
        if (user) {
            res.json({ success: true, username });
        } else {
            res.json({ success: false, message: "Invalid username or
password." });
        }
    } catch (error) {
        // Handle server errors
        res.status(500).json({ success: false, message: "Server error
during login." });
    }
});


// Define the registration route
app.post("/register", async (req, res) => {
    const { username, password } = req.body;

    try {
        // Check if the username already exists
```

```
        const existingUser = await User.findOne({ username });
        if (existingUser) {
            return res.json({ success: false, message: "Username already
exists." });
        }

        // Create a new user with an empty courses array
        const newUser = new User({ username, password, courses: [] });
        await newUser.save();
        res.json({ success: true, message: "Registration successful!"
});
    } catch (error) {
        // Handle server errors
        console.error("Error during registration:", error);
        res.status(500).json({ success: false, message: "Server error
during registration." });
    }
});


// Route to get all courses for a specific user
app.get("/courses", async (req, res) => {
    const { username } = req.query;

    try {
        console.log("Fetching courses for user:", username);

        // Find the user by username and select only the "courses" field
        const user = await User.findOne({ username }).select("courses");
        console.log("User data fetched from MongoDB:", user);

        // Check if the user exists
        if (!user) {
            return res.status(404).json({ success: false, message: "User
not found." });
        }

        // Return the user's courses or an empty array if none exist
        res.status(200).json(user.courses || []);
```

```
    } catch (error) {
        // Handle server errors
        console.error("Error during course fetching:", error);
        res.status(500).json({ success: false, message: "Server error
fetching courses." });
    }
});


// Route to add a new course for a specific user
app.post("/api/addCourse", async (req, res) => {
    const { username, courseId, courseName } = req.body;

    try {
        // Find the user by username
        const user = await User.findOne({ username });
        if (!user) {
            return res.json({ success: false, message: "User not found."
});
        }

        // Initialize the courses array if it doesn't exist
        if (!user.courses) {
            user.courses = [];
        }

        // Check if the course already exists in the user's courses
        const existingCourse = user.courses.find((course) =>
course.courseId === courseId);
        if (existingCourse) {
            return res.json({ success: false, message: "Course already
exists." });
        }

        // Add the new course to the user's courses array
        user.courses.push({ courseId, courseName });
        await user.save();
        res.json({ success: true, message: "Course added successfully!"
});
```

```
    } catch (error) {
        // Handle server errors
        console.error("Error adding course:", error);
        res.status(500).json({ success: false, message: "Server error
adding course." });
    }
});


// Route to delete a course for a specific user
app.delete("/api/deleteCourse/:courseId", async (req, res) => {
    const { username } = req.query;
    const courseId = req.params.courseId;

    try {
        // Find the user by username
        const user = await User.findOne({ username });
        if (!user) {
            return res.json({ success: false, message: "User not found."
});
        }

        // Remove the course from the user's courses array
        user.courses = user.courses.filter((course) => course.courseId
!== courseId);
        await user.save();
        res.json({ success: true, message: "Course deleted
successfully!" });
    } catch (error) {
        // Handle server errors
        res.status(500).json({ success: false, message: "Server error
deleting course." });
    }
});


// Start the server
app.listen(PORT, () => {
    console.log(`Server running at http://localhost:${PORT}`);
```

```
});
```

userModel.js

```
/**
* Author: Timur Rakhimov
* Date: 2024-11-12
* Description: Mongoose model for User collection, including user
details and enrolled courses.
*/

const mongoose = require("mongoose");

// Define the Course schema (sub-document of User)
const courseSchema = new mongoose.Schema({
   courseId: String,
   courseName: String,
});

// Define the User schema
const userSchema = new mongoose.Schema({
   username: { type: String, required: true, unique: true },
   password: { type: String, required: true },
   courses: [courseSchema], // Explicitly define the courses array
});

// Create the User model from the schema
const User = mongoose.model("User", userSchema);
// Export the User model to be used in other parts of the application
module.exports = User;
```

courseModel.js

```
/**
* Author: Timur Rakhimov
* Date: 2024-11-12
* Description: Mongoose model for the Course collection.
*/

const mongoose = require("mongoose");

// Define the Course schema
const courseSchema = new mongoose.Schema({
   courseId: String,
   courseName: String,
});

// Create and export the Course model
module.exports = mongoose.model("Course", courseSchema);
```

Challenges and Solutions

Database Structure Issues

Initially, there was confusion about whether to maintain a separate collection for courses or include courses as a subdocument array within the users collection. This caused inconsistencies and errors when fetching course data.

Solution

After evaluating the complexity, I decided to keep the courses as an embedded array in the users collection. This decision simplified data retrieval and aligned well with the overall project requirements. By restructuring the database schema, I ensured that each user has their own set of courses, making data management easier and reducing errors.

Data Fetching and Display Problems

While fetching course data for a user, there was an issue where the frontend did not display the courses correctly, even though the backend successfully retrieved the data from MongoDB. This problem was caused by the response data format not being handled properly in the AngularJS code.

Solution

To fix this, I carefully checked the backend response and adjusted the AngularJS controller logic to correctly handle the array of courses. I added explicit checks for the response data format and implemented mechanisms to handle unexpected responses. This resolved the issue and allowed the frontend to display the courses correctly.

Handling User Authentication

There was an error during user login due to mismatched credentials and inconsistent data retrieval from MongoDB. This resulted in repeated "Server error during login" messages.

Solution

I fixed this by refining the login route in the server code and improving error handling. Additionally, I ensured that the user credentials are properly trimmed and validated before querying the database. These adjustments improved the reliability of the login functionality and eliminated unnecessary error messages.