EVENT BAY

// TODO
Casey Kinnamon
Joe Niu
Gabriel Castro
Brayden Cloud

SOFTWARE DESIGN DOCUMENT
Version 1.0

Table of Contents

Introduction

The Software Design Document is designed to assist in the creation of our Progessive Web App, “Event Bay” or “Ebay” for short, by maintaining a blueprint for how the software should be created. The purpose of Event Bay is to allow people to coordinate informal meetups with friends and family. Our ultimate objective is to empower the connection of people by helping improve communication between one another through the use of Event Bay.

The Software Design Document is important because the software development cycle is constantly changing. It is a good idea to organize our project into a design document that we can refer to as the software development cycle progresses.

After the introduction, we will discuss the architecture of our app. In the architecture overview, we will discuss the components of our system. Next, we will discuss use cases. In this section, we will have a use case diagram of our whole system and important use case specifications. The use case diagrams show the interaction between actors in our system The use case specifications specify the details of a use case. In the next section, we discuss class diagrams and sequence diagrams. The class diagrams give an overview of the system’s classes. The sequence diagrams give an overview of our how system interactions occur. Following this, we discuss the state transition diagram. The state transition diagram includes finite states within our PWA. Lastly, we discuss the user interface of Event Bay, including mockups and design decisions. We conclude by summarizing the main points of our Progressive Web App.

Overview of Architecture

Event Bay is broken up into four main components. (See Figure 1)

- Client (PWA)
- Authentication Service (Auth0)
- API (Express API)
- Database (MongoDB Atlas)

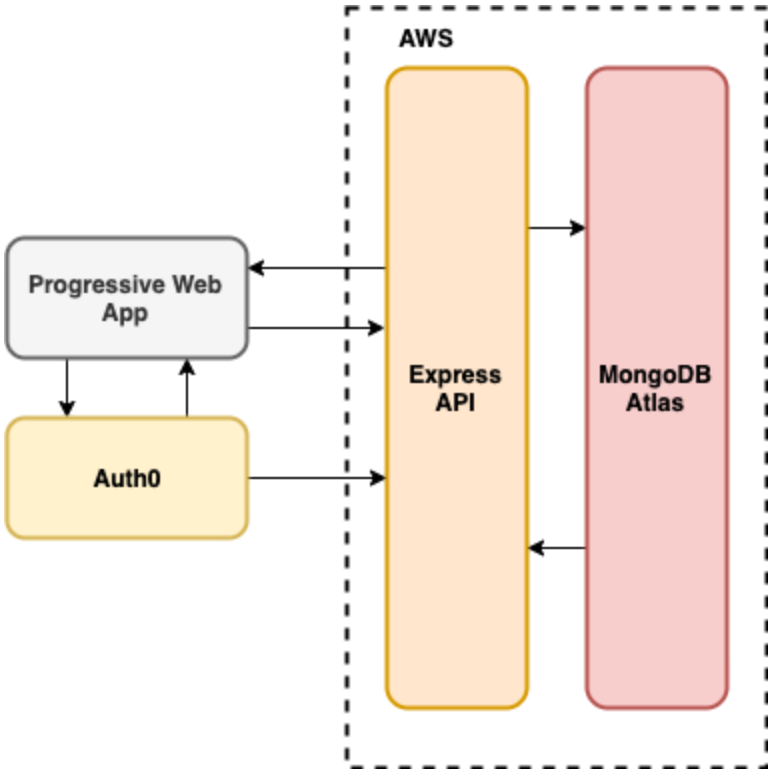


Figure 1. Architecture Diagram

Client

The Client is an Angular application which acts a Progressive Web App. It allows users to interact with the platform via the web browser or downloaded PWA. It interacts directly with two services: Auth0 and Express. Angular interacts with Auth0 via the Auth0 SDK when a user requires authentication. Auth0 handles OAuth services, token management, and security for the platform. Any other service the Client provides is handled by the Express API, including managing Events, and updating user preferences.

Auth0

Auth0 is a third party service which handles user creation, deletion, and authentication. It provides a user interface for creating an account and logging in. The tokens Auth0 creates are used to authenticate the user against the Express API. The line drawn from *Auth0* to *Express API* in Figure 1 demonstrates the hook to the backend when a user is created.

Express API

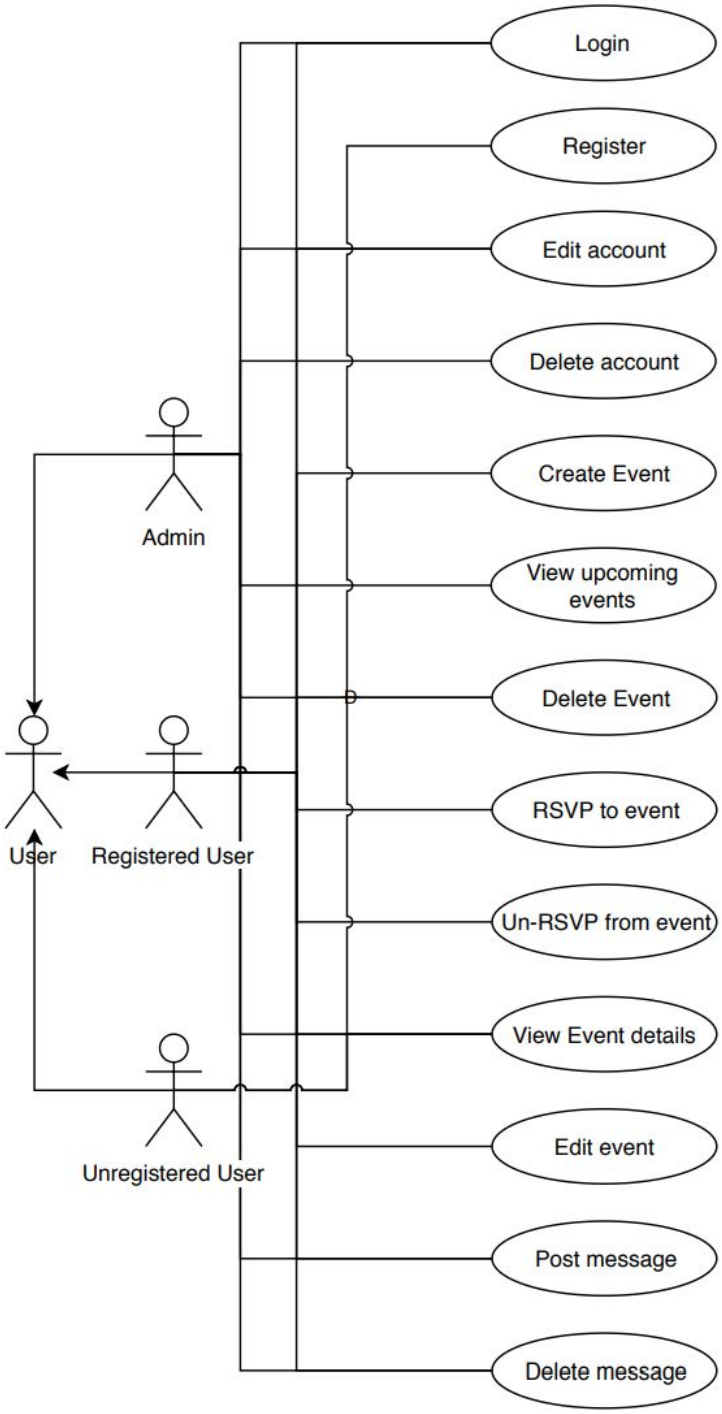
The API for Event Bay is written in TypeScript using the Express.js Node library. It's role is to accept requests from the Client and Auth0. It exposes a specific endpoint for Auth0 to call when a new user is created so that it can create a record for it in the MongoDB database. Additionally, it handles all Client requests to manipulate user or event data which it stores in MongoDB.

MongoDB Atlas

MongoDB is a NoSQL database which is used to store all data for the Event Bay platform. It interacts only with the Express API to ensure all queries are properly authorized and validated.

Use Cases

A. Use Case Diagram:



B. Use Case Specifications

Use Case Specification 1 - Log in

Use Case Number:	EB-0001
Use Case Name:	Log in
Overview:	Event Bay member logs in
Type:	Primary
Actors:	Event Bay member
Pre-condition:	Event Bay member is actually a member of Event Bay
Main Flow:	1. System requires Event Bay member ID and password 2. User submits email and password to system 3. If email and password combination are correct, system allows Event Bay member to log in
Alternate Flow:	If email and password combination are incorrect, inform user and prompt again
Post-condition:	True
Cross-reference:	Validate email and password

Use Case Specification 2 - Create Event

Use Case Number:	EB-0004
Use Case Name:	Create event
Overview:	User creates an event
Type:	Primary
Actors:	Registed User
Pre-condition:	User has registered, logged in, and navigated to the events tab
Main Flow:	1. Registed user presses create event button 2. User is taken to form where he enters the following: a. Name b. Location c. Date and time d. Invites 3. If data entered is invalid, feedback will be given for user to correct data 4. If data is valid, the event is posted to the API and created in the database
Post-condition:	User is taken to the event details screen
Cross-reference:	Validate user input for event

Use Case Specification 3 - View Events

Use Case Number:	EB-0002
Use Case Name:	View Events
Overview:	Event Bay member views upcoming events, can click on event to see details, and can add an event
Type:	Primary
Actors:	Event Bay member
Pre-condition:	Event Bay member has selected to view events pages
Main Flow:	1. Event Bay member is on View Events page 2. Event Baby member can click on specific event listed to look at details of event 3. Event Bay member clicks Create Event by clicking on plus sign at the bottom of View Events page.
Alternate Flow:	1. Event Bay member has no events listed with the only option of Create Event by clicking on plus sign at the bottom of View Events list
Post-condition:	True
Cross-reference:	Create Event

Use Case Specification 4 - View Profile

Use Case Number	EB-0007
Use Case Name	View profile
Overview	User views profile
Type	Primary
Actors	Registered user
Pre-condition	User has registered, logged in
Main Flow	1. Registered user pressed the navbar drawer 2. User presses the account screen button 3. User account screen is displayed, showing profile information
Post-condition	User is shown account page with profile information

Use Case Specification 5 - Create Account

Use Case Name:	Create Account
Overview:	Make an Account
Type:	Primary
Actors:	User
Pre-Condition:	Found website or is going through event invite
Main Flow:	1. System gives the user ways to sign up via google, number, email and password 2. User chooses one 3. System request the information 4. User enters information 5. System validates information and checks to see if user is new 6. If new creates account
Alternate Flow:	7. User was already made resets use case
Alternate Flow:	7.Information was incorrect use case is reset
Cross-Reference:	Auth0

Use Case Specification 6 - Delete Event

Use Case Name:	Delete event
Overview:	Remove event from Database
Type:	Primary
Actors:	User/Admin
Pre-Condition:	User that created event is on View Event page
Main Flow:	1. User clicks delete event button 2. System prompts are you sure 3. User clicks yes 4. System communicates with API to delete event. 5. API deletes data from MONGODB 6. Event is Deleted
Cross-Reference:	API, MONGODB

Use Case Specification 7 - Delete Account

Use Case Number	EB-0003
Use Case Name	Delete account
Overview	User deletes the account
Type	Primary
Actors	Registered user
Pre-condition	User has registered, logged in
Main Flow	1. Registered user pressed the navbar drawer 2. User presses the account screen button 3. User presses delete account button in the account information screen 4. Delete request posted to the API and user is deleted from the database, if success message is returned user is logged out of app
Post-condition	User is logged out of app and taken to home page
Cross-reference	Validate response message for request

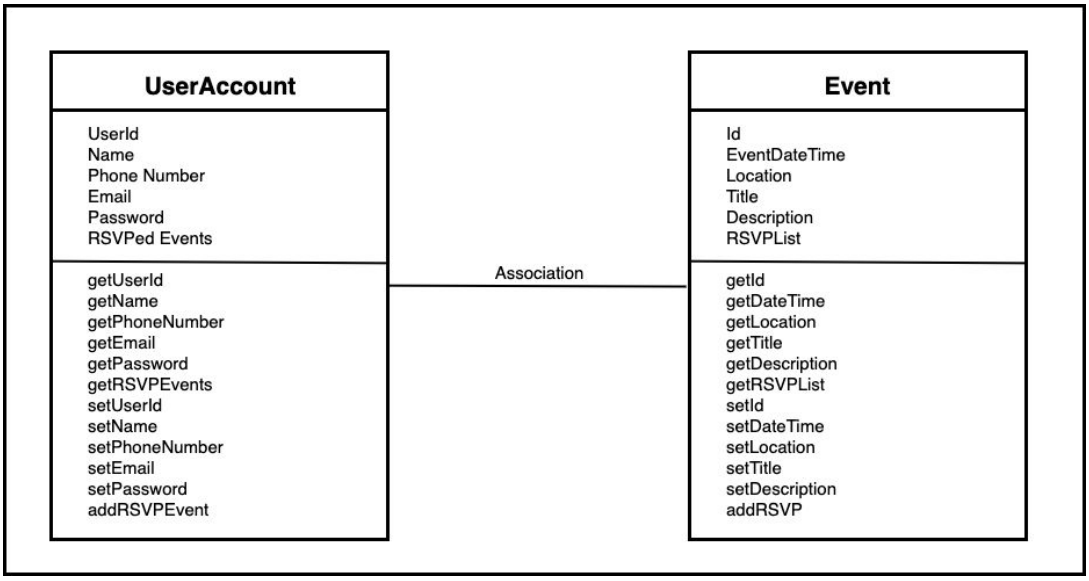
Use Case Specification 8 - Delete User From RSVP List

Use Case Number:	EB-0008
Use Case Name:	Delete from RSVP List
Overview:	Delete a user from an event's RSVP list
Type:	primary
Actors:	Registered User, Administrator
Pre-condition:	User is registered and is the creator of the event OR the user is an admin
Main Flow:	1. Navigate to event details page 2. Select the options drop-down 3. Select "Delete Event" 4. Confirm selection
Post-condition:	User or Admin is returned the page they were on prior to selecting the event, with that event removed

Classes and Class Interactions

A. Class Diagrams

a.



B. Class Details

a. UserAccount class

i. Attributes

Attribute name	**Type**	**Description**
UserId	String	Field used to store the value of a user id.
Name	String	Field used to store the name of the user.
Phone number	Integer	Field used to store the phone number of the user.
Email	String	The email address of the user.
Password	String	The password of the user.
RSVPed Events	String[]	Array of event ids from RSVPed events.

ii. Operations/Methods

Method Name	Arguments passed	Expected return value	Description
getUserId	none	String	Function used to retrieve the unique id of the user.
setUserId	String	none	Function used to set the unique id of the user.
getName	none	String	Function used to retrieve the name of the user.
setName	String	none	Function used to set the name of the user.
getPhoneNumber	none	Integer	Function used to retrieve the phone number of the user.
setPhoneNumber	Integer	none	Function used to set the phone number of the user
getEmail	none	String	Function used to retrieve the email of the user
setEmail	String	none	Function used to set the email of the user
getPassword	none	String	Function used to retrieve the password of the user
setPassword	String	none	Function used to set the password of the user
addRSVPEvent	String	none	Function to add an Event id to the

			RSVP'ed event array.
getRSVPEvents	none	String[]	Function to retrieve array of Event ids the user has RSVP'ed to.

iii.

b. Event class

i. Attributes

Attribute name	**Type**	**Description**
Id	String	Field to store the value of the event id.
EventDateTime	DateTime	DateTime object used to store event date and time.
Location	Double[]	Double array used to determine latitude and longitude of the event location.
Title	String	The title of the event
Description	String	The description of the event.
RSVPList	String[]	Array of user ids of the users who have RSVP'ed for the event.

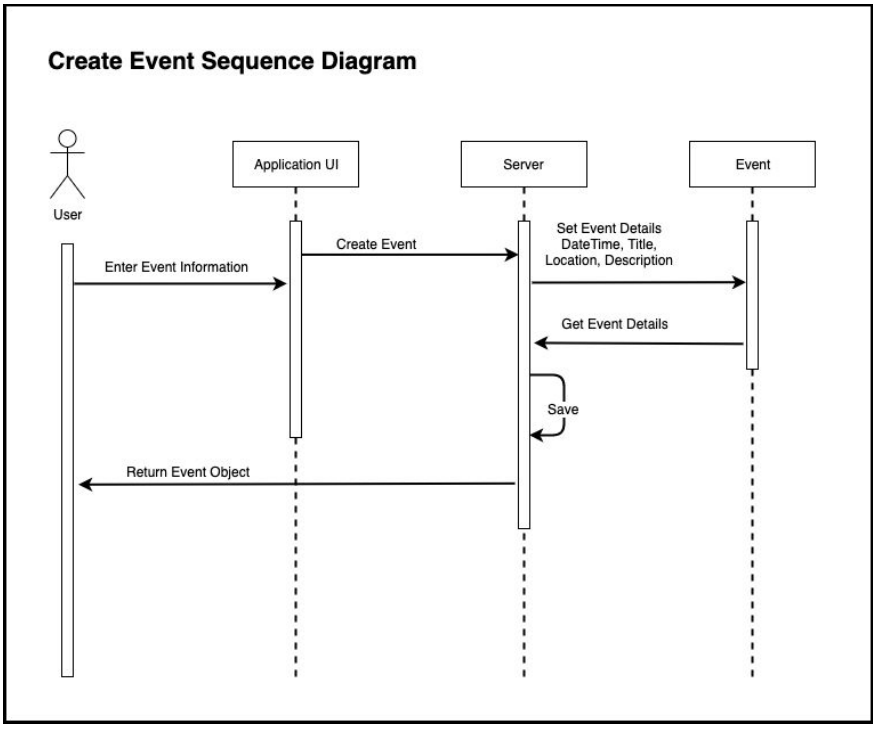
ii. Operations/Methods

Method Name	**Arguments passed**	**Expected return value**	**Description**
getId	none	String	Function used to retrieve the id of the event.
setId	String	none	Function used to set the id of the event.
getDateTime	none	DateTime	Function used to retrieve the date and time of the event.

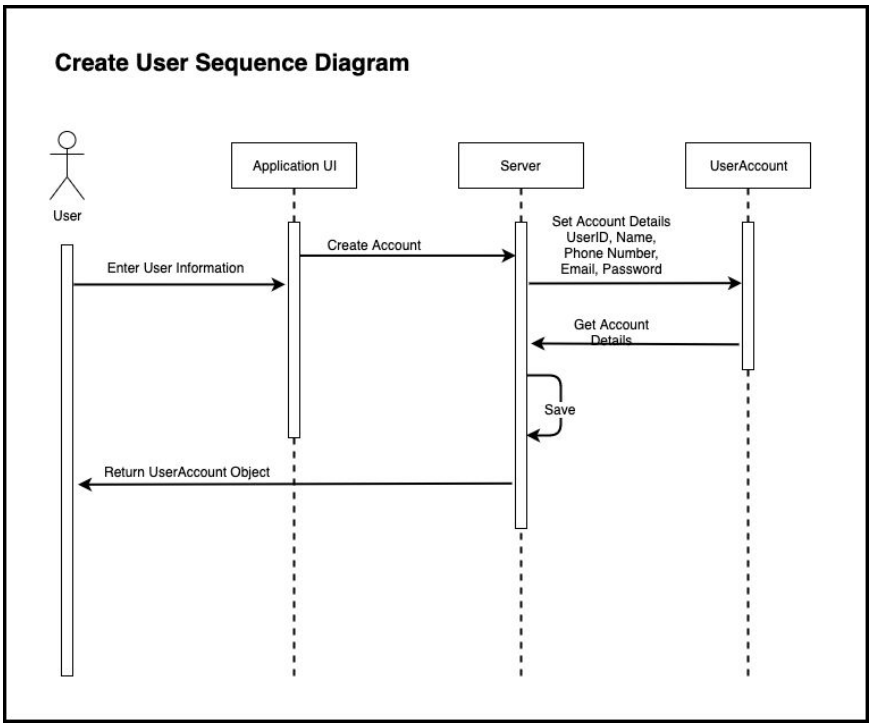
setDateTime	DateTime	none	Function used to set the date and time of the event.
getLocation	none	Double[]	Function used to retrieve the latitude and longitude of the event location.
setLocation	Double[]	none	Function used to set the latitude and longitude of the event location.
getTitle	none	String	Function used to retrieve the title of the event.
setTitle	String	none	Function used to set the title of the event.
getDescription	none	String	Function used to retrieve the description of the event.
setDescription	String	none	Function used to set the description of the event.
addRSVP	String	none	Function to add a User id to the RSVP array.
getRSVPList	none	String[]	Function to retrieve array of User ids of users that have RSVP'ed to the event.

C. Sequence Diagrams

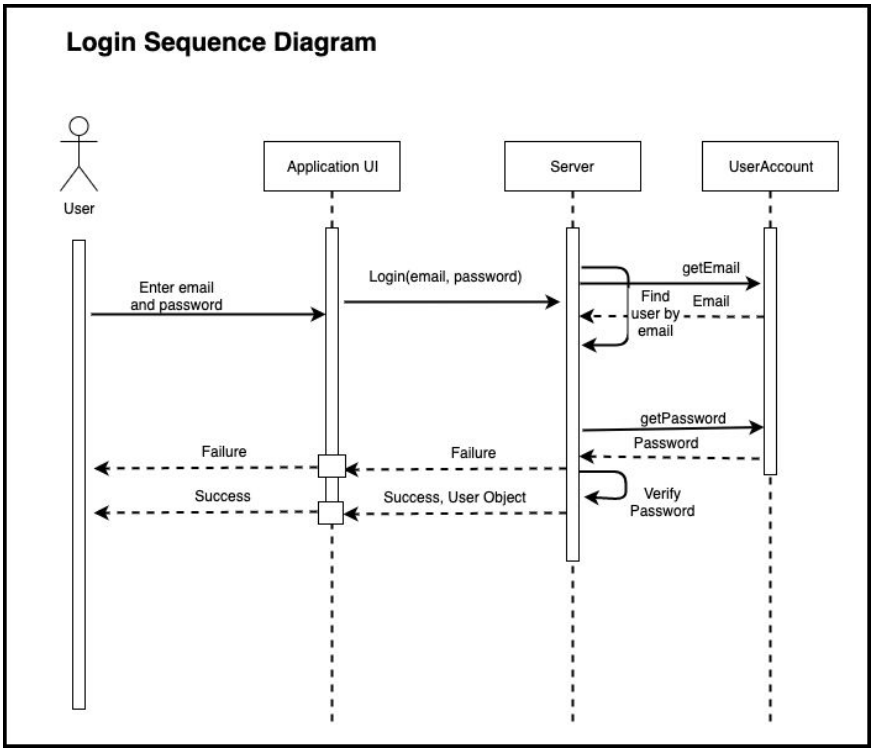
a. When creating an Event, the user interacts with the Application UI, entering the Event information. The application then sends a Create Event request to the server, the server creates an Event object, populating the data and saves it. It returns the just saved Event object to the user.



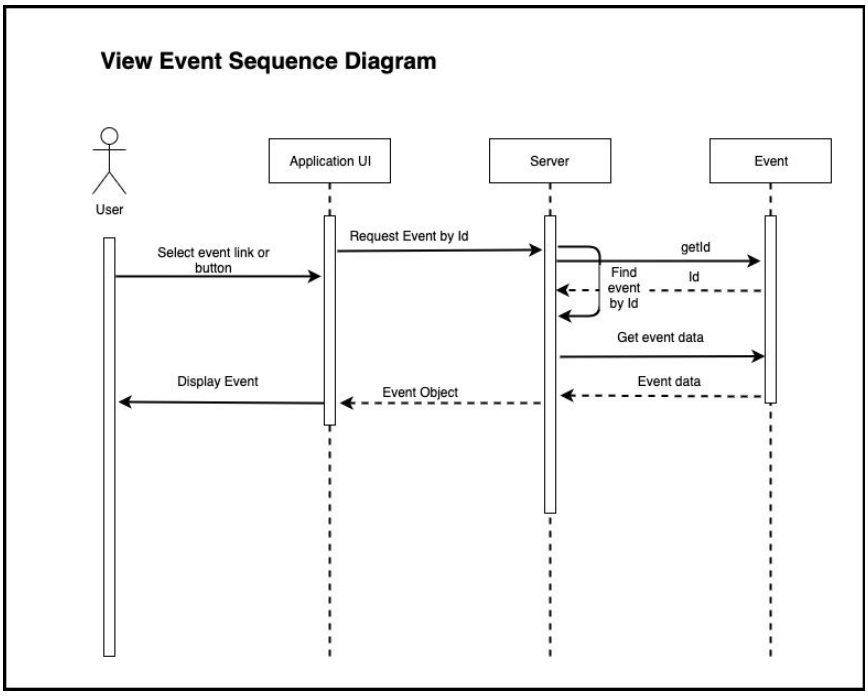
b. When creating a UserAccount, the user interacts with the Application UI, entering the UserAccount information. The application then sends a Create Account request to the server, the server creates a UserAccount object, populating the data and saves it. It returns the just saved UserAccount object to the user.



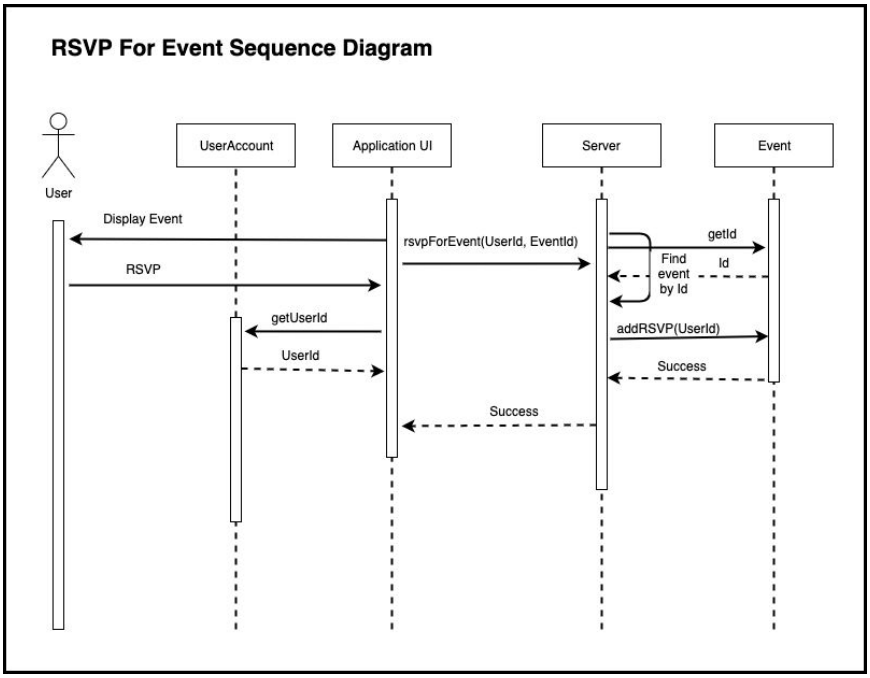
c. When logging in, the user enters an email and password to the Application UI and attempts to log in. The Application sends a Login request to the server with the email and password. The server finds the corresponding UserAccount by email, and verifies if the correct password was entered. If the verification was a Failure, a Failure message is sent to the Application and displayed to the user. If the verification was a Success, a Success message is sent to the application along with the UserAccount object and is displayed to the user.



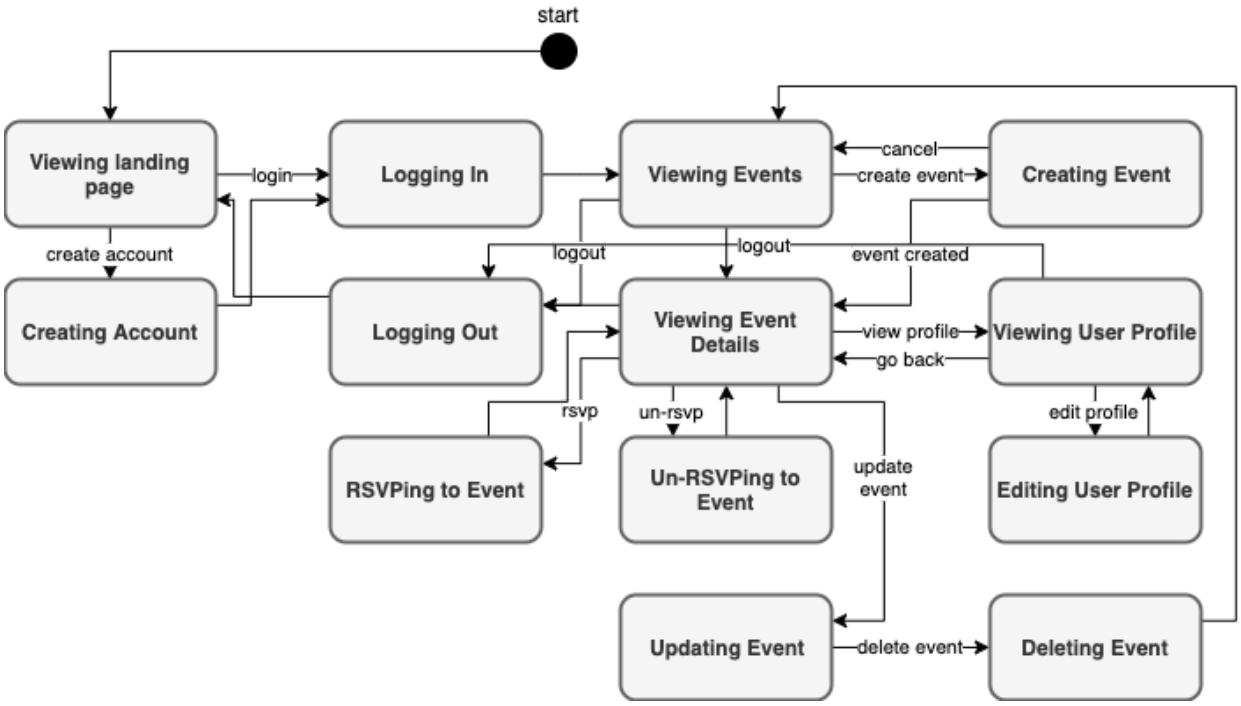
d. When a user selects an Event link or Event button in the application, a request to get the event object by id is made to the Server. The Server finds the corresponding Event by EventId and retrieves the Event data. It then returns the Event data as an Event object to the application, which will then display the event to the user.



e. When the user is viewing a displayed event, the User can select the RSVP button on in the Application. The Application will then get the UserAccount's Id and make an rsvpForEvent request to the server with the UserAccount Id and the EventId (would already have due to sequence necessary to display event). The server finds the corresponding Event by EventId and adds the UserAccount (by UserId) as an RSVP. The success messages are then back-propagated to the Application.



State Transition



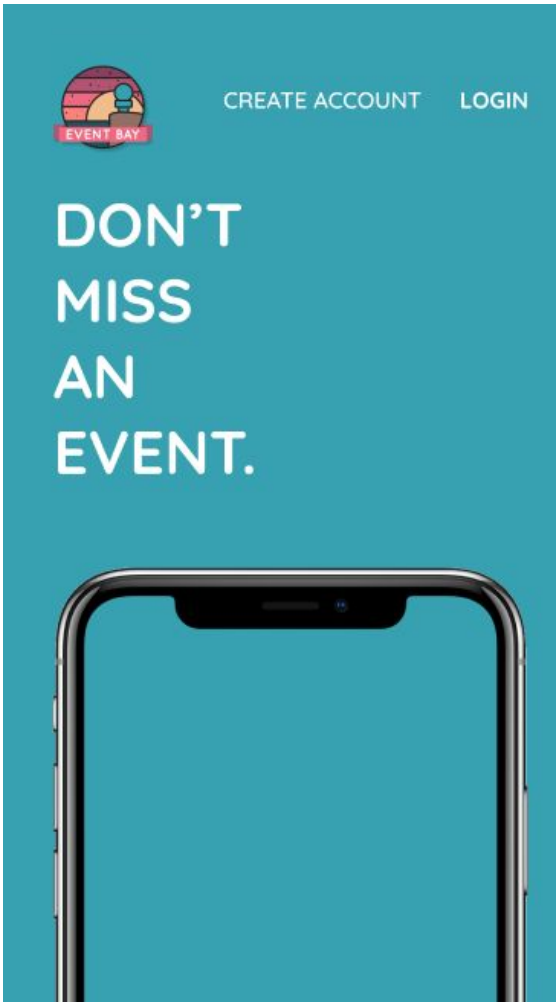
The above state transition diagram considers 13 individual states. The starting state is logically "Viewing landing page" because this is the first thing users will be doing when they arrive at the site. From there, the user can transition to "Creating Account" by pressing a Create Account button, or "Logging In" by pressing a login button. After these points, the user will be authenticated and in the "Viewing Events" state.

When viewing the events, the user can either create an event, view an event's details, or logout. Logging out transitions to the "Logging Out" state and then to "Viewing landing page" state. Creating an event will transition to the "Creating Event" state followed by the "Viewing Event Details" state. Which will also be active if the user clicks on an event.

When viewing event details, the user can either view a user profile, update the event, RSVP to the event, or Un-RSVP. Additionally if they update the event, they can also delete it. Pressing RSVP/Un-RSVP will take the user to that respective state and then back to "Viewing Event Details." Pressing edit will take the user to the "Updating Event" state, from which they can transition to the "Deleting Event' state. If they delete the event, they will be taken back to "Viewing Event" state.

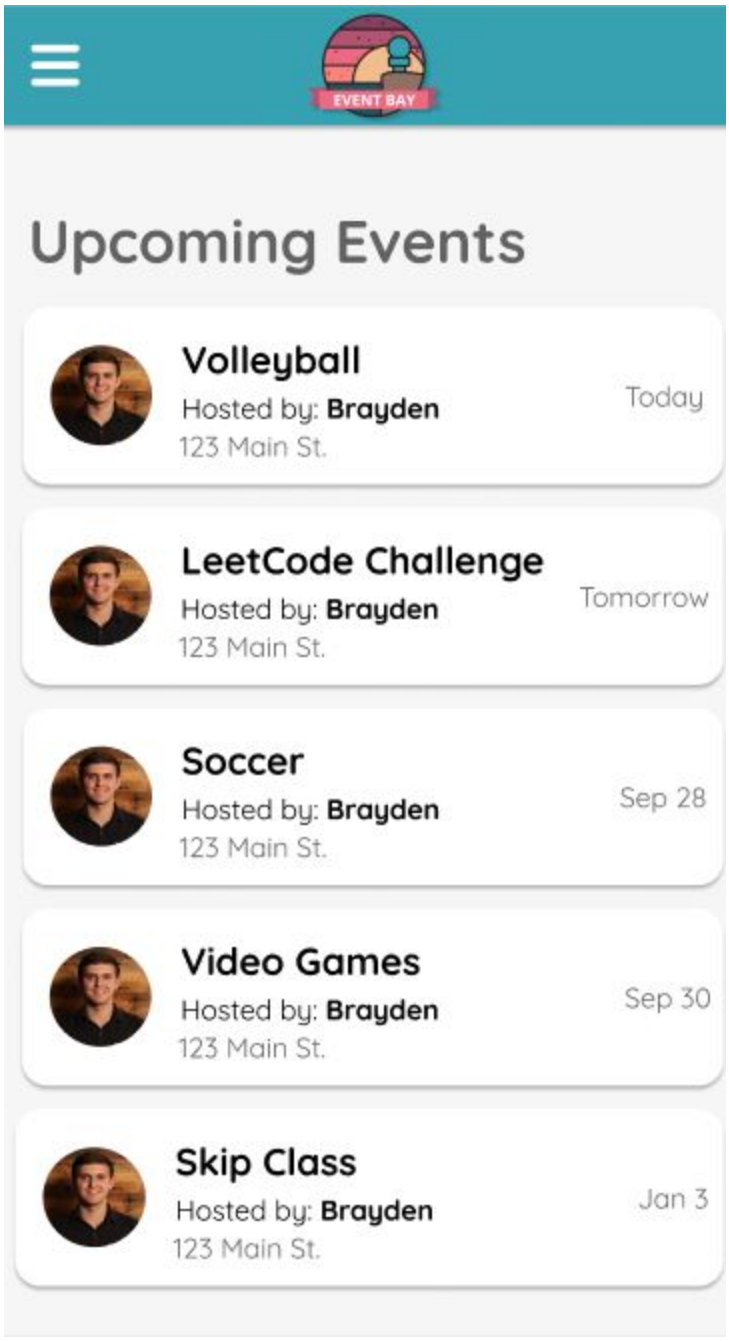
User Interface Design

Screen 1 - Landing Page



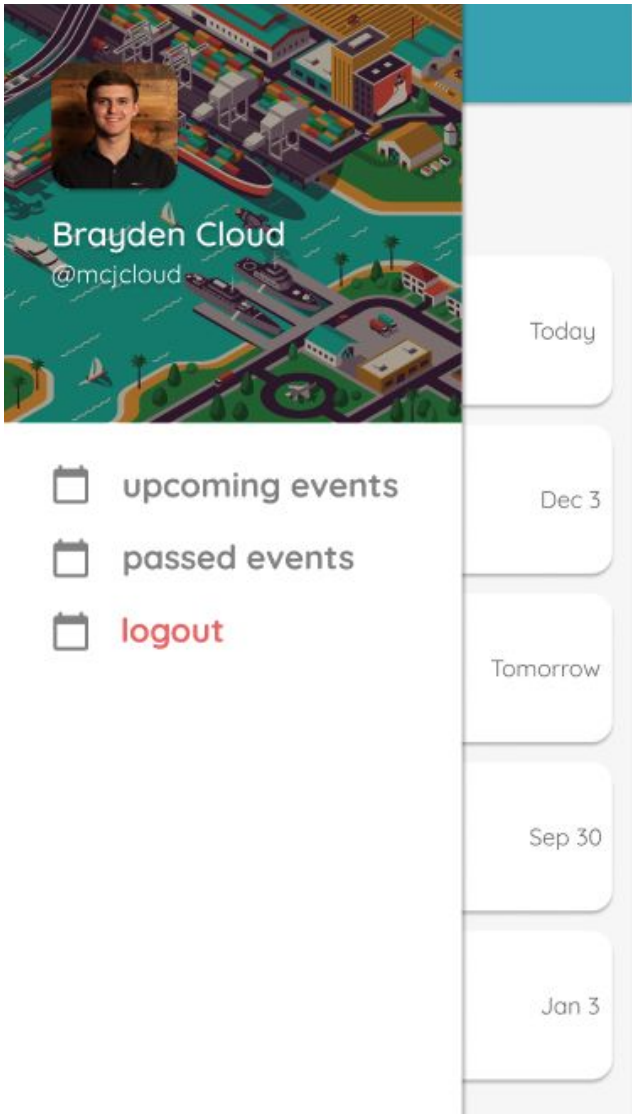
This is the landing page where it sends you if you are not logged in. The Create Account button is at the top of the page so it's clearly visible and when clicked it sends you to the create account page. Next to it is the login button which will direct you to the login page.

Screen 2 - Upcoming events page



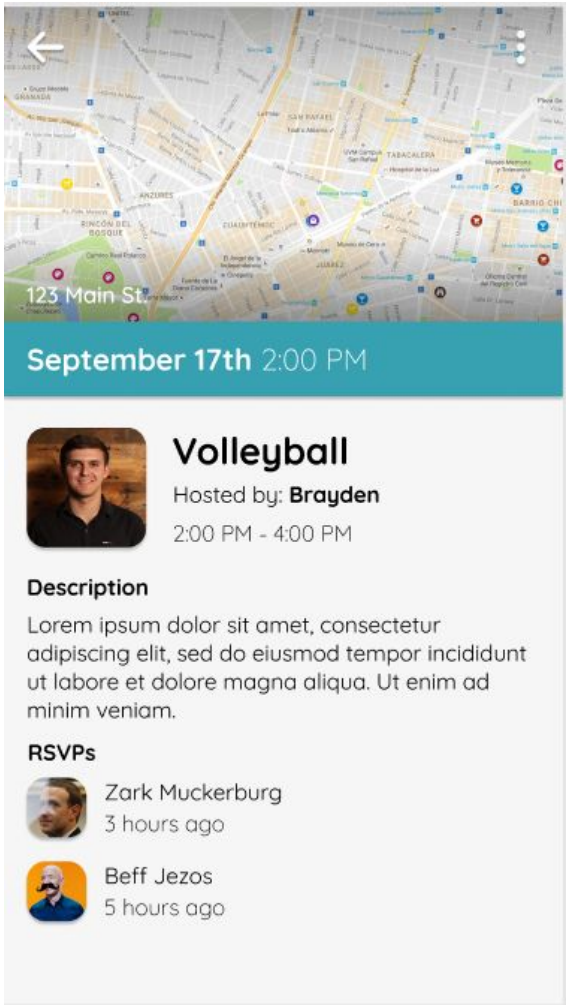
This is the page you will be directed to after logging in. This page displays a list of clickable cards that will direct you to the event page. The card shows the event name, who is hosting it, the date, and the address. The hamburger button at the top left pulls out a drawer.

Screen 3 - Drawer



The drawer pulls out onto the upcoming page. It will display the user information at the top of the drawer. Under the user information there are three clickable buttons. The upcoming events shows the events that haven't happened yet. The passed events button will only show the events that have passed. The logout button will logout the person. It will also send them to the landing page.

Screen 4 - event page



The event page will give details of the event such as the title, the host, the time, a description and the RSVPs. The person who created the event can click on the people who RSVP'd and take them off the event. At the top of the page there is a picture of the address on a map. At the top right there is a back arrow which will redirect you to the upcoming events page. At the top left is a button which will bring up some options about the event.

Conclusion

We learned that software design is a very important part of the development process. Starting without a design can lead to the whole process falling apart and a lot of reworking the app to fit in the specifications that were missed. With a good design, you have a clear direction of where you need to go. We didn't face any difficulties on the design portion of making the app. Now that we have it all laid out in the design document, it will mitigate the troubles we will have in the coding portion, since all the steps to make the app we want are clear cut.