



BoilerDate

CS 30700
Design Document

Team 5

Youngjun Yoo

Chaewon Lee

Jeongbin Lee

Hosung Ryu

Purpose

Currently, there are many dating apps available online, such as Tinder and Bumble. However, Purdue STEM students might find it challenging to find a suitable match on these platforms.

The purpose of this project is to create an alternative dating web application, BoilerDate, that specifically targets Purdue students, with a strong focus on those in STEM majors. This is because students in STEM are known for their academic strength, but are also often introverted and may have weak social skills. BoilerDate will be catered towards students who want to find their significant other, not only based on common dating app features but also based on common academic goals and interests. BoilerDate will enable users to create their profiles based on different features such as photos, interests, hobbies, and skills. Users can view other users' profiles and match with other users by liking or disliking them. When matched, users can interact with each other through a text message system that the application provides so that they can know each other better and form relationships.

Design Outline

High-Level Overview



This project will be a web application that allows Purdue students to find their potential match, whether it be a friend or significant other, based on their interests and preferences, and interact with each other. They will be able to filter other users and choose their best matches. Our application will leverage a client-server model with Express.js, a Node.js framework. The server efficiently manages concurrent client requests, handling data access or storage in the database, and providing feedback to clients as needed.

1. Client

- a. Client provides a user interface for our application.
- b. Client sends HTTP requests to the server.
- c. Client accepts and interprets the HTTP response from the server, performing needed action that updates the user interface.

2. Server

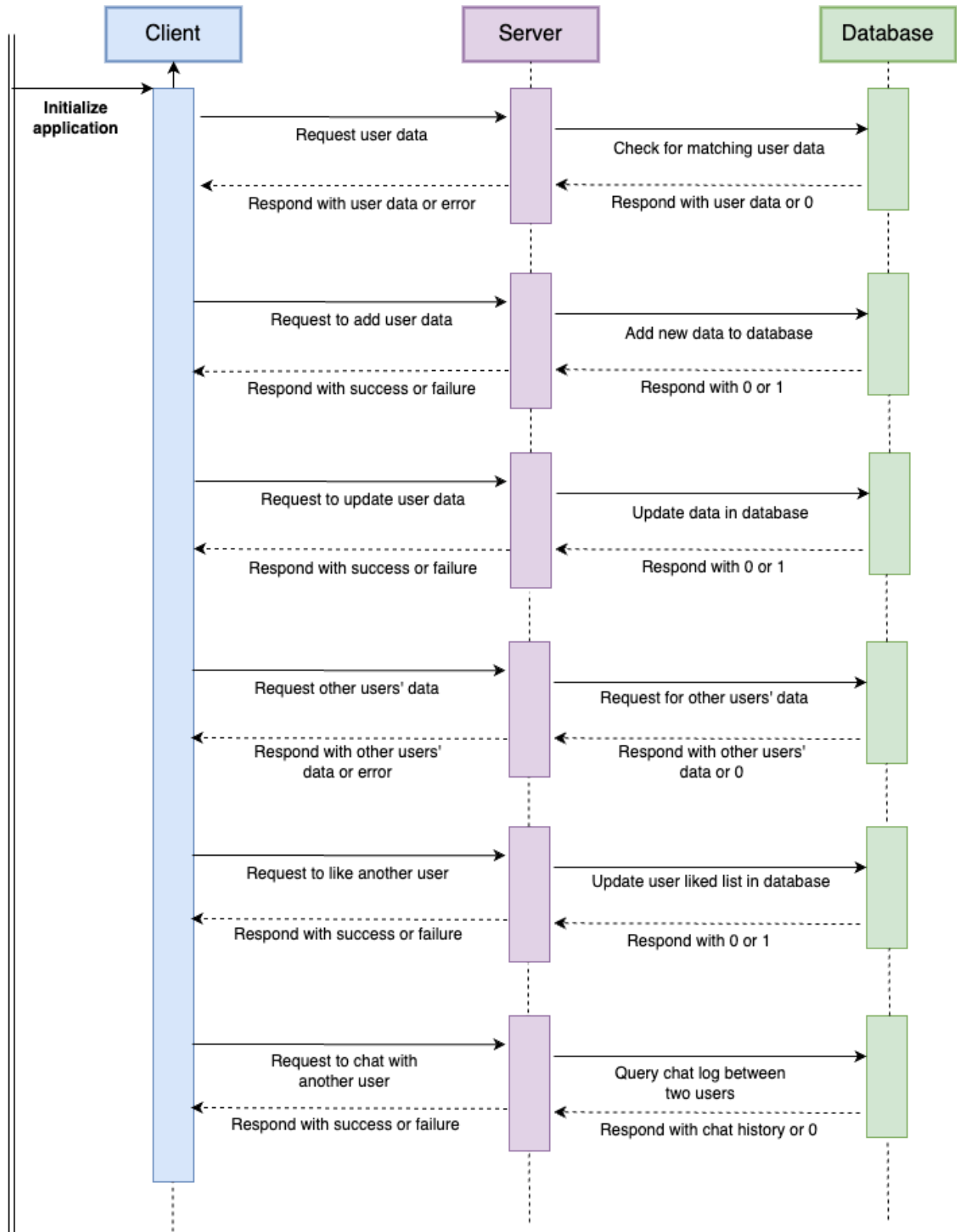
- a. Server manages and responds to HTTP requests received from clients.
- b. Requests are validated, and according to each valid HTTP request, the server interacts with the database, performing actions such as adding, modifying, or retrieving data.
- c. Server generates appropriate responses and sends the fetched data back to the specific clients over HTTP.

3. Database

- a. A non-relational database stores all application data, such as user information, potential matches list, user preferences, liked/matched list, etc.
- b. The database processes server queries and returns the retrieved data to the server using the JSON format.

Sequence of Events Overview

The sequence diagram below vividly presents how client, server, and database interactions are processed. The overall sequence is initially commenced by a user starting the web app. The user may log in or sign up, and the client sends appropriate requests to the server. The server processes the request sends a query to the database and receives the data from the database as well. After signing up/logging, the client sends more requests for other actions based on the user's choice, such as updating the user profile (including user preference and user settings), liking/disliking potential matches, chatting with matches, and any other events that require user interactions. The server will receive these requests and send queries to the database to gain corresponding data. The database will respond to these queries and return the data as requested back to the server.



Design Issues

Functional Issues:

1. What kind of information do we need for signing up?

- Option 1: Name, username
- Option 2: Name, Purdue email address
- Option 3: Name, gender, date of birth, Purdue email address

Choice: Option 3

Reason: Since our application is a dating website, we need essential user information such as name, gender, and date of birth, that are immutable. Also, we chose to require a Purdue email address instead of a username so that we can authenticate the Purdue email and check if the user is a Purdue student.

2. What kind of ways are there to verify a user's email address?

- Option 1: Search the Purdue Directory
- Option 2: Send verification code to user's Purdue email address
- Option 3: Send verification code to user's Purdue email address + Search the Purdue Directory

Choice: Option 3

Justification: To verify that a user's Purdue email address is valid, we can send a code to the user's Purdue email account, and the user has to type in the correct code.

Additionally, we will search the Purdue directory to verify once more that the Purdue email account exists and matches the user's name.

3. How should matched users communicate with each other?

- Option 1: Provide access to the phone number/email of the matched users
- Option 2: Provide a messaging service within the web app
- Option 3: Provide a video call service within the web app

Choice: Option 2

Justification: Since the very purpose of our project is to encourage Purdue STEM students to meet others, it is essential to empower the user to communicate with others spontaneously. Simply providing contact information is insufficient for introverted users to initiate communication. At the same time, jumping right up to video calls may even discourage users. Therefore, one of the less intimidating means to communicate with others is to chat by text.

4. In what kinds of formats can users input data for their profile?

- Option 1: Predefined tags only
- Option 2: Customizable tags only
- Option 3: Combination of predefined and customizable tags

Choice: Option 3

Justification: Everyone has their unique talents, skills, and strengths. Only selecting predefined tags will prevent users from appealing to their fullest. Therefore, we will have a section that allows users to create customizable tags about themselves. However, using customizable tags alone without predefined tags will make filtering matches difficult. Thus, having both predefined tags and customizable tags will allow users to appeal to others and choose matches in a way that customizable tags or predefined tags alone cannot fully cover.

5. How should users be able to block other users?

- Option 1: Block only liked users
- Option 2: Block only matched users
- Option 3: Block any users

Choice: Option 3

Justification: To ensure a safe and comfortable matching environment, users should have the ability to block any users they choose. They are given the choice to refrain from communicating with someone they do not desire, which can include a range of people from potential criminals to those they simply feel awkward around.

Non-Functional Issues:

1. What frontend language/framework should we use?

- Option 1: Vanilla JavaScript + HTML + CSS
- Option 2: TypeScript + Next.js
- Option 3: JavaScript + React.js + Tailwind CSS
- Option 4: JavaScript + Angular.js + Material UI

Choice: Option 3

Justification: JavaScript with React along with Tailwind CSS is an excellent choice for beginners due to its strong community support and rich resources online. JavaScript provides a solid foundation, React offers a component-based approach for UI development with a vast community, and Tailwind CSS simplifies styling through abundant utility classes. This modern stack allows beginners like us to create versatile projects and transition to more complex applications that we plan to develop. Moreover, the simplicity of Tailwind CSS also reduces the learning curve for styling, making it an accessible option for newcomers.

2. What backend language/framework should we use?

- Option 1: Spring Boot (Java)
- Option 2: Flask (Python)
- Option 3: Node.js/Express.js (JavaScript)
- Option 4: Ruby on Rails (Ruby)

Choice: Option 3

Justification: Opting for Node.js/Express.js for the backend aligns with a React + JavaScript frontend, forming a cohesive full-stack JavaScript application. The asynchronous nature of Node.js complements React's reactivity, ensuring efficient communication. The unified language stack enhances code consistency, and Node.js's scalability and performance synergize well with React's capabilities, providing a robust foundation for building modern, scalable web applications.

3. What kind of database should we use?

- Option 1: MySQL
- Option 2: PostgreSQL
- Option 3: MongoDB
- Option 4: SQLite3

Choice: Option 3

Justification: MongoDB is a compelling option due to its flexibility, scalability, and ease of development. As a NoSQL, document-oriented database, MongoDB accumulates dynamic data structures, making it suitable for projects with versatile requirements. Its schema-less design simplifies development, allowing developers to work with data more originally. Additionally, its JSON-like document format facilitates seamless integration with JavaScript, aligning well with the React + Node.js/Express.js stack.

4. Which API should we use to implement the chatting service?

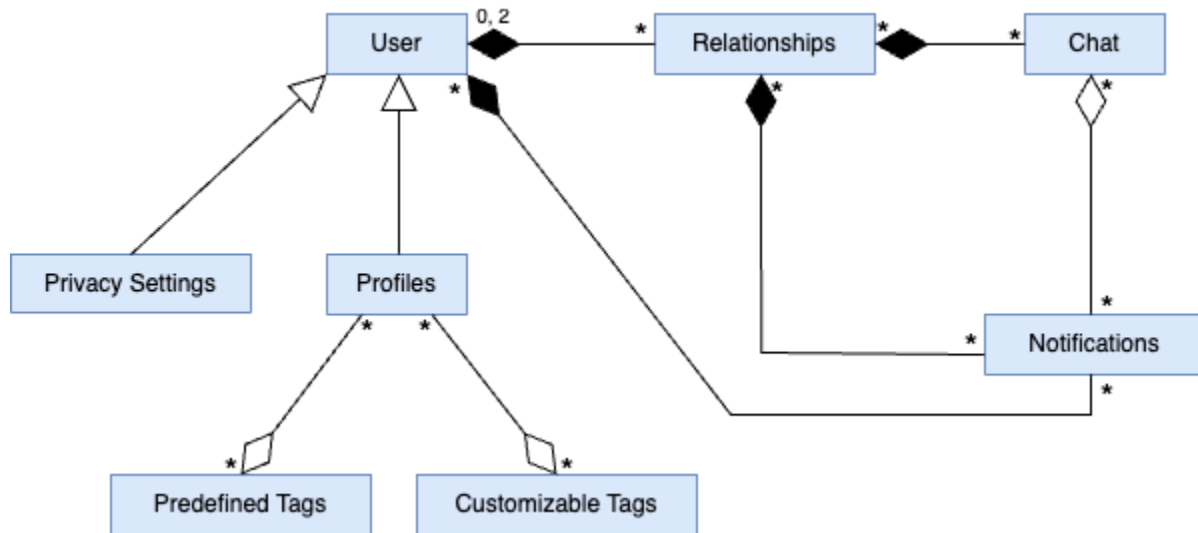
- Option 1: TalkJS
- Option 2: Sendbird
- Option 3: Rocket.Chat
- Option 4: DeadSimpleChat

Choice: Option 4

Justification: DeadSimpleChat is an excellent option for beginners implementing chat service due to its simplicity and ease of integration. Its user-friendly design and straightforward setup cater to those new to chat implementation, providing a smooth learning curve. Additionally, it's a cost-effective approach for medium-sized projects and its clear documentation ensures quick understanding and integration into the codebase. Its predefined themes and custom fonts/colors allow us to easily customize our chat interface.

Design Details

Class Diagrams



Our web application will have the following implementation of classes:

- Profiles and Privacy Settings classes are an extension of the User class.
- Profiles class has an aggregate relationship with Predefined Tags and Customizable Tags classes as the Profiles class can exist without these two Tags classes but does depend on them.
- Relationships and Notifications classes have a composite relationship with the User class since these classes cannot exist without the User class.
- Notifications and Chat classes have a composite relationship with the Relationships class since these classes cannot exist without the Relationships class.
- Notifications class has an aggregate relationship with the Chat class since the Notifications can exist without the Chat.

User
user_id: String
email: String
password: String
first_name: String
last_name: String
created_at: Date
date_of_birth: Date
gender: ENUM
get<Attribute>()
set<Attribute>()

Profiles
photos: String[]
my_description: String
height: Float
gpa: Float
major: ENUM
interests: ENUM[]
hobbies: ENUM[]
personality_type: ENUM
degree: ENUM
citizenship_status: ENUM
lifestyle: ENUM[]
relationship_goals: ENUM[]
skills: String[]
employment_history: String[]
career_goals: String[]
github_username: String
linkedin_profile: String
get<Attribute>()
set<Attribute>()
displayProfile()

Predefined Tags
height: Float
gender: ENUM
age: int
gpa: Float
major: ENUM
interests: ENUM[]
hobbies: ENUM[]
personality_type: ENUM
degree: ENUM
citizenship_status: ENUM
lifestyle: ENUM[]
relationship_goals: ENUM[]
get<Attribute>()
set<Attribute>()
searchBy<Attribute>()
searchHeightRange()
searchGPARange()
searchAgeRange()

Customizable Tags
skills: String[]
employment_history: String[]
career_goals: String[]
github_username: String
linkedin_profile: String
get<Attribute>()
set<Attribute>()

Relationships
one_way_like: hashmap<User, User>
one_way_dislike: hashmap<User, User>
match: hashmap<User, User>
unmatch: hashmap<User, User>
block: hashmap<User, User>
getRelationshipStatus()
setRelationshipStatus()
hideBlockedUser()
displayMatchedUsers()
displayUsersYouSentLike()
displayUsersWhoSentLike()

Privacy Settings
show_email: Boolean
show_github_username: Boolean
show_phone_number: Boolean
show_linkedin_profile: Boolean
display<Attribute>()
hide<Attribute>()

Notifications
target: User.email
likes: Boolean
match: Boolean
chat: Boolean
push_notification_agree: Boolean
allowNotifications()
blockNotifications()
allowChatNotifications()
blockChatNotifications()
allowRelationshipNotifications()
blockRelationshipsNotifications()

Chat
sender_id: String
receiver_id: String
content: String
status: ENUM
sent_at: timestamp
seen_at: timestamp
searchBySenderId()
searchByReceiverId()
displayChatStatus()

Description of classes and interaction between classes

1. Users

- User objects are created whenever a new user signs up.
- Each user will have a unique user ID.
- Each user will have to enter their email, password, first name, last name, birthdate, and gender to create an account.
- Each user will set up their profile based on predefined tags and customizable tags.
- Each user will be able to set up one's privacy settings.
- Each user will be able to choose to receive notifications.

2. Profiles

- Profile objects will be created when the user creates an account.
- Each profile will have a URL of photos uploaded by a user, profile description, height, GPA, major, interests, hobbies, personality type, degree, citizenship, lifestyle, relationship goals, skills, employment history, career goals, GitHub username, and LinkedIn profile.
- Each profile will have a short description that will be displayed on their main page along with their photos, name, major, and GPA.
- User profiles will later be used for other users to filter out and search for their desired (predefined) tags.

3. Predefined Tags

- Predefined tags are a list of options for each attribute.
- Each user will be able to choose one or more options (depending on the tag) from the list of predefined tags as one of their attributes.
- It will be used for filtering a list of other users for potential matches that match a user's preferences.

4. Customizable Tags

- Customizable tags are a list of attributes that each user can customize.
- Each user can choose to input one or more specific skills, employment history, and career goals for their profile.
- Each user can also choose to input their GitHub username and LinkedIn for their profile.
- Customizable tags cannot be used to filter other users for potential matches, but they can be displayed with each profile.

5. Relationships

- Relationship object is created when two different users interact and perform an action upon each other. There are 5 kinds of relationship types in the Relationship object.
- If only one user likes another user, it will form a relationship type of one-way like.
- If both the users like each other, it will form a relationship type of match.
- If one user dislikes the other, it will form a relationship type of one-way dislike.
- If one user decides to unmatched from the match, it will form a relationship type of unmatched.
- If one user decides to never see another user while using the app, it will form a relationship type of block.

6. Chat

- Chats object is created when there is a new value added in the match hashtable.
- Chat objects will have sender ID and receiver ID, which are defined in whichever sequence two users are mapped in the match hashtable.
- Chat objects will have content(chat log), status when the last message was sent, and when it was read.

7. Notifications

- Notification object is created when the Relationship object mapped to the user has changed (new like, new match, and new message).
- Notification objects will have a target, likes, match, chat, and push notification agreement.

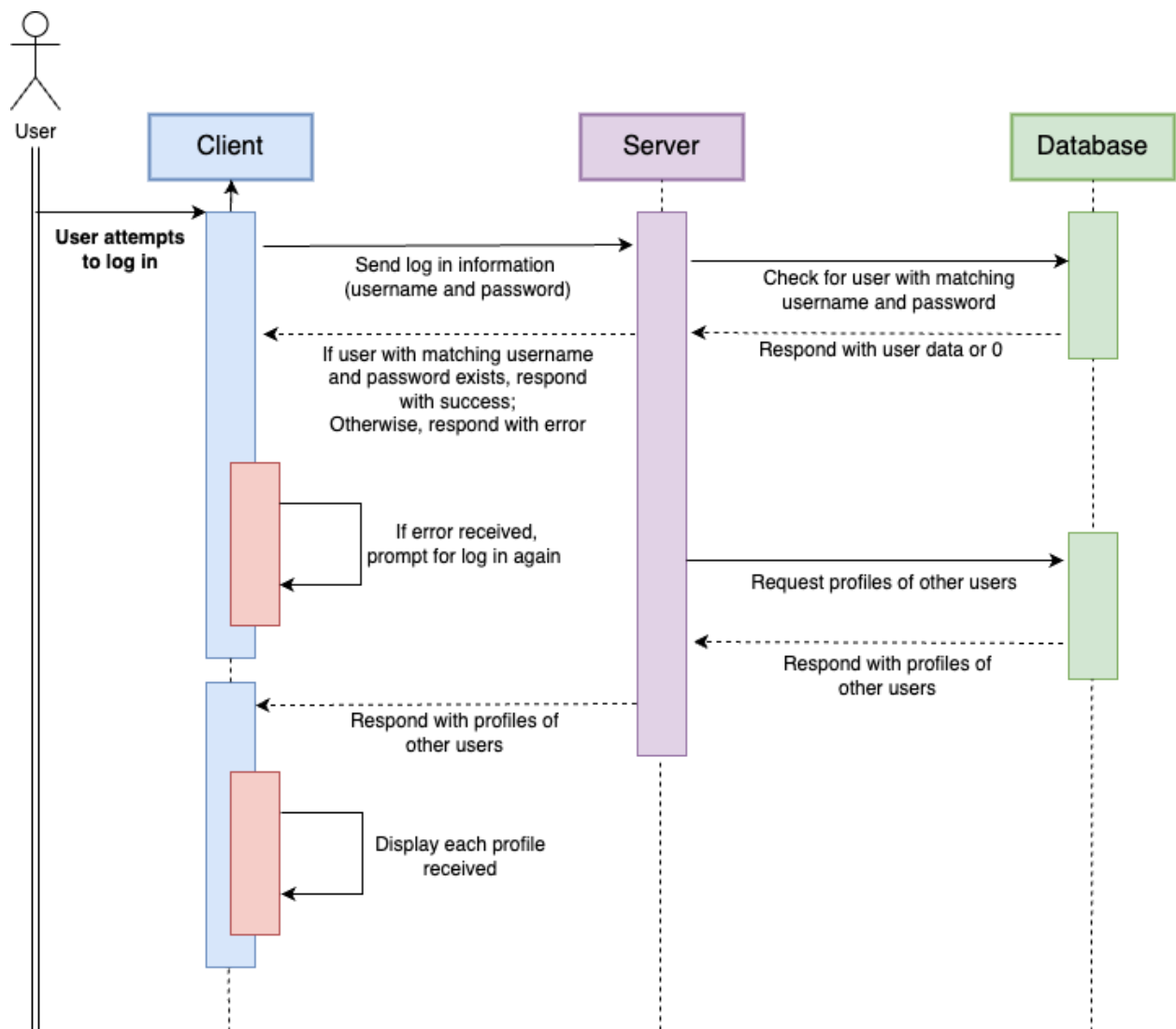
8. Privacy Settings

- Privacy Setting object is created when a new User object is created.
- Each privacy setting will have boolean variables to determine whether to display the User's email, GitHub username, phone number, or LinkedIn profile.
- Users will set each variable to True or False to ensure one's privacy.

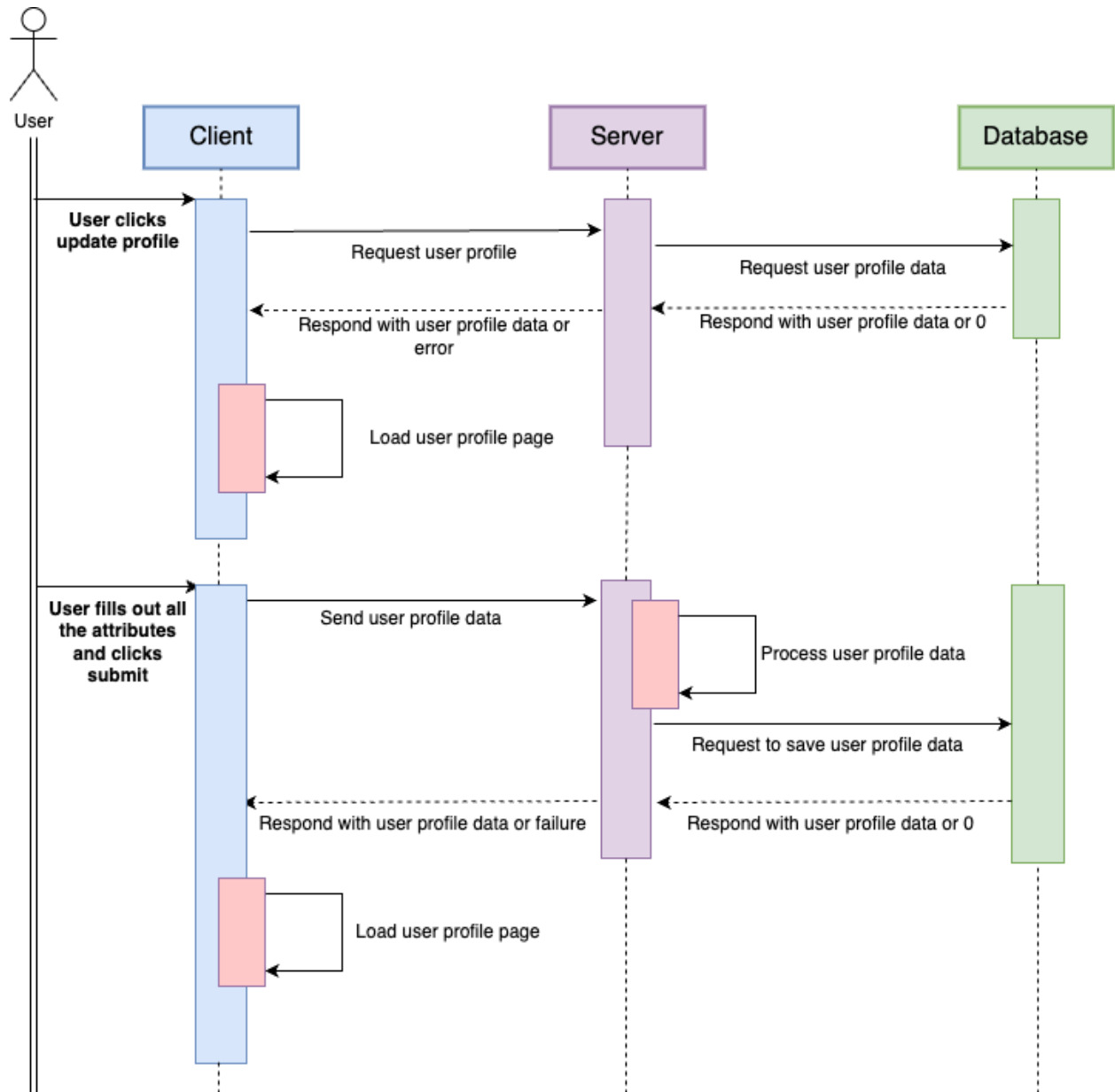
Sequence Diagrams

The following diagrams represent the series of major events in this application, including user login, creating a profile, liking/disliking, and matching with another user, applying a filter to view other users' profiles, and blocking another user. The following sequences demonstrate how the client, server, and database send requests and receive responses.

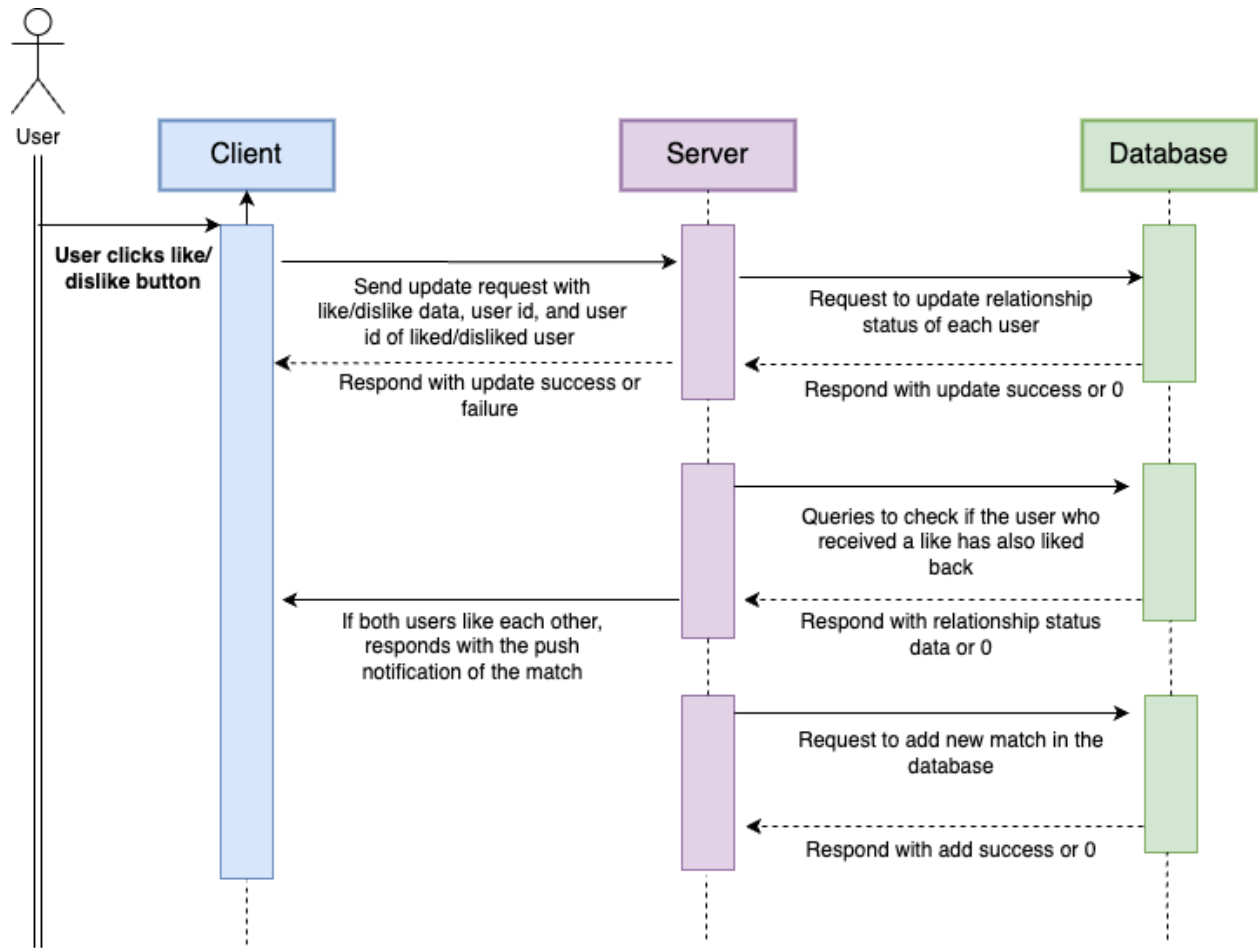
1. User logs in



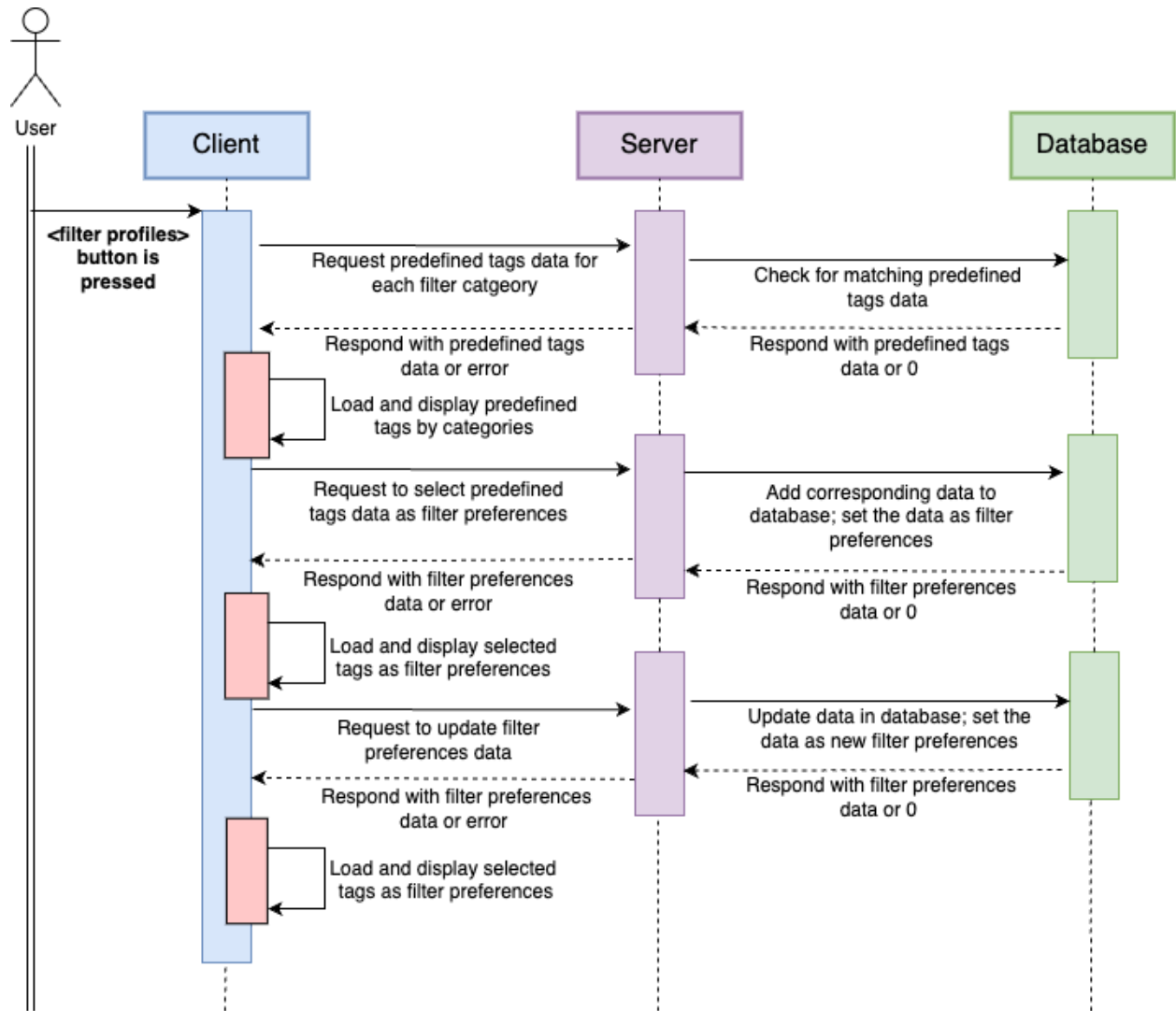
2. User creates profile



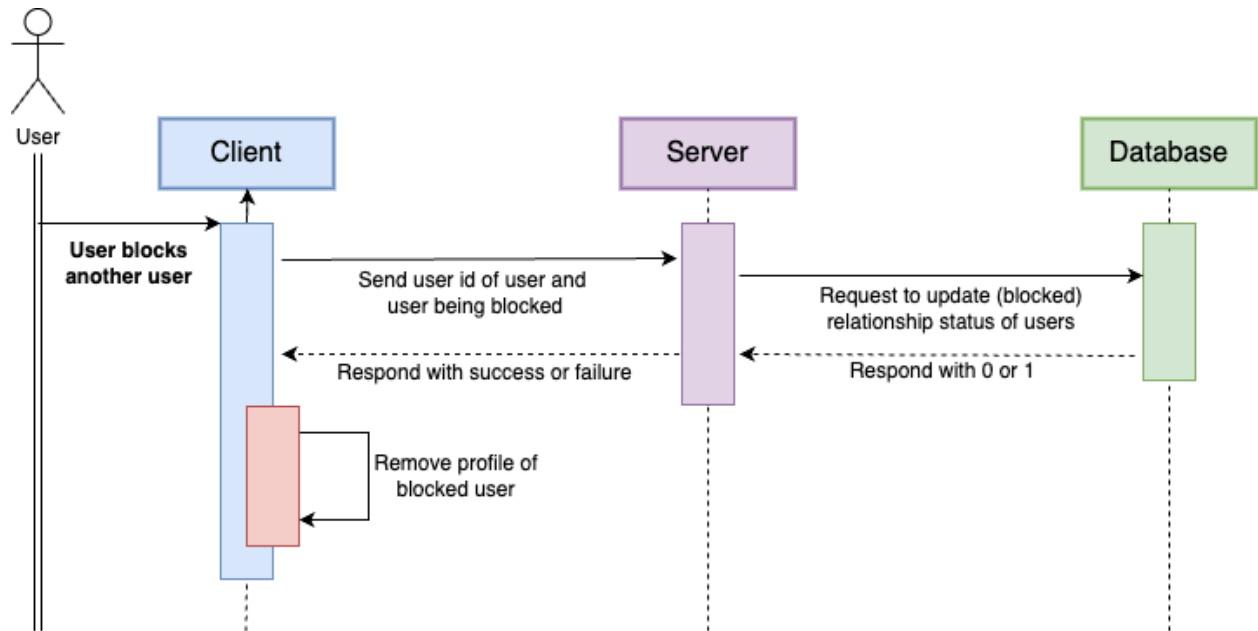
3. User likes/dislikes/match another user



4. User applies a filter for viewing others' profiles

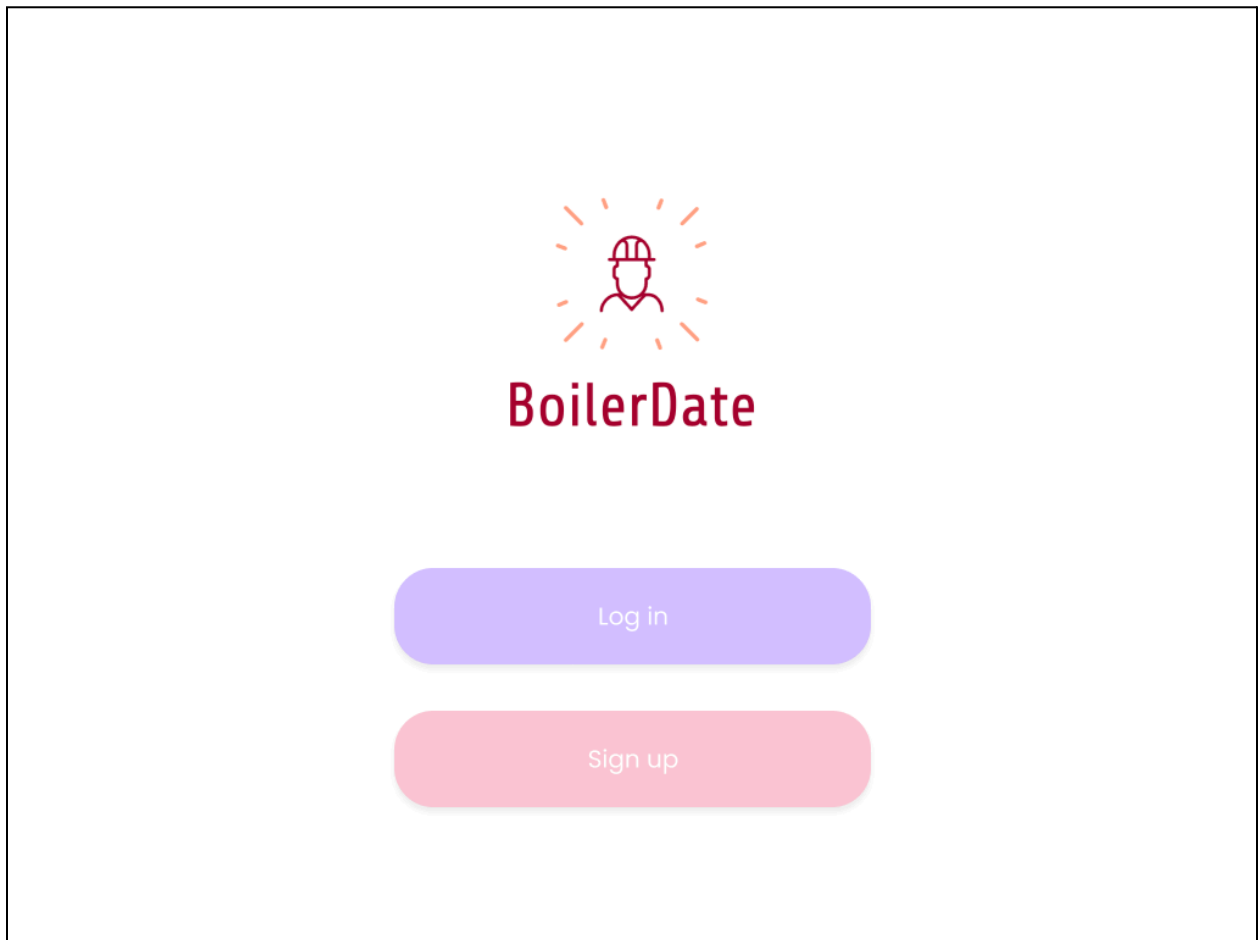


5. User blocks another user



UI Mockup

The following diagrams represent the possible UI of our application, including the landing page, sign-up pages, log-in page, and the interests, filtering, discovery, matches, and messaging page. We wanted to create a modern minimal UI that would be both easy for users to use and aesthetically pleasing.



This is the landing page the user will be able to see when they first open the application. There is a “Sign up” button for new users and a “Log in” button for returning users.

Create an account

Already have an account? [Log in](#)

1

2

3

Enter your email adress

Provide your basic info

Create your password

What's your Purdue email?

Enter your Purdue email address

Next

Create an account

Already have an account? [Log in](#)

1

2

3

Enter your email adress

Provide your basic info

Create your password

Profile name

Enter your profile name

What's your gender? (optional)

☐ Female

☐ Male

☐ Non-binary

What's your date of birth?

Month

Date

Year

Next

Create an account

Already have an account? [Log in](#)

1

2

3

Enter your email adress

Provide your basic info

Create your password

Password


Hide

Enter your password

Use 8 or more characters with a mix of letters, numbers & symbols

☒ By creating an account, you agree to the [Terms of use](#) and [Privacy Policy](#).

☒ I'm not a robot



Sign up

Already have an account? [Log in](#)

This is the sign-up page where the user will be providing their Purdue email address, and basic information, such as name, gender, and date of birth for the creation of their account. The user can also create his or her password for the account.



BoilerDate

User name or email address

Your password

 Hide

[Forgot your password](#)

Log in

Don't have an account? [Sign up](#)

This is a login page where users can enter their email address and password. Users may click “Forgot your password” if needed to reset their password. Users may click “Sign up” to create an account.

Filters

Clear

Interested in

Girls

Boys

Both

MBTI

ISFJ

>

Height

5' 4

>

Degree

Bachelor's

>

Major

Computer Science

>

Interests

Sleeping

>

Citizenship

United States

>

Lifestyle

Gymgirl

>

Relationship Goals

Grow together

>

GPA

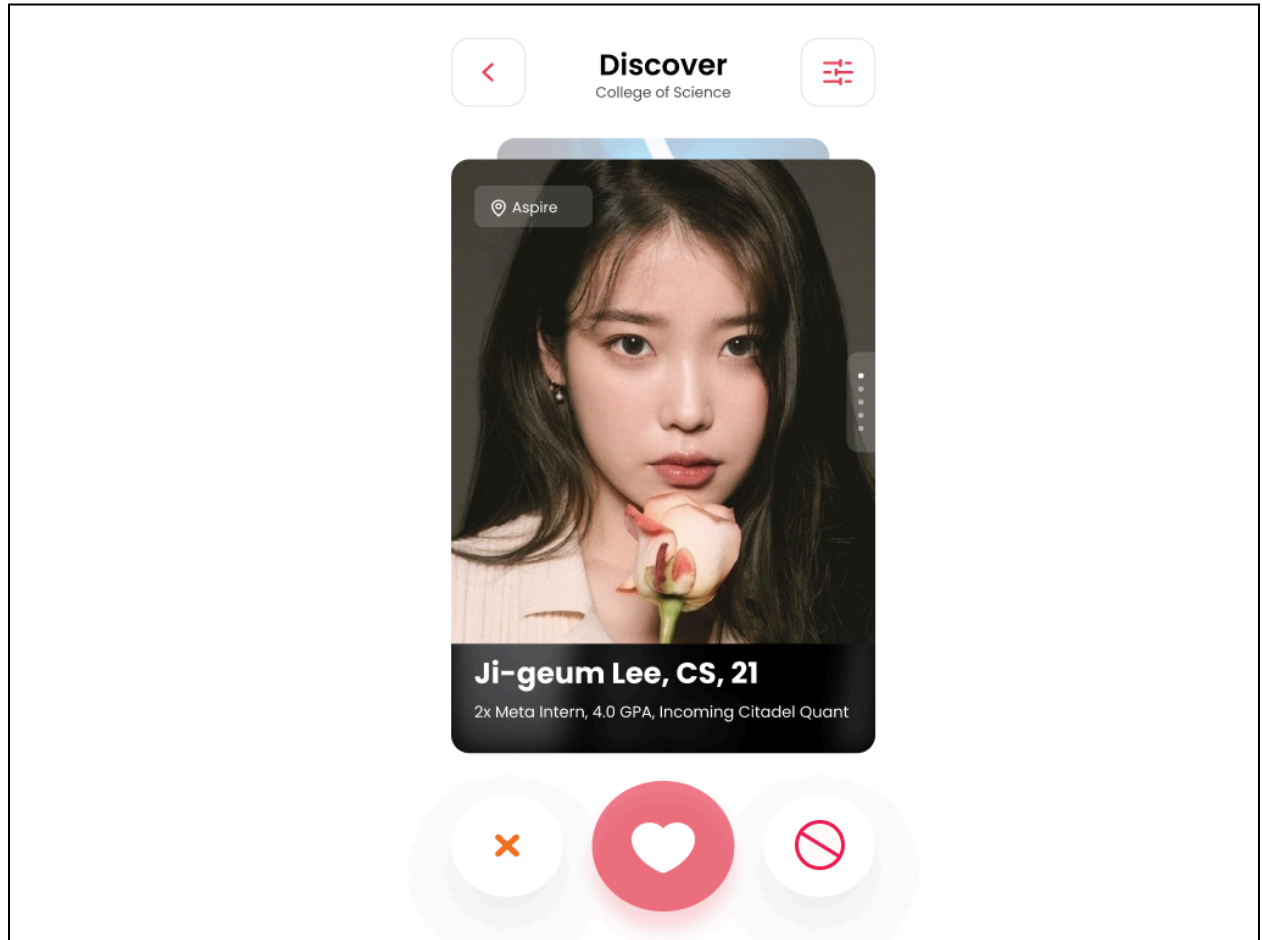
3.5-4.0

Age

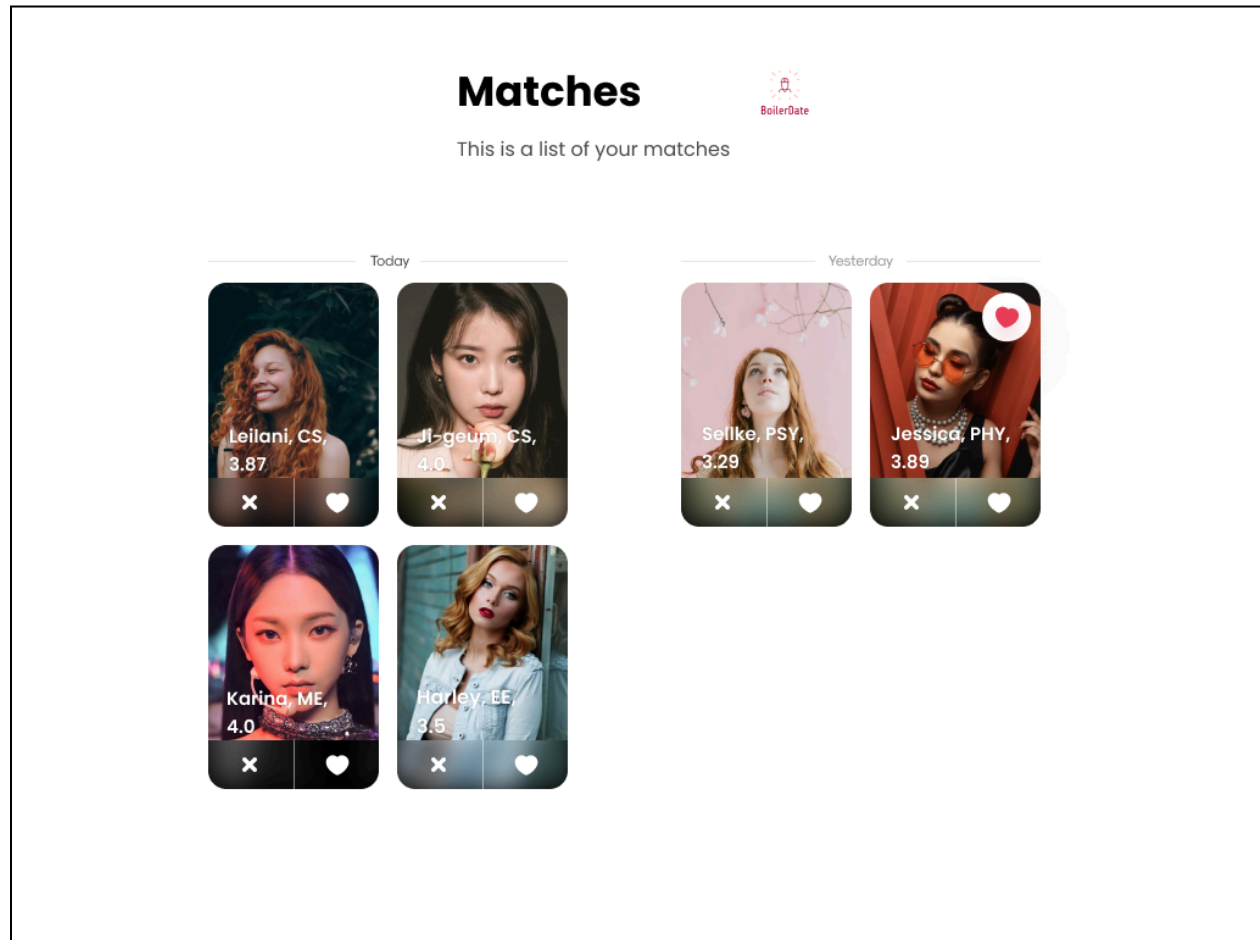
18-22

Continue

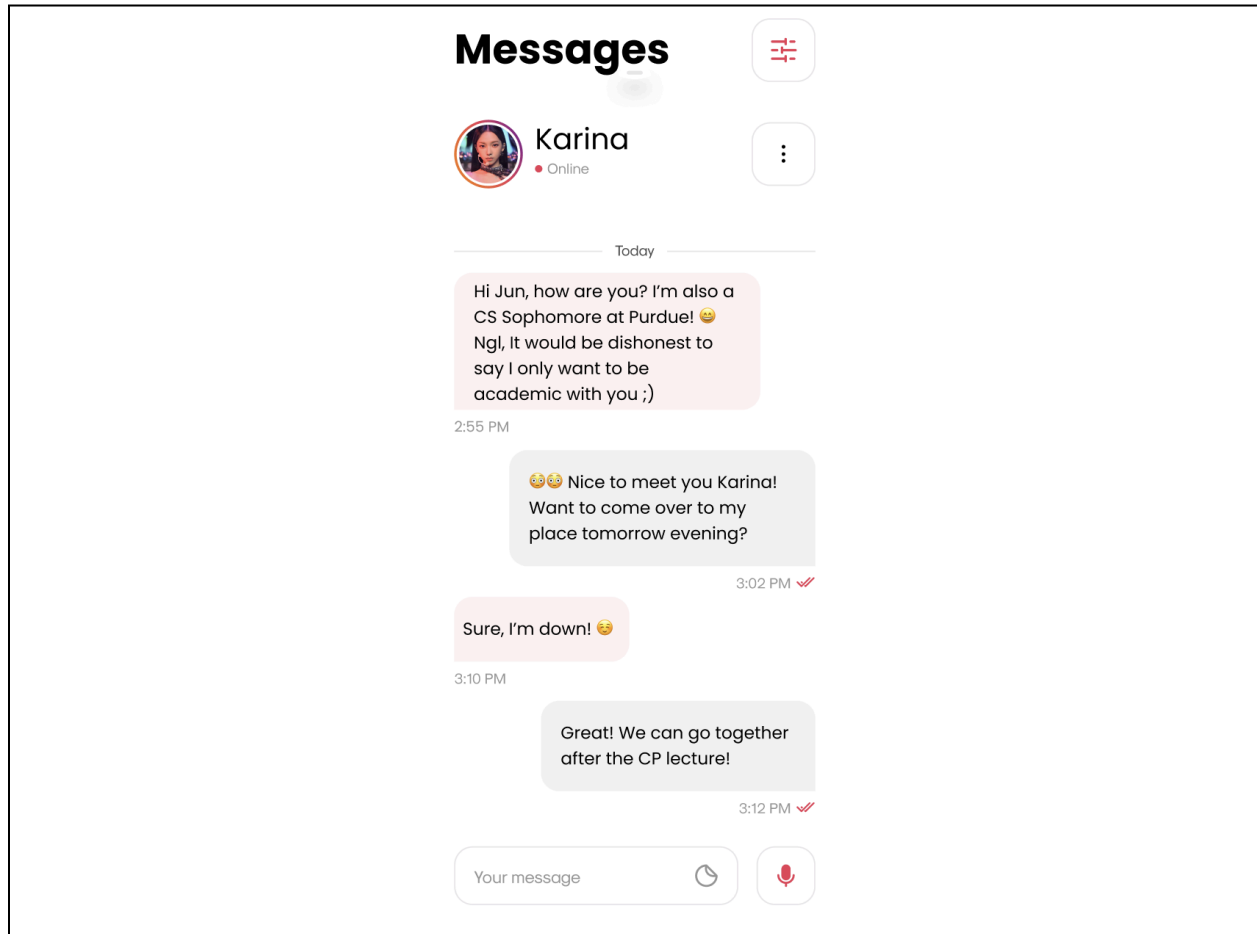
This is a filtering page where users can configure their preferences and filter out other users to find someone they would like. They may click “Continue” to move on to the next page, which contains another set of filters they can configure.



This is a discovery page where users will be shown other users' profiles. They may click the "X" button to dislike the user, click the "heart" button to like the user, or click the "block" button to block the user. The page will automatically reload and display the next user profile as soon as the user makes any selection.



This is the matches list page. The user can view the list of the profiles of the users he or she matched with. There will also be a likes sent and likes received page which will have a similar format as this page.



This is the messaging page. The user can send text messages to a matched user and also receive text messages from a matched user. The double-check mark indicates that the receiver has read the message.