Ryan Murphy and Brady Barnes

Professor Brinkley

CPSC 3720

31 August 2025

AI Usage Report

The use of AI for this project was immensely beneficial and contributed a lot to our success. ChatGPT was first used to create the SRS after being given the project scope, specific instructions, and constraints. This gave us a clear understanding of where to go and how to set up the first section. Having this outline was very helpful in starting the project. Once we approved the SRS we used AI to generate pseudo code that followed the SRS. We then reviewed the code and made edits so that it more closely resembled what we wanted the final product to look like. After we confirmed and agreed upon the pseudo code we used Claud AI to get the proper formatting for the directory. It was very helpful to have this structure before we started coding, having everything in a labeled directory made it much easier to then go in and code it since we knew what we were working on and where to find another file. We then used Claud AI to draft parts of the ReactJS pages since neither of us had very deep knowledge on ReactJS. A lot of corrections were made at this point, there was an error where the UI was only showing up on the left side of the screen, and the rest was a gray background. This took a lot of manual and AI trouble shooting to get it fullscreened. Another issue from using AI was it would constantly make text the same color as the background but this was easy to manually change the color. A particularly helpful area where AI was used on the backend was for formdata requests and saving user information to their profiles since this was also a new area for us. Once the front and back end were finished from the combination of manual coding and AI we then used AI to generate tests for us to go in and check. This was not necessary but due to the short time constraints of the project saving time was very helpful. In all, AI was incredibly important and helpful to this project and while it did require a moderate amount of corrections it did save us time, and the corrections helped us learn more about ReactJS and Javascript.

AI screenshots/ transcripts

Me-How do I make a test plan for this project?

ChatGPT said:

Here’s a structured way to create a test plan for your Study Buddy app. Since this is a React web app with profile, sessions, and matches functionality, the test plan should cover functional testing, UI testing, and edge cases.

Study Buddy Test Plan (Example)

1. Test Plan Overview

Objective: Ensure all major features of the Study Buddy app work correctly and the user can navigate, create/update profiles, manage sessions, and view matches without errors.

Scope: Test ProfilePage, SessionsPage, MatchesPage, and overall navigation.

2. Features to Test

Profile Management

View profile info (name, major, courses)

Edit profile info

Add and remove courses

Add and remove availability

Sessions

View existing sessions

Create a session

Join a session

Validation for required fields

Prevent creating or joining sessions for courses not enrolled

Matches

View suggested study buddies based on courses

Ensure duplicates are not shown

Display number of sessions each match participates in

Navigation

Switching between Profile, Sessions, and Matches pages

Active tab highlighting

Fullscreen layout and responsive design

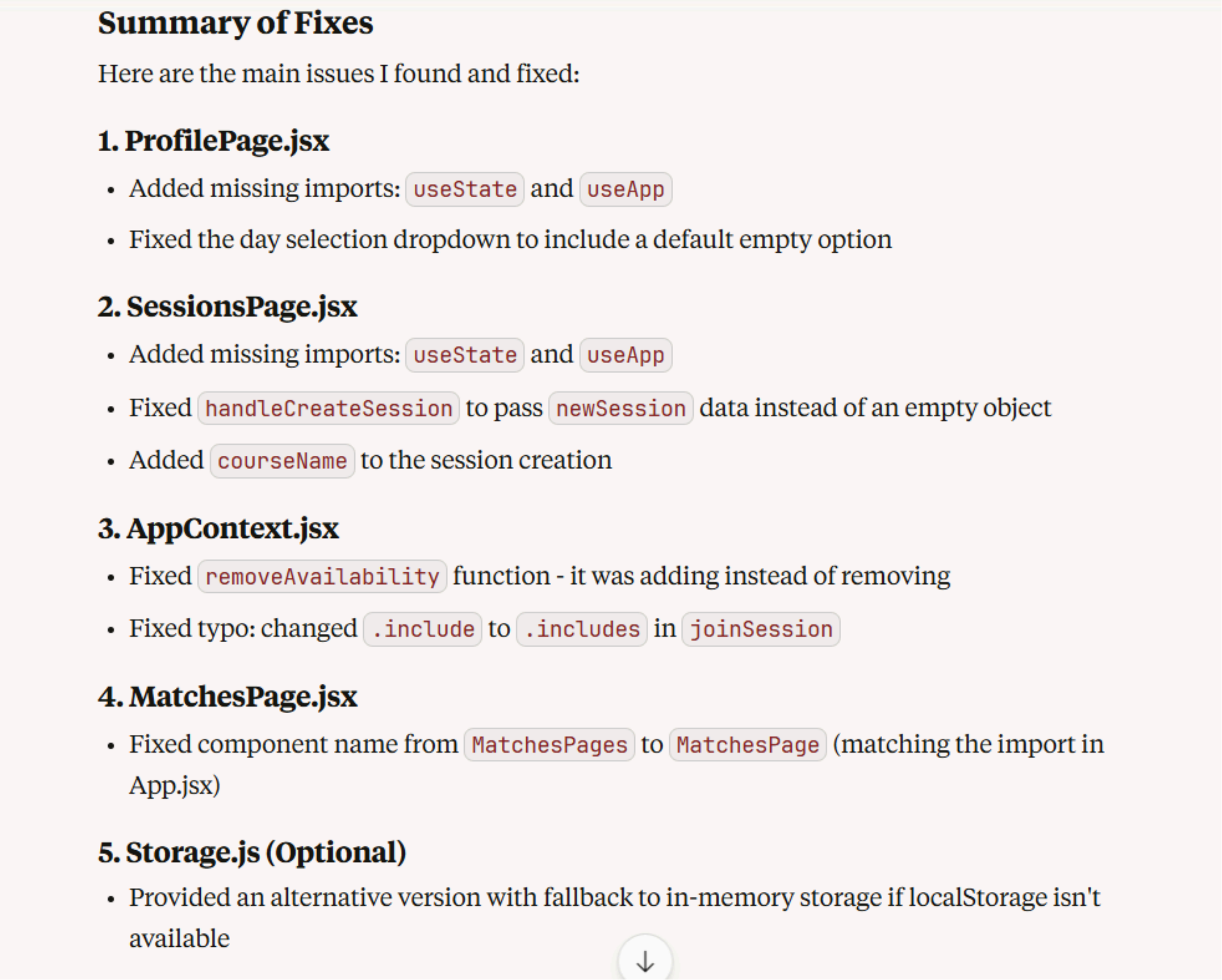
UI/UX

Form validation messages

Button enabling/disabling

Layout of cards, lists, and forms

Colors and contrast for accessibility (e.g., white text on light background)



Summary of Fixes

Here are the main issues I found and fixed:

1. ProfilePage.jsx

- Added missing imports: `useState` and `useApp`
- Fixed the day selection dropdown to include a default empty option

2. SessionsPage.jsx

- Added missing imports: `useState` and `useApp`
- Fixed `handleCreateSession` to pass `newSession` data instead of an empty object
- Added `courseName` to the session creation

3. AppContext.jsx

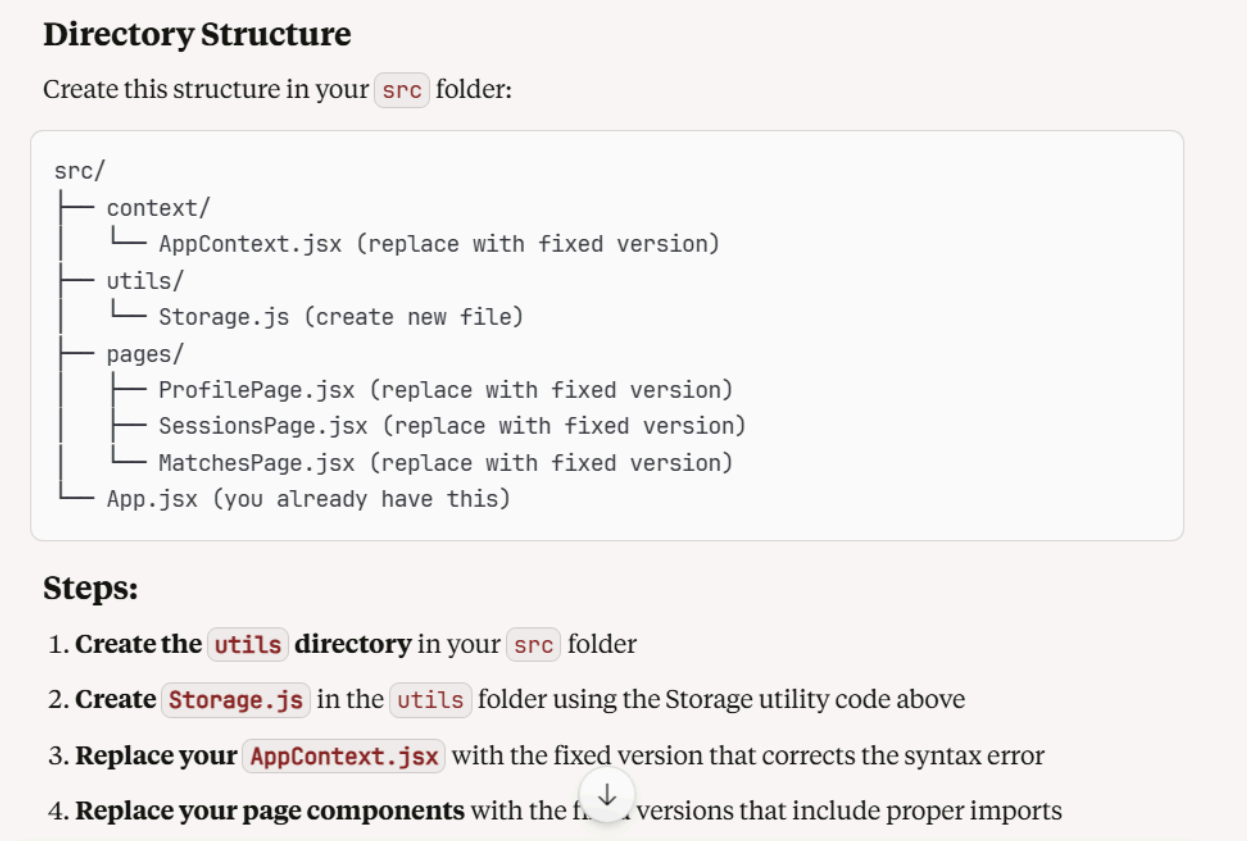
- Fixed `removeAvailability` function - it was adding instead of removing
- Fixed typo: changed `.include` to `.includes` in `joinSession`

4. MatchesPage.jsx

- Fixed component name from `MatchesPages` to `MatchesPage` (matching the import in App.jsx)

5. Storage.js (Optional)

- Provided an alternative version with fallback to in-memory storage if localStorage isn't available



Directory Structure

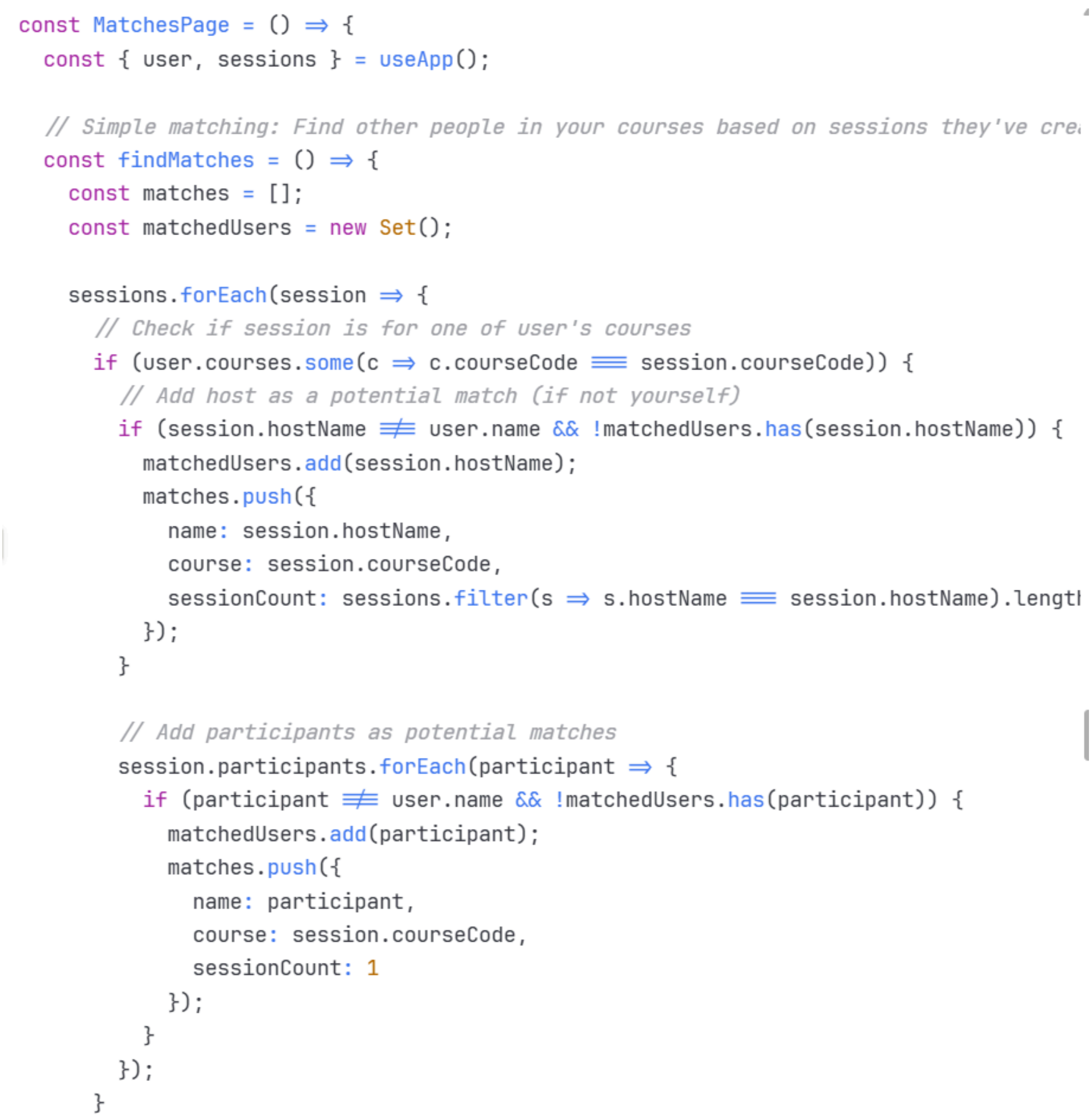
Create this structure in your `src` folder:

```
src/
├── context/
│   └── AppContext.jsx (replace with fixed version)
├── utils/
│   └── Storage.js (create new file)
├── pages/
│   ├── ProfilePage.jsx (replace with fixed version)
│   ├── SessionsPage.jsx (replace with fixed version)
│   └── MatchesPage.jsx (replace with fixed version)
└── App.jsx (you already have this)
```

Steps:

1. **Create the `utils` directory** in your `src` folder
2. **Create `Storage.js`** in the `utils` folder using the Storage utility code above
3. **Replace your `AppContext.jsx`** with the fixed version that corrects the syntax error
4. **Replace your page components** with the [illegible] versions that include proper imports

Matches Algorithm:



```
const MatchesPage = () => {
  const { user, sessions } = useApp();

  // Simple matching: Find other people in your courses based on sessions they've cre
  const findMatches = () => {
    const matches = [];
    const matchedUsers = new Set();

    sessions.forEach(session => {
      // Check if session is for one of user's courses
      if (user.courses.some(c => c.courseCode === session.courseCode)) {
        // Add host as a potential match (if not yourself)
        if (session.hostName !== user.name && !matchedUsers.has(session.hostName)) {
          matchedUsers.add(session.hostName);
          matches.push({
            name: session.hostName,
            course: session.courseCode,
            sessionCount: sessions.filter(s => s.hostName === session.hostName).lengt
          });
        }

        // Add participants as potential matches
        session.participants.forEach(participant => {
          if (participant !== user.name && !matchedUsers.has(participant)) {
            matchedUsers.add(participant);
            matches.push({
              name: participant,
              course: session.courseCode,
              sessionCount: 1
            });
          }
        });
      }
```