Scholasticate

Team 8 - Design Document

Daniel Karagory, Andrew Lanham, Jackson Rosenberg, Colston Streit, Devin Vering, Ethan Zhu

Table of Contents

- Purpose
    - Non-Functional Requirements
    - Functional Requirements
- Design Outline
    - High-Level Overview
    - System Components
    - Sequence of Events Overview
    - Activity Diagrams
- Design Issues
    - Non-Functional Issues
    - Functional Issues
- Design Details
    - Class Design and Diagrams
    - Sequence Diagrams
    - UI Mockups

Purpose

College campuses are known for their ability to bring students together for a variety of reasons, both socially- and academically-related. However, when it comes time to complete difficult assignments, many students are forced to work individually around campus because they don't know anyone in their class. It can be very difficult for students to break the ice and seek out the help that they need from their peers.

Scholasticate fixes this problem by using location services to find students who are studying the same topic nearby and by bringing these students together for a more productive and collaborative study session. Scholasticate will offer course- and section-level filtering so that students can easily find people who are not only studying the same subject, but who also are in the same course section. This is important because many courses with sections taught by different professors often have different homework assignments; therefore, students may want to focus their search for study partners on people with the same assignments and course experiences.

Overall, the purpose of Scholasticate is to make it easier for study groups to be formed outside of the classroom. We believe this could significantly boost academic cooperation while also helping students to meet new people and make friends in the process.

Non-Functional Requirements

1. **Performance Requirements**
   As a developer,
   a. I would like the back-end server to be able to handle at least 1,000 simultaneous requests.
   b. I would like each user's location to be updated at regular intervals.
   c. I would like the application to be fluid and stutter-free on all platforms.

2. **Client Requirements**
   As a developer,
   a. I would like the application to be available on an Android phone.
   b. I would like the application to be available on an iOS device.
   c. I would like the application to be available on a generic web platform.

3. **Server Requirements**
   As a developer,
   a. I would like the server to be able to support real-time communication with all connected clients.
   b. I would like the server to be able to save relevant data (users, current study groups, etc.) to a database.

4. **Usability Requirements**
   As a developer,
   a. I would like the front-end to be easily understandable to the average user.
   b. I would like the front-end to be aesthetically pleasing for all screen resolutions.

5. **Security Requirements**
   As a developer,
   a. I would like the users' account information to be encrypted.
   b. I would like the users' current location data to be encrypted.
   c. I would like the application to be resistant to SQL injection and other common exploits.

6. **Deployment Requirements**
   As a developer,
   a. I would like to use Docker to allow for easy compartmentalization of the back-end server.
   b. I would like rebuilding and testing to be done every time new code changes are committed to the repository to support continuous delivery and integration.

Functional Requirements

1. **Users can create, edit, and manage their accounts.**
   As a user,
   a. I would like to register for a Scholasticate account.
   b. I would like to login to my Scholasticate account.
   c. I would like to access my Scholasticate account from web browsers on both mobile devices and desktop computers.
   d. I would like to reset my password if necessary.
   e. I would like to delete my account and all my information if desired.
   f. I would like to select which course(s) I'm currently studying.
   g. I would like to edit my user profile's personal information (name, pronouns, profile picture, etc.).
   h. I would like to edit my user profile's academic information (expected graduation date, major, minors, etc.).
   i. I would like to add a small biography to my profile to describe study habits and personality traits.
   j. I would like to set times where I am available and unavailable.
   k. I would like to have my course history saved to my profile to inform other users who may request tutoring assistance.
   l. I would like to update wardrobe information in order to help find other people more seamlessly.
   m. I would like to have the option to display my profile as offline after finding another student to work with.

2. **Users can customize their preferences regarding location services.**
   As a user,
   a. I would like to manually refresh the location services in addition to having it be automatically updated at regular intervals.
   b. I would like to customize the location service's refresh rate.

3. **Users can view other active users both on a map and in a list.**
   As a user,
   a. I would like to see a list of users who are currently online.
   b. I would like to select and view individual user profiles within the list.
   c. I would like to see a map displaying everyone's locations.
   d. I would like to be able to zoom in and out on a map.
   e. I would like to select and view individual user profiles as a pop-up window on the map portal.
   f. I would like to see a list of online users sorted by distance.
   g. I would like to see a list of all friends who are currently online.
   h. I would like to see a distinction between my friends and other users.

4. **Users can manage their relationships and communications with other users.**
   As a user,
   a. I would like to send a message to another user.
   b. I would like to screen messages from non-friend users before beginning a direct message chain.
   c. I would like to delete a message chain so that I can't see it anymore.
   d. I would like to friend another user to potentially reconnect during another study session.
   e. I would like to remove users from my friends list.
   f. I would like to block another user if they are bothering me in some way.
   g. I would like to report another user if they are being inappropriate.

5. **Users can search for other users with whom to study using a variety of filters.**
   As a user,
   a. I would like to search for people studying the same course using keywords.
   b. I would like to search for people studying the same course using specific course number information.
   c. I would like to filter nearby students by school.
   d. I would like to filter nearby students by course.
   e. I would like to filter nearby students by course section.
   f. I would like to filter nearby students by my friends list or if I belonged to a shared study group in the past.

6. **Users can help contribute to the database by adding their own courses and schools so future users can find them.**
   As a user,

a. I would like to request that a new school be added to the database for later searches.
b. I would like to request that a new course be added to the database for later searches.
c. I would like to request that a new course section be added to the database for later searches.

7. **Users can interact with existing study groups.**
   As a user,
   a. I would like to have the option of publicly displaying active meetups, or study groups, I participate in.
   b. I would like to see all study groups distinctly marked on the map and list interfaces.
   c. I would like to view the study group profile and the individual profiles of all of the members present.
   d. I would like to request to join an active study group and send direct messages to all current active members.
   e. I would like to receive messages from individuals from shared study groups in a group chat format.
   f. I would like to discern the owner of an active study group.

8. **Study group owners can manage their own study groups.**
   As an owner of a study group,
   a. I would like to name the study group and have the name displayed on the user interfaces.
   b. I would like to edit a study group "profile" with a similar structure to the user profiles.
   c. I would like to create invitations for the group.
   d. I would like to restrict the number of members in the group, and remove members if necessary.
   e. I would like to have settings to schedule recurring iterations of the same study group at a certain time and location.
   f. I would like to transfer ownership of the group to another member.
   g. I would like to delete the study group from the system if it is no longer relevant.

9. **Users can receive and customize notifications.**
   As a user,
   a. I would like to receive pop-up notifications when a message is sent to my profile.
   b. I would like to receive pop-up notifications when a new member joins a current study group I participate in.
   c. I would like to receive pop-up notifications when a scheduled study group meeting is about to begin.
   d. I would like to change notifications preferences in my user profile and receive email notifications if desired.

Design Outline

In Scholasticate, many users will be sharing their location information with each other, as well as access shared information about study groups and user profiles. As such, the project uses the client-server model, with one central server handling requests and information from all clients. The server will communicate with an integrated SQLite database to store and access user information. As interactions within this project are primarily through requests from the client to the server, we use a transaction-processing design pattern.

The overall role of each component is as follows:

Client:

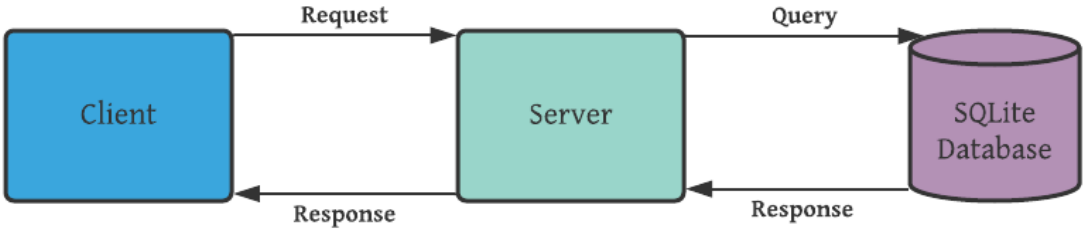
- Displays user profile, study groups and nearby user locations
- Sends requests to create, join or leave a study group to the server
- Creates search queries for profile and study groups
- Sends and receives direct messages and group messages
- Requests addition of new courses and universities

Server:

- Stores profile and study group info from clients into the database
- Aggregates and distributes geolocation info between clients
- Retrieves and sends a client's search results from the database
- Distribute client messages to recipients
- Creates new schools and courses after manual approval

Database:

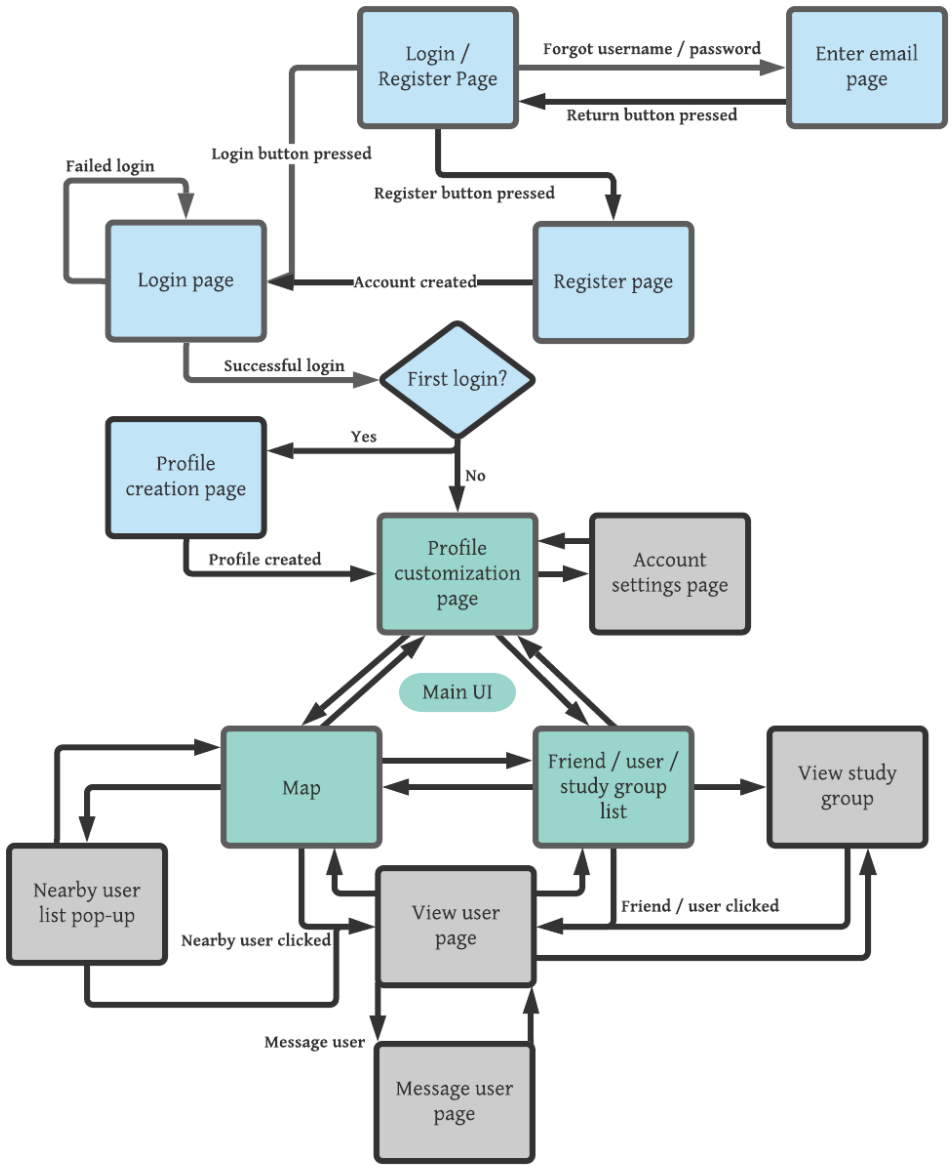
- Contains user profile and study group information
- Contains a list of schools and their courses



High-level view of the design

Activity / Navigation Flow Diagram

The following chart below details potential navigation paths the user can take while interacting with the application. The chart is broken into distinct sections by color, with blue representing the user authentication and profile-based actions and interfaces, green representing the main UIs and grey representing interfaces for connecting with other students. The main user navigation will normally be a series of transitions between searching for users in the map and list UIs and connecting with users in more intrapersonal-focused spaces.



Design Issues

Here are some design issues we considered while planning out our project.

Non-Functional Issues

1. **What programming language or framework should we use for the front-end?**
   Option 1: Pure JavaScript, HTML, and CSS
   Option 2: React.js
   Option 3: Python and Flask
   Choice: Option 2
   Justification: We chose to use the React framework largely because it allows for the easy construction of UIs due to the fact that everything in React is a component. Every component handles its own rendering and can be reused in other places, making it very straightforward to build consistent-looking UIs. In addition, there are a lot of great tutorials out there that will help us to quickly get into using React. Finally, due to its reliance on its component structure, React can easily be exported to mobile apps without any other code changes, meaning we can deploy to the web, to iOS devices, and to Android devices all using the same code base.

2. **How should we host the back-end?**
   Option 1: Google Cloud
   Option 2: Amazon Web Services
   Option 3: A traditional server owned by one of our members
   Choice: Option 3
   Justification: Despite Google Cloud's and AWS's free tiers or discounts for students, we opted to just use a traditional server that was already owned by one of our members. Using our own server, we wouldn't have to worry about paying any extra service fees. In addition, Daniel is already very familiar with his own server and is looking forward to giving it another purpose. If it turns out his server cannot handle the kind of load we're hoping our app can handle, then we may switch to either of the other options which should be trivial due to the fact that the back-end will be compartmentalized with Docker. For now, however, we're excited to be using our own server.

3. **What location service API should we use?**
   Option 1: Google Geolocation API
   Option 2: JS Geolocation API
   Choice: Option 2
   Justification: We've decided to just use the Geolocation API included within JavaScript due to the fact that we wouldn't have to be worrying about paying any costs included with the use of the Google Geolocation API. We wouldn't have to worry about API keys for the JavaScript API as well. In addition, even though there are subtle differences between the two APIs regarding how they get their location data, both of them at the end of the day spit out a latitude and a longitude that we can later display using the Google

Maps API or something similar. Therefore, overall, it just seemed simpler to use the JavaScript API.

4. **How do we want to deploy our code changes?**
   Option 1: Manual building and testing
   Option 2: Automatic building and testing with Docker
   Choice: Option 2
   Justification: We opted to use Docker so that we could automatically build and test our code with each committed code change. While it might take extra work to set it up, we believe it will save us time in the end, especially since this will be a large-scale project. In addition, this would support a continuous delivery / integration pipeline, which is an important part of most applications being created today. It would be good for us to practice this skill. Finally, using Docker will additionally allow for easy compartmentalization of the back-end server.

5. **Which platforms should we explicitly provide support for?**
   Option 1: iOS devices
   Option 2: Android devices
   Option 3: The Web
   Option 4: All of the above
   Choice: Option 4
   Justification: While it might seem like a lot of extra work to plan to support all three of the major platforms out there today, we believe that it will be doable because we are using the React framework to design the front-end. As mentioned previously, everything in React is a component, and each component handles its own rendering. Because of this, React apps can be exported as mobile applications rather seamlessly. That's why we believe that it will only take a little extra work to be able to support iOS and Android devices in addition to our primary goal of supporting the Web in general. Besides, we were already planning on making our UIs accessible for all screen resolutions, so it shouldn't take much extra work to make them look nice on mobile devices too.

6. **Which database should we use?**
   Option 1: PostgreSQL
   Option 2: MySQL
   Option 3: SQLite
   Choice: Option 3
   Justification: While PostgreSQL and MySQL can provide superior performance for large data sets, they also add significant extra complexity to development and deployment. Both mandate an outside server to be used to manage just the database, meaning a larger attack surface and another point of failure to deployments. SQLite, on the other hand, has no outside server and uses only a single file for the entire database. While it can suffer performance constraints on large datasets, the data we expect to store and retrieve is not expected to exceed this threshold. For some datasets it can even have a

significant performance increase due to there being no round-trip delay like there is with a client-server design.

Functional Issues

1. **How should schools and courses be added to the database?**
   Option 1: Manually add each and every school and course to the database ourselves
   Option 2: Allow students to add them to the database themselves
   Option 3: Have students request for a school or course to be added if it doesn't yet exist
   Choice: Option 3
   Justification: Many existing platforms that followed the same idea as Scholasticate were limited by their scope. For example, many schools weren't supported on the StudyBuddy app. This effectively prevented students attending those schools from making the most of the app. That's why we didn't want to manually add anything ourselves: there would always be schools we would miss. However, if we just allowed students to add whatever they wanted to the database without some kind of approval process, many duplicate or inappropriate items could be added to the database. That's why we've opted for Option 3. If a school or course is not yet present in the database, users can request that they be added. These requests will then be approved or denied by some kind of system we implement. It could be an admin account manually approving the requests, a system flagging any requests containing references to illicit or irrelevant topics, or something else. In any case, we will construct a system that only accepts appropriate requests, allowing our app to be flexible but also safeguarded from database bloating and inappropriate content.

2. **How should our location services be updated?**
   Option 1: Refreshed at regular intervals
   Option 2: Manually by the user
   Option 3: Both
   Choice: Option 3
   Justification: We've opted for a dual approach to updating location services. We don't want to unnecessarily drain the user's battery or slow down the app by constantly querying their location data as well as those of other users. That said, location plays a vital role in our app. That's why we've decided to refresh location services at a set rate (perhaps every couple minutes) so that locations are still mostly up-to-date. However, if a user for whatever reason wants to refresh the locations more frequently, we believe they should be allowed to do so. That's why we're planning on allowing them to customize the refresh rate as well as refresh it manually.

3. **How should active users be displayed on the UI?**
   Option 1: On a map for easy view of proximity to others
   Option 2: In a list for more convenient viewing
   Option 3: Both
   Choice: Option 3

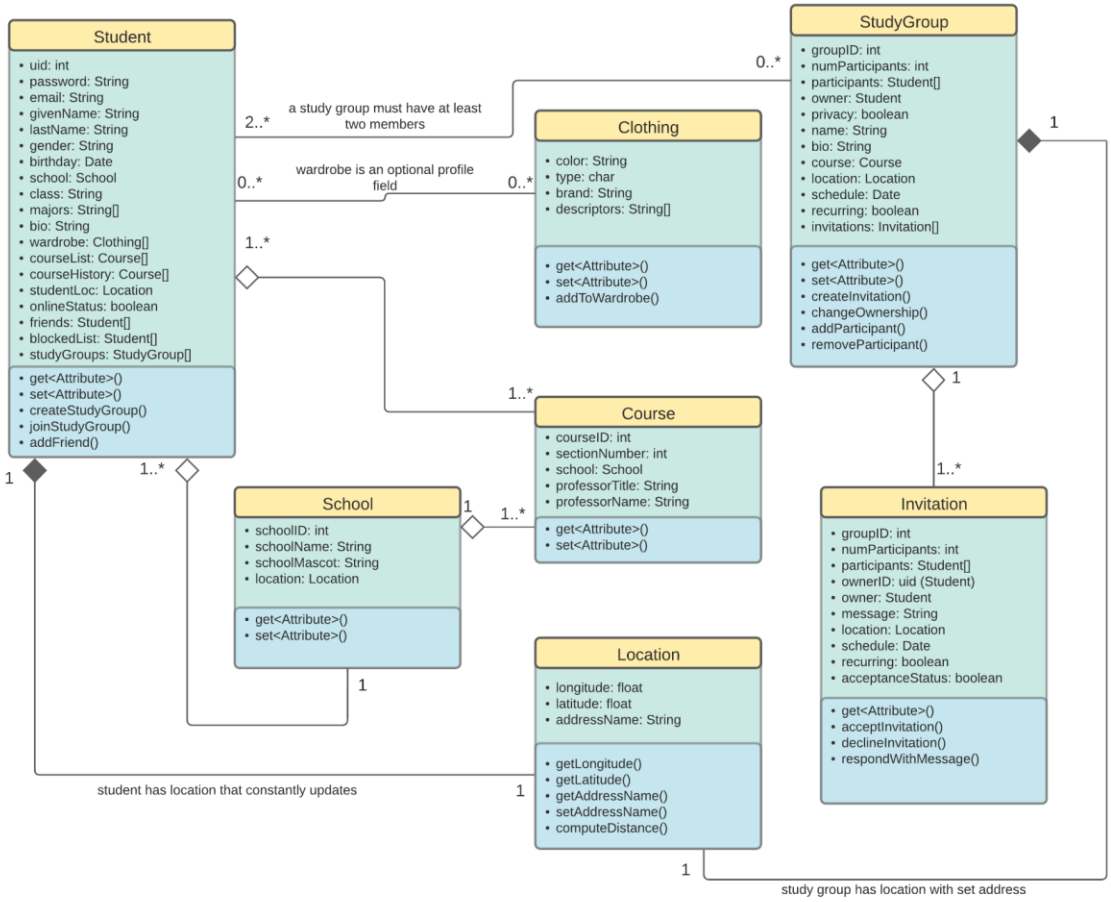
Justification: There are two main ways one could use this app: you could look around on a map for people near you to study with, or you could search through a list. A map works well when people are very close to you, but it can be easy to get lost if you have to pan or zoom around a lot to find other users. On the other hand, a list is easy to look through, but it can get really big if there are a lot of people online. Different people might have different preferences, so we've simply opted to support both approaches. We'll provide a map on which people can see other active users, which can be fun to look at. But we'll also provide a list of nearby users sorted by distance for those who prefer lists.

4. **Should course searches be complicated by the specification of course section numbers?**
   Option 1: Yes
   Option 2: No
   Option 3: Let the user decide
   Choice: Option 3
   Justification: We've decided to support this option but not require it. We expect that most users would simply use the subject name or the course number to search for specific study partners, and that would be enough. However, because many courses have multiple sections that are taught by different professors, oftentimes students within the same course will have different homework assignments. Therefore, students in one section may see no point in studying with students of a different section because assignments and exams may not match up. Allowing them to additionally specify the course section while searching for study partners will save them this trouble.

5. **How should study groups be represented within the system / on the UI?**
   Option 1: Formally, as their own dedicated structure
   Option 2: Informally, as groups of individuals shown together on a map
   Choice: Option 1
   Justification: We've decided to make active study groups their own formal structure because it would allow for much more interaction between members of these study groups. Currently, we plan on allowing study groups to have a name, an owner, optional scheduled meeting times, and even a group chat feature. These useful functionalities would not be easily possible if we did not recognize study groups as their own entity. If instead we just showed the users close together on a map (like Snapchat does), there wouldn't be any easy way to save the state (messages, title, meeting times, etc.) of that group of individuals. That's why we believe study groups should be their own entity within our design.

Design Details

Class Design



Descriptions of Classes and Interaction Between Classes

The class design is centered around the instances of students and their ability to create and participate in study groups. Each class centers around defining the attributes of student profiles and the connections between student objects. Here is a list of detailed explanations regarding each class found in the diagram, including their attributes and their functionalities.

Student

- Each student will have a unique ID number (uid) that will serve as an identifier for searching for Student objects.
- Each student will have login credentials, a valid email address and a password of appropriate strength.
- Each student will have a user profile that they can customize to their liking and display to other users. The attributes of the profile that can be edited include the following:
    - Preferred First and Last Names
    - Gender Pronouns
    - School Information: Enrolled Courses, School Name, Class, Major(s)
    - Short Biography
    - Current Wardrobe Information (to allow for identification)
- Each student will have a list of the current courses they are taking as well as a list of all previous courses completed on their profile as they choose to display them.
- Each student will have a current location that updates in real-time as their longitude and latitude change. Any student can also specify the address name of their location, with automated checkpoint locations available as well.
- Each student will have the ability to display themselves as online to enable location services.
- Each student will have a list of friends with marked differences from other users as well as a list of blocked users who will not appear on their profile. Students will have the ability to add and remove students from either of the lists.
- Each student will have the ability to create or join Study Groups.

School

- Each student will have a school they currently attend
- Each school will have a name, location, mascot, and an ID. All attributes can be accessed by the user.
- A student can request a new school to be added to the database.

Course

- Each student will have a list of classes they currently are enrolled in.
- Each course will have a name, course ID, section number (if applicable), and professor attributes including title and name. All attributes can be accessed by the user.
- A student can request a new course to be added to the database.

Clothing

- Clothing objects make up the current wardrobe information of a Student.
- A list of character codes will be used to represent general types of Clothing (i.e. shirt, pants, shoes, etc.).

- Each object will have a type (specified by the encoding), a color and a brief description provided by the Student, if desired.

Location

- Every Student and StudyGroup will have a location. Student locations will update in real-time, and StudyGroup locations will be static and set by the owner of the StudyGroup.
- Each location is represented by longitude and latitude coordinates. Any location can be further specified by an address name provided by the user (specifically applicable in static contexts)
- Locations will be updated automatically by the server at a specified time interval, or updated on the request of the user if they choose to turn off automatic refreshing. Locations will not be updated automatically if they are specified to be static.
- Each student will be able to view the computed distance between themselves and all other objects that also have a location attribute.

StudyGroup

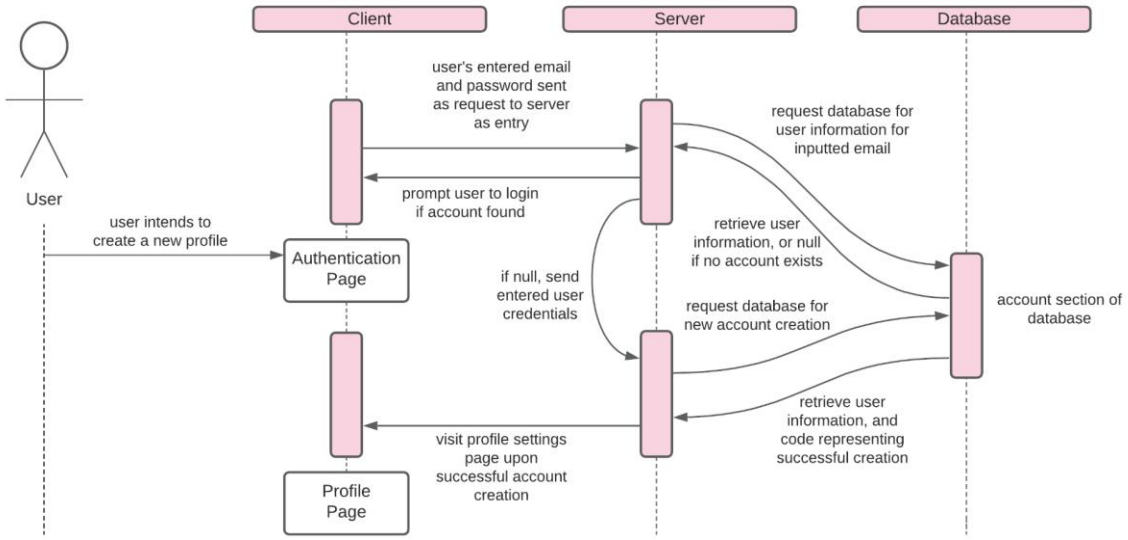
- Each study group will have a dynamic list of participants supporting additions and removals of other students. Each study group must have at least 2 participants
- Each study group will have a unique ID (groupID) to easily identify study groups and search for them.
- Each study group will have a name, course specification, and short biography to display in a profile similar to that of a Student object
- Each study group will have an owner with a unique ID (ownerID) with special permissions to change different attributes of the group or change ownership of the group to another active user.
- Each study group will have a privacy setting to determine if the group is looking for additional members or is closed to other students.
- Each study group can be recurring and have set scheduling, including a static location and a set list of students.
- The owner of a study group can create invitations to send to other users.

Invitation

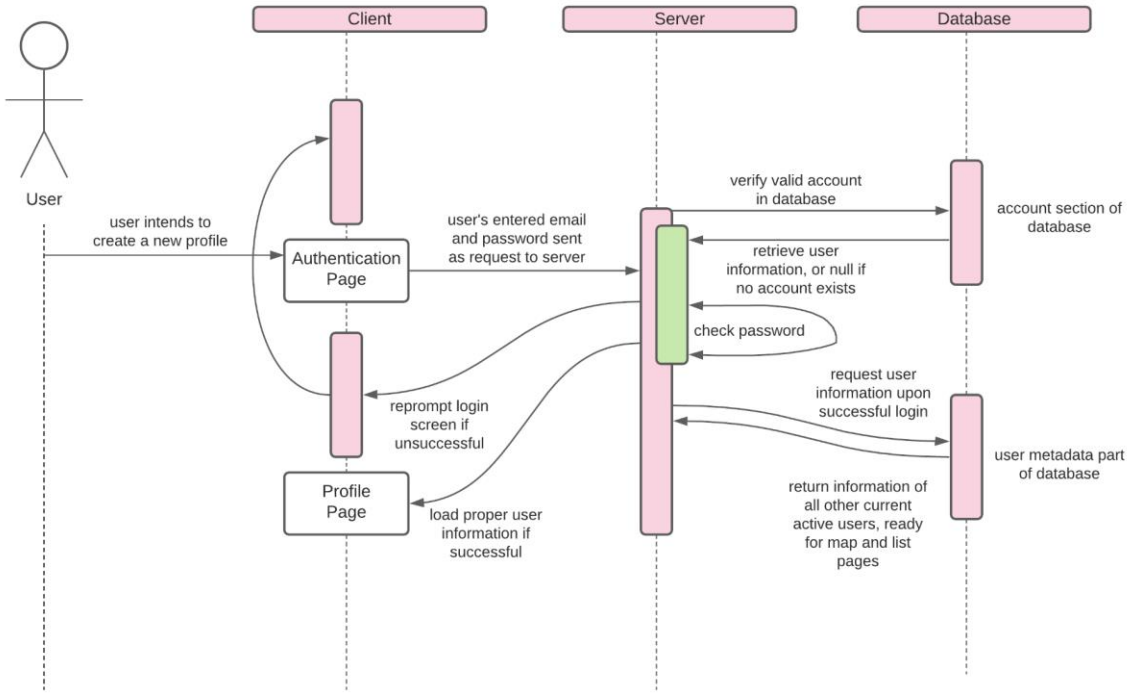
- Each invitation will store the ID of the study group embedded within the request to join the respective group, as well as the data of who already belongs to the group.
- Each invitation will specify the owner of the study group and contain a short message from the owner.
- Invitations will show the location and all details of the group, including the following attributes:
    - Name, Course Information and Description
    - Scheduling and Recurrence Specification
    - Privacy of the Group
- Each student can choose to accept or decline the invitation.
- Each student can respond to the owner of the invitation with a message directly from the invitation interface.

Sequence Diagrams

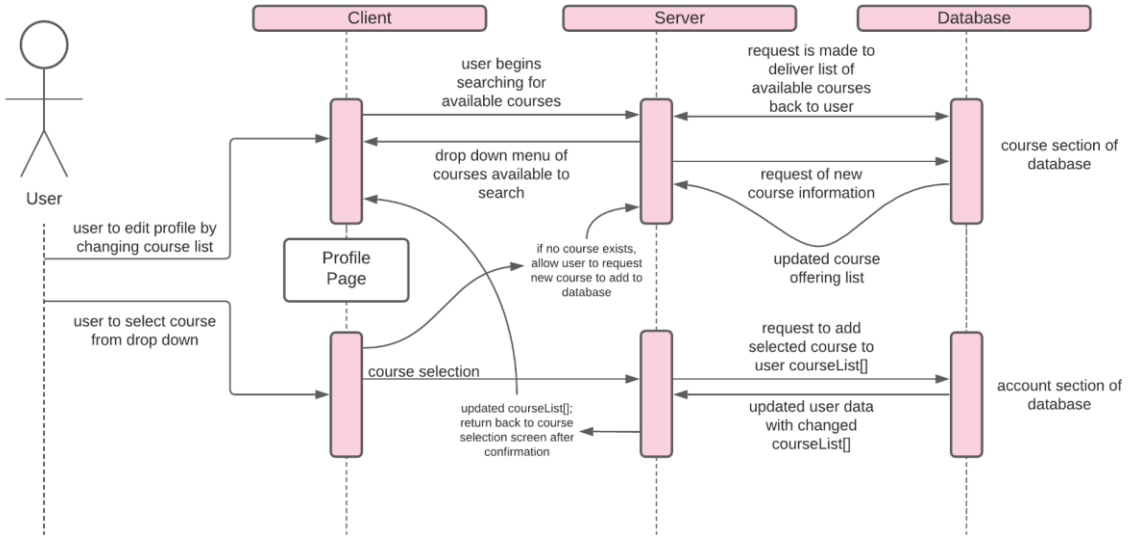
User Profile Creation



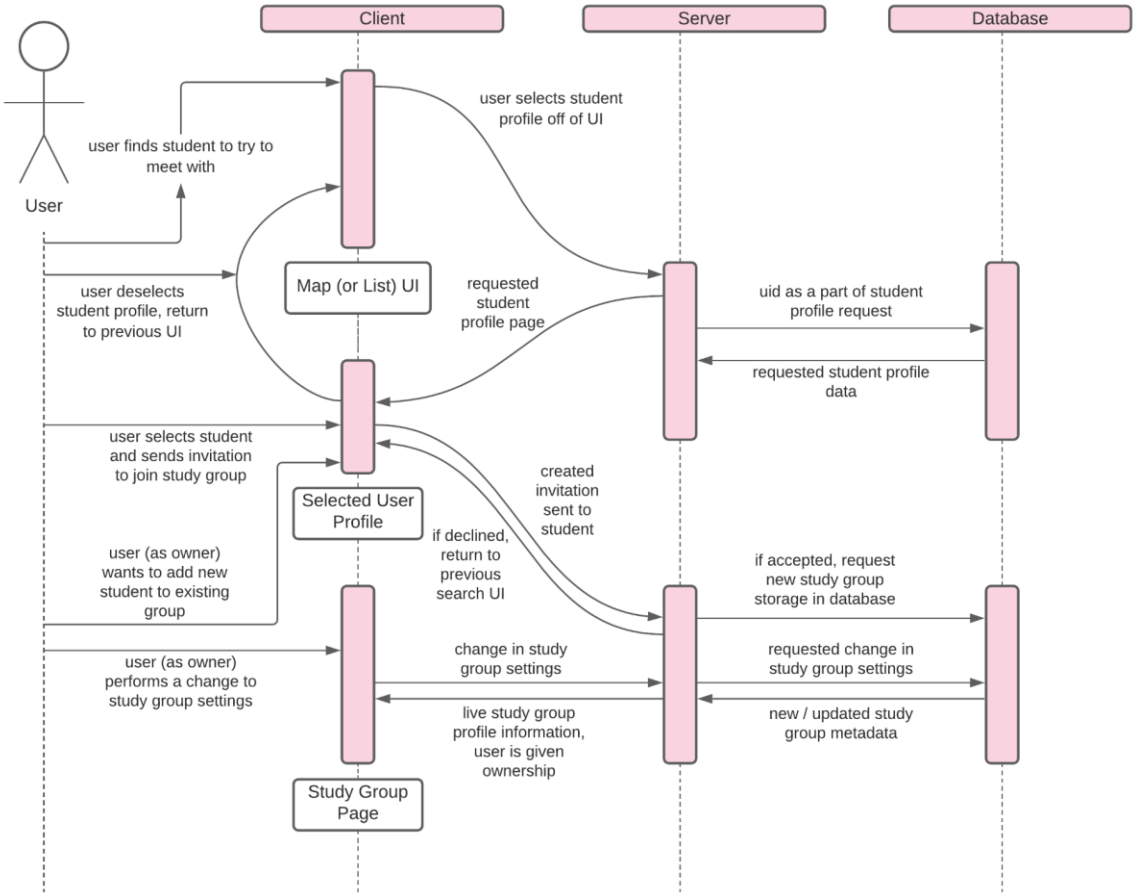
User Authentication



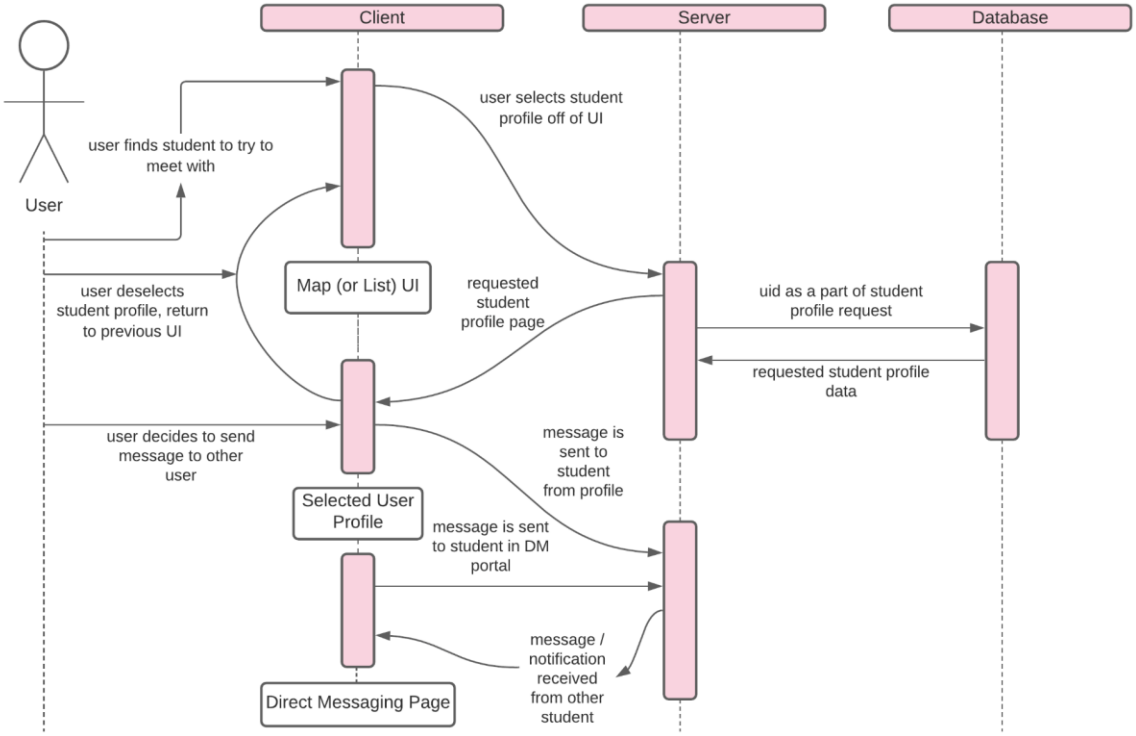
Updating User Profile



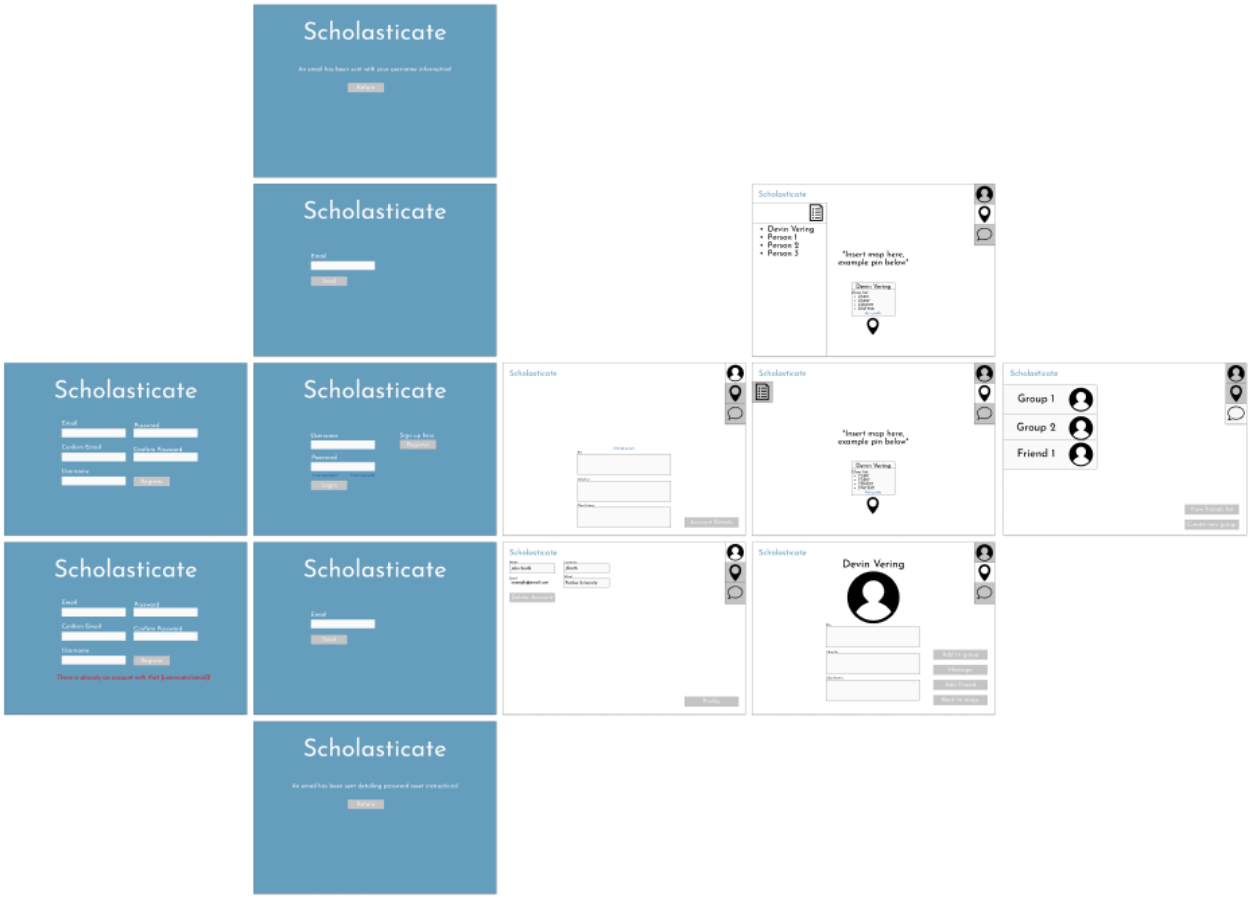
Finding Another Student To Match With / Formation of Study Groups



Messaging Another Student



UI Mockups


Scholasticate
Scholasticate
Scholasticate
Scholasticate
Scholasticate
Scholasticate
Scholasticate
Scholasticate
Scholasticate
Scholasticate
Devin Vering
Person 1
Person 2
Person 3
"Insert map here, example pin below"
Group 1
Group 2
Friend 1