UNSW SYDNEY

Deliverable 2:

Software Architecture & Design

Fish & Chips

Mentor: Ali Darejeh

Wednesday 3:20 PM

Ahmet Karatas	z5311022
Daniel Ouyang	z5308495
Humza Saeed	z5309373
Jacky Xie	z5257079
Martin Le	z5208859

Contents

Software Architecture

For our project, we will be accessing data from four main external sources; UNSW Handbook API created by UNSW CSESoc, Facebook and Google's API for Oauth2 and Google's heatmap API for locational data, and our own webserver containing persisted data such as user data, stored course data and locational data.

External Data Sources

In order to fulfill our objective of creating a university planner for students, we needed to get relevant course information from the University to display to the users. To do this, we used the UNSW courses API which showed all relevant information about every course in the university, such as course names, course codes, units of credit, etc. in a JSON format. Since the API is in a messy JSON format, we will use a web scraper to extract the relevant information.

After inspecting and researching on the API, we realised some of the data was altered in such a way that no relevant information could be extracted from it. For example, the information on which terms a course will run were stored in a 32bit hex string which was intelligible for a normal reader. To solve this problem, we realised that these hex strings were repeating and multiple courses would share the same pattern. Through this intuition, we were able to decipher the relevant patterns and match it with the correct corresponding term.

Term 1: bc067bf7db7993009c2c403c3a961994
Term 2: 03067bf7db7993009c2c403c3a961997
Term 3: 601633b7db7993009c2c403c3a9619ab
Summer Term: B8f53777db7993009c2c403c3a9619e4

We plan to store the retrieved information from the API in the MongoDB database, which stores our data in a JSON format, through the Express backend and when users request information through the React frontend, the relevant information will be rendered onto the subsequent page.

OAuth2 from Google and Facebook

OAuth2 is an authorisation framework that enables applications to obtain limited access to user accounts on a HTTP service such as Facebook, GitHub, Google etc. It works by delegating user authentication to the service that hosts the user accounts, authorising third party applications to access the user account.

OAuth has 4 roles:

- Resource owner
- Client
- Resource server
- Authorisation server

Resource owner: User

- User who authorises an application to access their account
- Access is limited to the 'scope' of authorisation granted (r/w access)

Resource/ Authorisation server: API

- Hosts the protected user accounts and verifies the identity of the user then issues access tokens to the application
- Fulfills both resource and authorisation server roles

Client: Application

- The app that wants to access the user's account

OAuth2 works in a way that bypasses the backend's need to check credentials within the program's databases, generating a pair of access(authentication) tokens and
then using the access tokens to let the user in.

OAuth2 essentially uses an API call to the target authorisation server, in our case Facebook or Google, and asks them to authenticate the user (i.e. making sure they are who they say they are), once authenticated, the API will send over authorisation tokens for the App to send to the user to store in the web browser as access token cookies. As a Result, during the period in which the access cookie is valid, the user Is allowed access onto the website without having to re-authenticate.

OAuth2 is good because it does not require the app to store user credentials in its databases. This minimises the damage in major data breaches, as user data is only limited to the in-app data and does not even include data such as user email addresses (since the Authentication server only sends over an access token). Furthermore, user account information is mainly just stored in the major tech company's databases making it more secure.

Aside from security benefits, the client (app) does not need to worry about storing, hashing and retrieving user credentials and dramatically narrows down the work to just an API call, thereby making it much more developer friendly in terms of architectural implementation.

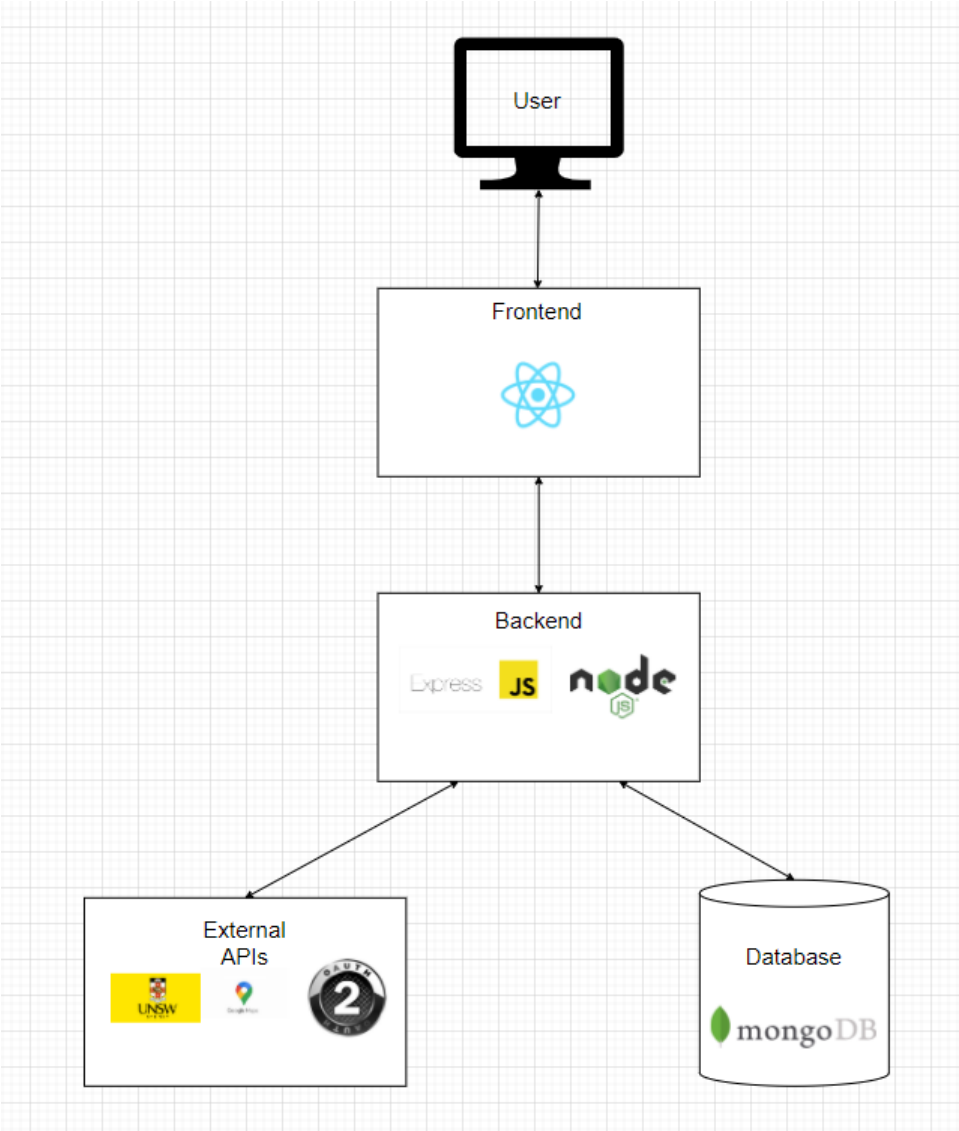
Heat-map from Google Maps

For our "Study Space Finder", we intend to use the Google Maps API, specifically using the heatmap layer. We plan to use it by first making an API request which will return information showing how busy a location is using a contrasting colour scheme. Using this returned information, we intend to then extract that information and then place it on a 1-4 scale, 1 being not busy and 4 being the busiest. This linear scale will make it easier for us to both visually and textually represent our data. To convert the coloured heatmap to a numerical scale, we will use a planned colour gradient to compare with the heatmap and a function will then convert the matched colour to a relevant number on the scale. This information is then stored in our backend (MongoDB and Express) and when a user requests information on a location, the information will be shown and rendered. For example, if a student wanted to know how busy the Main Library is, they will press the pin on top of the Main Library and a small popup will appear on top of the pin textually showing how busy it is.

Software Components

The languages and framework that we will be using are:

Web Stack -> MERN stack
Frontend -> React.js
Backend -> Node.js and Express.js
Framework -> MongoDB
External APIs -> UNSW course APIs, Google Maps API and OAuth2



Since we are choosing React for front end, we thought it would be good to also use a JavaScript library for the backend rest API since we can stay consistent in terms of programming style and language, furthermore since JavaScript has native async support it saves us the hassle of downloading async.io in python nor do we need to deal with threads and blocking functions. We decided to use a NoSQL framework (MONGODB) for the database since we are creating a JavaScript app, where json objects are prevalent. Thus, it makes sense to use a NoSQL language where the data there is represented in a JSON like format instead of a table style schema language like PostgreSQL. Our choice of platform would be chrome, which is supported on any modern operating system.

Summary of Architectural Choices

By utilising a popular web stack framework (MERN stack), we are guaranteed to have architectural components that work and communicate with each other well:

- **React**
  - Easy to learn
    - Since only a few of us had experience with front-end development, by picking React, it allows for an easier learning-curve

    - Faster rendering
        - React has incorporated a virtual DOM (document object model) to solve the issues of bottlenecks which are caused when small changes at upper levels causes a ripple effect to the interface.
        - Virtual DOM is a virtual representation of a document object model, which means that all changes are applied to the virtual DOM and then an algorithm calculates the minimal scope of necessary DOM operations.
        - Guarantees better user experience and higher app performance
    - SEO friendly
        - Deals with common search engine failure to read JavaScript-heavy apps.
        - React runs on the server, rendering and returning the virtual DOM to the browser as a regular webpage

- **Express/Node**
    - Being able to create a REST API server
        - Since all of us have experienced and utilised the REST API in COMP1531 we are all comfortable in working with it
    - Uses same language as frontend
        - Since we are primarily coding in JavaScript for our frontend, being able to stick to one main language is beneficial for time spent learning the language during development

- **MongoDB**
    - Schema-less
        - MongoDB's NoSQL document model allows any kinds of data structure to be modelled and manipulated easily in their BSON data format.
        - This allows us to access it our preferred language
    - Free to use
    - Horizontal Scalability
        - MongoDB is designed to be a distributed database.
        - The collection of documents creates multiple clusters which sustain performance and scale horizontally

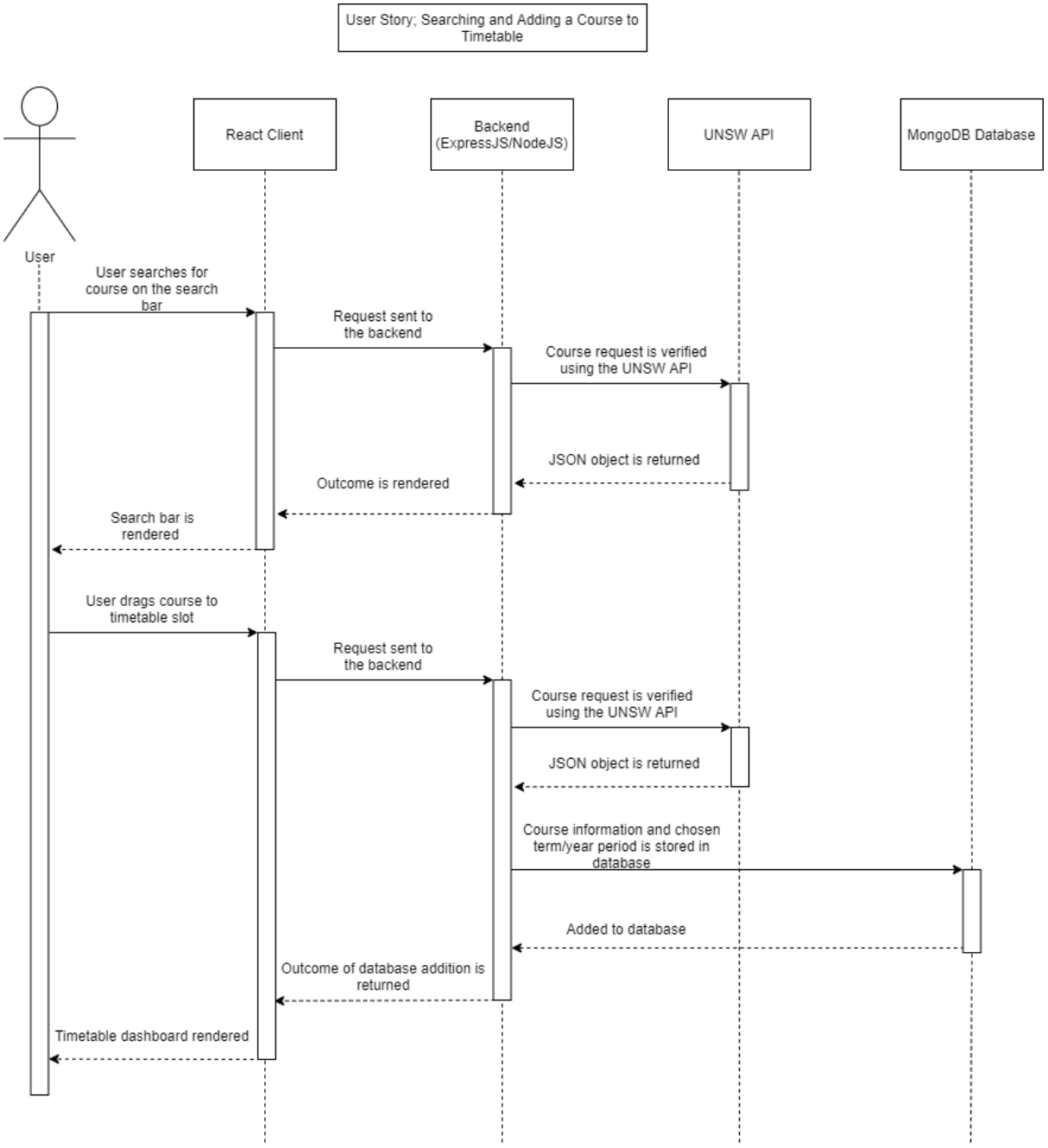
Software Design

Feature 1: Degree Planner

User	Student
Requirement	Students currently do not have a responsive platform which allows them to plan their degree and courses
Feature	An interactive course planner page where students can search up courses and then drag and drop them onto the calendar without having to navigate to other pages
Description	As a student, I want a platform where I can search up courses and find and then drag them onto an interactive calendar, so that I can effectively track and plan my degree on a clean and responsive visual interface.
Flow	**Given** that I am on the degree planner calendar **Then** I can see my personal degree plan **When** I search up a course in the search bar **Then** I can click and drag that course onto my calendar.

User	Student
Requirement	Students currently do not have a responsive platform that is easy to navigate and provides a brief overview of each course
Feature	A single page course planner which has all the information required for students to plan their courses and set them out as they wish
Description	As a student, I want a platform where I can drag and drop courses onto an interactive course planner so that I can quickly plan out my degree without having to navigate to multiple pages to learn about the course overview.

Flow	**Given** that I am on the degree planner **Then** I can see my personal degree plan **When** I click a course from the search bar **Then** I can click and drag that course onto my calendar and see the courses overview on the same page

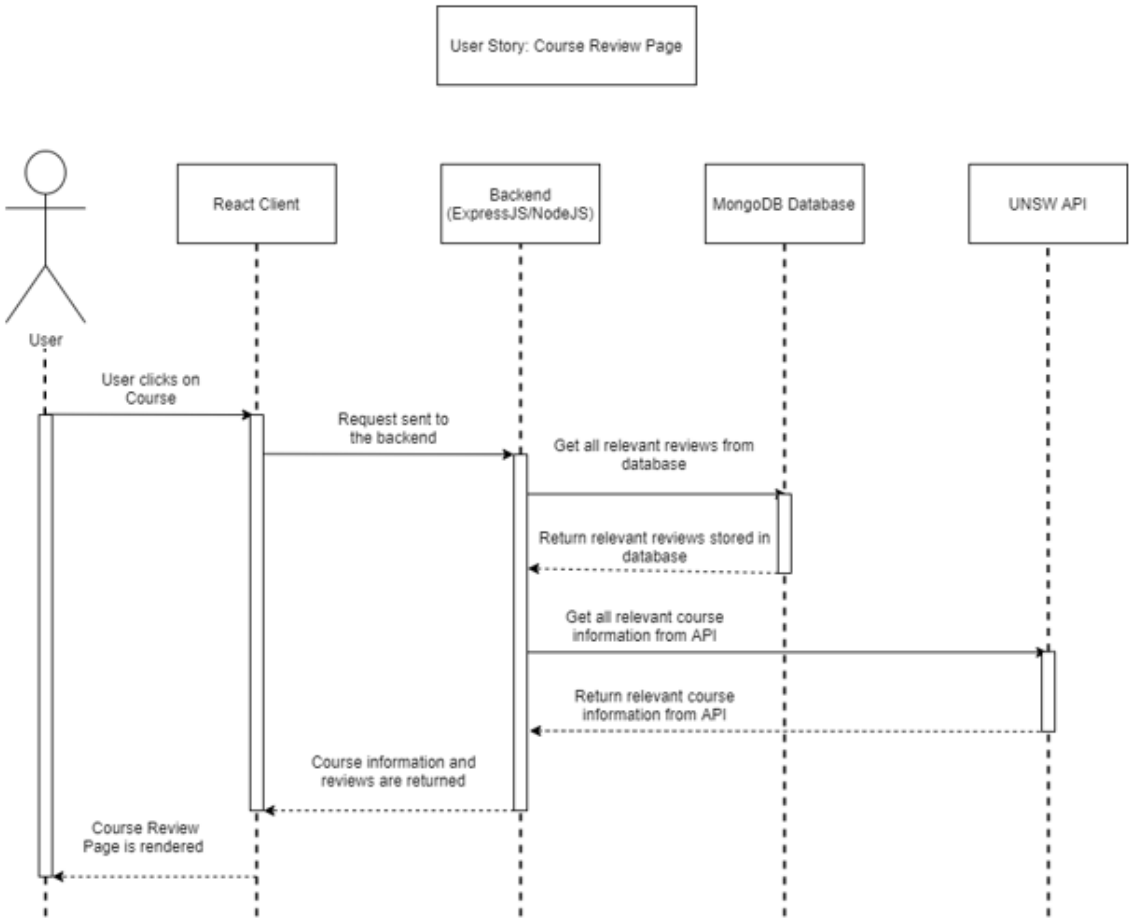


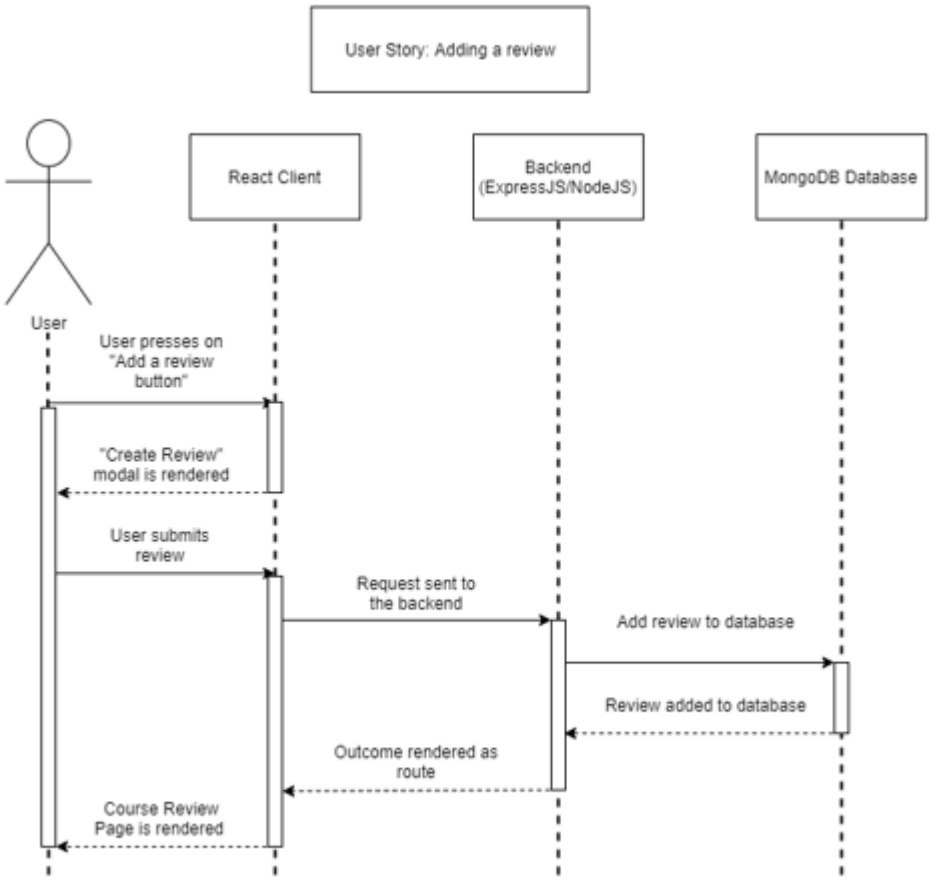
Feature 2: Course Review

User	Student
Requirement	There is no existing UNSW course review system for students to review a course
Feature	A course review system where students are able to review and provide a star rating on a course and provide important comments.
Description	As a student I want to be able to see reviews of courses I may do in the future by older students so that I can be prepared for the amount of content and work I need to do
Flow	**Given** that I have already completed the course **When** I search up a course and have clicked on it **Then** I can press the “add a review” button **Then** I can write and submit my review

User	Staff
Requirement	There is no way to check a student’s opinion on a subject besides the myExperience survey at the end of term
Feature	A dashboard which collates all the course reviews that are relevant to the admin/staff user and creates infographics to visually depict the student sentiment
Description	As the admin/staff, I want a platform where I can readily view the reviews for my courses, so that I can make improvements and decisions to improve student satisfaction.

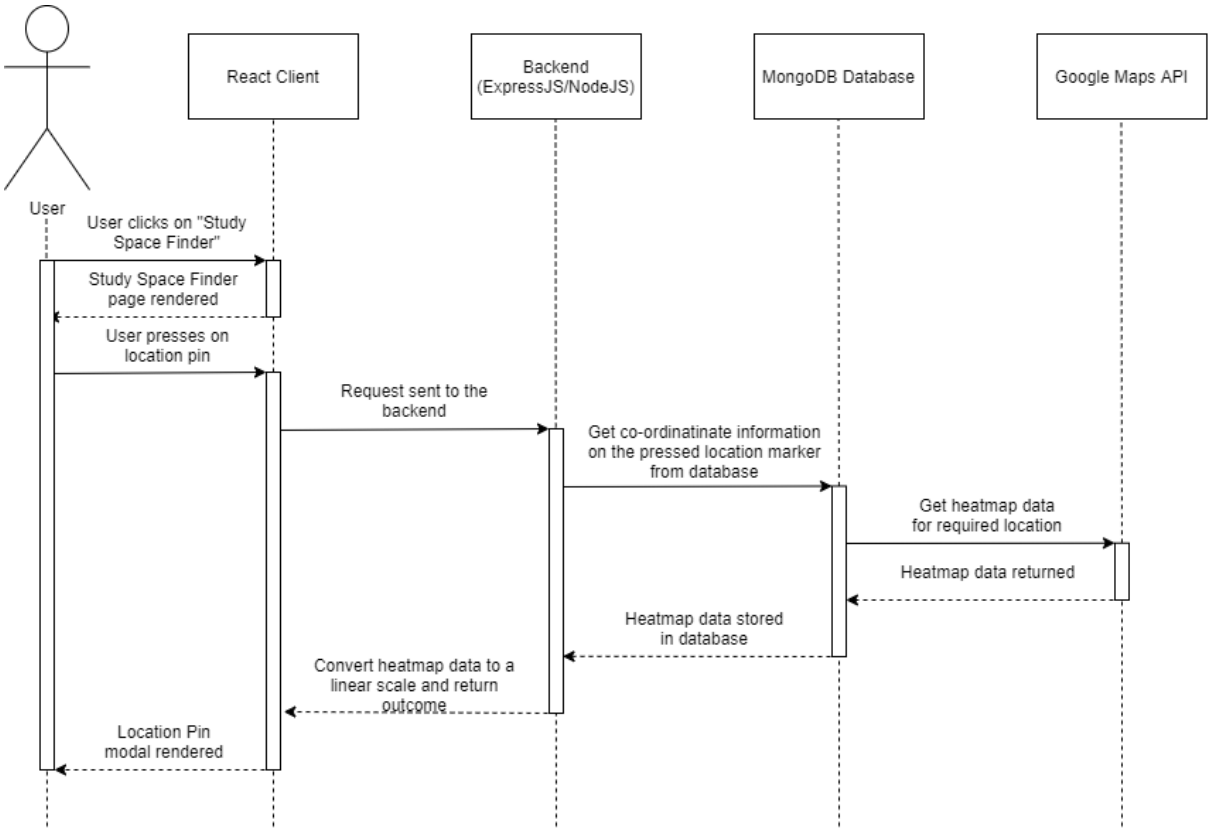
Flow	**Given** that I am a lecturer of the course **When** I login **Then** I can see a dashboard with reviews and feedback of the course/s that I teach so that I may improve my teaching.




User Story: Adding a review
User
React Client
Backend (ExpressJS/NodeJS)
MongoDB Database
User presses on "Add a review button"
"Create Review" modal is rendered
User submits review
Request sent to the backend
Add review to database
Review added to database
Outcome rendered as route
Course Review Page is rendered

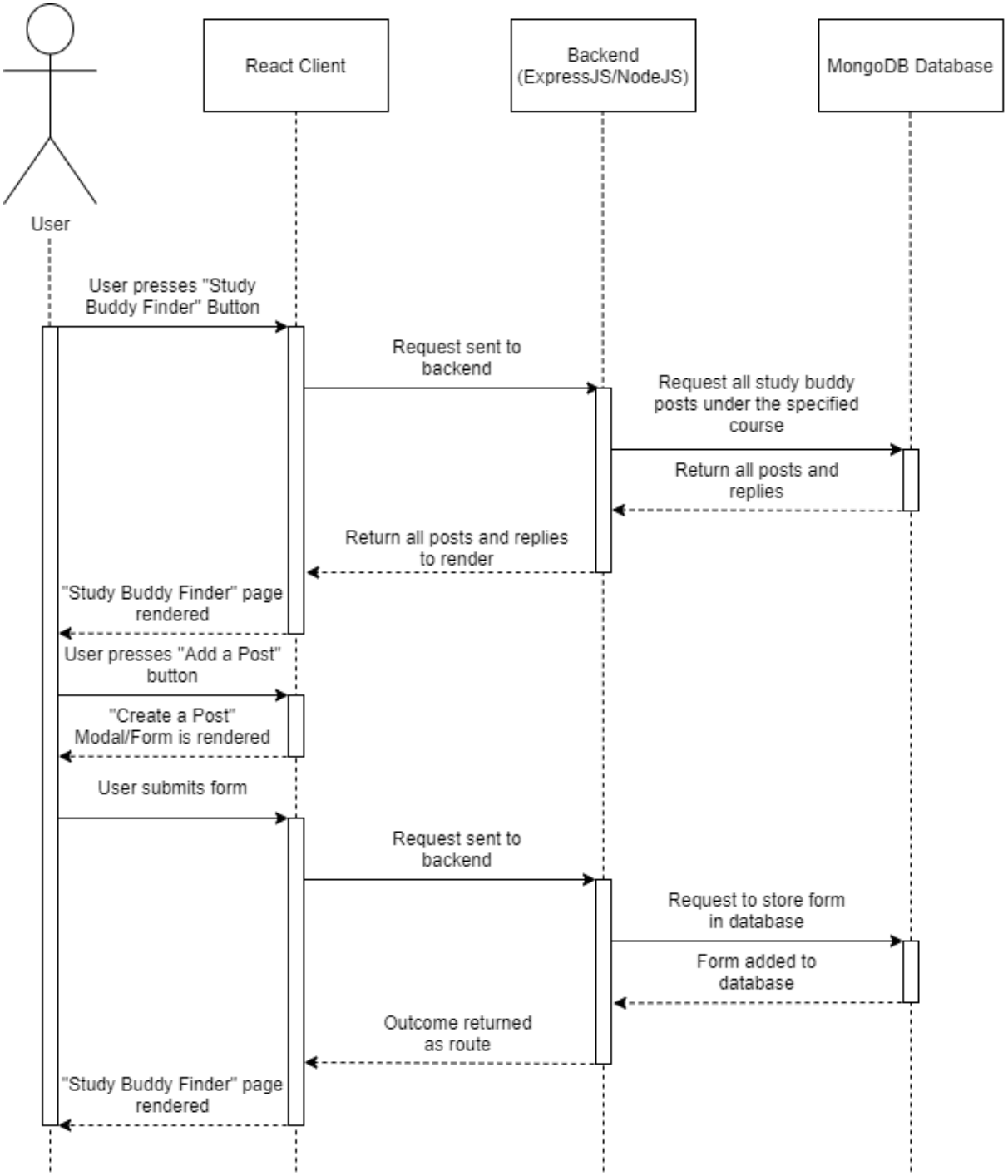
Feature 3: Study Space Finder

User	Student
Requirement	There is no way to find empty study spaces besides having to manually travel to the study space
Feature	A map where students can search up a location and pinpoints will display on the map with empty/near-empty study spaces
Description	As a student, I want to be able to find a quiet study space, so that I can study effectively and in an environment where I'm not heavily distracted
Flow	**Given** that I am under the Study Space Finder Tab **When** I search for study spaces near my location **Then** I can see a visual pinpoint on the map that shows places with empty study spaces



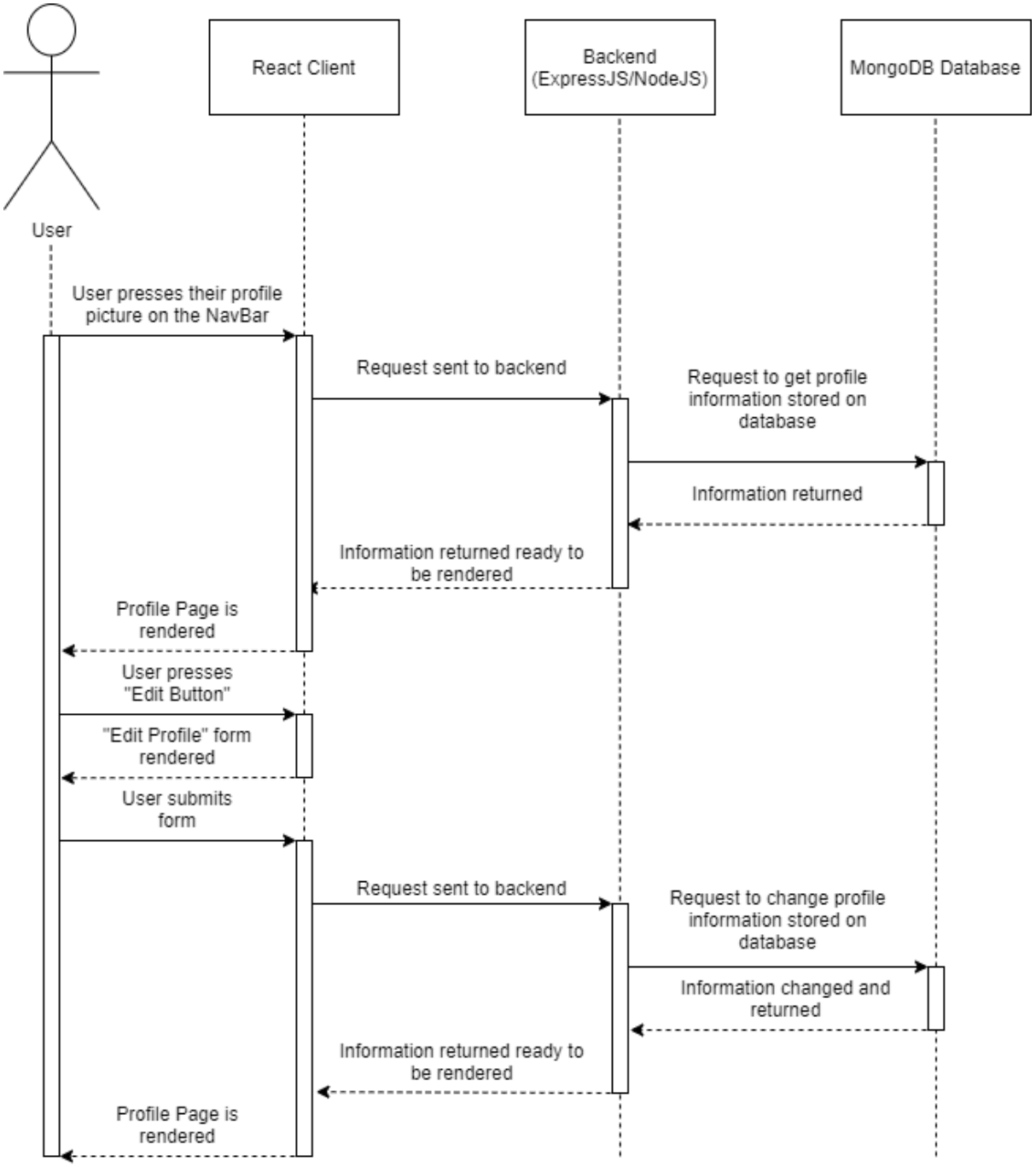
Feature 4: Study Buddy Finder

User	Student
Requirement	Currently there is no effective way for students to create study groups or find assignment groups
Feature	A feature in which students can find other students who are looking to form a study group or assignment groups and add each other.
Description	As a student I want a platform where I can find other students to form a study group or an assignment group with, so that I can make friends with others as well as be in a group with people who share the same goals and hobbies as me.
Flow	**Given** that I am under the “Find a Group” tab **When** I search for a subject **Then** I can see a list of open groups or other students who are looking for students to form a group **Then** I can choose to message the group or student in order to join or make a group


User
React Client
Backend
(ExpressJS/NodeJS)
MongoDB Database
User presses "Study Buddy Finder" Button
Request sent to backend
Request all study buddy posts under the specified course
Return all posts and replies
Return all posts and replies to render
"Study Buddy Finder" page rendered
User presses "Add a Post" button
"Create a Post" Modal/Form is rendered
User submits form
Request sent to backend
Request to store form in database
Form added to database
Outcome returned as route
"Study Buddy Finder" page rendered

Feature 5: Profile Page

User	Student
Requirement	Students should have a profile page which contains information about them
Feature	A profile page which displays a student's academic status and hobbies
Description	As a student, I want a profile page which contains basic information about me so I can keep track of my academic progress and become study buddies with other people who share similar interests and hobbies.
Flow	**Given** that I am registered on the Web-App **When** I press on the profile tab **Then** my profile is displayed which displays my degree plan and features **Then** I can customise my profile page by pressing the settings button


User
React Client
Backend
(ExpressJS/NodeJS)
MongoDB Database
User presses their profile picture on the NavBar
Request sent to backend
Request to get profile information stored on database
Information returned
Information returned ready to be rendered
Profile Page is rendered
User presses "Edit Button"
"Edit Profile" form rendered
User submits form
Request sent to backend
Request to change profile information stored on database
Information changed and returned
Information returned ready to be rendered
Profile Page is rendered