# Team 6 - Milestone #7

Andrew Wang, Keyu Lin, Lexie King, Shrey Bahadur

## Project Overview

Repository: https://github.com/voidpls/326_Course_Recommendation_System.git

Issues: https://github.com/voidpls/326_Course_Recommendation_System/issues

Milestones:

Description: The user will input the courses they have already taken, and optionally any academic or career interests they have, and the app will recommend future courses they might be interested in. The recommendations will also consider prerequisite/eligibility requirements to ensure only available courses are suggested. A visual CS major course progress map will show completed courses, current courses, and suggested future courses to help students plan their academic path.

Key Features:

- User Input: Students can enter courses taken, interests, and career goals.
- Course Recommendation: The system recommends appropriate courses based on inputs and provides course details (e.g., prerequisites, credits, schedule, etc.).
- Course Schedule Builder: Students can select courses and generate schedules, the system will indicate time conflicts and provide suggestions for adjustments.
- Course description search up: Students can search up interested courses through the course details tab.
- Course Reviews: Students can look at and leave reviews for courses

## Roles

Lexie King:

- Role: Project Manager
- Issues: General UI Design, Past Course Storage, Course Details, Course Recommendation Home Page

Shrey Bahadur:

Role: Note Taker

Issues: Course Review Page, User login data,

Andrew Wang:
Role: Integration Manager
Issues: Ensuring UI, data structures, and the various pages are compatible with each other. Course Progress Chart page.

Keyu Lin:

Role: User Experience Tester

Issues: User Data Structure, Course Recommendation Home Page, User Experience Tester

## Historical Timeline

Front-end hookup: Mar 1 – April 15

Back-end build & testing: Apr 15 – May 7

## Lexie - Assigned tasks & Pull Request

- Project manager
- General UI Design
- Course Recommendation Home Page UI
- Course Details Page UI
    - Course Data Structure
    - Course search within Course Details Page
- User Data Structure
- Past Course Storage
- Course Recommendation Algorithm
- Login Page
    - Login and create account features.

https://github.com/voidpls/326_Course_Recommendation_System/pull/100
https://github.com/voidpls/326_Course_Recommendation_System/pull/107

## Lexie - Completed Task Description

- Login-Page
- Back-end
  - Switched from in-memory arrays to a real SQLite database (data/app.db) via the sqlite3 package.
  - Express routes & controllers
    - Routes in api/routes/*.js mounted under /api:
      - GET /api/courses, GET /api/courses/:id
      - POST /api/users (signup), PUT /api/users/:id (profile update)
      - POST /api/preferences, GET /api/recommendations, DELETE /api/recommendations/:id
  - Controllers in api/controllers/ rewritten to use the sqlite3 driver.
  - Session-based authentication
  - Installed and configured express-session in server.js
  - Error handling & validation
  - Centralized error handler in server.js catches DB errors and returns { error: message } with appropriate HTTP codes.
  - Basic input checks (e.g. ensure username/password present before attempting login).
  - Integration & front-end hookup
    - Updated front-end fetch calls in main.js / Recommendations.vue to point at the new SQLite-backed endpoints.
  - Handled CORS and JSON parsing (app.use(cors()); app.use(express.json());).

## Lexie - Features

Features Implemented:

User can Create Account (with unique-username check)

User can Log In (with bcrypt-protected passwords)

Guest Mode (stores a random ID in localStorage)

Completion Progress

Front-End:  forms, view toggles, fetch calls, alerts, redirects

Back-End:  /api/signup, /api/login, db.js, session handling, SQLite storage, user.db, auth_controller.js, auth.js, user profile.js

Git Branch : login-page

## Lexie: Code Structure & Organization

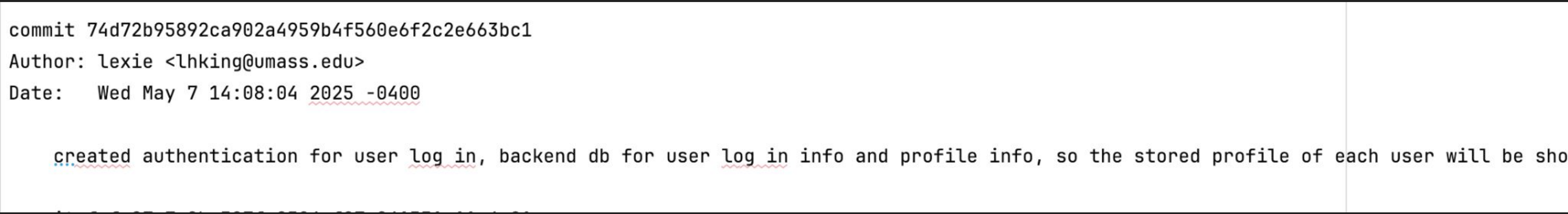
Separation of Concerns:

- Front-End lives entirely under /src (HTML, CSS, client JS)
- Back-End lives under /server (Express, SQLite, bcrypt, sessions)

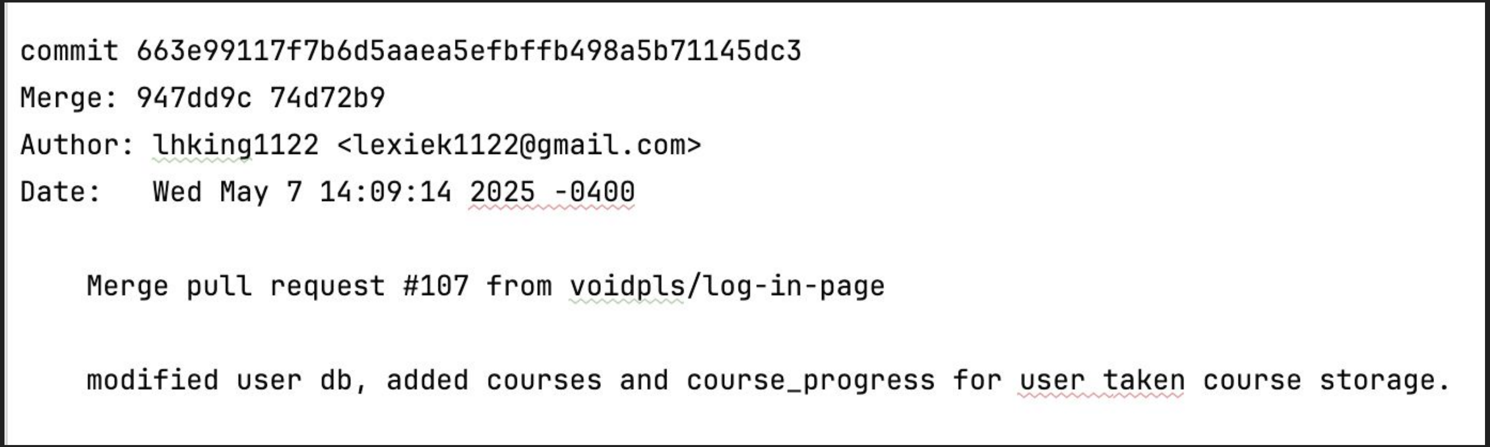
Key Components:

- login.html (UI + event listeners)
- server.js (API routes + middleware)
- routes/ (modular course-related endpoints)

# Lexie Work Summary - commits

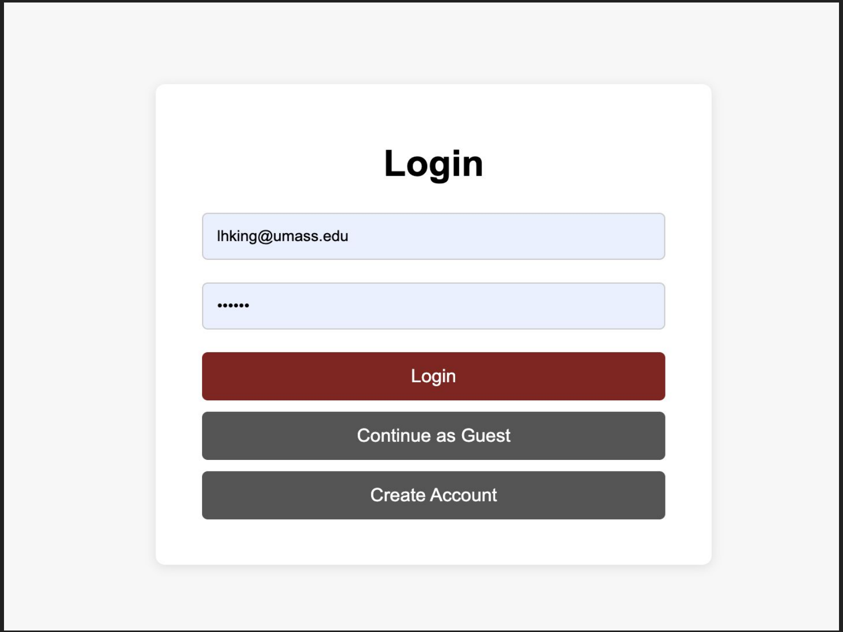


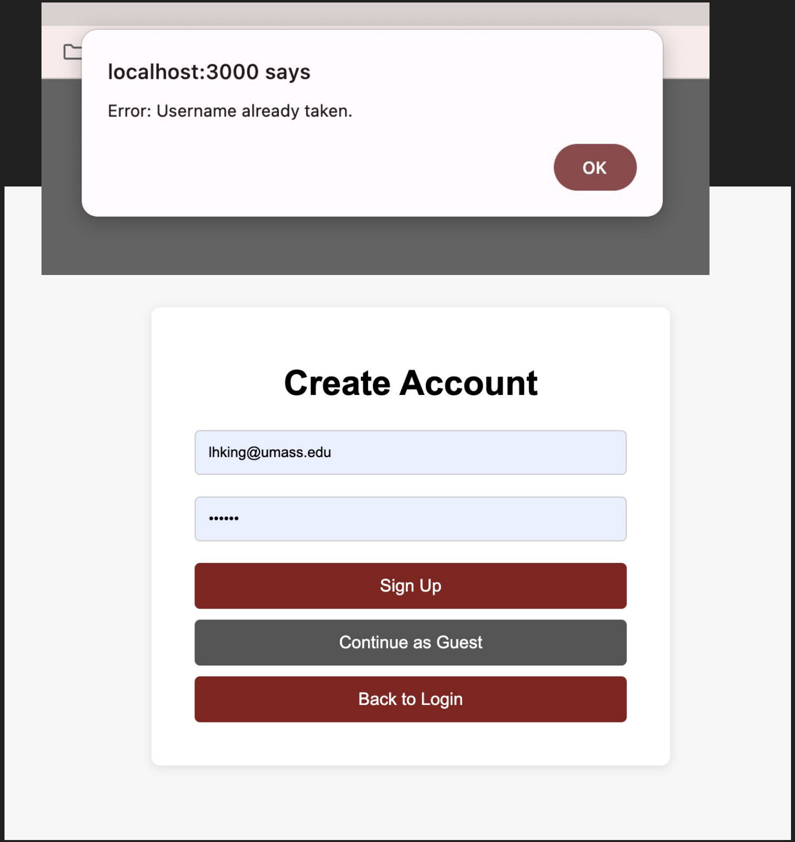
```
commit 74d72b95892ca902a4959b4f560e6f2c2e663bc1
Author: lexie <lhking@umass.edu>
Date:   Wed May 7 14:08:04 2025 -0400

    created authentication for user log in, backend db for user log in info and profile info, so the stored profile of each user will be sho
```



```
commit 663e99117f7b6d5aaea5efbffb498a5b71145dc3
Merge: 947dd9c 74d72b9
Author: lhking1122 <lexiek1122@gmail.com>
Date:   Wed May 7 14:09:14 2025 -0400

    Merge pull request #107 from voidpls/log-in-page

    modified user db, added courses and course_progress for user taken course storage.
```

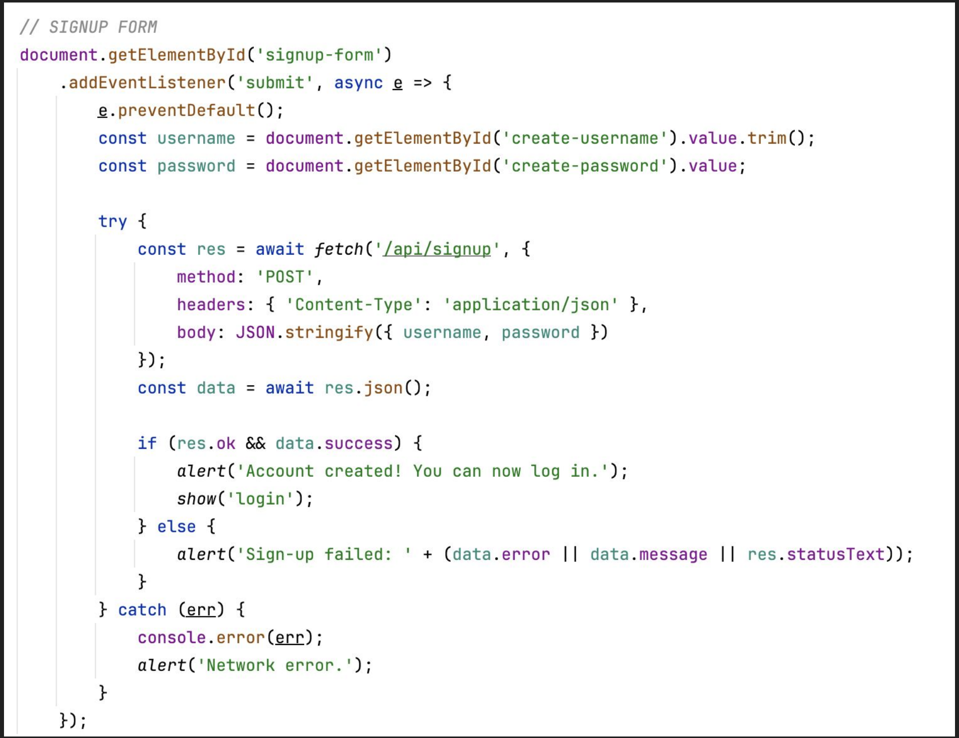
## Lexie - Screen Shots of UI Implementation #1




localhost:3000 says

Error: Username already taken.

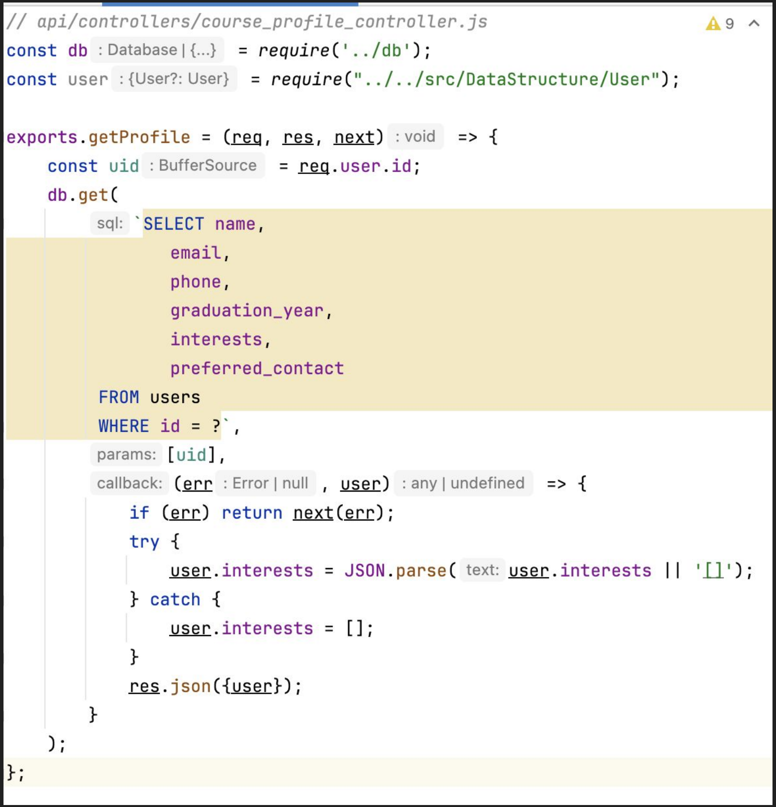
## Lexie - Code Snippet and Explanation (Front-End)



```
// SIGNUP FORM
document.getElementById('signup-form')
    .addEventListener('submit', async e => {
        e.preventDefault();
        const username = document.getElementById('create-username').value.trim();
        const password = document.getElementById('create-password').value;

        try {
            const res = await fetch('/api/signup', {
                method: 'POST',
                headers: { 'Content-Type': 'application/json' },
                body: JSON.stringify({ username, password })
            });
            const data = await res.json();

            if (res.ok && data.success) {
                alert('Account created! You can now log in.');
                show('login');
            } else {
                alert('Sign-up failed: ' + (data.error || data.message || res.statusText));
            }
        } catch (err) {
            console.error(err);
            alert('Network error.');
        }
    });
```

- The fetch('/api/signup', …) call hits the Express route, which creates the user in your SQLite DB and responds { success: true }.
- Async/Await & Error Handling
- async/await makes the code read top-to-bottom.
- The try/catch block catches both JSON parsing errors and network failures.
- Inside the if, inspect both res.ok (HTTP 2xx) and data.success flag.
- UI FloOn success call show('login') ( helper that hides the signup form and shows the login form).
- On failure, pop up an alert explaining what went wrong.
- This pattern ensures the page never reloads, gives immediate feedback, and cleanly ties front-end form to the back-end signup logic.

# Lexie - Code Snippet and Explanation (Back-End)



```
// api/controllers/course_profile_controller.js                              ⚠9 ^
const db : Database | {...}  = require('../db');
const user : {User?: User}  = require("../../src/DataStructure/User");

exports.getProfile = (req, res, next) : void  => {
    const uid : BufferSource  = req.user.id;
    db.get(
        sql: `SELECT name,
                email,
                phone,
                graduation_year,
                interests,
                preferred_contact
         FROM users
         WHERE id = ?`,
        params: [uid],
        callback: (err : Error | null , user) : any | undefined  => {
            if (err) return next(err);
            try {
                user.interests = JSON.parse( text: user.interests || '[]');
            } catch {
                user.interests = [];
            }
            res.json({user});
        }
    );
};
```

- The getProfile function serves as the Express controller for the profile-view endpoint. It reads the authenticated user’s ID from req.user.id, then executes a SELECT query via db.get to retrieve the corresponding row (name, email, phone, graduation_year, interests, and preferred_contact). Any error during the database call is forwarded to the next middleware. After fetching, it attempts to parse the interests field (stored as a JSON string) into an array—defaulting to an empty list if parsing fails—and finally sends the resulting user object back in the JSON response.

## Lexie -  Challenges and Insights

During Milestone 7, we hit a few bumps. Moving from in-memory arrays to SQLite meant planning our tables carefully and handling callback functions properly. We also had to adjust CORS and cookie settings so sessions worked between front-end and back-end. Parsing JSON fields like interests taught us to include safe defaults in case parsing fails. Converting our sqlite3 calls to use promises let us write cleaner async/await code and made testing much easier. On the teamwork side, managing database updates across multiple branches meant doing lots of code reviews and short check-in meetings. We helped one another write basic tests and stayed in close contact. Working together closely made integration smoother and kept our code organized and well-tested.

## Lexie - Future Improvements & Next Steps

Enhance the recommendation engine by tracking more user signals—such as prereqs, course views, and ratings—and using those to weight suggestions dynamically. On the back end, make data structure will let us experiment with simple collaborative-filtering or hybrid ranking models without slowing down API responses.

## Lexie - Links to Issues

https://github.com/voidpls/326_Course_Recommendation_System/issues/90

https://github.com/voidpls/326_Course_Recommendation_System/issues/89

https://github.com/voidpls/326_Course_Recommendation_System/issues/79

https://github.com/voidpls/326_Course_Recommendation_System/issues/49

## Shrey Work Summary

- Set up the courses and reviews tables as well as sending local json course data to the db
- Implemented a get endpoint to get course details as well as a get endpoint to get the average review score for a course
- Rewrote the controllers for the review endpoints to connect them with the db
- Fixed some of the review page js file

All Issues were closed

PR links: https://github.com/voidpls/326_Course_Recommendation_System/pull/128

https://github.com/voidpls/326_Course_Recommendation_System/pull/120

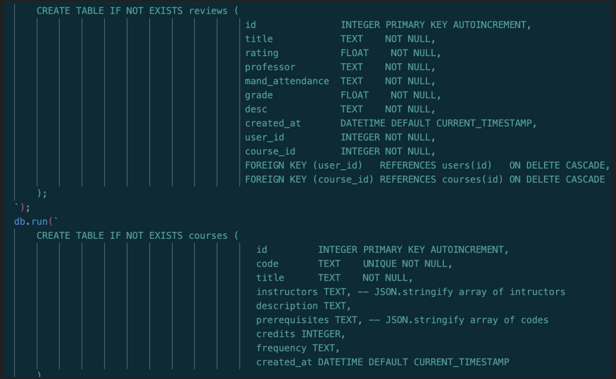
https://github.com/voidpls/326_Course_Recommendation_System/pull/115

https://github.com/voidpls/326_Course_Recommendation_System/pull/110

Work done in the shrey/db branch and shrey/db2 branch

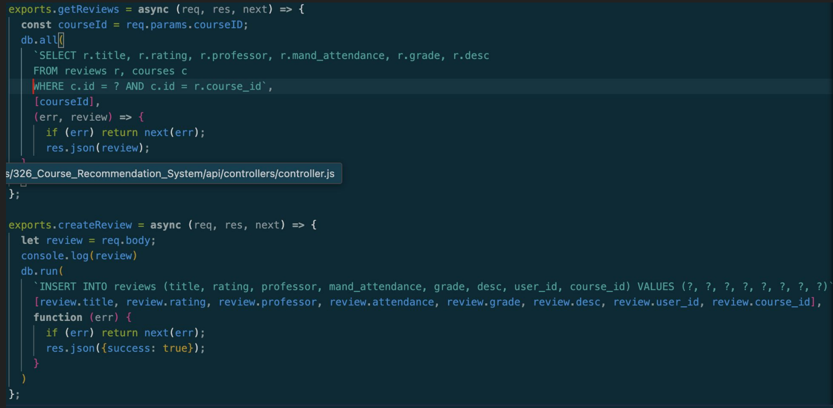
# Shrey Feature Demonstration

## Database set up



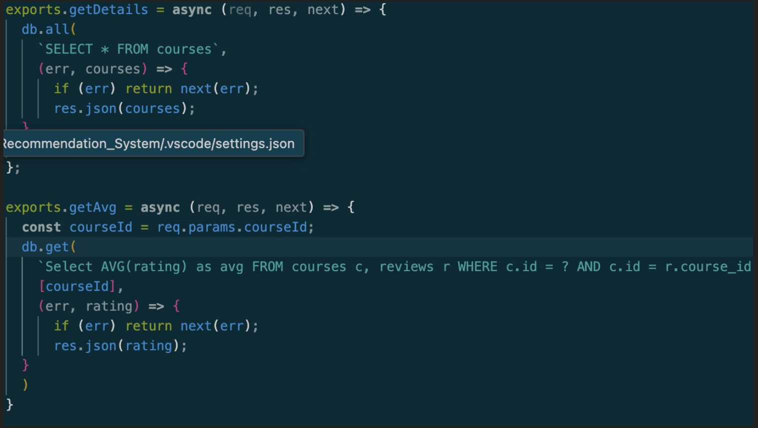
```
    CREATE TABLE IF NOT EXISTS reviews (
                                          id                INTEGER PRIMARY KEY AUTOINCREMENT,
                                          title             TEXT    NOT NULL,
                                          rating            FLOAT    NOT NULL,
                                          professor         TEXT    NOT NULL,
                                          mand_attendance   TEXT    NOT NULL,
                                          grade             FLOAT    NOT NULL,
                                          desc              TEXT    NOT NULL,
                                          created_at        DATETIME DEFAULT CURRENT_TIMESTAMP,
                                          user_id           INTEGER NOT NULL,
                                          course_id         INTEGER NOT NULL,
                                          FOREIGN KEY (user_id)   REFERENCES users(id)   ON DELETE CASCADE,
                                          FOREIGN KEY (course_id) REFERENCES courses(id) ON DELETE CASCADE
    );
`);
db.run(`
    CREATE TABLE IF NOT EXISTS courses (
                                            id         INTEGER PRIMARY KEY AUTOINCREMENT,
                                            code       TEXT    UNIQUE NOT NULL,
                                            title      TEXT    NOT NULL,
                                            instructors TEXT, -- JSON.stringify array of intructors
                                            description TEXT,
                                            prerequisites TEXT, -- JSON.stringify array of codes
                                            credits INTEGER,
                                            frequency TEXT,
                                            created_at DATETIME DEFAULT CURRENT_TIMESTAMP
```

## Reviews controllers



```
exports.getReviews = async (req, res, next) => {
  const courseId = req.params.courseID;
  db.all(
    `SELECT r.title, r.rating, r.professor, r.mand_attendance, r.grade, r.desc
    FROM reviews r, courses c
    WHERE c.id = ? AND c.id = r.course_id`,
    [courseId],
    (err, review) => {
      if (err) return next(err);
      res.json(review);
s/326_Course_Recommendation_System/api/controllers/controller.js
};

exports.createReview = async (req, res, next) => {
  let review = req.body;
  console.log(review)
  db.run(
    `INSERT INTO reviews (title, rating, professor, mand_attendance, grade, desc, user_id, course_id) VALUES (?, ?, ?, ?, ?, ?, ?, ?)`
    [review.title, review.rating, review.professor, review.attendance, review.grade, review.desc, review.user_id, review.course_id],
    function (err) {
      if (err) return next(err);
      res.json({success: true});
    }
  )
};
```

## Courses Controllers

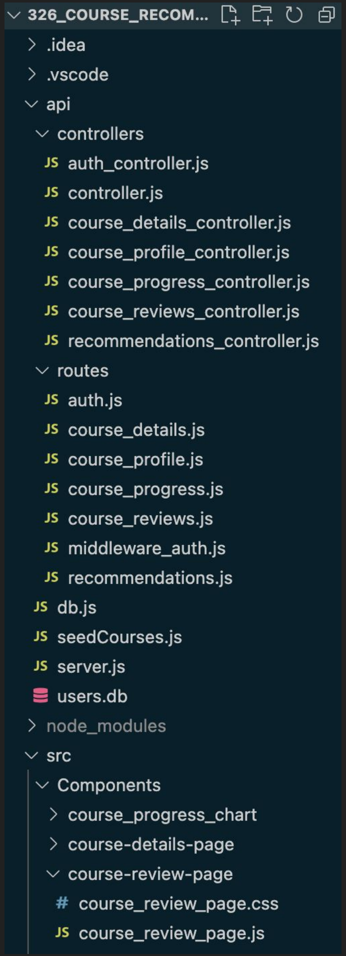


```
exports.getDetails = async (req, res, next) => {
  db.all(
    `SELECT * FROM courses`,
    (err, courses) => {
      if (err) return next(err);
      res.json(courses);
Recommendation_System/.vscode/settings.json
};

exports.getAvg = async (req, res, next) => {
  const courseId = req.params.courseId;
  db.get(
    `Select AVG(rating) as avg FROM courses c, reviews r WHERE c.id = ? AND c.id = r.course_id
    [courseId],
    (err, rating) => {
      if (err) return next(err);
      res.json(rating);
    }
  )
}
```

## Shrey Code structure

Our frontend is in the src folder and our backend is in our api folder. Our endpoints are broken down into the routes and controllers folders. Our db related stuff is also in the api folder. Our frontend pages components (css and js) are in the components folder within src.

The files I worked on this milestone are the controllers course_details and course_reviews as well as db.js and seedCourses.js. I also worked on my frontend js file, course_review_page.js.

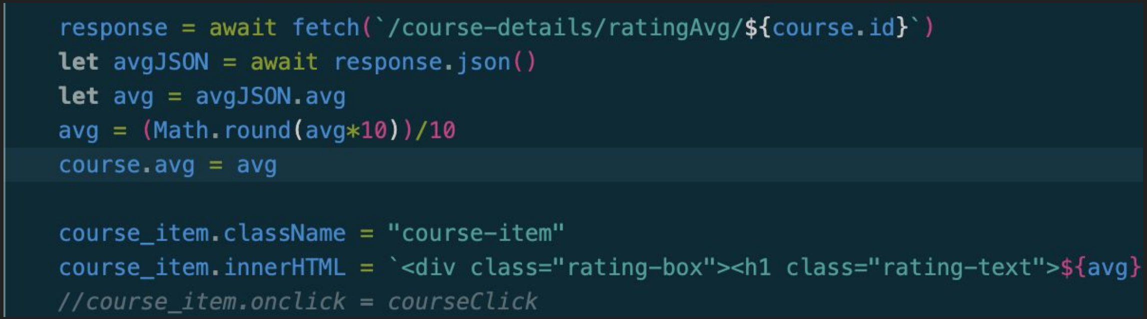


```
326_COURSE_RECOM...
  .idea
  .vscode
  api
    controllers
      auth_controller.js
      controller.js
      course_details_controller.js
      course_profile_controller.js
      course_progress_controller.js
      course_reviews_controller.js
      recommendations_controller.js
    routes
      auth.js
      course_details.js
      course_profile.js
      course_progress.js
      course_reviews.js
      middleware_auth.js
      recommendations.js
    db.js
    seedCourses.js
    server.js
    users.db
  node_modules
  src
    Components
      course_progress_chart
      course-details-page
      course-review-page
        course_review_page.css
        course_review_page.js
```

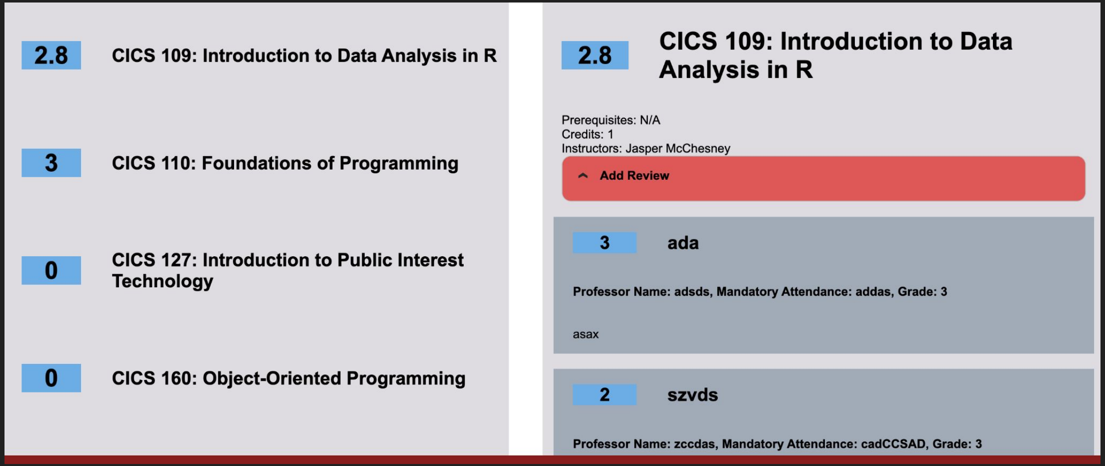
# Shrey Front-end Implementation

Added average review rating endpoint and integrated it into the frontend to display in the course list.

The biggest challenge faced when working on the frontend was removing static filler code and replacing it with data from the backend.



```
response = await fetch(`/course-details/ratingAvg/${course.id}`)
let avgJSON = await response.json()
let avg = avgJSON.avg
avg = (Math.round(avg*10))/10
course.avg = avg

course_item.className = "course-item"
course_item.innerHTML = `<div class="rating-box"><h1 class="rating-text">${avg}
//course_item.onclick = courseClick
```



2.8 CICS 109: Introduction to Data Analysis in R

3 CICS 110: Foundations of Programming

0 CICS 127: Introduction to Public Interest Technology

0 CICS 160: Object-Oriented Programming

2.8 CICS 109: Introduction to Data Analysis in R

Prerequisites: N/A
Credits: 1
Instructors: Jasper McChesney

Add Review

3 ada

Professor Name: adsds, Mandatory Attendance: addas, Grade: 3

asax

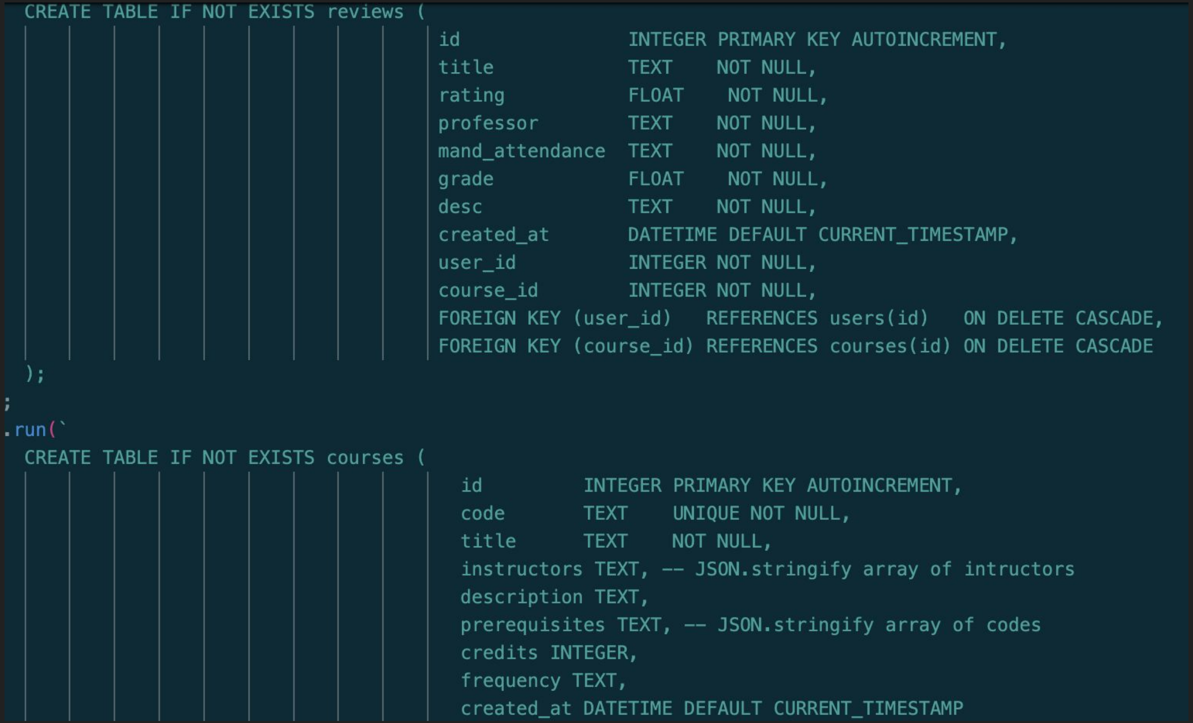
2 szvds

Professor Name: zccdas, Mandatory Attendance: cadCCSAD, Grade: 3

## Shrey Back-end Implementation

Created reviews and courses table. We then fill the courses table with web scraped data we have in a json.

The biggest challenge was thinking about how to structure the data based off of our needs and what constraints should be on the db/certain fields.



```
  CREATE TABLE IF NOT EXISTS reviews (
                                       id                INTEGER PRIMARY KEY AUTOINCREMENT,
                                       title             TEXT    NOT NULL,
                                       rating            FLOAT    NOT NULL,
                                       professor         TEXT    NOT NULL,
                                       mand_attendance   TEXT    NOT NULL,
                                       grade             FLOAT    NOT NULL,
                                       desc              TEXT    NOT NULL,
                                       created_at        DATETIME DEFAULT CURRENT_TIMESTAMP,
                                       user_id           INTEGER NOT NULL,
                                       course_id         INTEGER NOT NULL,
                                       FOREIGN KEY (user_id)   REFERENCES users(id)   ON DELETE CASCADE,
                                       FOREIGN KEY (course_id) REFERENCES courses(id) ON DELETE CASCADE
  );
;
.run(`
  CREATE TABLE IF NOT EXISTS courses (
                                         id          INTEGER PRIMARY KEY AUTOINCREMENT,
                                         code        TEXT    UNIQUE NOT NULL,
                                         title       TEXT    NOT NULL,
                                         instructors TEXT, -- JSON.stringify array of intructors
                                         description TEXT,
                                         prerequisites TEXT, -- JSON.stringify array of codes
                                         credits INTEGER,
                                         frequency TEXT,
                                         created_at DATETIME DEFAULT CURRENT_TIMESTAMP
```

## Shrey Challenges and Takeaways

The biggest challenge for me was creating the db structure as it required communication between everyone so we can understand what kind of data each of our pages needs and then turn that into a well structured database. My key takeaway is that effective communication is important when setting up changes that affect the entire application as poor communication can bog you down or lead to future problems.

## Future improvements

A future improvement that can be made is the ability for users to delete their reviews. Although I have some baselines code for it, I never got to properly integrate that feature which I think can be useful for users.

https://github.com/voidpls/326_Course_Recommendation_System/issues/133

## Andrew Work Summary

**For this milestone I did the following:**

- Implement an LLM recommendations endpoint (POST /recommendations) that interacts with the Google Gemini API to generate a list of recommended courses to take
- Add course progress backend endpoints to save data to SQLite DB (as opposed to dummy in-memory data store)
- Modify course progress frontend to persist data to backend, and to prioritize backend persisted data over local
- Add LLM course recommendations functionality to homepage

**PRs Closed:**

- https://github.com/voidpls/326_Course_Recommendation_System/pull/104
- https://github.com/voidpls/326_Course_Recommendation_System/pull/112
- https://github.com/voidpls/326_Course_Recommendation_System/pull/116
- https://github.com/voidpls/326_Course_Recommendation_System/pull/122
- https://github.com/voidpls/326_Course_Recommendation_System/pull/126
- https://github.com/voidpls/326_Course_Recommendation_System/pull/132

# Andrew Work Summary

**The issues that I resolved for this milestone are:**

Persist user inputs on course selection page through backend
https://github.com/voidpls/326_Course_Recommendation_System/issues/78

Migrate LLM API from dummy in-memory data store to SQLite DB
https://github.com/voidpls/326_Course_Recommendation_System/issues/121

Migrate from dummy data store to SQLite database
https://github.com/voidpls/326_Course_Recommendation_System/issues/113

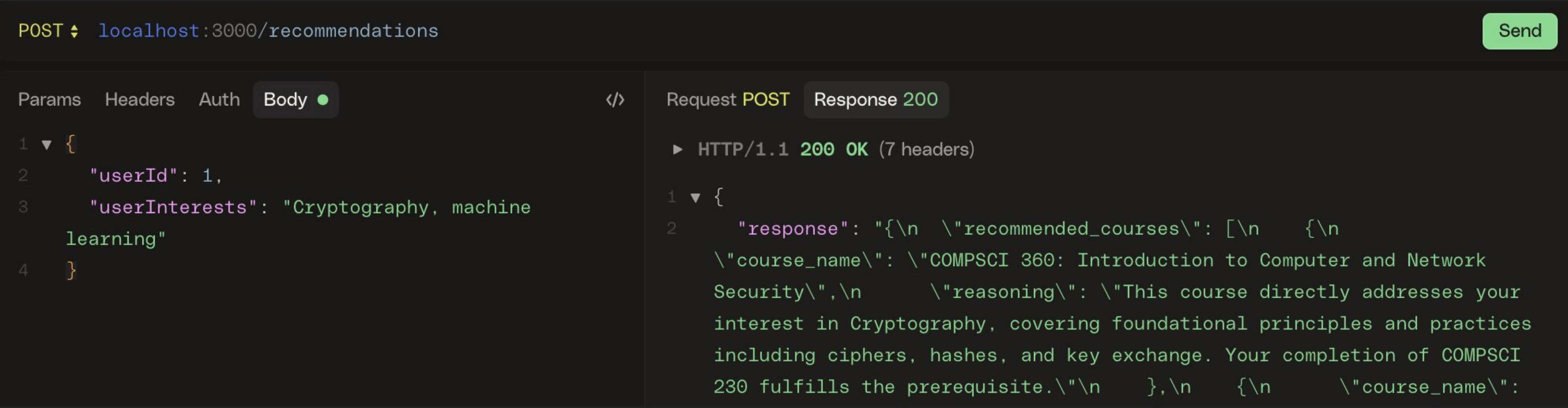
Back-end user course handling
https://github.com/voidpls/326_Course_Recommendation_System/issues/79

Course Recommendation Home Page
https://github.com/voidpls/326_Course_Recommendation_System/issues/7

# Andrew Work Features

**Feature:** Implement an LLM recommendations endpoint (llm-api branch)

User queries endpoint with userId and list of interests, API pulls user course progress from database, complete course list from local filesystem, and queries the Google Gemini API with all of the information. The Gemini API returns a structured output (JSON) response, which is passed on by this API.



```
POST localhost:3000/recommendations
```

```
{
  "userId": 1,
  "userInterests": "Cryptography, machine learning"
}
```

```
HTTP/1.1 200 OK (7 headers)
{
  "response": "{\n  \"recommended_courses\": [\n    {\n      \"course_name\": \"COMPSCI 360: Introduction to Computer and Network Security\",\n      \"reasoning\": \"This course directly addresses your interest in Cryptography, covering foundational principles and practices including ciphers, hashes, and key exchange. Your completion of COMPSCI 230 fulfills the prerequisite.\"\n    },\n    {\n      \"course_name\":
```

## Andrew Work Features

**Feature:** Course progress backend persistence functionality on frontend + backend (course_selection branch)

For backend, new endpoints for fetching from or updating local SQLite DB with course progress information. For frontend, the client queries backend for data for persistence, and prioritizes that over localstorage, if available. When the user submits the form, the client queries backend with updated data, and the backend saves to SQLite.

SQLite table:

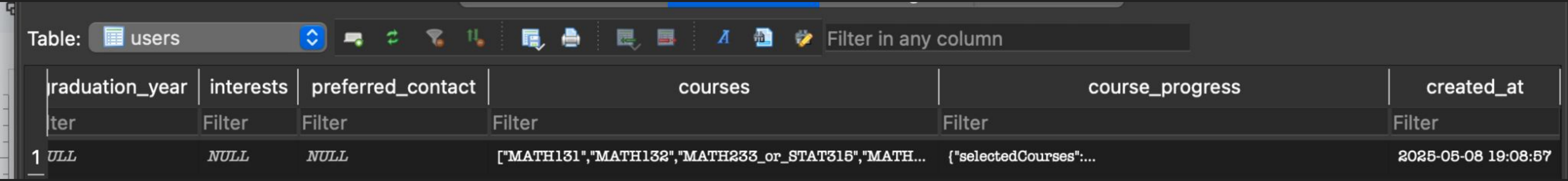


Table: users

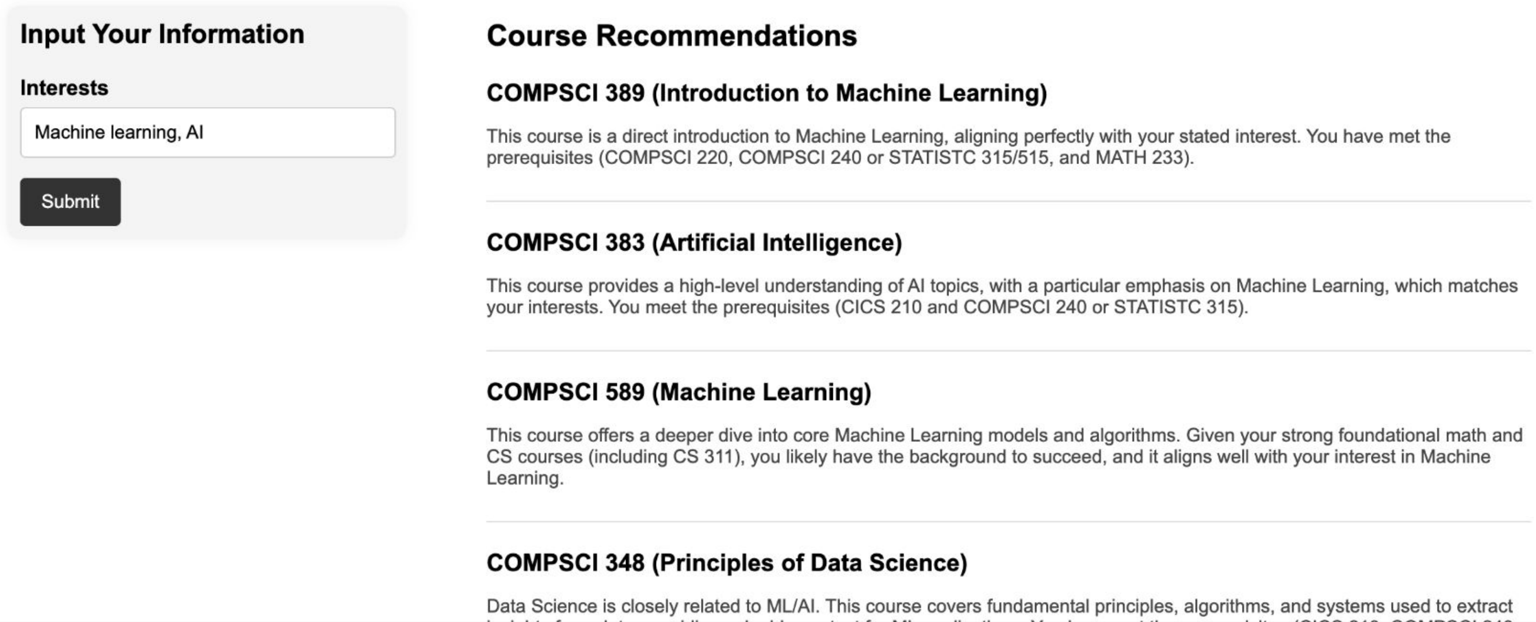
Filter in any column

| | raduation_year | interests | preferred_context | courses | course_progress | created_at |
|---|---|---|---|---|---|---|
| | lter | Filter | Filter | Filter | Filter | Filter |
| 1 | *ULL* | *NULL* | *NULL* | ["MATH131","MATH132","MATH233_or_STAT315","MATH... | {"selectedCourses":... | 2025-05-08 19:08:57 |

# Andrew Work Features

**Feature:** Add LLM course recommendations functionality to homepage (course_selection branch)

The homepage where users submits interests for course recommendations is now functional. The user submits their interests, and a POST request to my LLM Recommendation endpoint is made with the interests and userId. The server returns the recommendations and the page is updated with the list.



Course Recommendations

**COMPSCI 389 (Introduction to Machine Learning)**

This course is a direct introduction to Machine Learning, aligning perfectly with your stated interest. You have met the prerequisites (COMPSCI 220, COMPSCI 240 or STATISTC 315/515, and MATH 233).

**COMPSCI 383 (Artificial Intelligence)**

This course provides a high-level understanding of AI topics, with a particular emphasis on Machine Learning, which matches your interests. You meet the prerequisites (CICS 210 and COMPSCI 240 or STATISTC 315).

**COMPSCI 589 (Machine Learning)**

This course offers a deeper dive into core Machine Learning models and algorithms. Given your strong foundational math and CS courses (including CS 311), you likely have the background to succeed, and it aligns well with your interest in Machine Learning.

**COMPSCI 348 (Principles of Data Science)**

Data Science is closely related to ML/AI. This course covers fundamental principles, algorithms, and systems used to extract

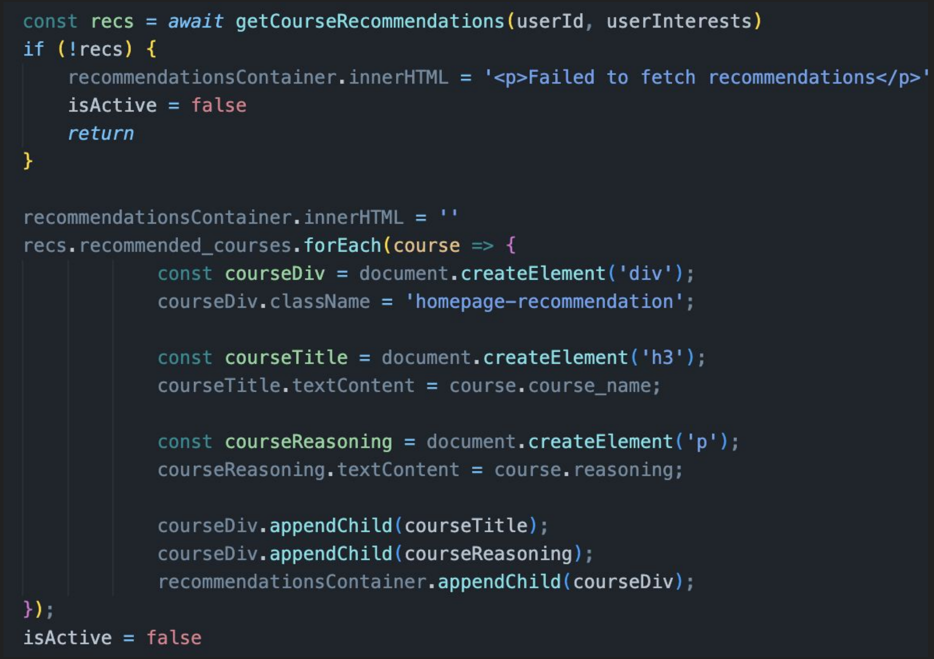
## Andrew Work Code Organization

Backend functionality (eg. API for updating database, API for LLM recommendations) are done under our /api directory, where our backend database functionality also lies.

Frontend functionality (eg. frontend persistence calls, accessing LLM API) are done in under our /src directory.

There is a key separation in that we try to do most of the processing, especially with user data, on the backend, while providing the frontend an easy-to-interact with API to simplify a lot of operations.

# Andrew Work Front End Implementation

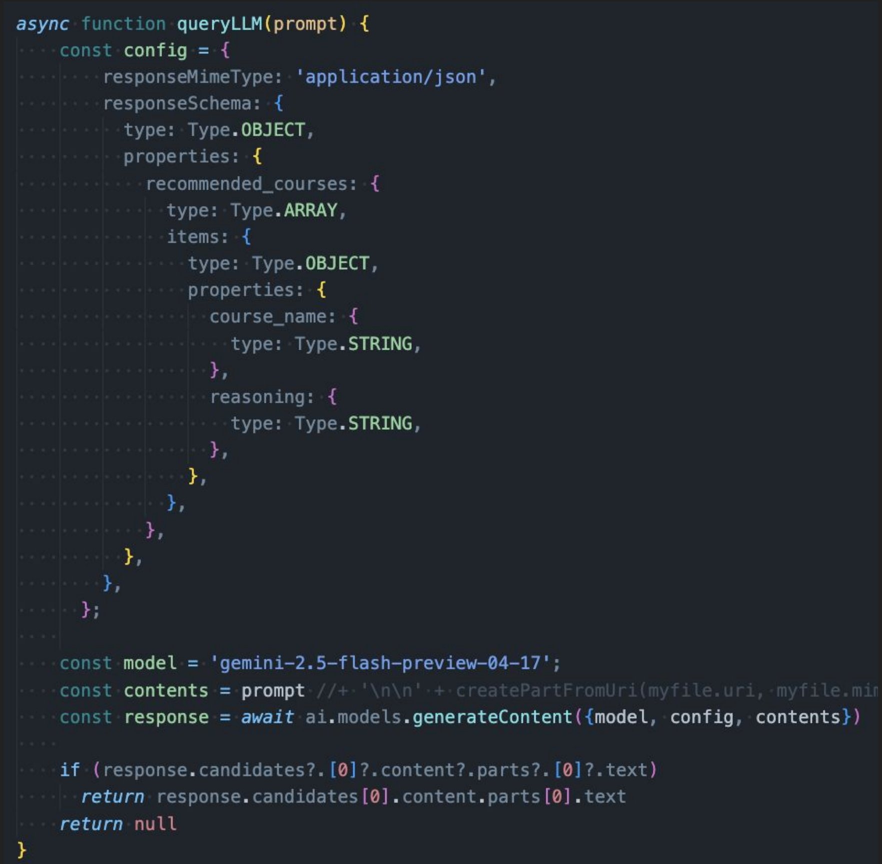


```
const recs = await getCourseRecommendations(userId, userInterests)
if (!recs) {
    recommendationsContainer.innerHTML = '<p>Failed to fetch recommendations</p>'
    isActive = false
    return
}

recommendationsContainer.innerHTML = ''
recs.recommended_courses.forEach(course => {
            const courseDiv = document.createElement('div');
            courseDiv.className = 'homepage-recommendation';

            const courseTitle = document.createElement('h3');
            courseTitle.textContent = course.course_name;

            const courseReasoning = document.createElement('p');
            courseReasoning.textContent = course.reasoning;

            courseDiv.appendChild(courseTitle);
            courseDiv.appendChild(courseReasoning);
            recommendationsContainer.appendChild(courseDiv);
});
isActive = false
```

This code is for our homepage/recommendations page. It fetches recommendations from our LLM API, using a helper function, and if successful, it appends an element for each recommendation.

Since the LLM call takes ~30 seconds to complete, I also have an isActive boolean variable that essentially disables the button/function if a call is already being made. This prevents duplicate calls to our API.

# Andrew Work Back End Implementation



```
async function queryLLM(prompt) {
    const config = {
        responseMimeType: 'application/json',
        responseSchema: {
          type: Type.OBJECT,
          properties: {
            recommended_courses: {
              type: Type.ARRAY,
              items: {
                type: Type.OBJECT,
                properties: {
                  course_name: {
                    type: Type.STRING,
                  },
                  reasoning: {
                    type: Type.STRING,
                  },
                },
              },
            },
          },
        },
      };

    const model = 'gemini-2.5-flash-preview-04-17';
    const contents = prompt //+ '\n\n' + createPartFromUri(myfile.uri, myfile.mi
    const response = await ai.models.generateContent({model, config, contents})

    if (response.candidates?.[0]?.content?.parts?.[0]?.text)
      return response.candidates[0].content.parts[0].text
    return null
}
```

This a helper function from my LLM API, for interacting with the Google Gemini API.

One of the key features is that it takes advantage of Gemini's structured outputs capabilities, in which I can define a JSON schema for the API to always respond with. This makes it so that the LLM output no longer needs to be parsed, which makes it 100% consistent if the API request succeeds.

An example response using the defined schema would be:

{
    "recommended_courses": [
        "course_name": "COMPSCI 360…",
        "reasoning": "This course directly addresses your interest in Cryptography, ….."
    ……]
}

## Andrew Work Challenges and Insights

The biggest challenge for this milestone was that many features relied on other features. For example, we couldn’t start implementing backend persistence before we finished designing and implementing our database, and we couldn’t start implementing some features like using user data across features/endpoints (eg LLM API using course progress data) before we implemented a basic authentication system so that we have user IDs.

However, after we did complete everything, I realized the usage of a database made cross-feature data sharing a lot easier than it would’ve been without, if even possible. So I think it was worth the challenge to implement it.

## Andrew Work Next Steps

Although we have finished the project as of now, there are still many bugs and areas of improvement. For example, because we didn’t use the most expensive AI model, and the nature of LLMs itself, sometimes the model would not accurately take into account your prerequisites, so there is room to improve there.

Professor Chiu suggested that we send our project to someone higher up in the department, so there may be many features that would be required for any kind of official usage as well.

## Keyu Work Summary

Added some details like checking user input.
Implemented several methods for HTTP

GET /course-profile/:id to retrieve

POST /expenses/:id to create

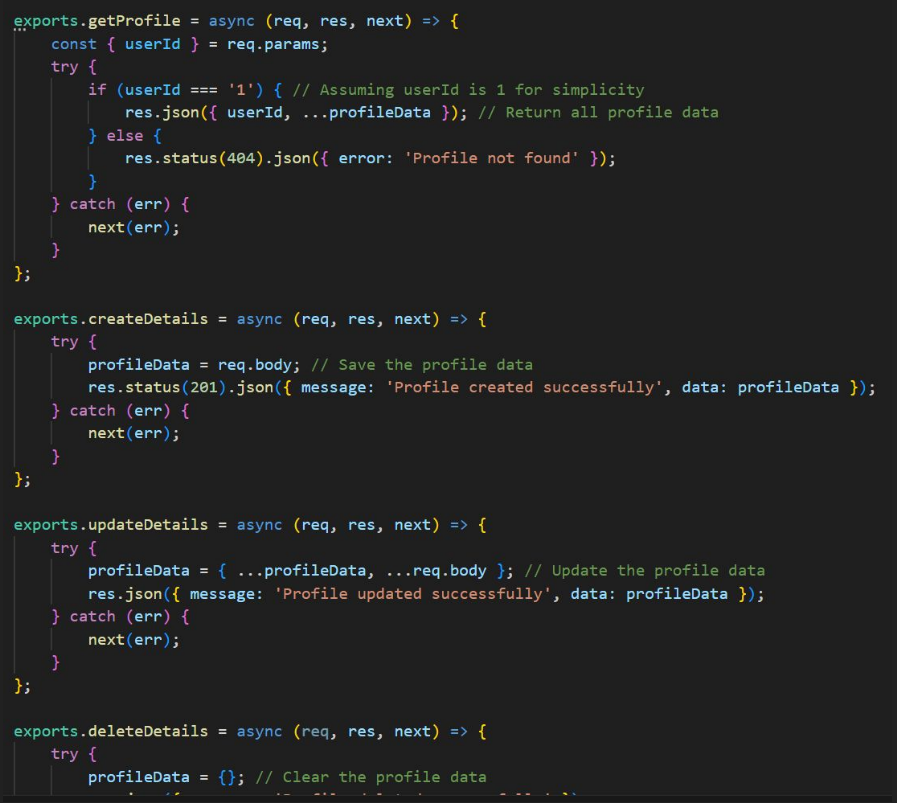
PUT /expenses/:id to update

DELETE /expenses/:id to remove

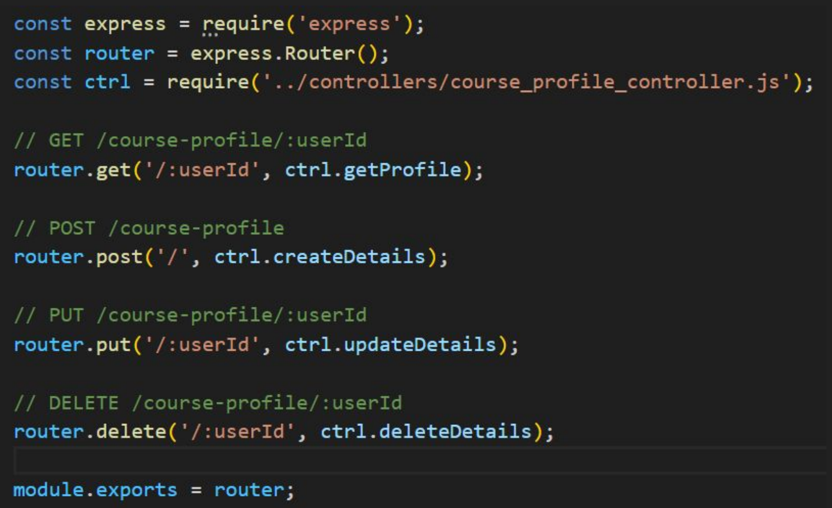
## Keyu - Pull requests

https://github.com/voidpls/326_Course_Recommendation_System/pull/80
https://github.com/voidpls/326_Course_Recommendation_System/pull/81
https://github.com/voidpls/326_Course_Recommendation_System/pull/91

# Keyu - Code snippet

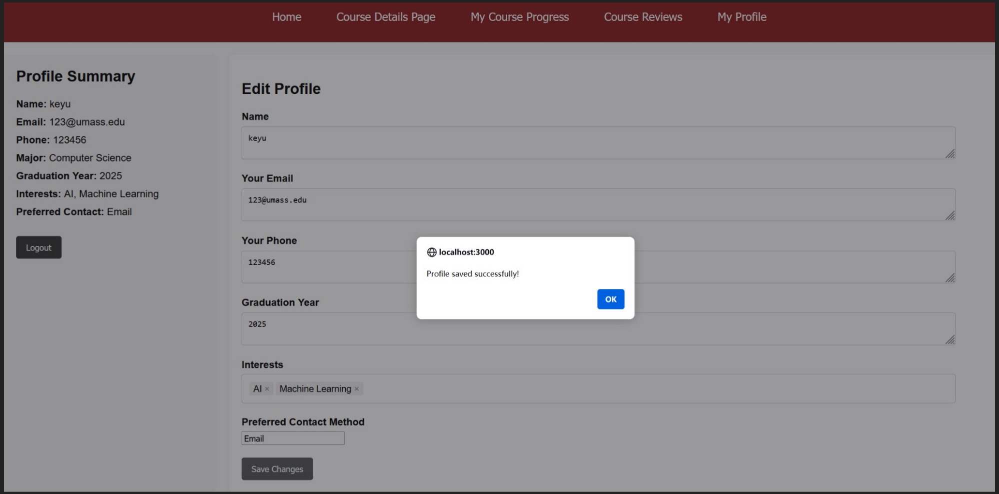


```javascript
exports.getProfile = async (req, res, next) => {
    const { userId } = req.params;
    try {
        if (userId === '1') { // Assuming userId is 1 for simplicity
            res.json({ userId, ...profileData }); // Return all profile data
        } else {
            res.status(404).json({ error: 'Profile not found' });
        }
    } catch (err) {
        next(err);
    }
};

exports.createDetails = async (req, res, next) => {
    try {
        profileData = req.body; // Save the profile data
        res.status(201).json({ message: 'Profile created successfully', data: profileData });
    } catch (err) {
        next(err);
    }
};

exports.updateDetails = async (req, res, next) => {
    try {
        profileData = { ...profileData, ...req.body }; // Update the profile data
        res.json({ message: 'Profile updated successfully', data: profileData });
    } catch (err) {
        next(err);
    }
};

exports.deleteDetails = async (req, res, next) => {
    try {
        profileData = {}; // Clear the profile data
```



```javascript
const express = require('express');
const router = express.Router();
const ctrl = require('../controllers/course_profile_controller.js');

// GET /course-profile/:userId
router.get('/:userId', ctrl.getProfile);

// POST /course-profile
router.post('/', ctrl.createDetails);

// PUT /course-profile/:userId
router.put('/:userId', ctrl.updateDetails);

// DELETE /course-profile/:userId
router.delete('/:userId', ctrl.deleteDetails);

module.exports = router;
```

# Keyu - Implementation



**Profile Summary**

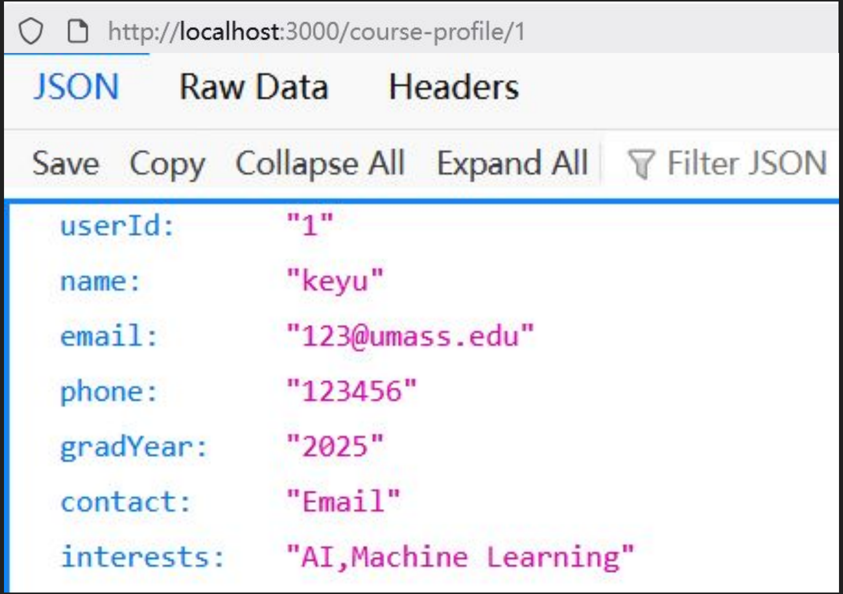
**Name:** keyu
**Email:** 123@umass.edu
**Phone:** 123456
**Major:** Computer Science
**Graduation Year:** 2025
**Interests:** AI, Machine Learning
**Preferred Contact:** Email

**Edit Profile**

| | |
|---|---|
| Name | keyu |
| Your Email | 123@umass.edu |
| Your Phone | 123456 |
| Graduation Year | 2025 |
| Interests | AI × Machine Learning × |
| Preferred Contact Method | Email |

localhost:3000

Profile saved successfully!



http://localhost:3000/course-profile/1

```
userId:      "1"
name:        "keyu"
email:       "123@umass.edu"
phone:       "123456"
gradYear:    "2025"
contact:     "Email"
interests:   "AI,Machine Learning"
```

## Keyu - Challenges

After changing to the API there were many problems with data storage and calls, such as data not being saved or read correctly. Due to incomplete understanding of the new knowledge, a lot of time spent in writing the code to test it over and over again, as well as part of the previous code was modified or rewritten from time to time.

## Keyu - Next steps

At present, the main functional modules of the project have been developed according to the plan and reached the expected goals. Next, we will organize internal communication within the team to evaluate the existing functions through collective discussion, focusing on the analysis of the optimization space and potential function expansion needs.