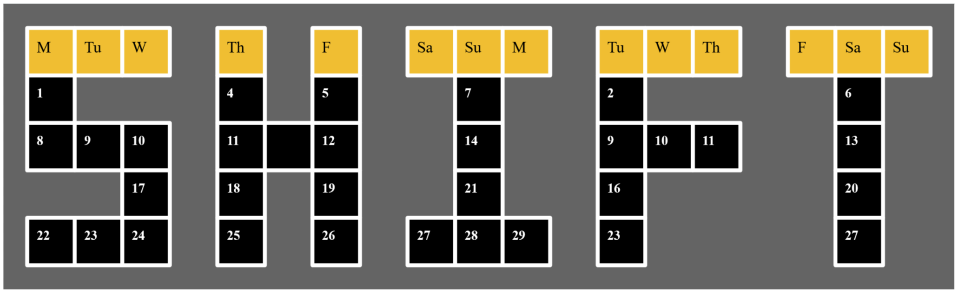


Design Document

FEBRUARY 9TH 2024

TEAM 18

Danielle Ejiogu

Daniel Fakunle

Murtuza Kagalwala

Lucas Munteanu

Jenna Rigdon

Samson Tesfagiorgis

Table of Contents

Key Terms

Here are some key terms that have been used throughout this document, in order of appearance. Please refer to this if you need to check what the abbreviation stands for.

UR: University Residences

RA: Resident Assistant

REA: Resident Education Assistant

REC: Resident Education Coordinator

PUID: Purdue University Identification

Purpose

University Residences' (UR) current system for assigning duty shifts to Resident Assistants (RA) is archaic, time-consuming, and often leads to asymmetric workloads. For example, at Meredith Hall scheduling duty involves generating a randomized list of the names of the 12 RAs and using this list to assign duties to approximately 70 time slots. Fairly and effectively assigning these slots for a given month can take time, especially when accounting for exceptions. Shift offers intuitive features like customizable duty types (think holidays, weekdays, and more) and duty statuses (lead, secondary, tertiary, desk) to ensure every RA gets a fair share of responsibilities. Unlike other platforms such as When2Meet, WhenIWork, or SignUpGenius, Shift is specifically tailored to the unique needs of university residences, offering a comprehensive solution that addresses the complexities of duty shift management. Shift stands out from existing solutions by offering a highly customizable and user-friendly interface, empowering both RAs and administrators to efficiently manage duty schedules.

Join us in revolutionizing duty shift management – let's Shift the way university residences work, one schedule at a time!

Functional Requirements

Account Management

As a user,

1. I would like to be able to create my account
2. I would like to login to my account.
3. I would like to be able to reset my password.
4. I would like to delete my account and all of its associated data.
5. I want a dashboard where I can view relevant information, such as my display name, position, and work schedule.
6. I would like to be able to edit my account details and set up more secure methods of authentication like logging in with social provider accounts (eg: Sign in with Google).
7. I would like to be able to specify my hall, such that I can search for other users in my hall.

As an REA or REC,

8. I would like to be able to create accounts for RAs as needed and supply them with a temporary password to access it.
9. I would like to view at a glance the current status of RAs, e.g., active, inactive, unavailable.

As an REA,

10. I would like to be able to temporarily disable or delete accounts of the RAs.

As an REC,

11. I would like to be able to temporarily disable or delete accounts of the RAs and REAs.

Customization

As a user,

12. I would like to be able to select from a variety of themes, allowing me to personalize my visual experience.
13. I want to be able to set notification preferences, allowing me to choose how and when I receive alerts about shifts, announcements, or other important updates.
14. I would like to be able to reorient the layout of the widgets (e.g., calendar, tabs, taskbar, etc.) on my account.
15. I would like to be able to create a quick view of any key messages, shifts, or events I would like to see after I login.

As an REA or REC,

16. I would like to have the option of creating a welcome page that outlines the rules of scheduling and a brief overview of the website, when I add an RA who is just starting their new job with us.

Scheduling

As a user,

17. I would like to be able to submit my availability to the scheduler.
18. I would like to add, edit, and delete my personal events.
19. I would like to clear my schedule based on hour, day, week, or month.
20. I would like to export my schedule for use on other calendar applications/websites such as Google Calendar.
21. I would like to be able to synchronize my schedule with an external calendar service such as Google Calendar or Outlook Calendar.
22. I would like to be able to download the schedule as a PDF.
23. I would like to be able to schedule, change, and cancel meetings with other users.

24. I would like to receive email notifications to be notified of schedule changes or upcoming events.
25. I would like to see whether or not an RA has clocked in for their shift.
26. I would like to search for important events in my hall or other halls.
27. I would like to customize the level of detail displayed on my calendar view, allowing me to toggle between daily, weekly, and monthly views based on my preference and workflow.
28. I would like to be able to view future days, weeks, and months of the schedule.

As an RA,

29. I would like to be able to indicate a preference for when to schedule certain shifts.
30. I would like to be able to select whether or not I can view and select conflicting shifts (i.e., I can access shifts which may overlap with my other events because I am unable to attend any other available ones).
31. I would like to be able to request to shift drops assigned to me in the face of any personal emergencies.
32. I would like to be able to view shifts that are available due to late shift drops.
33. I'd like to be able to see which RA's are available on any given day (for duty switches, etc.).

As an REA or REC,

34. I would like to set a minimum/required number of shifts each RA has to select.
35. I would like to approve of/confirm all RA schedules as well as accept or decline shift drops, depending on an RA's given reasoning.
36. I would like to create, edit, and delete events in bulk (blocking different shifts of certain intervals, e.g., 15, 30, 60 minutes, etc., within a certain time range) to make it easier to create events and shifts as they will be back to back.
37. I would like to assign time slots randomly to my RAs based on their priority such that the shifts can be distributed fairly while also accommodating for RAs with more time constraints.
38. I would like to be able to manually assign time slots to RAs.
39. I would like to be able to add special rules to shifts (e.g., certain halls, such as coeds, must have RAs of the opposite gender while performing certain duties).
40. I would like to view the number as well as the types of shifts completed by an RA.
41. I would like to edit RA schedules (for last minute shift drops, etc.).

42. I would like the option to generate a daily, weekly, or monthly report detailing the shifts completed by RAs (e.g., hours worked, duties completed, absences, etc.).

Messaging

As a user,

43. I would like to chat with other users privately.
44. I would like to have access to a public chat of all users in the same residence hall.
45. I would like to be able to send hyperlinks or documents.
46. I would like my messages to be encrypted.
47. I would like to be able to see the most recent chat field at the top of the list.
48. I would like to be able to report inappropriate messages in the chat.
49. I would like the ability to delete individual messages from a conversation and to delete the message group itself.
50. I would like the ability to block other users from messaging in private or group conversation.
51. I would like to pin messages that I would like to save for future reference.
52. I would like to schedule messages to be sent in the future.

As an REA or REC,

53. I would like to set rules for spamming in group chats.
54. I would like to have an emergency response chat that will be visible on any page of the website when the emergency is active.

Non-Functional Requirements

Architecture

The website will be split into 2 layers: front-end and back-end. The front-end will be built using React, to give users a simple and easy to navigate interface. The back-end will be implemented with Java using the Spring Boot framework. We will be using SQL as our database to store information. We will be using REST APIs to connect our back-end with our front-end. For better performance, users should be able to store and access their data for at least a 6 month time period. Users will only be able to access messages from the past 7 days, unless older messages are pinned.

Security

We will be utilizing Auth0 to handle user authentication and authorization. This will provide a more secure and streamlined way for users to create and login to their accounts. We will also be using a framework (Spring Boot) that prevents SQL injections. To preventDoS/ DDoS we will be limiting the number of messages that a user can send in a conversation to 500 messages a day, the number of button clicks will be limited to 1 every 5 second during times of high traffic. We would also like to fix bugs within a 48-hour time period from when the bug was discovered, depending on the criticality of the bug.

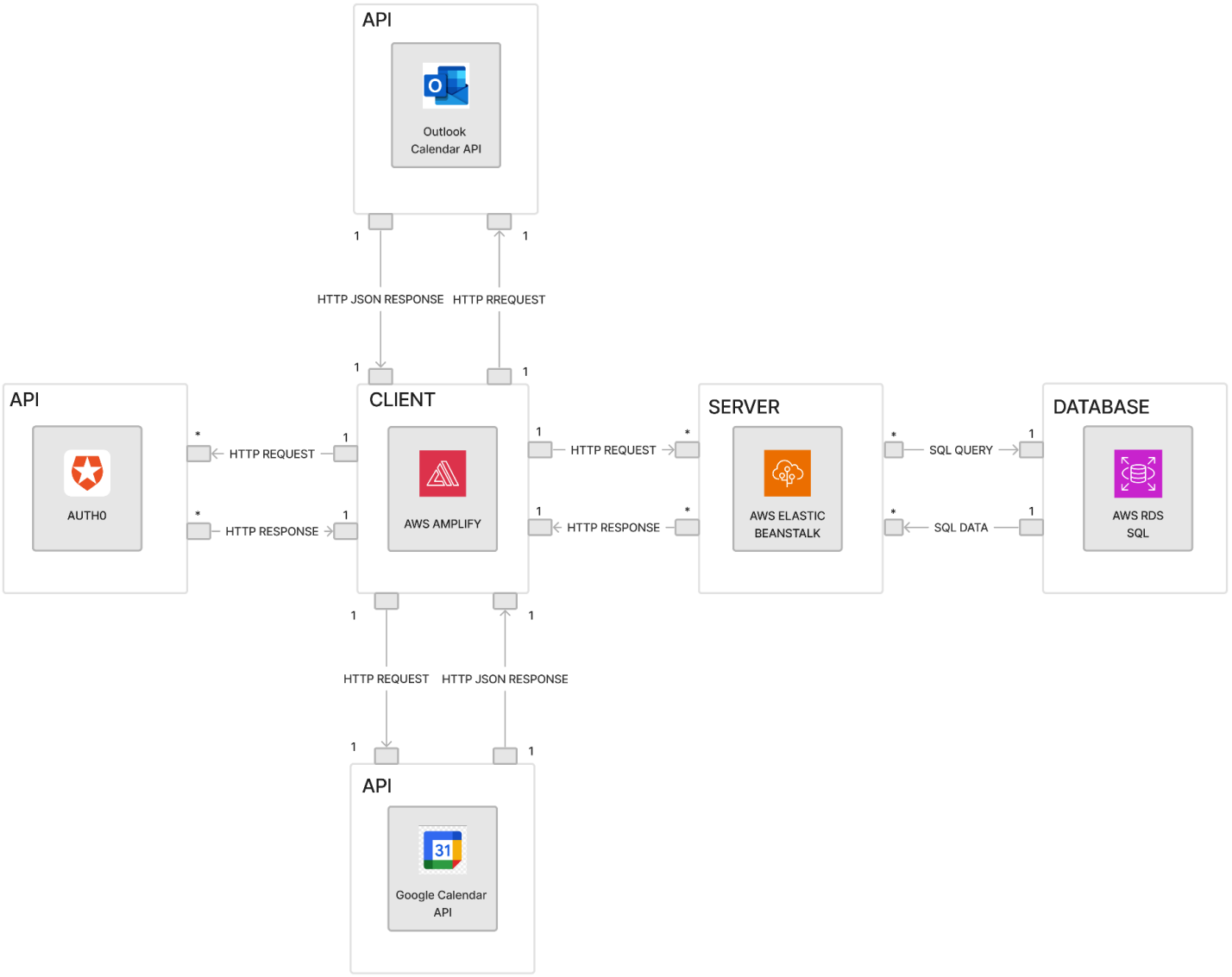
Usability

To ensure usability, our front-end will adhere to essential design principles such as visual hierarchy, consistency, spacing and appropriate use of color. We will also make it more accessible to a wide range of users, with the website being able to operate 24/7 as well as being able to handle at least 1000 users at a time, to allow for high traffic. Furthermore, we will make sure the website is still user-friendly while using it from a mobile device. If time permits, there is a scalable option to expand to more universities. Additionally to increase accessibility, our solution will include the option for high-contrast color schemes and accommodations for color-blind individuals.

Deployment/Hosting

We would like to host the website on AWS which will simplify our deployment and will mainly be hosting our front-end through the AWS service called amplify and we will be hosting the backend on the AWS service called Beanstalk. The database will be hosted using an AWS service called RDS. (We will try asking Purdue's hosting service to make things easier.) If time permits, we will be using Docker and Jenkins to build a CI/CD pipeline.

Design Outline



High level overview

Shift is a full-stack web application that helps optimize the duty scheduling process for Purdue's University Residences. We'll be using a centralized system with resources from the AWS toolkit, and using a Client-Server model. Our development process will divide the programming work into a Front-End and a Back-End. The Client will be developed using Amplify to access the React framework and Auth0 for security features and the Back-End will implement Server functionalities using the Java language through Elastic Beanstalk and SQL through RDS to use database features.

1. **Front-End**

   Amplify- Amplify provides the full libraries of many other development frameworks and the one we'll be specifically using in our implementation is **React**. Here we'll create our **Client** , which is where the user will interact with the program. The **Client** will send HTTP requests to the server.

   Auth0- The client's HTTP requests go through Auth0 before they are sent to the server. The converse is true as well, with the HTTP responses going through Auth0 before they reach our client again. Auth0 will provide us with the security needed, as it uses the OAuth 2.0 protocol, which is a widely used open-source industry standard for authorization.

2. **Back-End**

   Elastic Beanstalk - Elastic Beanstalk supports applications written in Java, which is the language of our choice. In this framework we implement Server-Side functionalities, as a request-handler it handles the bulk of our scheduling needs and will provide information to the Front-End for its UI.

3. **Database**

   RDS- This is a managed relational Database that is provided by AWS, that supports SQL databases, allowing us to use secure database instances. So we will be utilizing SQL Queries for the storing of data such as user details and scheduler details. The security and scalability features provided makes it ideal for accessing through our backend which is deployed on the Beanstalk service provided by AWS. Using RDS as our database hosting service also provides us with the option to securely recover our data.

4. **APIs**

   Google & Outlook Calendar - The integration of the Google and Outlook Calendar APIs with our server involves establishing a communication channel between our server and the API to manage users' schedules efficiently. Our server will initiate HTTP requests to the API to perform operations like fetching events from the calendar, updating events, deleting, etc. The calendars, upon receiving the requests, will perform the requested operations by generating HTTP JSON responses that our server will handle accordingly.

Design Issues

Functional Issues

1. **How should the platform implement Messaging?**
   - **Choice a.** Group Chats
   - **Choice b.** Direct Messages
   - **Choice c.** Both

   Decision: **Choice C**

   Having both allows for a direct channel of communication between parties, and also allows for Residence hall-wide communications. Singling it down to just direct messages or only group chats comes with flaws. With only direct messages, sending an important announcement for everyone would prove strenuous as one message would have to be sent to each individual user. However, by only allowing group chats, messaging will be clustered as one - one conversations won't be catered too. Providing both options guarantees a solid user messaging experience. This way, users will be able to engage in proper communications and efficiently communicate with one another.

2. **How long should messages stay visible for?**
   - **Choice a.** One Day
   - **Choice b.** One Week
   - **Choice c.** One Month
   - **Choice d.** One Semester
   - **Choice e.** One Academic Year
   - **Choice f.** Indefinitely

   Decision: **Choice B**

   Allowing messages to have a one week lifespan allows for users to have access to relevant messages while alleviating the program load. If we made the time span shorter, there would be a great risk that important messages sent would be deleted by the time the intended recipient(s) open the application. Any other time frame chosen would have been an unnecessary strain on data storage but with little to no change in user usability. A key additional feature we will include is the ability for users to pin messages. We understand there are exceptions and some messages should be maintained. So, pinned messages will be indefinitely viewable to the user to compliment that their relevance may extend beyond the week for the users.

3. **How should account setup be organized for users?**
   - **Choice a.** All users are free to make an account and choose role
   - **Choice b.** Restrict all account creations to REC
   - **Choice c.** Users free to sign up but are restricted until approved by REA/REC

   Decision: **Choice C**

   To uphold security and prevent the creation of accounts with unauthorized privileges, we opted to allow user creation by anyone, although with limited functionality. This way ensures accounts created cannot initially assume REC/REA privileges and proper role distribution is maintained. Choice c also alleviates the workload on an REC as they only have to approve RA accounts. Subsequently, other RA and REA approval requests handling can be distributed among the valid REA/REC accounts.

4. **How do we implement the chat functionality?**
   - **Choice a.** Interface with Private Chats and GroupChat classes implementing
   - **Choice b.** Chat and ChatHistory Classes
   - **Choice c.** A chat and message class

   Decision: **Choice C**

   We opted in making two separate classes to make up messaging. A Chat class and a Message class. The chat class would contain attributes: members and groupChat (boolean), which itself eliminates the need for distinct classes for group messaging and private messaging. Choice c proves to be the best option for retrieval of chat data as its linked list structure makes it easy to retrieve and display the chats in sequential order. This implementation with the direct connection from message to chat to RA simplifies the storage and handling of all unique chats with the least amount of classes and proper data storage.

5. **What class should the RA be able to interact with in order to relay their availability?**
   - **Choice a.** Shift
   - **Choice b.** Schedule
   - **Choice c.** Scheduler

   Decision: **Choice B**

   We decided the best point of access for an RA to detail their availability is the Schedule class. This allows each RA to have proper access to edit and manipulate their own schedule without the ability to manipulate everyone else's. Because we allow for the REAs access to their list of RAs, they can access their schedules and add that said schedule to the scheduler classes' list of RA schedules to be processed later.

Non-Functional Issues

1. **What web server do we use for hosting?**
   - **Choice a.** AWS
   - **Choice b.** Heroku and Netlify
   - **Choice c.** Third - Party base hosting

   Decision: **Choice A**

   With a goal of deployment in several residence halls on campus, AWS will provide better scalability and reliability than other options. As the user base grows, AWS services such as Auto Scaling will monitor our system and allow for automatic handling of varying traffic influx. This will keep our application at a low cost efficiency while also preventing the need for manual intervention. Additionally, Amazon Web Services also provides a reliable and secure database, Amazon RDS.

2. **What frontend language/framework will we use?**
   - **Choice a.** Javascript
   - **Choice b.** React
   - **Choice c.** Angular

   Decision: **Choice B**

   While most of our group lacks experience in JavaScript coding, we aim to deliver a professional-looking user interface efficiently. React emerges as the ideal choice given its reputation for being beginner-friendly, supported by extensive documentation and online tutorials. As young professionals who are preparing for industry roles, learning React is crucial, considering its widespread use and technological relevance. By opting for React now, we ensure we're equipped with skills that align with industry standards.

3. **How do we deal with user account creation and login?**
   - **Choice a.** Create and save information in plain text
   - **Choice b.** Create information and use hashing techniques to map users
   - **Choice c.** Use open-source authorization service

   Decision: **Choice C**

   Security is an important aspect that we at Shift take into consideration while making our decisions. We knew that store information in plain text was a big no for us. We considered using hashing techniques to encrypt user data, however this was not our best option since we could use the industry standard for authorization which is the OAuth 2.0 protocol. OAuth 2.0 adheres to our security objectives best with its secure authentication flow. Compliance with high level security standards is essential to mitigate risks and data breaches. This is also open-source so we will have constant updates to our program in terms of user security ensuring protection against evolving threats over time. It is crucial to maintain high security standards for successful implementation of Shift at Purdue.

4. **Which AWS database service do we use?**
   - **Choice a.** RDS
   - **Choice b.** Aurora
   - **Choice c.** Dynamo DB

   Decision: **Choice A**

   RDS aligns well with our plans in storing user data such as login information, personal info, and individual schedules per its structured design. Most of us are familiar with SQL and RDS works as a more familiar environment compared to the other too. Aurora and Dynamo DB do work with high volumes of data well, but at the cost of more provision. It is important to us that we put more focus on the development of the project and other aspects rather than the maintenance of the data. RDS provides that with that familiarity and secure data storage.

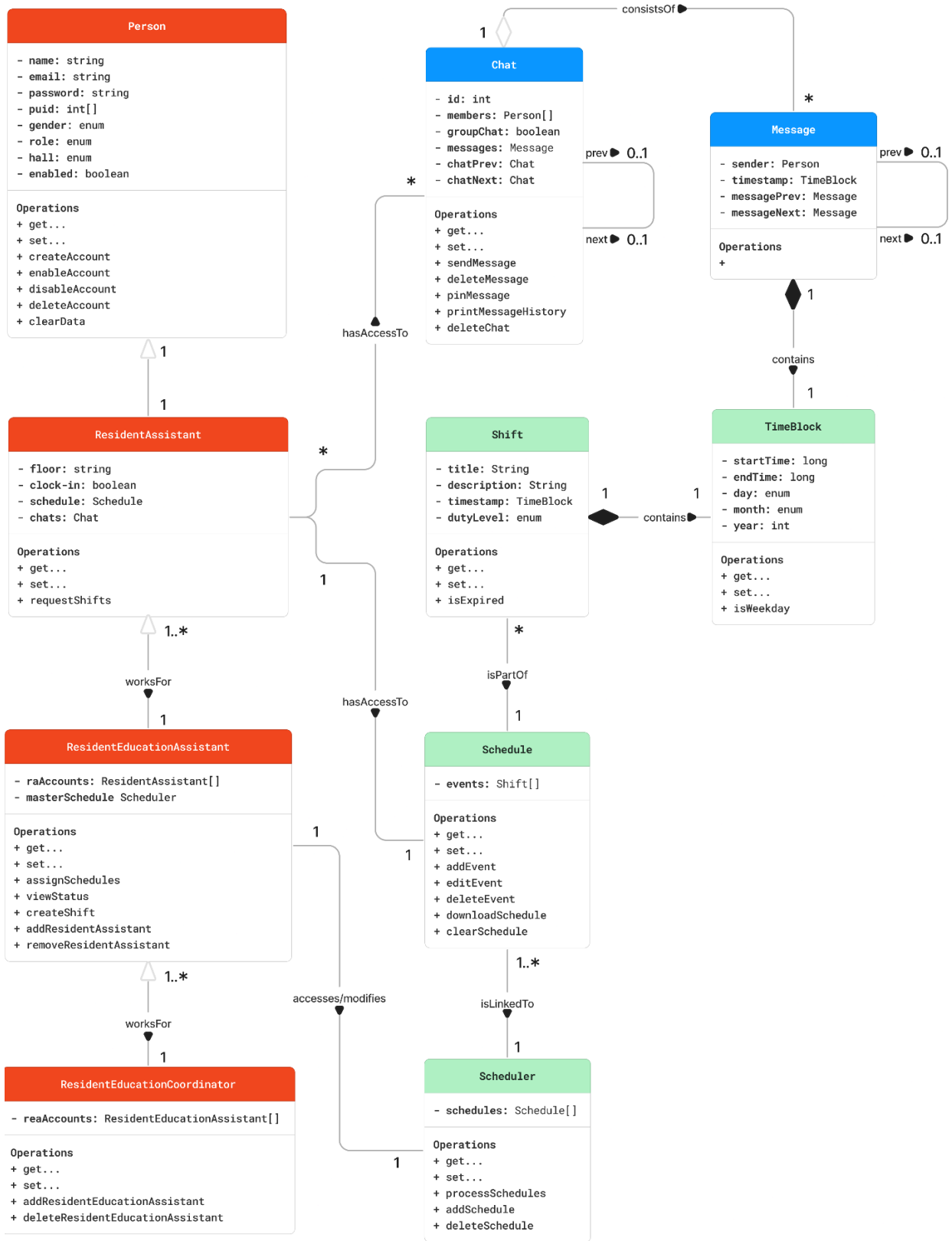
5. **How do we organize the class hierarchy of a User?**
   - **Choice a.** User class extending Person class containing only login info
   - **Choice b.** One user class with login and personal info
   - **Choice c.** Not creating user class at all

   Decision: **Choice B**

   In the initial stages of our design process, we considered splitting a user into two classes: a Person with only necessary login info and then a User with additional personal details. Upon further discussion, we noticed inefficiencies due to a one to one association between the two classes. The original design resulted in instances of redundant data retrieval from Person to User whereas a unified class solution would eliminate that problem and improve the design.

Design Details

Class Diagram



Description of Class Diagram

Our Design consists of 10 classes that help portray just how we intend to implement and connect these functionalities together. We use inheritance extensively in the diagram, as we found it was the best way to reflect the relevant hierarchical issues within the University Residences system. Additionally many classes operate as the smallest unit of different functionalities, reflecting the moving parts of the chat and scheduling mechanisms.

1. **Person**

The generalized **Person object** exists at the point of account creation as a temporary store of information as the user, being an RA or REA in this case, waits to be approved by a higher ranked user (only a RA or REA). The Person object holds an email and a password to be used for login, whilst the PUID, role, and hall are all used for authentication purposes. It serves as a blueprint for the more specific user types the system will employ.

a. The **RA object** is created once a person object with the role "RA" has been approved. The floor attribute will then specify that RA's jurisdiction. Additionally, the RA will be able to specify their availability by requesting shifts which will modify their own respective schedule attribute.
b. The **REA object** is created once a person object with the role "REA" has been approved. The REA object has a list of the RA's they preside over. Behaviors of the objects of this class and whatever class extends it, in our case only the REA class, include general managerial ones such as viewing the status of their workers, adding or removing RA's from their team, and creating Shifts that must be filled.
c. The **REC object** requires no authentication at account creation, and exists as an extension of the REA class with its own unique attribute— a list of the REA's they preside over.

The inheritance structure of these classes is deliberate, as we want our design to account for the hierarchy within University Residences and the distinct capabilities of each person in the system. Both the REA and REC's have an attribute called the master scheduler, which contains the schedules of the RA' s within their jurisdiction. Allowing them to access the availability of their teams and produce a single schedule from that set.

2. **TimeBlock**

The TimeBlock object is how our program represents things like the duration, start, and end of events within the scheduler. The operations of the class such as isWeekday serve to increase the efficiency of later algorithms by the system. IsWeekday is important as weekday and weekend shifts are distinct kinds of duty within the UR system the assigning of which will be weighted differently in the suggestion of a schedule.

3. **Shift**

The Shift object is the smallest unit of the overall Scheduling functionality. As seen in the class diagram, the Shift objects have built-in behaviors to check if a Shift has expired or distinguish it as weekend or weekday Shift. Shift objects have a **TimeBlock** object as an

attribute to represent the shifts temporal qualities. The various types of Person objects will modify shifts in order to indicate availability or the need for someone to take a shift.

4. **Schedule**

The Schedule object is used by all the users. The RA object has one as an attribute that they are able to modify, and it represents their availability and duty preferences. 1 schedule may be associated with many shift objects.

5. **Scheduler**

Those Schedules of theirs are then linked to the attribute in the Scheduler class. The managers have access to this master scheduler which consists of all the schedules of the RA's under their jurisdiction. They can perform operations on this array of **Schedules** to produce a single schedule object that will perform as the final duty schedule for RAs. 1 **Scheduler** may have associations with many schedule objects.

6. **Message**

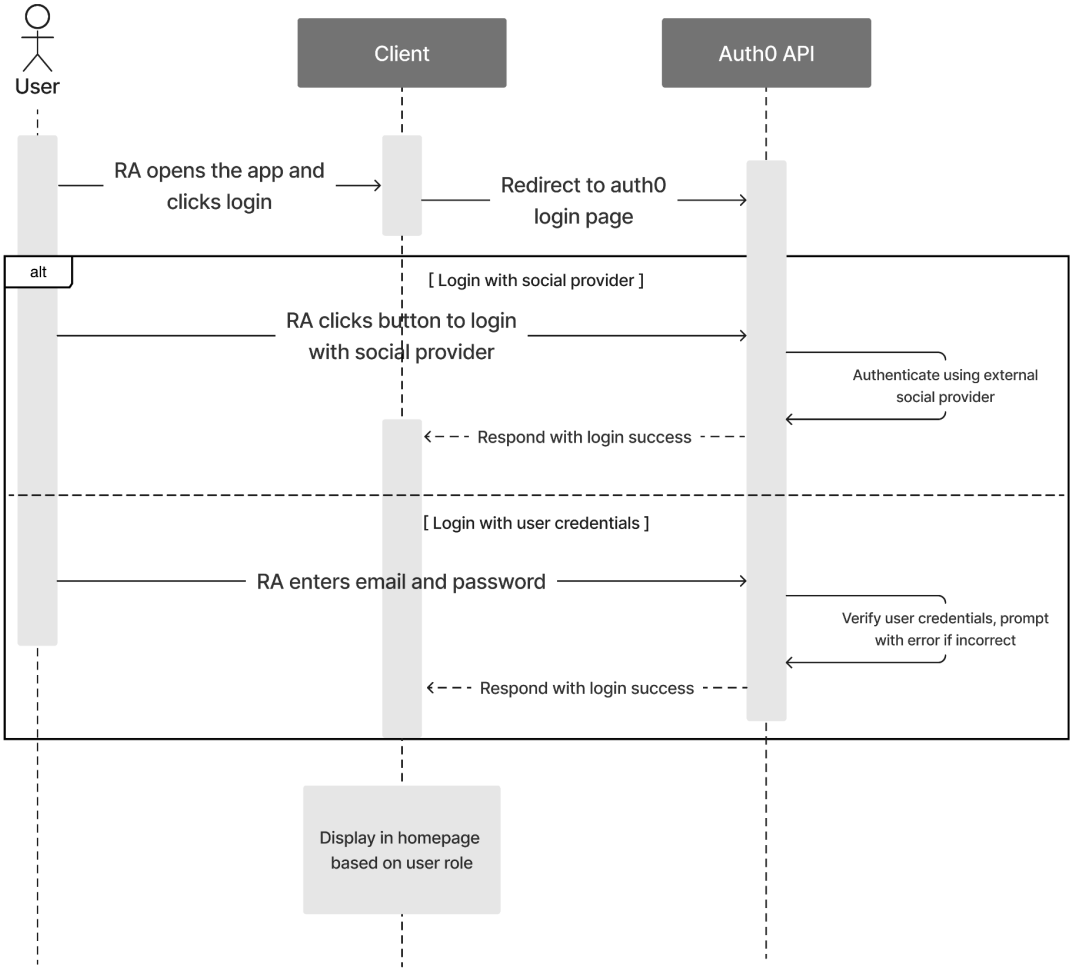
The Message object is the smallest unit of the chat functionality. The Message object has attributes sender, timestamp, and pointers to the next and previous messages. It is associated with the Chat class.

7. **Chat**

The chat object is central to Shift's user communication functionalities, it holds a list of all persons in the chat and whether or not the chat is a group one or not. The chat objects are also in a linkage structure, this is to aid with displaying the chats in the preferred order of the most recent message sent to the oldest message sent.

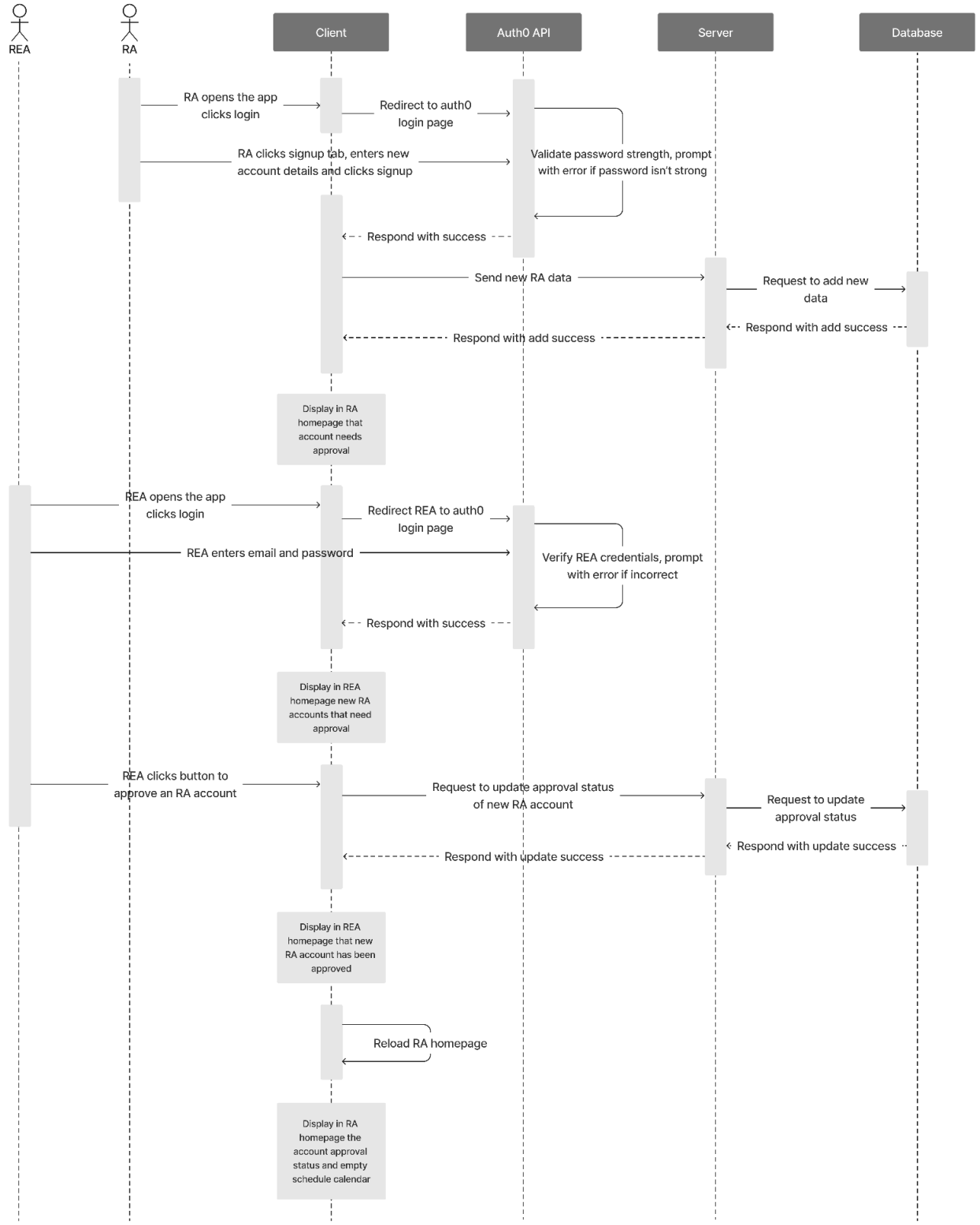
Sequence Diagrams

Sequence of events when a user logs in.



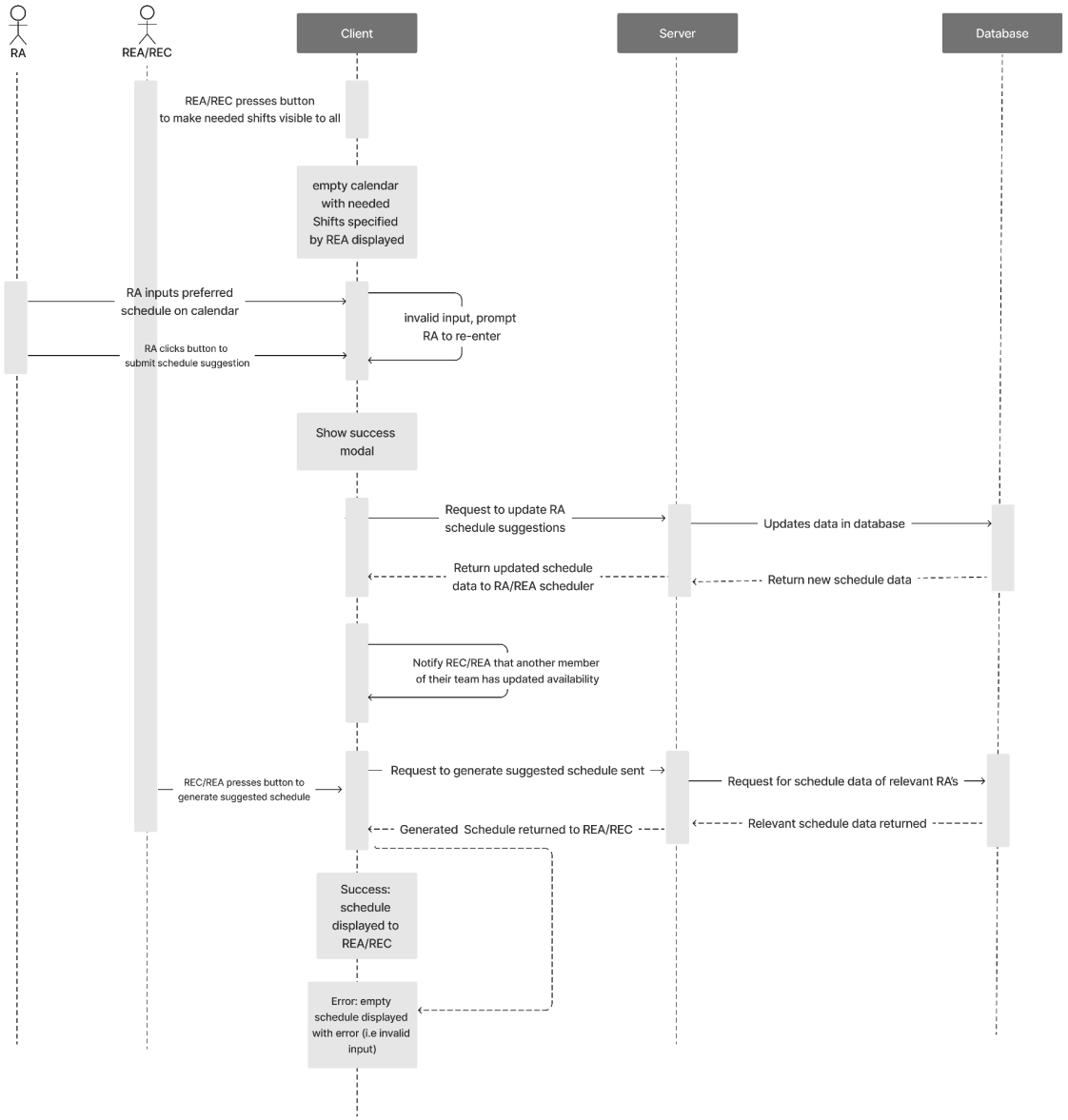
This diagram shows what happens when a user logs in on the website. The user's login information will be redirected to Auth0 service, where the user will be able to login with either their credentials or through a social provider. Once Auth0 verifies the user, the user is taken to the homepage of the application where they can start being productive with Shift.

Sequence of events when an RA creates a new account.



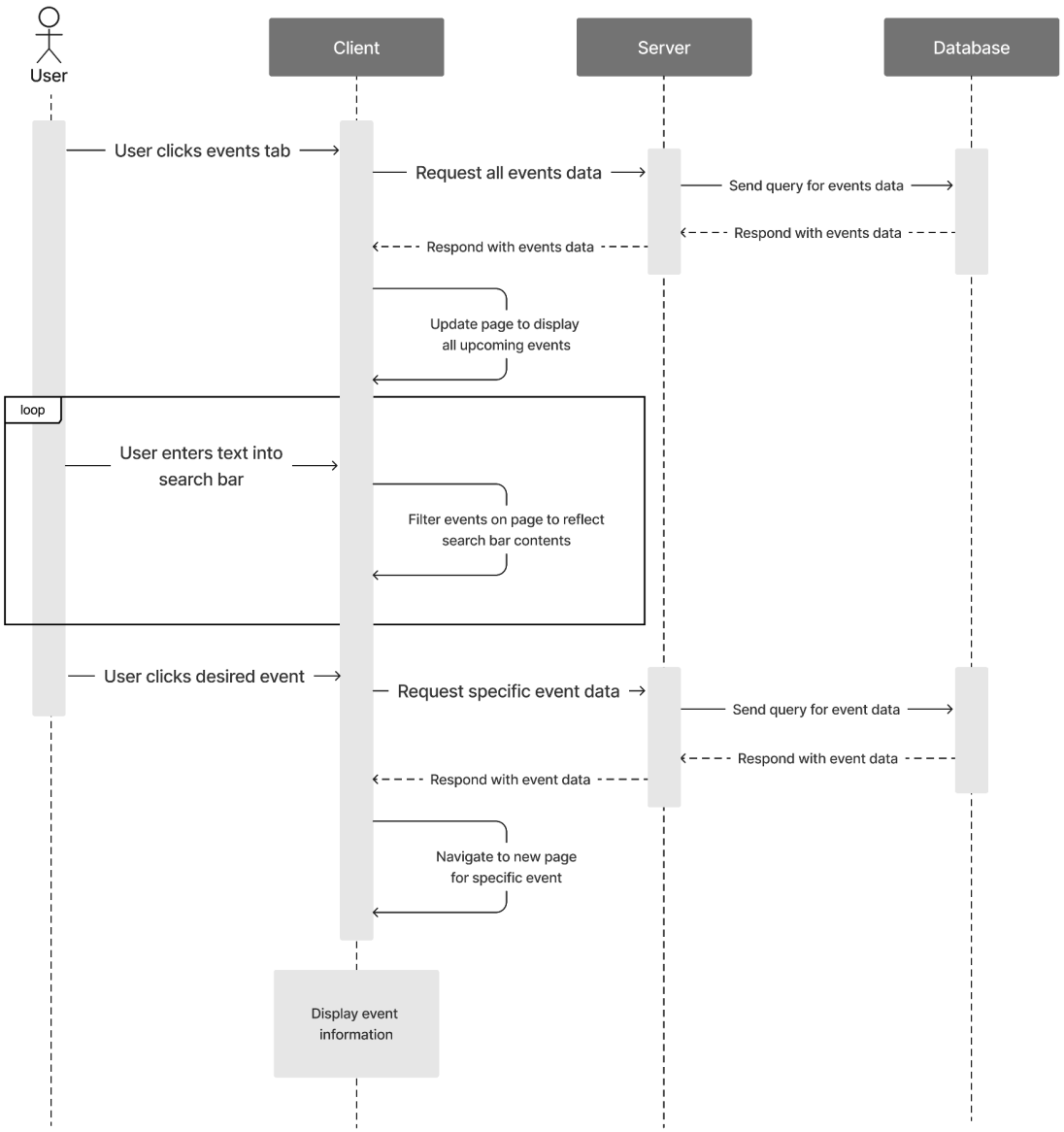
The sequence diagram above details the flow of events when an RA first starts the application to create an account. Two different actors play a part in this process. Take note of the account creation pathway which requires not only a test of password strength, but also that the new accounts have an initial probationary period in which before their accounts are approved by an REA/REC.

Sequence of events when an RA suggests their preferred schedule.



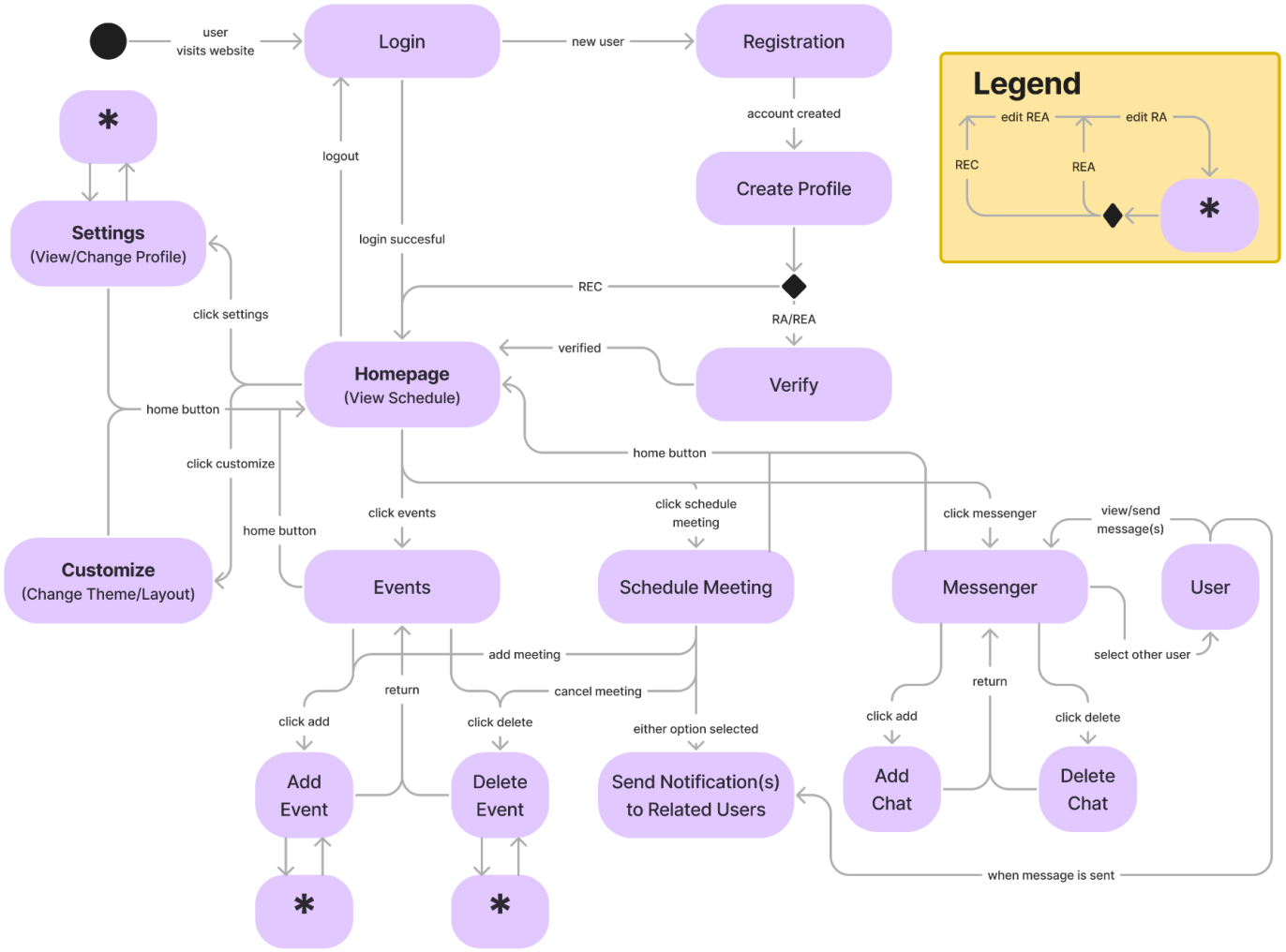
This sequence diagram portrays the scheduling mechanism and the flow of it. Two actors play a part in this. Take note of the fact that the RA's actions in the process are sandwiched between REA/REC action.

Sequence of events when user searches for important events.



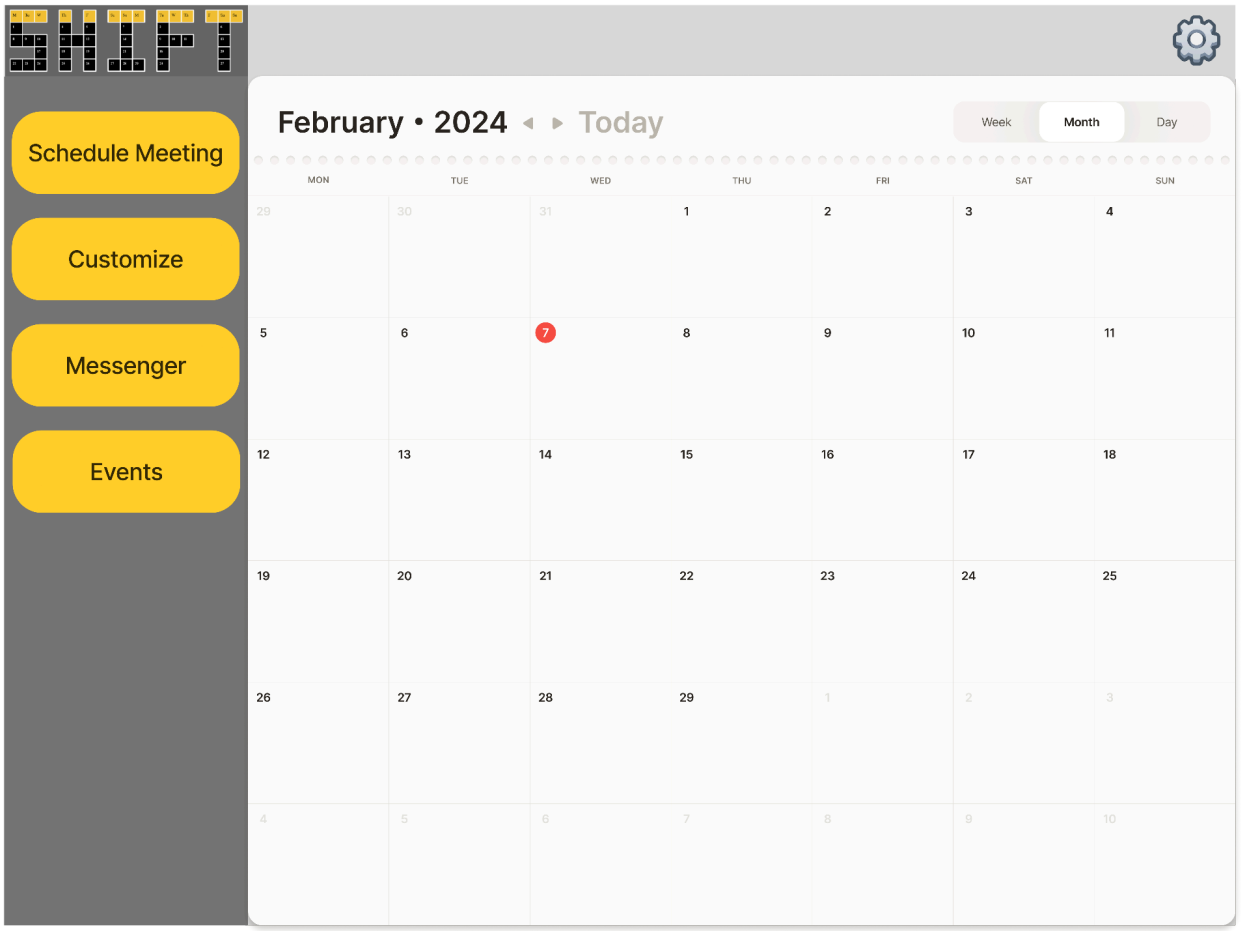
The sequence diagram above displays what happens when a user searches. The search functionality will be specific to events, and features. A loop that will be running to check the status of the search bar and filter the page to reflect what's in there.

Navigation Flow



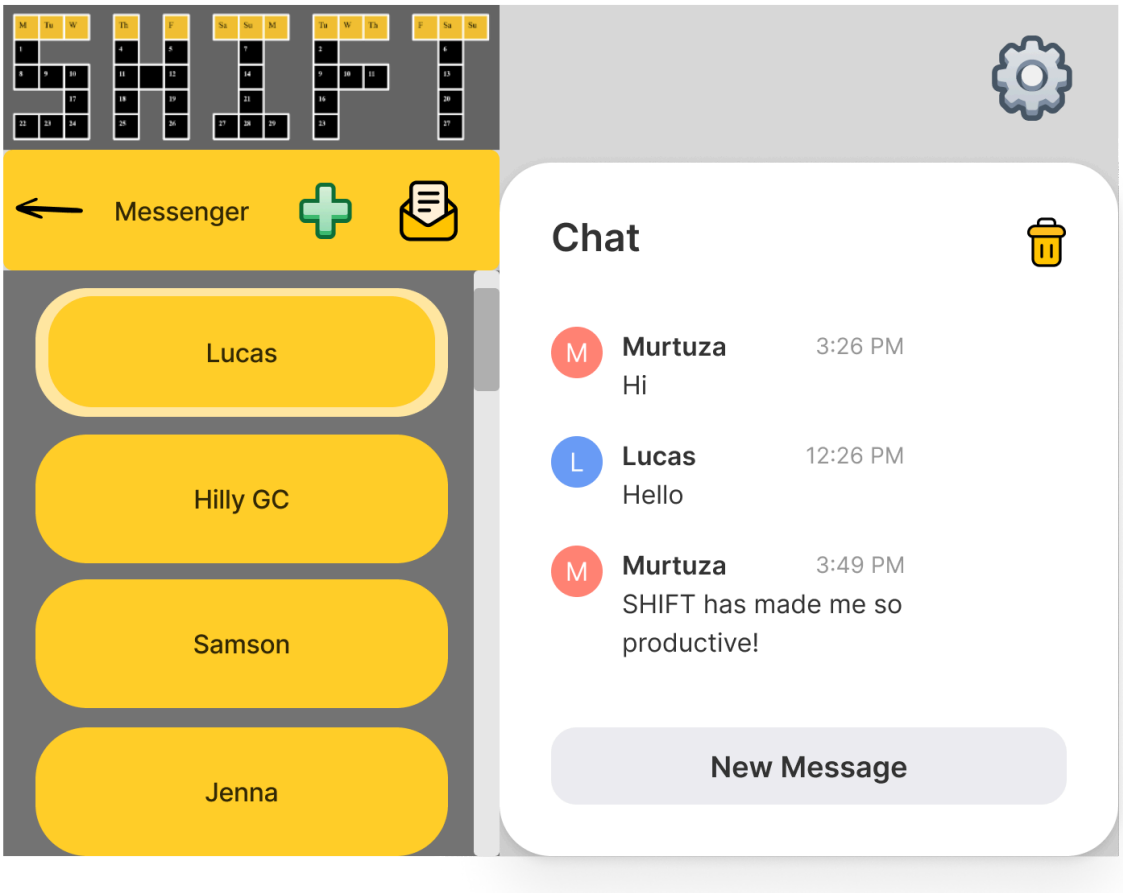
We at Shift want our application to be simple for users, since difficult to use applications and websites will generally hinder productivity. Our application is simple to use, you just login or create your account and then login, and then you are taken to the homepage where you can view your schedule as your first glance. You can access the messenger, schedule a meeting, add/delete events and customize Shift to suit your needs.

UI Mockup



Here is how the home page of Shift will look like. We have a Settings icon on the top-right corner, where the users can access accessibility settings, change their passwords and so on. Next, we have a Calendar where the users will be able to see their schedules and events at a glance. The users can change the calendar view based on the month, week and the day.

On the left the user has more options where they can customize their themes, schedule a meeting, access the messenger and they can add the events.



This is a UI mock-up of how our messenger will look like. You can click the messenger button from the home page and you can select a chat or a group chat and talk with your fellow colleagues. Here you can see the chat with “Lucas” is selected. You can also go back to the home page by either clicking on the Shift icon or by clicking the back button. You can create a new chat by clicking on the + icon. You can also email your colleague by clicking on the envelope icon. You can delete a chat by clicking on the trash icon by going to the chat you want to delete. This is done to avoid accidental deletion of chats.