BoileResources

Design Document

CS 307: Team 25

Created by Arya Shukla, Suhesh Venkatesh, Pranavi Chaganti, Zain Sohail

Index:

Purpose:

Students at Purdue University often face challenges in finding optimal resources for their individual classes. The scattered and unorganized nature of available resources can lead to wasted time searching for relevant materials and difficulty in connecting with peers for study groups. Existing platforms like Coursicle and Boilerclasses do not offer a centralized platform for class resources that provides tailored recommendations or tools to achieve these goals; they primarily focus on class schedules and basic search features.

The purpose of this project is to develop a comprehensive and interactive web application tailored for Purdue students. This application will combine resource recommendations, a personalized class management dashboard, and features such as study group matching and in-app chat. By simplifying resource discovery and fostering collaboration, our app aims to help students achieve academic success more efficiently.

Functional Requirements:

1. Users can manage their accounts
    a. As a user, I would like to be able to register for a BoileResources account.
    b. As a user, I would like to be able to log in to my account.
    c. As a user, I would like to manage my account to change the username and other personal details (position, grade, major, etc.).
    d. As a user, I would like to be able to reset my password if I forget it.
    e. As a user, I can add, modify, or delete my profile picture.

2. Users can manage their classes
    a. As a user, I would like to be able to add the classes that I am signed up for.
    b. As a user, I would like to be able to edit and delete the classes that I am signed up for.
    c. As a user, I can filter classes I search for depending on subject, credit, or class type.
    d. As a user, I would like to create study groups for classes that I am in.

e. As a user, I would like to join study groups of classes that I am in.
f. As a user, I can mark classes that I've already taken and this is updated in my profile.
g. As a user, I would like to be able to communicate to my study group via an in-app chat.

3. Users can manage their calendar
    a. As a user, I would like to be able to view my course calendar.
    b. As a user, I would like to be able to export my course calendar to Google calendar.
    c. If time allows, as a user, I can view my exam timings for a certain class on my calendar.
    d. If time allows, as a user, I can input my courses into a scheduling assistant that will tell me what my week would look like based on the courses that I added.
    e. If time allows, as a user, I can add tasks to a personal planner.

4. Users can interact with posted resources
    a. As a user, I would like to receive recommended study resources that scrape Google for each class that I signed up for.
    b. As a user, I would like to be redirected to a Reddit search on the specific class through a button.
    c. As a user, I would like to be redirected to a RateMyProfessor search on the specific class through a button.
    d. As a user, I would like to be able to see information about class timings and class information.
    e. As a user, I would like to be able to see information about professors (classes taught, contact information, and typical class size for classes taught), for the classes I signed up for.
    f. As a user, I would like to be able to receive grade distributions and class descriptions about the classes I signed up for.
    g. As a user, I can post resources for classes taken already/are taking for other students to see.
    h. As a user, I can bookmark posted resources for each individual added class that can be easily viewed for future reference.

i. As a user, I can comment and view comments on posted resources.
j. As a user, I can edit my posted resources and comments.
k. As a user, I can share posted resources to others via a share button that copies a link to clipboard.
l. As a user, I would like to react to posted resources by upvoting/downvoting them for visibility.
m. As a user, I would like to react to comments on posted resources by upvoting/downvoting them for visibility.
n. As a user, I would like to filter recommended resources by type (e.g., PDFs, lecture slides, YouTube videos, textbooks).
o. If time allows, as a user, I can view the past exam averages for the exams of a certain class.

5. Users can receive notifications
    a. As a user, I would like to receive email notifications for chat groups I am in.
    b. As a user, I would like to receive email notifications when the page for a class I am in is updated.
    c. As a user, I would like to get emails about new resources regarding the classes I signed up for.
    d. If time allows, as a user, I would like to receive youtube video recommendations for the course I am taking.

6. Users can customize UI elements
    a. If time allows, as a user, I can customize the layout of my dashboard to prioritize the classes and resources most relevant to me.
    b. As a user, I would like to switch between dark and light mode.

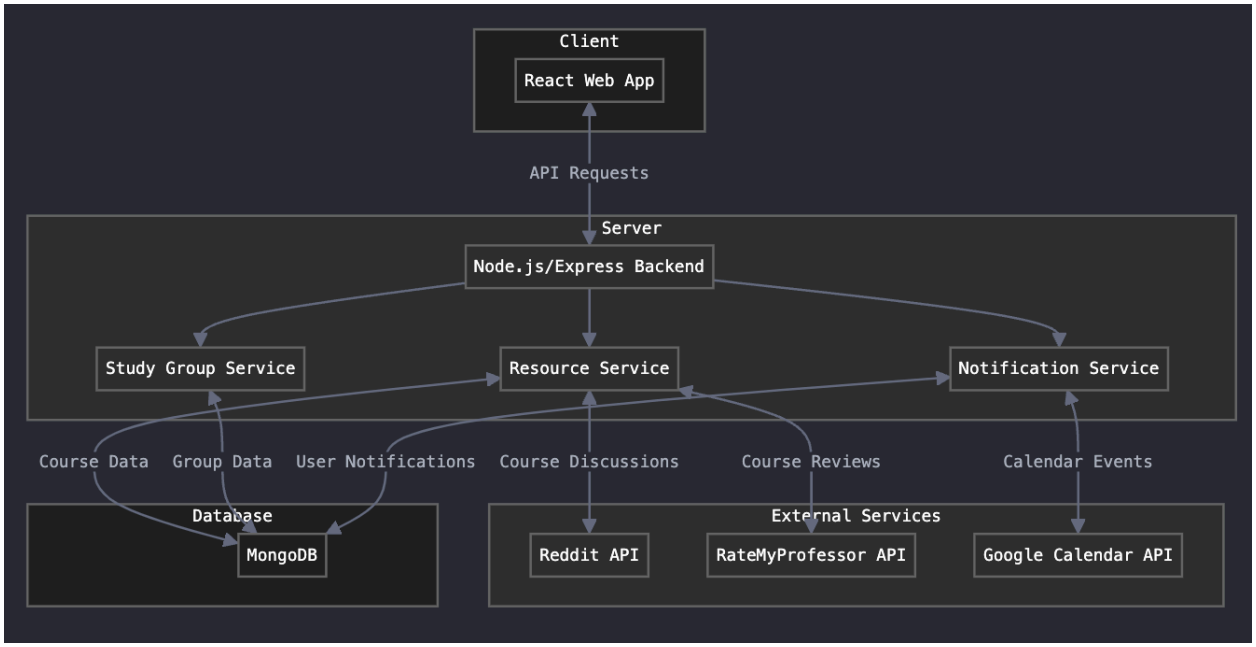
7. Users can comment about the website
    a. As a user, I can view and respond to a feedback form for comments/questions about the site.
    b. As a user, I can report inappropriate content/behavior on the platform.

Non-Functional Requirements:

1. Architecture and Performance
    a. The application will have a completely separate frontend and backend to make collaboration and compatibility easier.
    b. Node.js and Express will be used to build the backend, providing a RESTful API.
    c. MongoDB will handle data storage.
    d. The frontend will be developed using React, fetching data via API requests
    e. Cloud storage and automated management features will also be provided by the application.
2. Security
    a. The application must protect sensitive user information and scheduling data from unauthorized access.
    b. The application must validate all user input to prevent injection attacks.
    c. The application must implement rate limiting to mitigate brute-force attacks, limiting users to a maximum of 100 requests per 5 minutes.
    d. Users must not be able to view other users' account details or scheduled classes unless explicitly permitted.
    e. The application must enforce secure login and registration processes, including email verification and strong password policies.
    f. The application must provide a user reporting and flagging system to identify and address potential security concerns.
3. Usability
    a. The application must feature a responsive design compatible with all modern devices and browsers.
    b. The interface must be intuitive and easy to navigate for all users.
    c. Each user must have a centralized profile dashboard displaying their activity on the platform.
    d. All students must have access to the same layout, but past students and TAs must have additional privileges to endorse posted resources.
4. Hosting/Development?
    a. The backend and frontend must be deployed separately to allow independent modifications and scalability.

b. The frontend must be deployed on Vercel.
c. Deployment configurations must be managed through Vercel's dashboard to ensure streamlined updates and maintenance.

Design Outline:



High Level Overview

BoileResources is a web application designed to help Purdue students efficiently find and share class resources while connecting with peers for study groups. The platform aims to centralize course materials, AI-powered study recommendations, and collaboration tools into a single, user-friendly hub. The application will follow a client-server model, where a single backend server manages multiple client connections. The backend will be developed using Node.js with the Express framework, while the frontend will be built using React. The server will handle client requests, process and store data in a MongoDB database, and respond accordingly to the clients via a RESTful API.

1. Client

a. The client provides users with an interactive interface for accessing class resources and study groups.
b. The client sends AJAX requests to the server for user authentication, resource

retrieval, study group management, and other functionalities.
c. The client processes AJAX responses and dynamically updates the user interface based on the received data.

2. Server

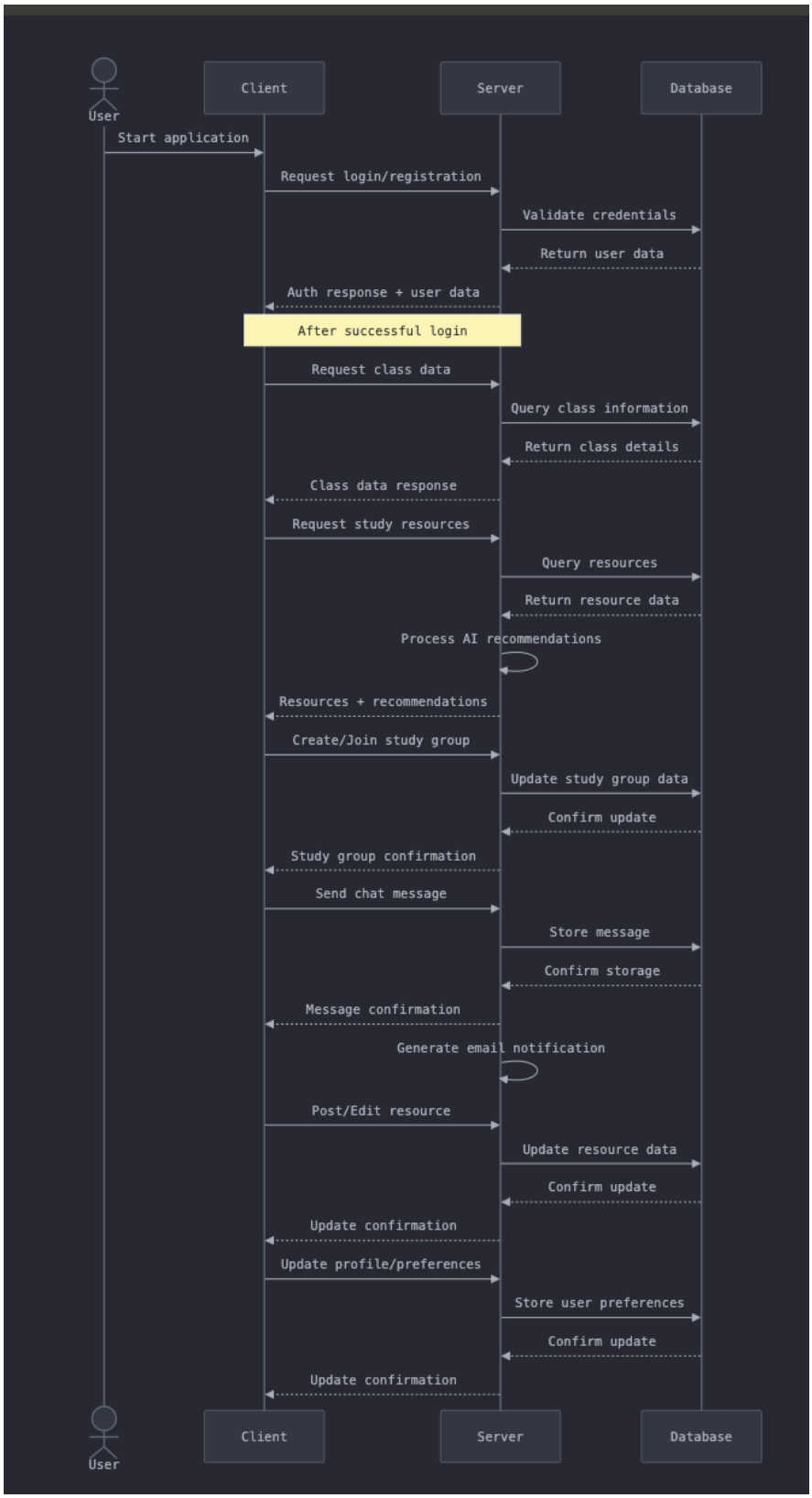
a. The server receives and processes AJAX requests from clients, handling operations such as authentication, resource management, and study group coordination.
b. The server validates requests, processes business logic, and queries the database to retrieve, modify, or store relevant data.
c. The server generates appropriate responses and sends them back to the requesting clients via a RESTful API.

3. Database

a. A NoSQL database (MongoDB) stores all essential application data, including user profiles, class resources, study groups, and chat messages.
b. The database responds to queries from the server and efficiently delivers requested data to support real-time updates and interactions.
c. The database ensures data integrity and optimized indexing for faster resource retrieval and search functionality.

Sequence of Events Overview

The sequence diagram below illustrates the typical interaction among clients, the server, and the database. The sequence begins when a user starts the BoileResources web application. Upon logging in, the client sends a request to the server. The server processes the request, queries the MongoDB database, and retrieves the relevant user data. Once logged in, the client can send additional requests to the server, such as searching for class resources, uploading study materials, forming study groups, and endorsing resources. The server validates these requests and interacts with the database to store or retrieve data as needed. The database responds to the server with the requested data or confirmation of updates. The server then formats the response and sends it back to the client, ensuring a seamless user experience.


User
Client
Server
Database
Start application
Request login/registration
Validate credentials
Return user data
Auth response + user data
After successful login
Request class data
Query class information
Return class details
Class data response
Request study resources
Query resources
Return resource data
Process AI recommendations
Resources + recommendations
Create/Join study group
Update study group data
Confirm update
Study group confirmation
Send chat message
Store message
Confirm storage
Message confirmation
Generate email notification
Post/Edit resource
Update resource data
Confirm update
Update confirmation
Update profile/preferences
Store user preferences
Confirm update
Update confirmation
User
Client
Server
Database

Design Issues

Functional Issues:

1. What information do we need for signing up for an account?
    a. Option 1: Purdue email and password only
    b. Option 2: Purdue email, password, and major
    c. Option 3: Purdue email, password, major, and academic year

Choice: Option 2 Justification: A Purdue email is essential to verify student status and ensure platform security. Passwords are necessary for account protection. Including major helps with resource recommendations and study group matching, as students often take similar courses within their major. However one disadvantage is that students with an undeclared major or students who do not want to give out that information may not use the app. Academic year, while potentially useful, isn't crucial for core functionality and can be optionally added to the profile later. This combination provides enough information for personalization without creating a cumbersome registration process.

2. How should we organize and categorize study resources?
    a. Option 1: Fixed categories (Lecture Notes, Practice Problems, Study Guides, etc.)
    b. Option 2: User-defined tags with no restrictions
    c. Option 3: Fixed primary categories with user-added tags

Choice: Option 3 Justification: Having predefined primary categories (like Lecture Notes, Practice Problems, Study Guides) makes resource organization and searching intuitive. However, allowing users to add their own tags provides flexibility for specific topics or resource types not covered by the main categories. This hybrid approach maintains organizational structure while giving users the ability to add detailed context to their shared resources. A

disadvantage is if users add tags in unpredictable or redundant ways, searching and filtering will become less effective over time.

3. How should we determine resource quality and relevance?
    a. Option 1: Upvote/downvote system only
    b. Option 2: Star rating system (1-5 stars)
    c. Option 3: Combined approach with upvotes and endorsements from past students/TAs
    d. Option 4: Endorsements from past students/TAs

Choice: Option 3 Justification: A simple upvote/downvote system might not provide enough granularity, while star ratings alone don't distinguish between peer and authority feedback. The combined approach lets regular students quickly indicate useful resources through upvotes, while allowing past students and TAs to provide authoritative endorsements. This creates a more reliable quality signal for users searching for resources. However, the downside to this would be having to wait for people with merit to use the app to create the endorsements.

4. How should we handle study group formation?
    a. Option 1: Automatic matching based on course and availability
    b. Option 2: Browse-and-join system with group descriptions
    c. Option 3: Hybrid system with both browsing and smart recommendations

Choice: Option 3 Justification: While automatic matching could be efficient, it might not account for personal preferences and study styles. A pure browse-and-join system could be overwhelming with many options. A hybrid approach allows students to browse available groups while receiving smart recommendations based on their courses, major, and study preferences. This provides flexibility while helping students find compatible study partners. One disadvantage is that integrating both browsing and smart recommendations

might make implementation difficult and require more resources to maintain an effective matching algorithm.

5. How should we implement the resource recommendation system?
    a. Option 1: Simple keyword-based matching
    b. Option 2: AI-powered recommendations based on course content only
    c. Option 3: AI recommendations incorporating user behavior, course content, and peer usage patterns

Choice: Option 3 Justification: Simple keyword matching wouldn't provide enough context for quality recommendations. Course content alone might miss valuable resources that peers have found helpful. Using AI that considers multiple factors (course content, user behavior, peer usage, and resource ratings) will provide more relevant and personalized recommendations. This approach aligns with our goal of helping students find the most useful resources efficiently. One disadvantage is that it introduces complexity in data collection, processing, and model training and it might require substantial computational resources and careful handling of user privacy concerns.

Non-Functional Issues:

1. What cloud platform should we use for deployment?
    a. Option 1: Vercel + MongoDB Atlas
    b. Option 2: Google Cloud
    c. Option 3: AWS
    d. Option 4: Heroku

Choice:Our choice of Vercel and MongoDB Atlas is driven by the need for separate frontend and backend deployments. Vercel excels in supporting React applications with automatic deployments, serverless functions, and edge network distribution, while MongoDB Atlas offers a free tier for database hosting that integrates seamlessly with our Node.js backend. This combination

provides an optimal balance of ease of use, cost-effectiveness, and alignment with our tech stack. However, there are tradeoffs to consider: Vercel's serverless functions may introduce cold start latency, impacting performance for time-sensitive applications, and the free tier of MongoDB Atlas has limitations on storage and performance, which might become a bottleneck as our application scales. Additionally, relying on these managed services can lead to vendor lock-in, complicating future migrations to other providers or self-hosted solutions. Despite these tradeoffs, we believe this setup offers the best overall benefits for our current needs.

2. What backend framework should we use?
    a. Option 1: Django (Python)
    b. Option 2: Express.js (Node.js)
    c. Option 3: Spring Boot (Java)
    d. Option 4: FastAPI (Python)

Choice: Express.js is chosen for our project because it's lightweight, flexible, and perfect for our needs. It has excellent middleware support, making it easy to implement rate limiting (100 requests per 5 minutes) and security features. Plus, it integrates well with MongoDB through Mongoose, which is great for our database setup. The large Node.js ecosystem also offers plenty of packages for features like email notifications and authentication. However, there are some tradeoffs. While Express.js is powerful, it can require more manual setup and configuration compared to some other frameworks, which might mean more initial development time. Additionally, because it's so flexible, there's a risk of inconsistent code quality if not managed properly. Despite these downsides, we believe Express.js is the best choice for our project due to its flexibility, strong community support, and seamless integration with our tech stack.

3. What frontend framework should we use?
    a. Option 1: React
    b. Option 2: Vue.js
    c. Option 3: Angular

d. Option 4: Svelte

Choice: React Justification: Our choice of Vercel and MongoDB Atlas is driven by the need for separate frontend and backend deployments. Vercel excels in supporting React applications with automatic deployments, serverless functions, and edge network distribution, while MongoDB Atlas offers a free tier for database hosting that integrates seamlessly with our Node.js backend. This combination provides an optimal balance of ease of use, cost-effectiveness, and alignment with our tech stack. However, there are trade offs to consider: Vercel's serverless functions may introduce cold start latency, impacting performance for time-sensitive applications, and the free tier of MongoDB Atlas has limitations on storage and performance, which might become a bottleneck as our application scales. Additionally, relying on these managed services can lead to vendor lock-in, complicating future migrations to other providers or self-hosted solutions. Despite these tradeoffs, we believe this setup offers the best overall benefits for our current needs.

4. What database solution should we use?
    a. Option 1: PostgreSQL
    b. Option 2: MongoDB
    c. Option 3: MySQL
    d. Option 4: Firebase

Choice: MongoDB Justification: MongoDB is particularly well-suited for our application because it offers a flexible schema design that can handle varying resource types, efficient querying for our recommendation system, and excellent scaling capabilities. Its document-based structure is ideal for storing nested data like study group chats and resource comments. Additionally, MongoDB's aggregation pipeline is perfect for generating resource recommendations and analyzing user interactions. However, there are some tradeoffs. MongoDB's flexibility can lead to inconsistent data structures if not carefully managed, and its performance can be impacted by complex queries on

large datasets. Despite these downsides, we believe MongoDB is the best choice for our project due to its versatility, powerful querying capabilities, and scalability.

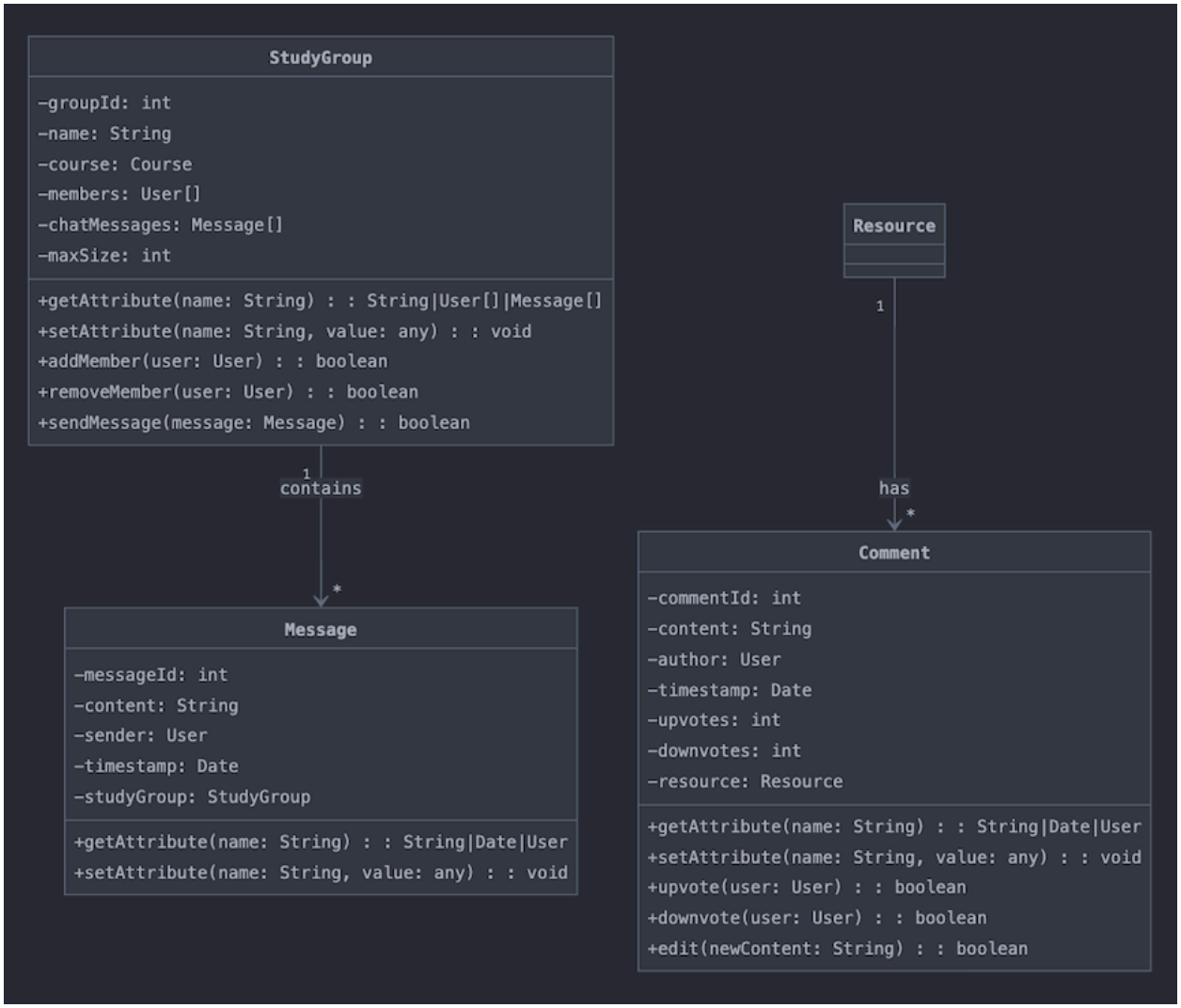
5. What authentication system should we implement?
    a. Option 1: JWT with local storage
    b. Option 2: Session-based authentication
    c. Option 3: OAuth 2.0 with Purdue CAS
    d. Option 4: Firebase Authentication

Choice: JWT (JSON Web Tokens) authentication has been chosen to meet our security requirements. JWT provides secure token-based authentication for our API endpoints and allows us to implement the necessary rate limiting while maintaining stateless authentication. However, there are some tradeoffs. Implementing JWT can add complexity to our authentication flow and may require additional development time to ensure seamless integration. Despite these challenges, we believe this approach offers the best balance of security and user access control for our platform.

Design Details

Class Design


Course
-courseId: int
-courseCode: String
-title: String
-professor: String
-description: String
-creditHours: int
-type: String
-subject: String
-resources: Resource[]
-studyGroups: StudyGroup[]
-schedule: Schedule[]
-professorEmail: String
+getAttribute(name: String) : : String|int|Resource[]
+setAttribute(name: String, value: any) : : void
+addResource(resource: Resource) : : boolean
+createStudyGroup(name: String, maxSize: int) : : StudyGroup
+getRedditLink() : : String
+getRMPLink() : : String
1
contains
*
1
has
*
Resource
-resourceId: int
-title: String
-type: String
-url: String
-description: String
-datePosted: Date
-upvotes: int
-downvotes: int
-comments: Comment[]
-postedBy: User
-fileType: String
-shareLink: String
+getAttribute(name: String) : : String|Date|Comment[]
+setAttribute(name: String, value: any) : : void
+upvote(user: User) : : boolean
+downvote(user: User) : : boolean
+addComment(comment: Comment) : : boolean
+generateShareLink() : : String
Schedule
-scheduleId: int
-course: Course
-startTime: DateTime
-endTime: DateTime
-dayOfWeek: String
-location: String
+getAttribute(name: String) : : String|DateTime
+setAttribute(name: String, value: any) : : void
+exportToGoogleCalendar() : : boolean



Descriptions of Classes and Interaction between Classes:

● User

- User object is created when a student registers for a BoileResources account
- Each user is assigned a unique user ID and must use their Purdue email
- User profile includes username, password, major, academic year, and optional profile picture
- Users can set their preferred layout and dark/light mode preferences
- Users maintain two course lists: currently enrolled courses and completed courses
- Users can bookmark resources and join study groups for their courses

- Users can post resources, comment on resources, and participate in study group chats
- Users receive notifications for updates in their courses and study groups
- Users can customize their email notification preferences

● Course

- Course object is created in the system database and can be added to user profiles
- Each course has a unique course ID, course code, title, and professor information
- Courses contain details like credit hours, type (lecture/lab), and subject area
- Each course maintains a collection of shared resources and active study groups
- Courses include grade distribution data and class size information
- Course schedule information is stored for calendar integration
- Professor details include contact information and office hours
- Courses link to external platforms (Reddit, RateMyProfessor) for additional information

● Resource

- Resource object is created when a user shares study material for a course
- Each resource has a unique ID, title, description, and URL/file content
- Resources are categorized by type (PDF, lecture slides, YouTube videos, textbooks)
- Resources track upvotes/downvotes for visibility and quality assessment
- Resources can be bookmarked by users for easy access
- Resources maintain a comment thread for discussion
- Resources can be shared via generated links
- Resources are associated with specific courses and their posting user

● StudyGroup

- StudyGroup object is created when a user initiates a study group for a course
- Each group has a unique ID, name, and maximum member limit
- Groups are associated with a specific course
- Groups maintain a list of members and chat message history

- Groups facilitate real-time communication between members
- Members can join or leave groups within their enrolled courses
- Groups can generate email notifications for new messages

● Comment

- Comment object is created when users interact with shared resources
- Comments include content, timestamp, and author information
- Comments can receive upvotes/downvotes for visibility
- Comments can be edited by their original authors
- Comments help facilitate discussion about specific resources
- Comments notify resource owners and other participants of new interactions

● Notification

- Notification object is created for various system events
- Notifications are generated for new resources, study group messages, and course updates
- Each notification includes type, message, timestamp, and recipient
- Notifications can trigger email alerts based on user preferences
- Notifications track read/unread status
- Notifications help users stay updated on relevant activity

● Schedule

- Schedule object is created for each course section
- Schedules contain meeting times, days, and locations
- Schedules can be exported to Google Calendar
- Schedules help users organize their course commitments
- Schedules are associated with specific courses

● Feedback

- Feedback object is created when users submit site feedback or reports
- Feedback includes content, timestamp, and status
- Feedback can be about the platform or specific content
- Feedback helps maintain platform quality and address issues
- Feedback can receive administrative responses

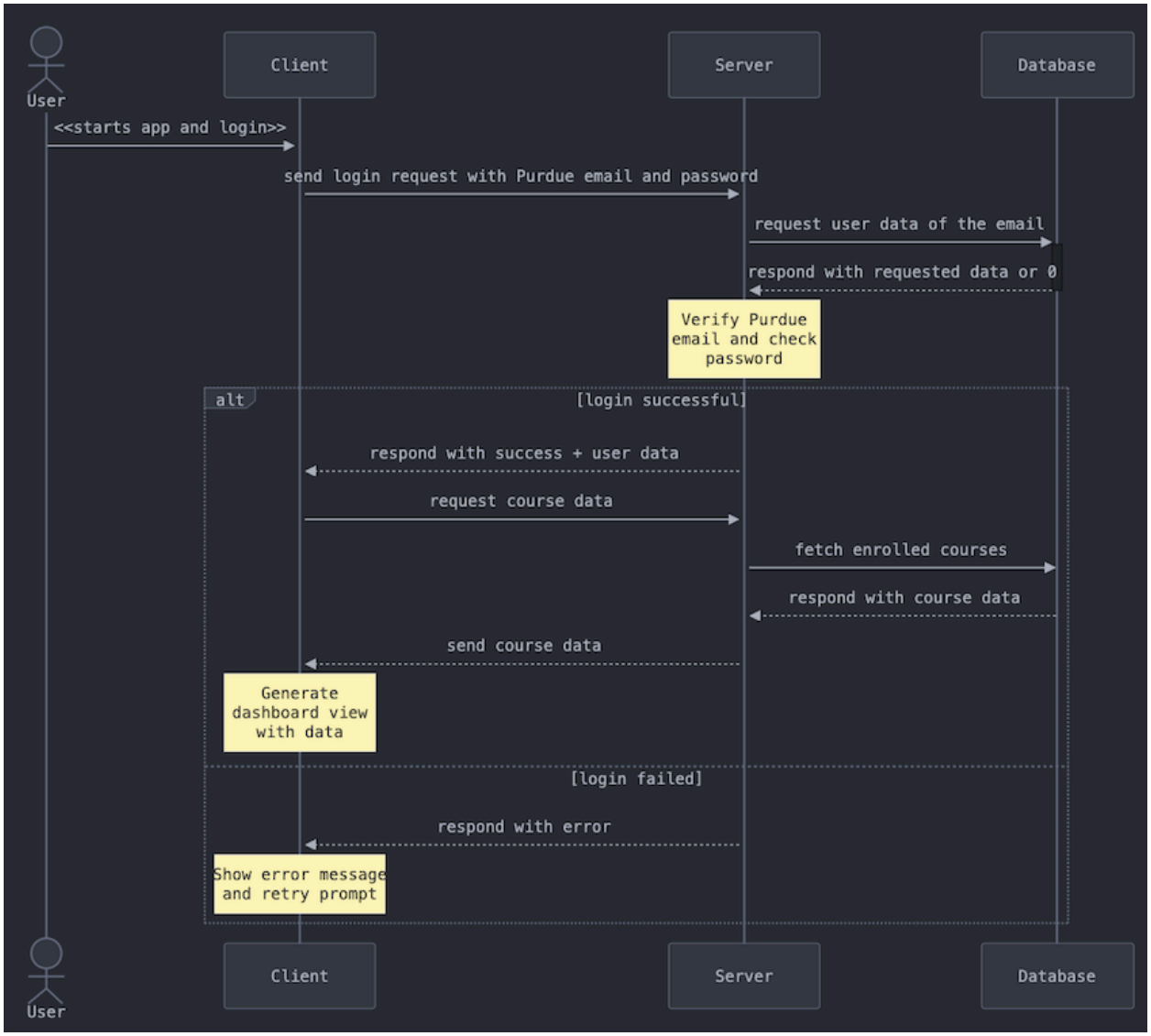
Key Interactions:

1. When a user enrolls in a course:
    - User's course list is updated
    - Access is granted to course resources
    - User can join study groups
    - Course schedule is added to calendar
2. When a resource is shared:
    - Resource is linked to specific course
    - Notifications sent to enrolled students
    - Comment thread is initialized
    - Resource becomes available for bookmarking
3. When a study group is created:
    - Group is associated with course
    - Chat functionality is enabled
    - Members can be added up to maximum
    - Notification system is activated
4. When notifications are generated:
    - Recipients are determined based on context
    - Email notifications are sent if enabled
    - Notification appears in user dashboard
    - Read status is tracked

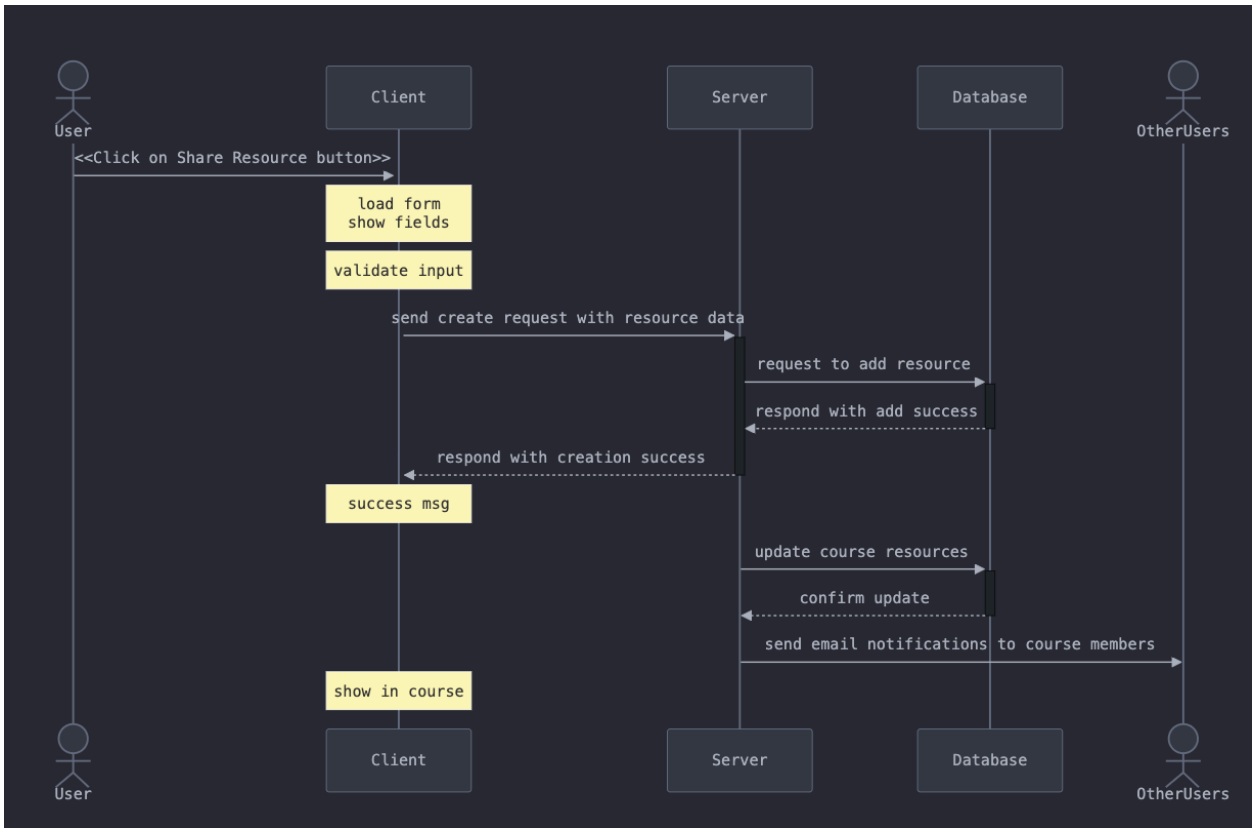
Sequence Diagram:

The following diagrams depict the sequence of the major events in BoileResources, including user login, resource sharing, study group participation, resource searching, and group chat interactions. The sequence shows how messages exchange in a client-server model. When a user performs an action on the frontend user interface, the client sends a request to the server and the server will send a query to the database to acquire the data needed. The server may process the data (such as applying AI recommendations for resources) and send a response back to the client. The client will process the data and update the UI display accordingly.

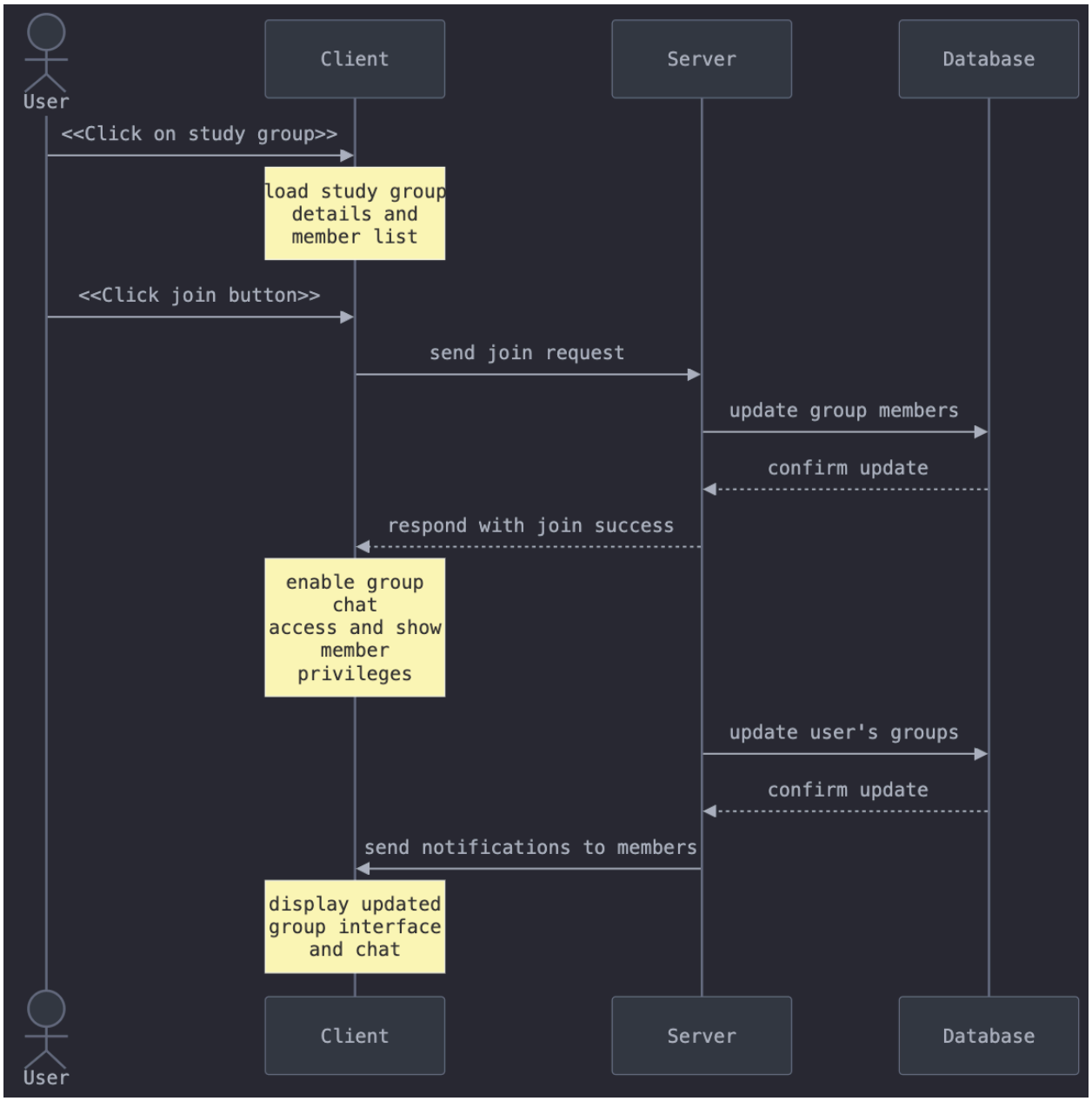
1. Login Sequence



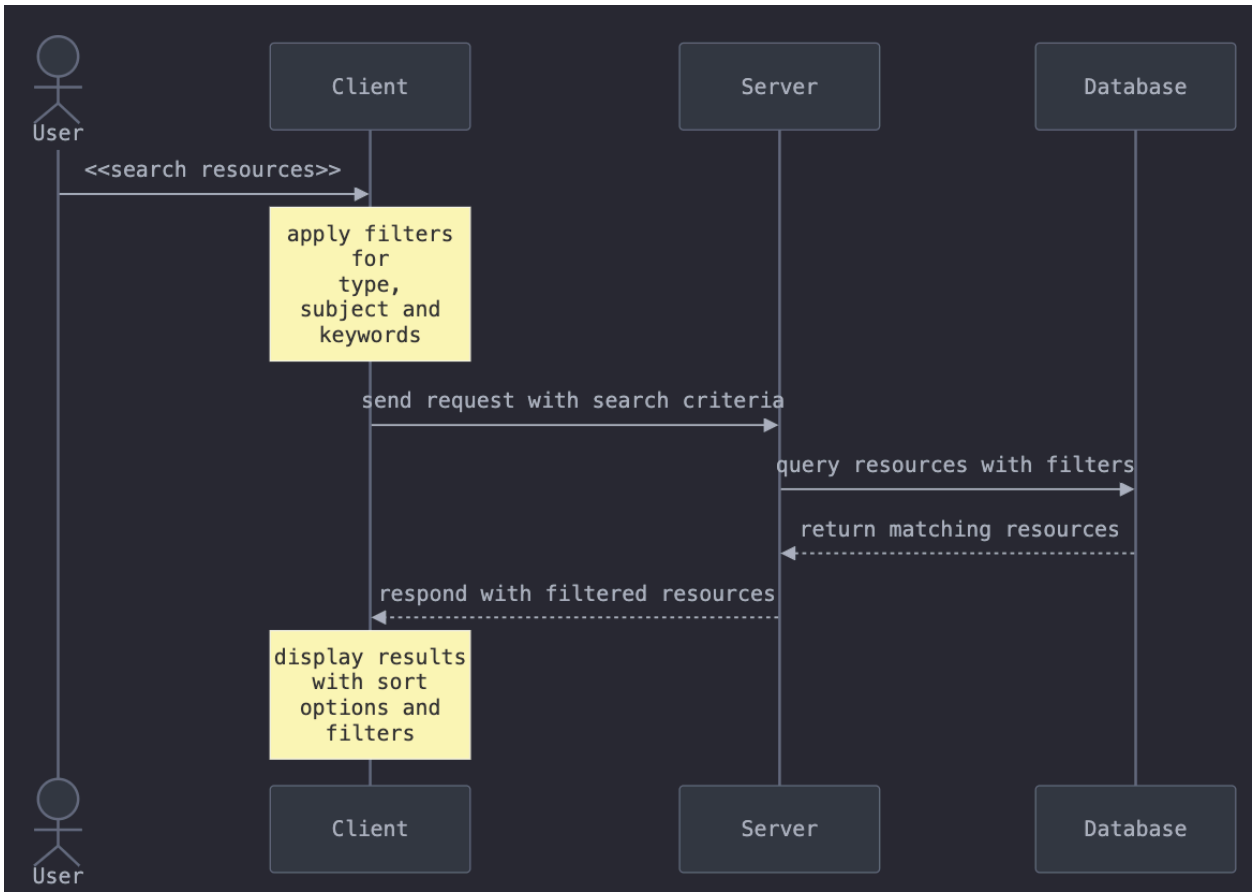
2. Resource Creation Sequence



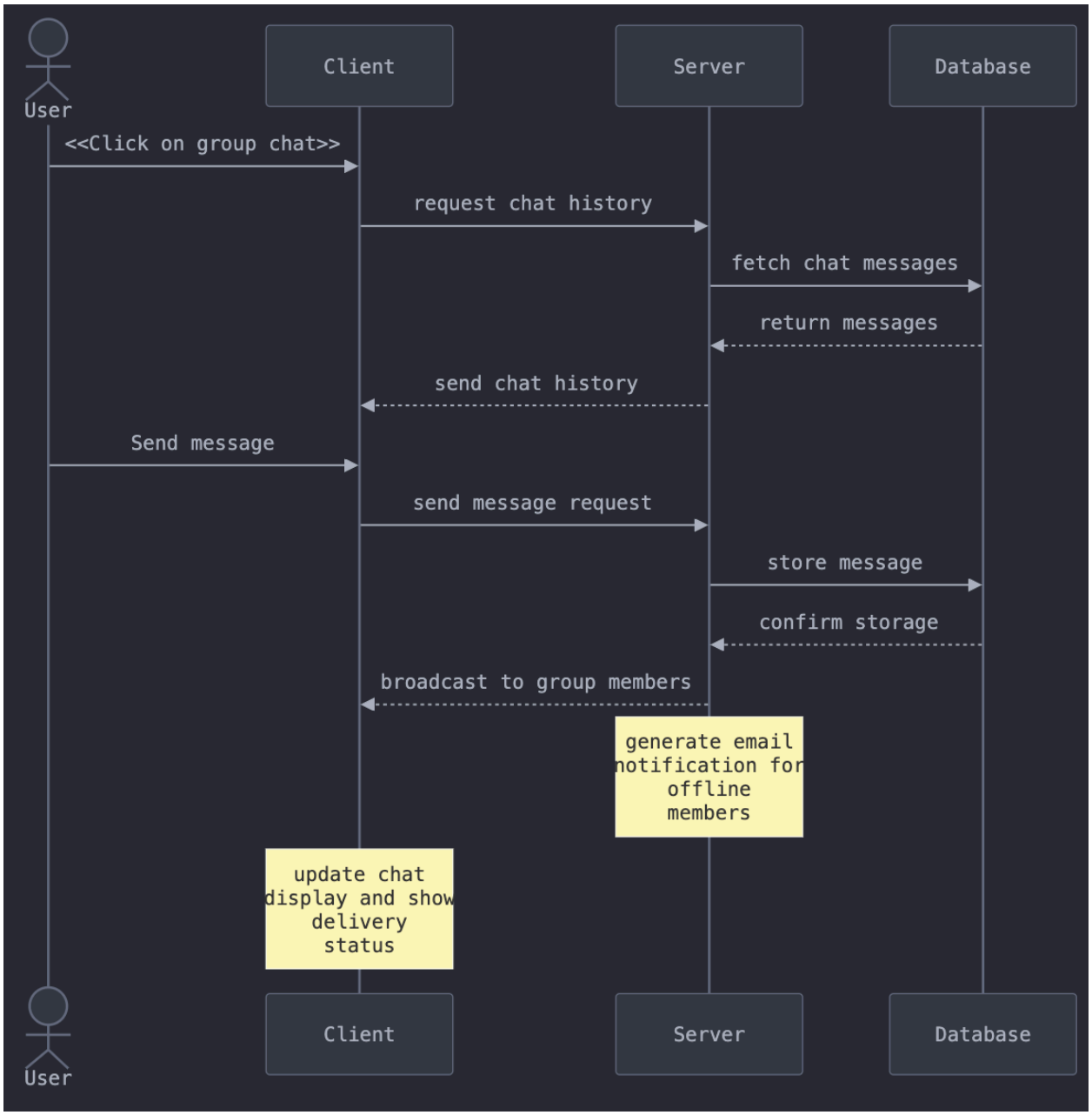
3. Study Group Join Sequence



4. Resource Search Sequence



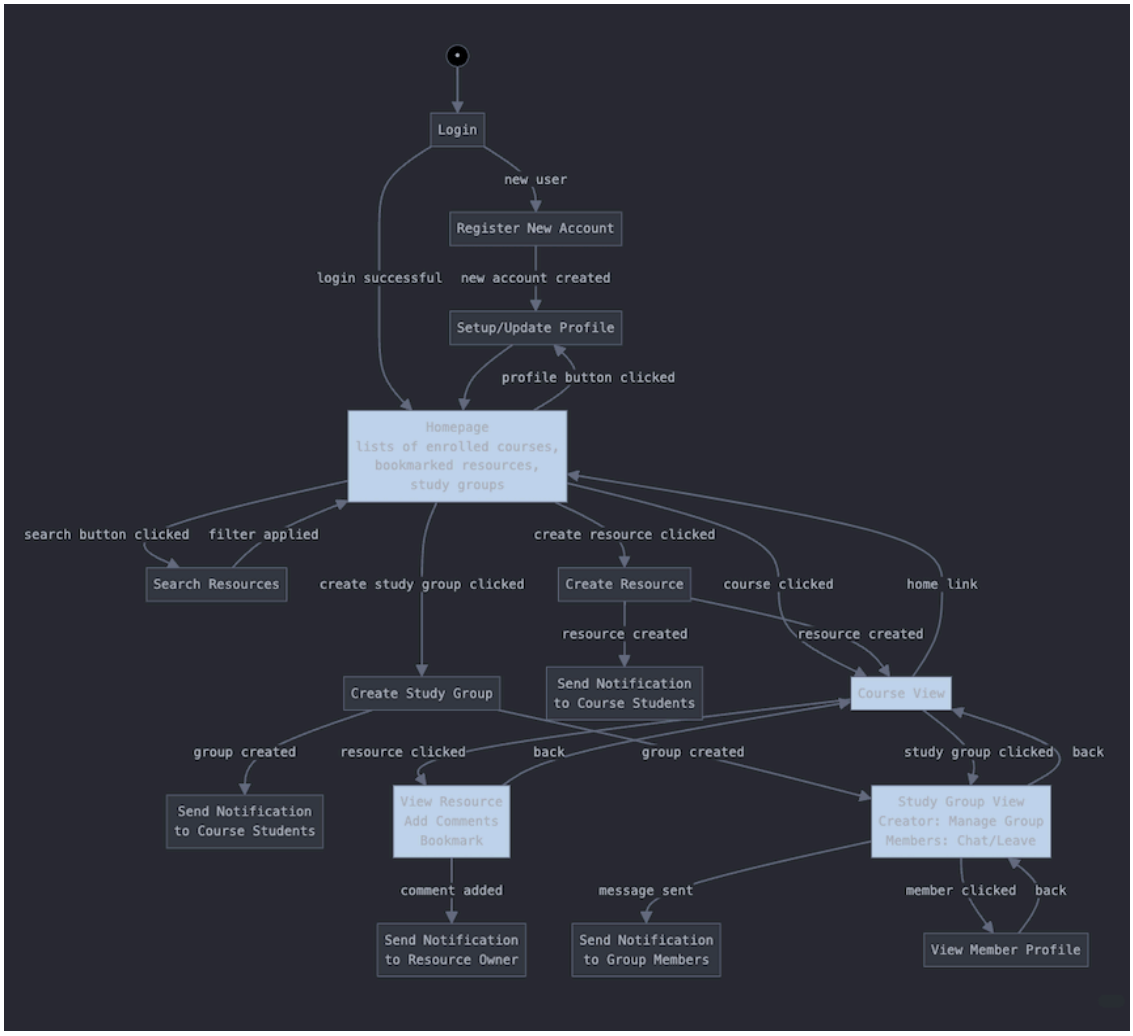
5. Study Group Chat Sequence



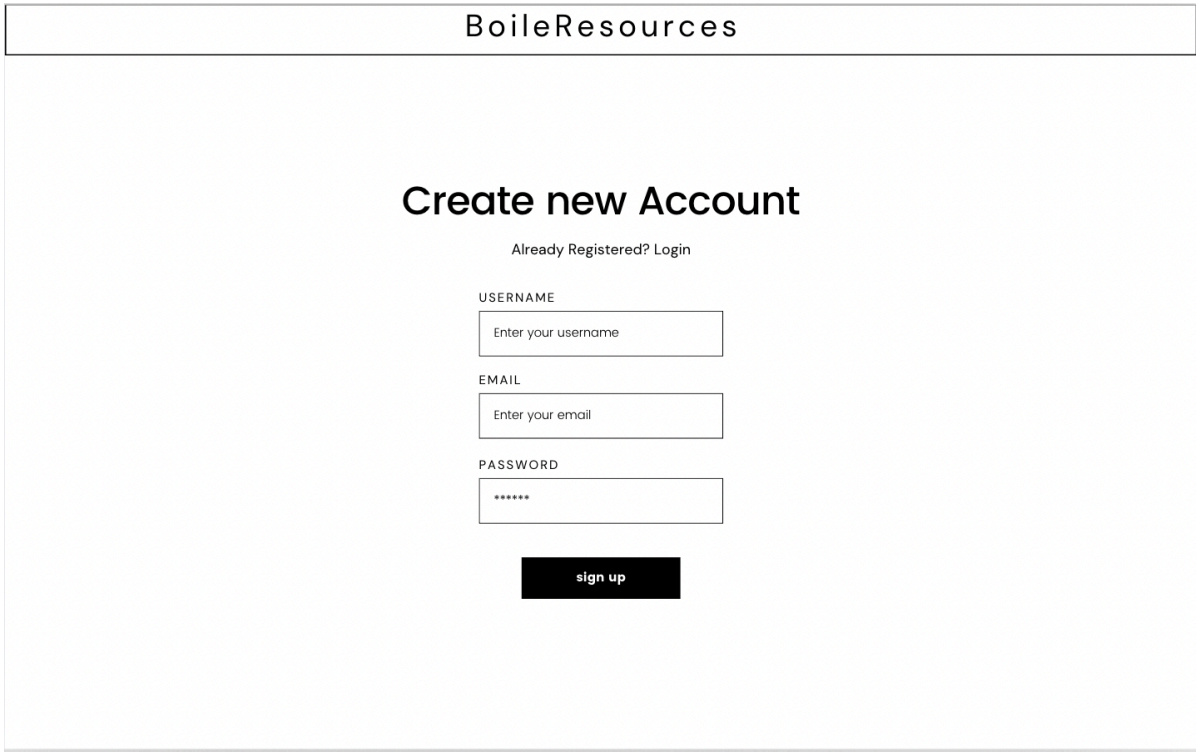
Navigation Flow Map:

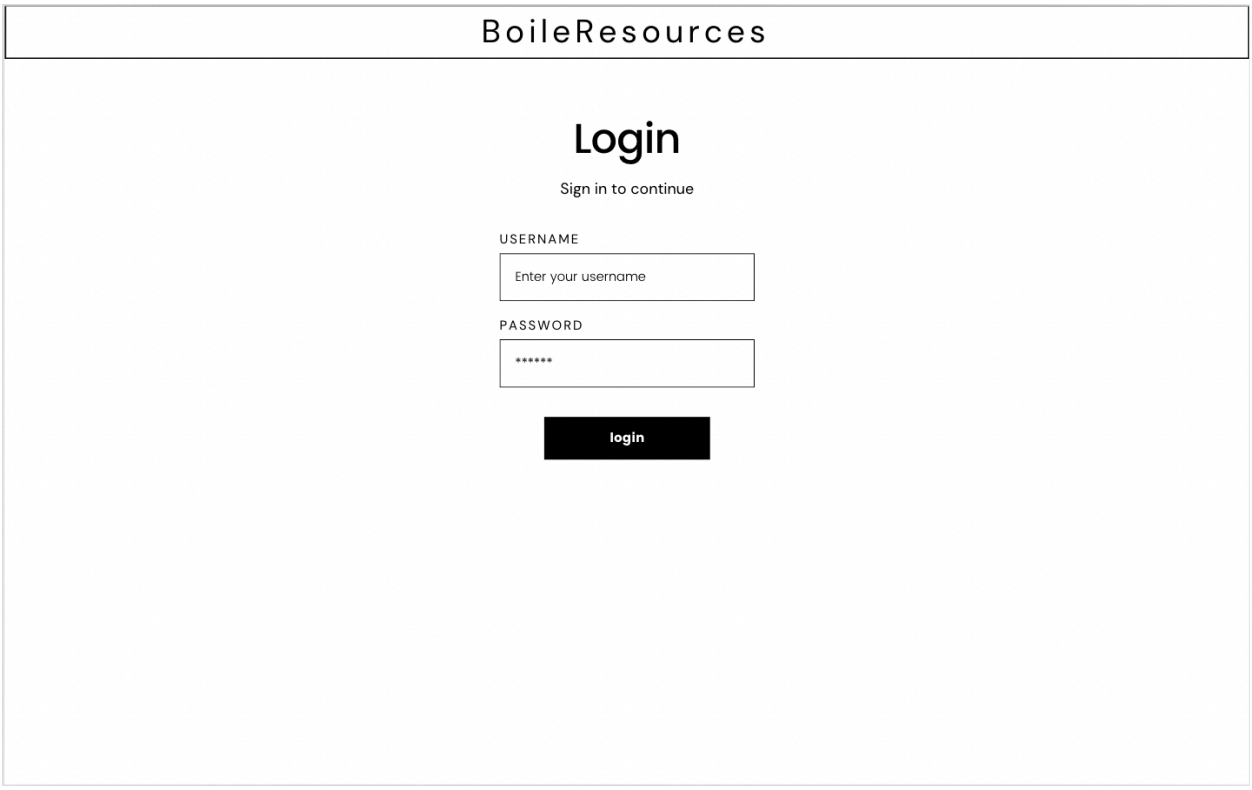
The design of BoileResources prioritizes simple, efficient access to academic content. Users log in with their Purdue credentials to access their personalized homepage, which displays enrolled courses, saved resources, and study groups. A consistent navigation bar provides quick access to core features, while a search bar enables filtering of resources. Course pages centralize all related content including shared materials, study groups, and professor details. The intuitive layout allows students to easily manage their academic resources and collaborate with peers.

Diagram shows how all features are accessible through the main navigation bar, with course-specific content organized within dedicated pages. This structure helps students quickly locate and share academic resources within the context of their courses.



UI Map:


BoileResources
Create new Account
Already Registered? Login
USERNAME
Enter your username
EMAIL
Enter your email
PASSWORD
******
sign up


BoileResources
Login
Sign in to continue
USERNAME
Enter your username
PASSWORD
******
login

User01's Classes:

CS30700

Software Engineering I

GPA: 3.34

Credits: 3

CS381000

Introduction to The Analysis of Algorithms

GPA: 3.01

Credits: 3

MA261000

Multivariate Calculus

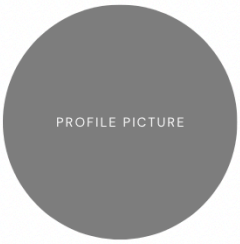
GPA: 3.23

Credits: 4

BoileResources

Settings

log out



edit profile picture

USERNAME

Example username

edit username

EMAIL

Example email

edit email

PASSWORD

******

edit password

MAJOR

Example major

edit major

BoileResources

My Calendar

TIME	MONDAY	TUESDAY	WEDNESDAY	THURSDAY	FRIDAY
09.00					
10.00	CS 180		CS 180		CS 180
11.00					
12.00		SCLA 101		SCLA 101	
13.00					
14.00					
15.00					
16.00					
17.00					