SW Engineering CSC648-848-05 Spring 2023

BucketLyst

Team 1

Marie Shimizu[1], Arielle Riray[2], Francis Quang[3], Tommy Truong[4],
Kenneth Gee, Rabin Karki, Aaron Kuo

[1] Team Lead mshimizu4@sfsu.edu
[2] Front End Lead
[3] Back End Lead
[4] Github Master

Milestone 2

Milestone	Date
M1V1	3/2/2023
M1V2	4/2/2023
M2V1	4/3/2023

Table Of Contents

1. Data Definitions

1. Users

Users are people that use our platform. After registering and successfully logging on, they can use our customer facing features. This includes creating, sharing, deleting lists, or viewing provided data analytics of your business. There are two types of users, and each user type have access to the specific functionalities.

a. PersonalUser

PersonalUser intends to use this app to make and view lists for themselves. They will only be able to create, read, update, and delete lists given appropriate permissions. They can make changes to their account details. Fields they have include the followings:

- **ID**: unique integer that refers to the user. Primarily used for internal user searching.
- **Username**: unique user-decided String, used to be searchable by other users.
- Email: unique String, for communication between BucketLyst admin and user.
- **Password**: String.
- **isPublic**: privacy setting for user.
- **FollowerList**: a list of PersonalUsers that have this user in their Following list.
- **FollowingList**: a list of PersonalUsers that have this user in their Follower list.
- **BlockedList**: a list of PersonalUsers that this user does not want in their following list. This should always be private.
- **LocationLists**: Lists of LocationList that this user can edit or view. Lists may be viewable based on the visibility setting.
- **Profile**: PersonalUser has a Profile to store their name, decription and location

b. BusinessUsers

BusinessUsers are users that own a business that uses our application to promote their company. They promote their company by running ads on BucketLyst. Their

functions include creating and removing ads, viewing active ads. When they register, they will select a business type account with their company details.

- **ID**: unique integer that refers to the user. Primarily used for internal user searching.
- **Username**: unique user-decided String, used to be searchable by other users.
- **Email**: unique String, for communication between BucketLyst admin and user.
- **Password**: String.
- **PhoneNumber**: String, a phone number of this business account
- **CompanyName**: String, the BusinessUser's compnay
- **PaymentInformation**: JSON of their payment information
- **runAds**: boolean, option for BusinessUsers to choose whether the advertisements are active or inactive.

2. **Profile**

Profile is a class associated with PersonalUsers. It is a page that contains the user's name, description, and where they located. Fields of profile will have the following:

- **FirstName**: String, first name of the user
- **LastName**: String, last name of the user
- **Description**: String, description of themselves
- **Location**: Optional for users to set where they are located

3. **Comment**

Comment allows PersonalUsers and BusinessUsers to express their opinions, provide feedback, or ask questions about a place. The comment field are the following:

- **ID**: unique id used to identify and track comments. To ensure comments are organized and analyzed.
- **Author**: refers to the PersonalUser
- **addedDate**: Portrays date when comment was added
- **isEdited**: boolean, shows that comment was edited or not
- **Content**: String for description

4. **LocationList**

LocationList is a list of Locations. It's used by PersonalUsers to edit, view and share their list of locations they like. It's editable by multiple users, has visibility options, and a description. It can only be deleted by the original owner. Thus, the following fields:

- **ID**: unique integer that refers to this LocationList object. Primarily used for internal searching.
- **Name**: String, a searchable name for PersonalUsers
- **Tags**: Collection of Strings to describe the contents of this LocationList
- **Description**: String, descriptor
- **Location**: The general location or area where all these Locations are located
- **Owner**: PersonalUser that owns this LocationList
- **Collaborators**: a list of PersonalUsers that can edit this LocationList.
- **Locations**: Collection of Locations that this LocationList contains
- **isPublic**: Indicates visibility of List

5. **Location**

Location is a container for a Location Data Object where a Personal user may edit fields they want to describe this Location.

- **Creator**: original user that created this specific Location
- **Name**: Name that the Personal user would want to call this specific Location
- **Description**: String descriptor
- **Tags**: Collection of Strings to describe this Location
- **GoogleMapLink**: link to their Google Maps page, grabbed from Location Data Object
- **LocationDataObj**: Location Data Object that contains internal data about this location

6. **LocationDataObject**

This is an internal reference to the specific location. Separating the location data details from the Personal user to access allows for individual customizability for Personal users.

- **DateAddedToDatabase**: date first created by a user
- **GPSCoordinate**: From a Google Maps link, parse the specific longitude and latitude for location manipulation
- **GoogleMapsLink**: Direct string of the location on Google Maps
- **IncludedInList**: number of people that include this in their list

2. Prioritized Functional Requirements

Priority 1:

New User:

1. User should create their username
2. User should enter their email
3. User should create a password
4. Same username and email cannot exist in database

Registered User:

5. User should be able to sign in with their registered details
6. Users should be able to reset their password in case they forget it
7. User can create a list
8. User can manage a list
9. User can own multiple lists
10. User can name a list
11. User can add tags
12. User can edit lists
13. User can edit location data fields
14. User can edit user profile
15. User can edit notes on the lists
16. User can edit their account details
17. User can edit the visibility of the list (public, only with friends, private)
18. User can add lists
19. User can add their followers
20. User can add additional information to profile
21. User can add notes on the lists
22. User can add a location to list
23. User can delete lists
24. User can delete notes on the lists
25. User can delete their account
26. User can delete location from their list
27. User can share their list with others with link
28. User can search for locations by tags
29. User can search for locations by categories
30. User can search other users on the platform
31. User can follow other users
32. Users can see other's profiles

33. User can sign out
34. User can explore lists through the recommendations
35. Users can view a map that displays the location of their saved places and recommendations.
36. Users can import their existing lists from Google Maps

Location:

37. Location contains google map link that allow user to jump to the location on google map seamlessly
38. Location has “Navigation” that allow users to open google map navigation

Priority 2:

Registered User:

1. User has option to register using Google account
2. User has option to register using facebook account
3. User can delete followers
4. User can delete their following list
5. Users can own shared lists as a group so that everyone can edit them
6. User can search for locations by radius from the user’s current location
7. User can search for locations by open now
8. User can search public lists with keyword
9. User can send a follower request to other users
10. User can accept follower request
11. User can deny follower request
12. Users can rate a place on a scale 1-5 (tier)
13. Users can view notifications
14. Users can turn off notifications
15. Users can turn on notifications
16. User can be notified via email
17. User gets notified for new activities in other people's list
18. Using the data to create popular lists like “most favorited places”
19. Users can view recommendations from other users
20. User can compare their list with selected list and see the difference

Business User:

21. Business user should enter their company name
22. Business user should enter their email
23. Business user should create a password
24. Business user should be able to sign in with their registered details
25. Business user should be able to reset their password in case they forget it
26. Business user can sign out
27. Business user can market their establishment on platform

Priority 3:

Registered User:

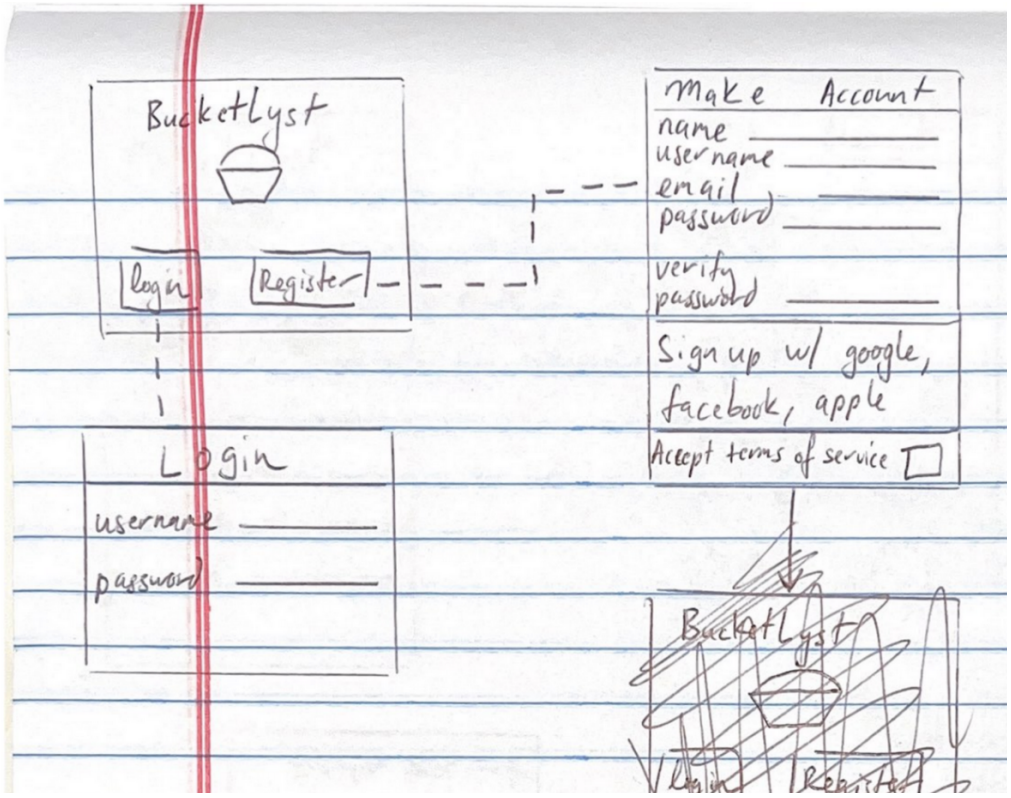
1. User can copy other's list to make it their own list
2. User can temporarily disable their account
3. User can choose the default privacy setting for their list
4. User has ability to choose to get notified - changes on the list
5. User has ability to choose to get notified - changes friend's activity in general
6. Users can report abuse, such as inappropriate comments or content

3. UI Mockups and Storyboards

Use case 1: Signing up

Actor: Peter

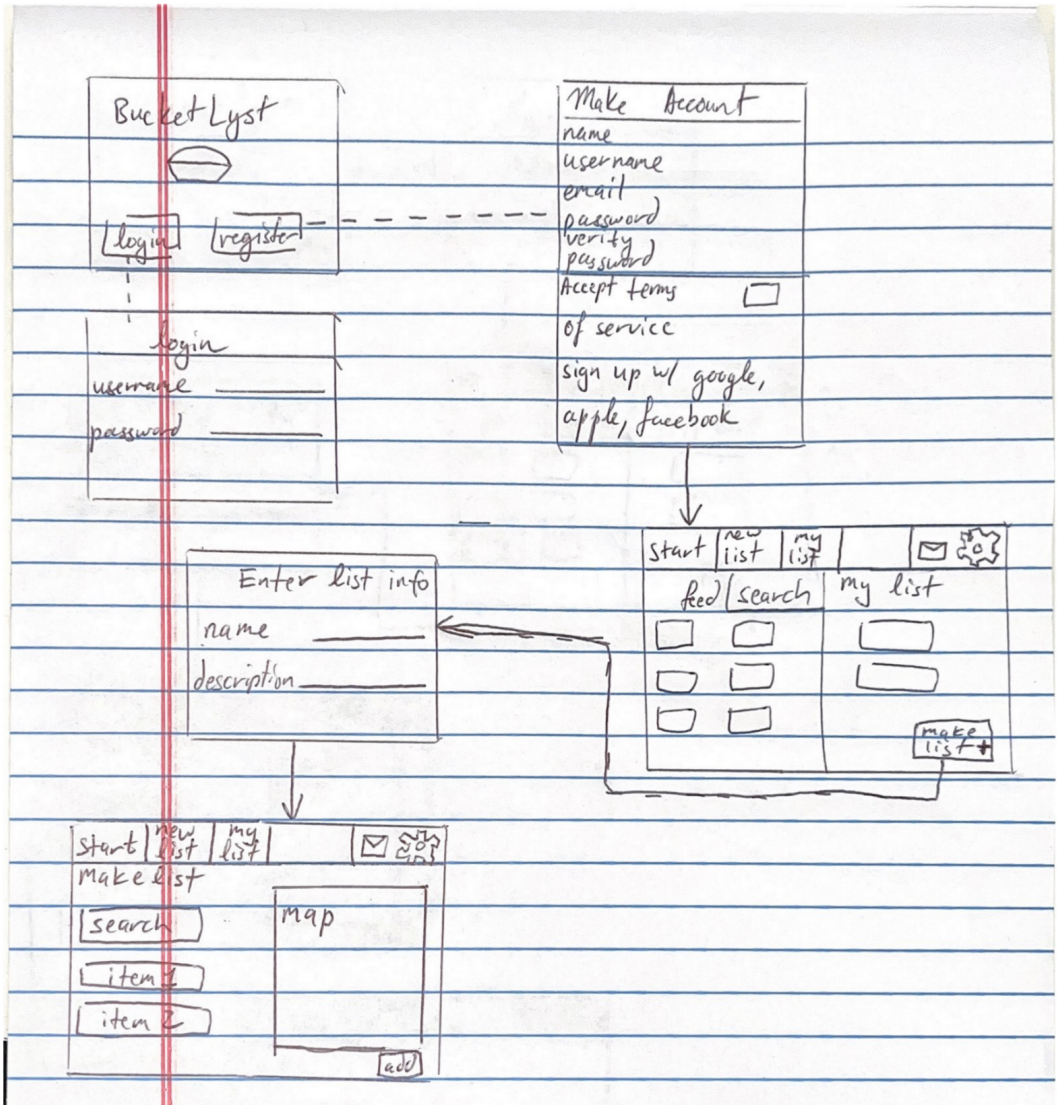
Peter loves to ask for recommendations on bars and restaurants from his friends, but he forgets their names too easily. His friend shows him BucketLyst, where he can create an account and have access to his friends' lists on the app when needed. His friend shares the URL, and he opens the website. He scrolls through the website and finds the "Sign Up" option at the top, which allows him to create an account and see his friends' lists. After clicking the "Sign Up" button, he is prompted to fill in details, including his first name, last name, and other information. He also has the option to create an account with Google or Facebook. Since he uses Gmail, he chooses to sign up using his Google account. However, he sees an error message because he has not yet agreed to the terms of service. After agreeing to the terms of service, he tries again, and the sign-up is successful.



Use case 2: Create a list

Actor: Alice

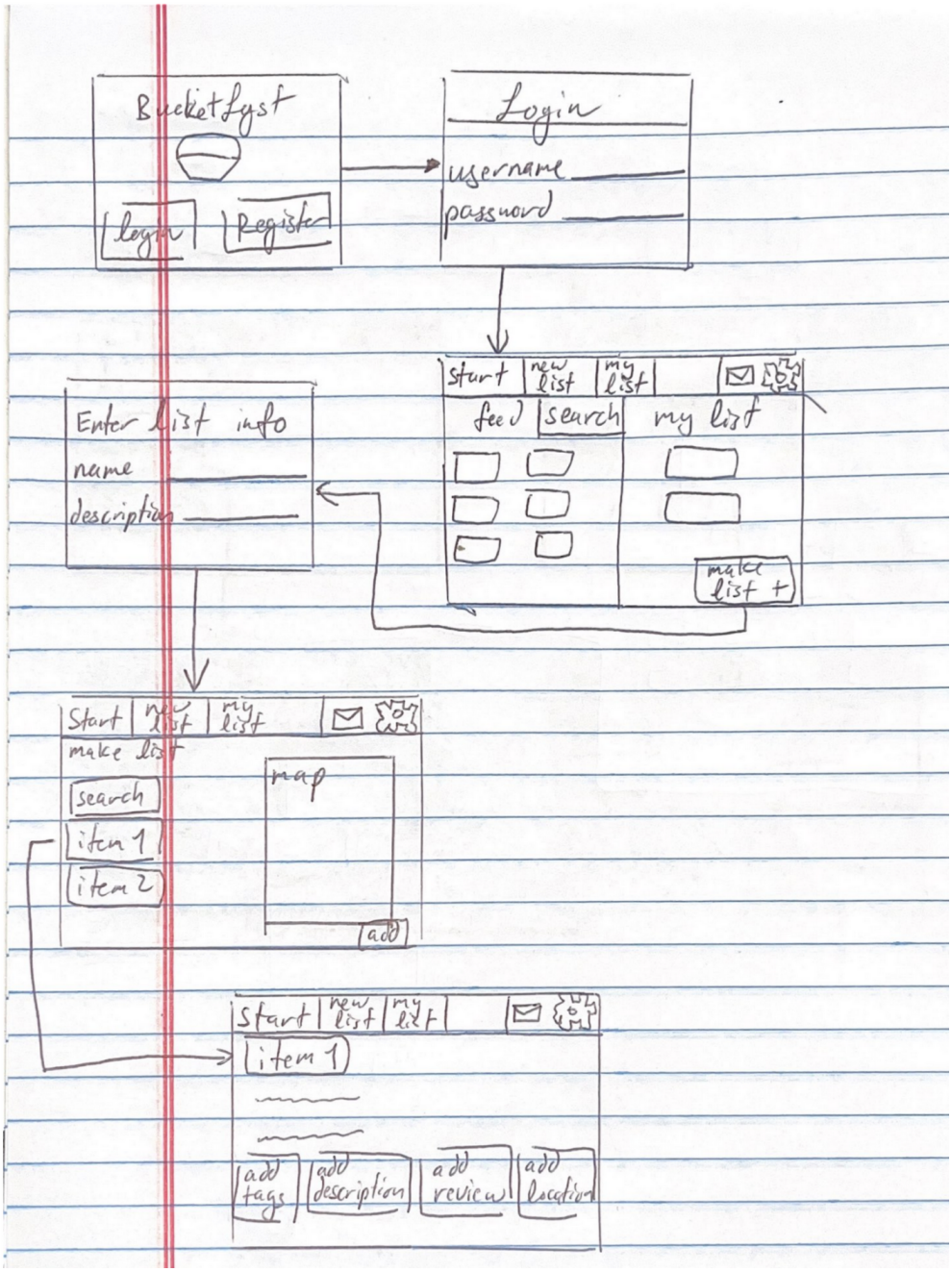
Alice wants to create a list to store her favorite cafes and the places her favorite influencer frequents more efficiently. She feels her iOS notes get too messy and unorganized. When she searches on Google for "how to create a list of favorite places", the second search result shows her BucketLyst with a short description: "Organize your favorite places with ease using our app. Create and customize lists, share them with friends, and filter and search through saved locations. Never forget a great spot again!" She is intrigued, and with only a few steps, she already has an account. Now she can create a list. BucketLyst pops up a tutorial to guide her in creating her first list. Alice follows the arrow and goes to "Lists," where it is empty, and the arrow appears again, pointing to the "+" sign at the top. She clicks it and is prompted to enter the name and description of the list. After creating the list, she is guided to add locations to the list, where she can search for the name and it will show the result, just like on Google Maps. She selects the result, and it auto-populates the rest of the information like the address and store hours. Now she has a list with her favorite cafes!



Use Case 3: Add Details on One Location

Actor: Drew

Drew is an adept user of Google Maps and is familiar with its lists feature. He's interested in migrating to BucketLyst because he just saw an ad for it and wants to use it as a central hub for all his lists, including his own reviews and labels for his followers to see. He logs into BucketLyst directly as he already has an account, and creates a new list by clicking on the appropriate button. He then adds an appropriate name and description for the list. Now, he can add a new location to the list by selecting the corresponding button and filling out the required fields, such as the name and the location search. Although the description and tags are optional, he fills them out for the convenience of his followers.



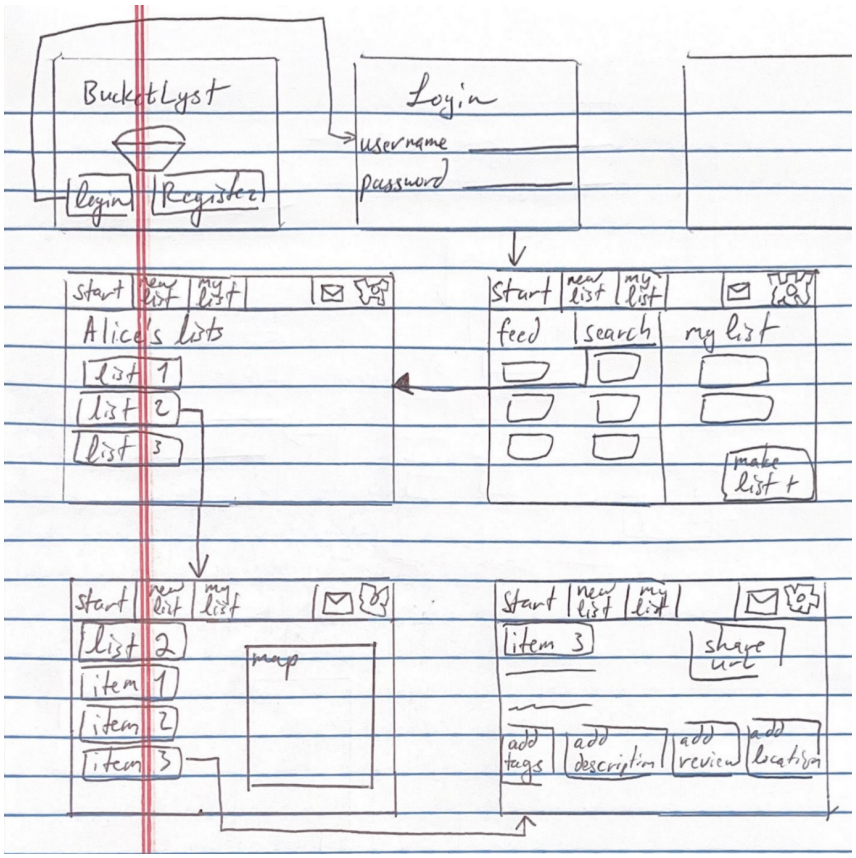
Use Case 4: Share a list

Actor: Drew, Alice

Drew is a renowned YouTuber who predominantly uploads content related to traveling and food reviews on his channel. He is very transparent with his fans and audience, and he shares his favorite things with them on YouTube. Since joining BucketLyst a couple of weeks ago, Drew cannot stop talking about the platform's amazing features. He was impressed by the idea of creating a list of favorite things in multiple categories. He created his own list of favorite things in his account. Moreover, he gained over 500 followers within a short period, and his followers provided constructive feedback on his lists through comments.

Alice has been Drew's biggest fan since high school, and she never misses any of his videos. She is an organized individual who loves traveling to different places and trying out new cuisines. In Drew's recent YouTube video, he talked about BucketLyst's interactive features, such as sending follow requests, commenting on lists, creating personalized lists, and creating groups to maintain lists among users. Drew is very social and aims to connect people through BucketLyst user profiles by sharing his favorite lists with others. As a result, he decided to provide his link in the description section of his YouTube channel.

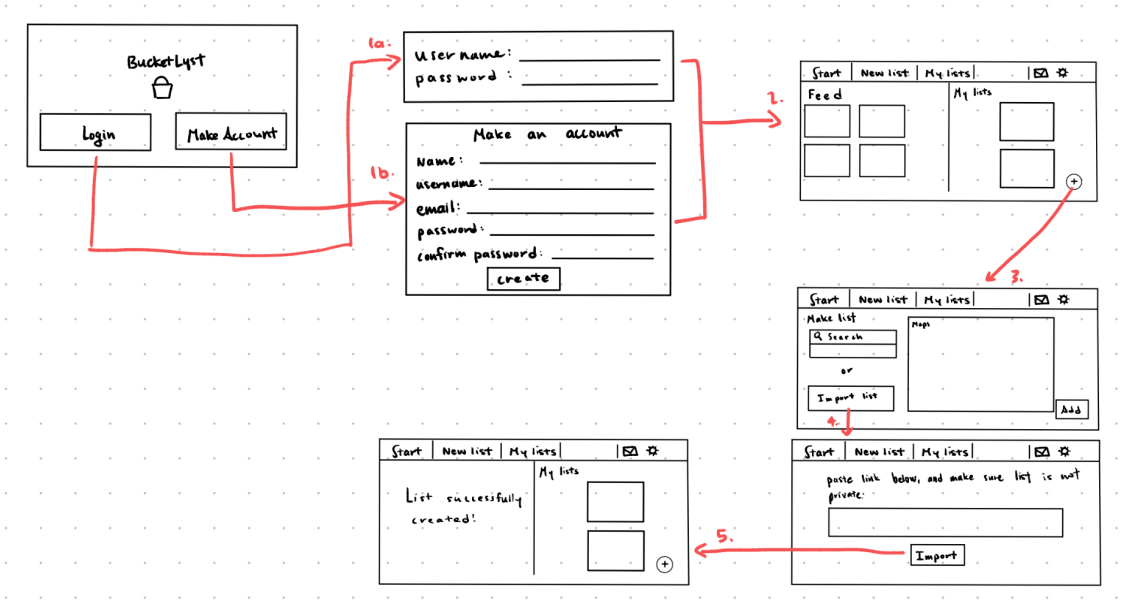
When Alice watched the video, she paused it in the middle and immediately followed her idol through the link from the description section. Since Drew is an influencer, he always makes his list public to his fans. When she visited the link, she signed into the app first before being directed to Drew's profile. Alice got excited about some of his lists, such as his favorite hiking places in SF. She followed him on BucketLyst and created her own list based on Drew's list. She also decided to share some of her lists with her group of friends, allowing others to edit her list with feedback in notes or descriptions.



Use Case 5: Import a List from Google Maps

Actor: Mary Jane

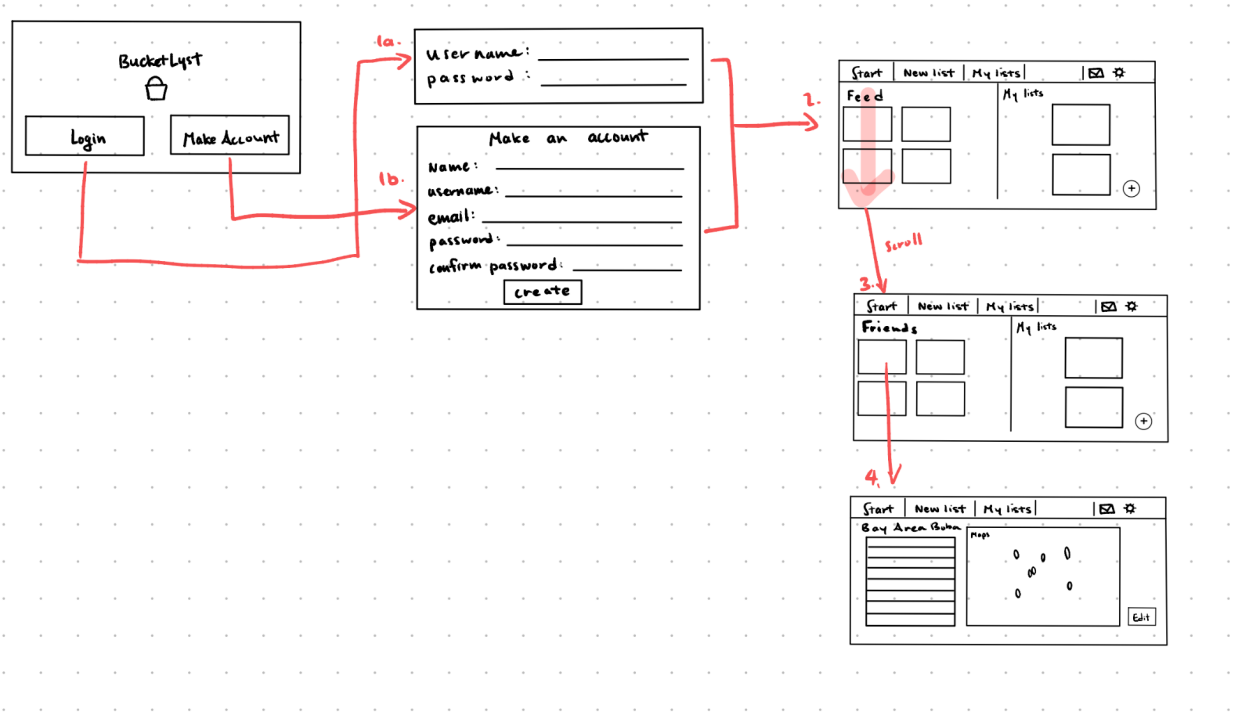
Mary Jane is a frequent traveler and digital nomad who creates extensive lists on Google Maps for every country or city she visits. However, she finds it challenging to navigate through her Google Maps list as it is one big list. She has to scroll repeatedly to find each item, making it tedious to plan out her trip. Her best friend, who shares the same name, Mary, tells her about BucketLyst and how the startup created an engaging and useful web app for solving this kind of issues. After learning about BucketLyst's powerful sorting and filtering tools, Mary Jane decides to import her lists from Google Maps. She navigates to BucketLyst and creates a new account. After logging in, she creates a new list with a meaningful name and clicks on the option to import from the web. Mary Jane then enters the URL of her Google Maps list, and now she has the Google Map list on BucketLyst with additional features to manipulate it!



Use Case 6: Exploring New Places from Friends' List

Actor: Peter

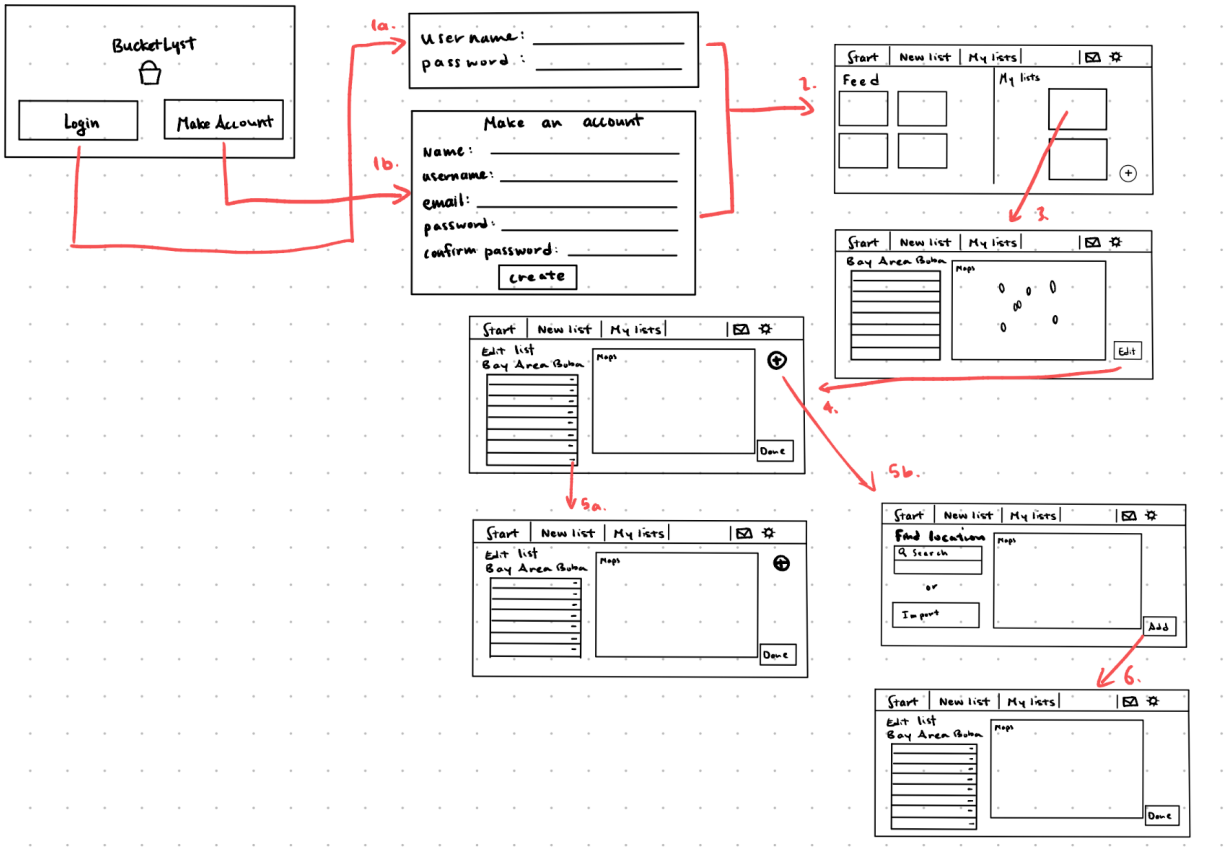
Peter is an adventurous and spontaneous person who seizes any opportunity to go out and explore. He loves discovering new places and trying out new restaurants with good food. One day, as Peter is searching for new places to try, he comes across an advertisement for BucketLyst and decides to give it a try. He tells his friends about BucketLyst and discovers that he can see their recommendations on the platform. Usually, Peter's friends are busy with work, so they may not be able to respond promptly to his requests for recommendations. However, with BucketLyst, he no longer has to wait for his friends to respond. He clicks on his friend's list and starts browsing through the recommendations that they have created.



Use Case 7: Search within a List with Criteria

Actor: Mary Jane

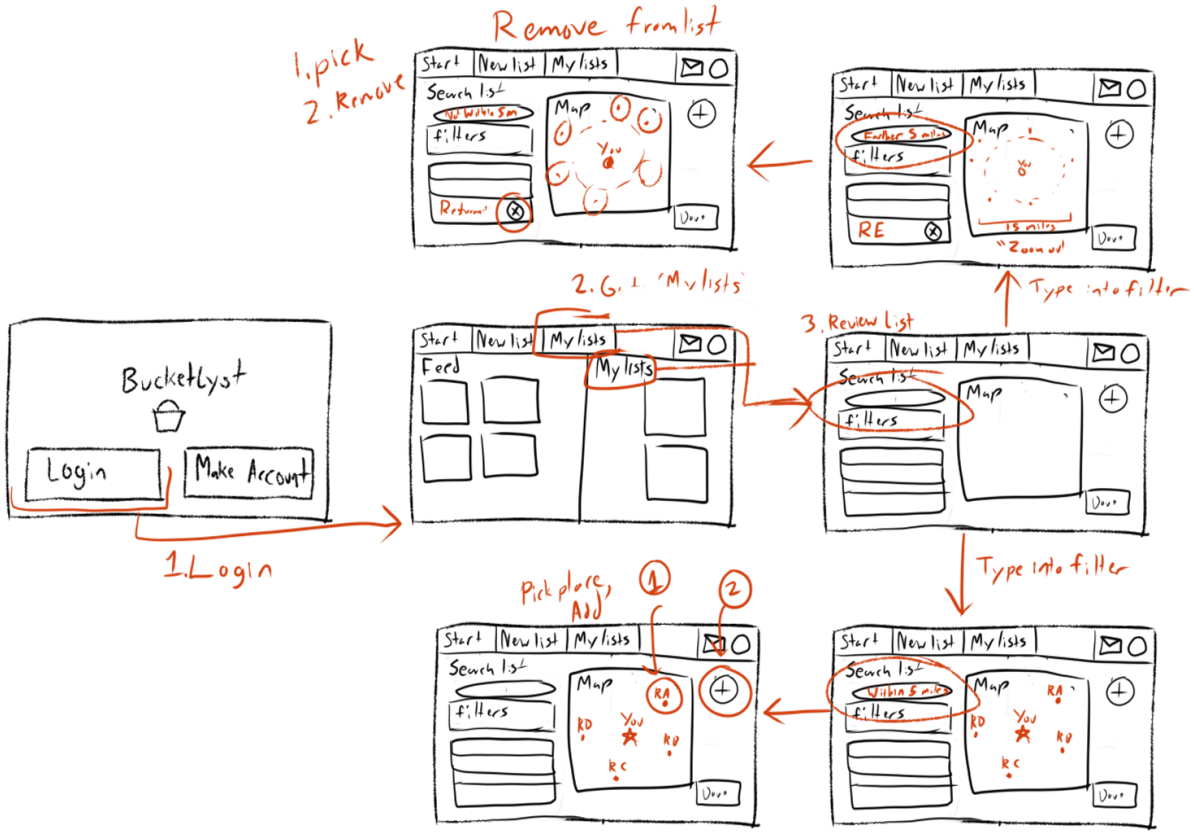
Mary Jane is on her way to Rio de Janeiro for a month to work remotely after the holidays. Before her trip, she had friends send her many recommendations that she accumulated over time. Her Google Maps list has over 90 locations! Before arriving, Mary Jane develops a strong craving for "Asian food," so she begins by importing her huge Google Maps list into BucketLyst. After importing her list, Mary Jane becomes elated at the idea of being able to search through her lists for some Asian food. She opens up BucketLyst and logs into her account. Once she is in, she sifts through her lists to find the one she labeled "San Jose, California." Once inside her list, she clicks on the filter search bar and enters "Asian food." Her original list of 99 spots is now reduced to 10 of her listed restaurants.



Use Case 8: Edit an Existing List

Actor: Lisa

Lisa is a recruiter who has a list of favorite lunch spots she likes to go to during her break. Recently, Lisa's company has been growing and becoming increasingly busy. Because of this, her main focus is time efficiency, so Lisa wants to update her list of favorite lunch spots by finding the ones closest to her office. She remembers some places that took too much time during her break, so she will use the search function to find those specific restaurants or if she does not remember, then she will filter by distance from location. If a restaurant is too far away, she will remove it from her BucketLyst. After removing some, she realizes the list is too small, and she wants to broaden her variety of choices. So, she plans to add new restaurants close to her company to her list.

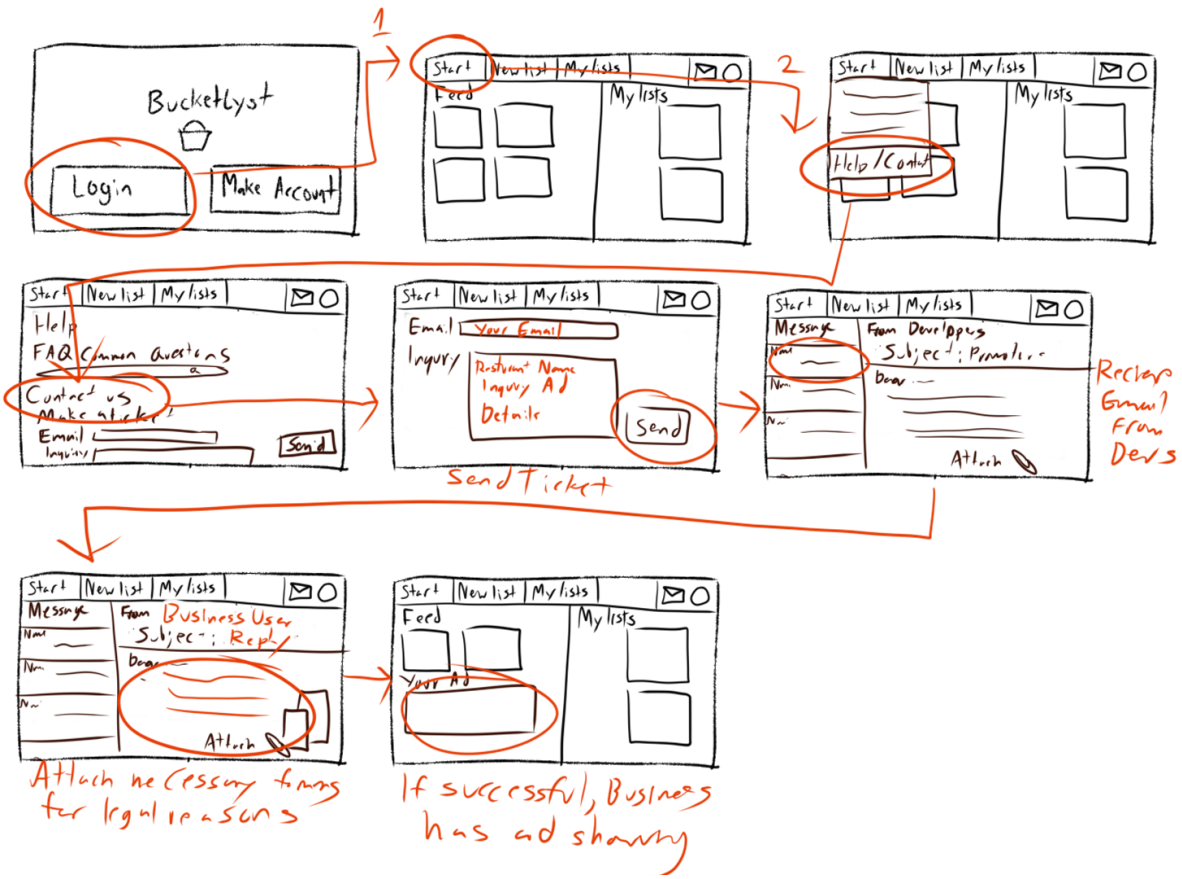


Use Case 9: Boost a Business Presence

Actor: Olivia

Olivia is the owner of a restaurant. One day, she walks in during the lunch hour and surveys the restaurant. Although the restaurant isn't completely full, there is a sizable number of patrons. However, Olivia feels that the remaining empty tables could be filled. She approaches the remaining customers and strikes up a conversation, asking how they found the restaurant. The customers show Olivia that they found the restaurant on the BucketLyst app, after being recommended it by their friends. Olivia is delighted to hear this and notices that there are sponsored results for other nearby businesses at the top of the results page. After business hours, Olivia thinks about her conversation with the customers and decides to get in contact with the BucketLyst app developers through the contact form on their app. After some negotiation via direct messages, verification through identification details, filling in legal forms, and signing a contract, Olivia and the app developers strike a deal to sponsor her restaurant. The sponsorship allows the restaurant to appear at the top of the search results page randomly for users within a 10-mile radius.

Overall, the BucketLyst app developers get paid for the sponsorship, users see verified recommendations for restaurants, and the restaurant receives a boost in customers. It's a mutually beneficial relationship where both the business and the app developers benefit.



Use Case 10: Gaining a follow on BucketLyst

Actor: Brandon (Traveler)

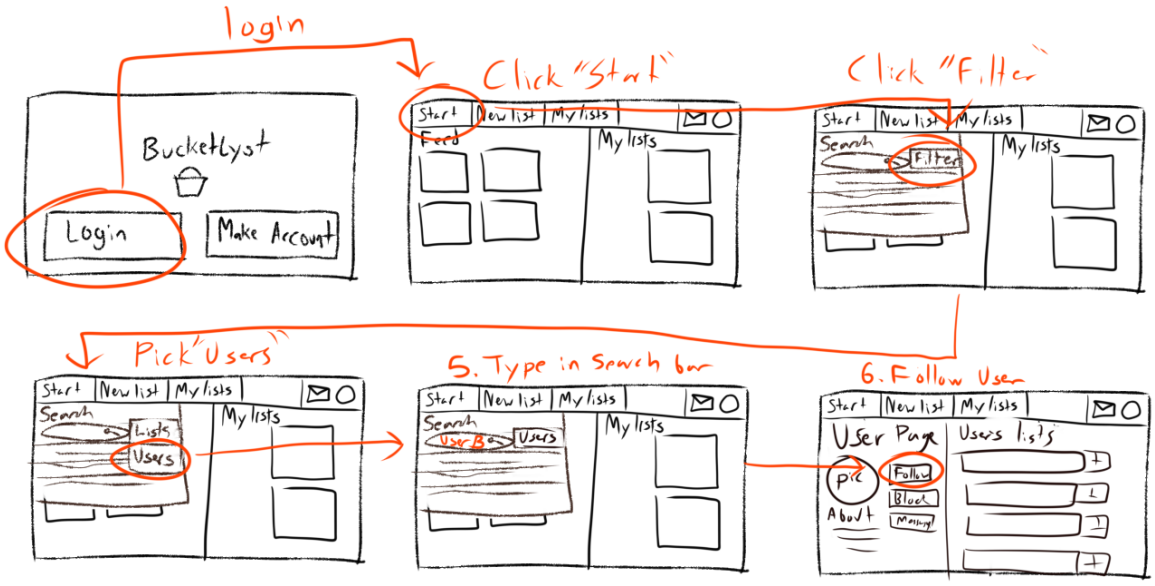
Enter Brandon, a well-off traveler living what some may call the "Dream-Life." Due to his high-paying remote job, he is free to venture throughout the world wherever he pleases at any time of the year (granted the place has Wi-Fi). Despite all the talks he'd have with the locals of the places he'd visit, they were only one-time conversations, and then he'd go off and never see them again. He'd learn the stories of hundreds of people but never got to tell his own. They were all bundled up in the camera roll of his phone, only there for his eyes to see. Everyone else seemed to have a home, but Brandon felt the opposite. He felt lost.

With all the places he has visited, it's quite overwhelming trying to recall which one was his favorite. Thankfully, he comes across the BucketLyst app, and he begins using it. Finally, a place where he can lay out the history/stories of his travels! As he's building up his list with places, he visits a beach in the San Francisco Bay Area. There on the beach, he meets a woman and has a well-meaning conversation that goes a bit deeper than he expected. She was interested in his travels, which was a pleasant surprise since he hasn't met anyone else who has done so. Then Brandon proceeds to show her his profile on the BucketLyst app and the list of places he has visited. It becomes very apparent to Brandon that he very much enjoyed her company. The woman looked to share the same sentiment as well. Unfortunately, they had to soon part ways.

Later that night, Brandon reflects on their encounter and realizes he never got her contact info or even her name. He finds himself feeling a deep sense of regret and sadness that he'd most likely never see her again, just like those hundreds of others he met on his travels. In the morning, Brandon wakes up to an unusual notification. Someone by the name of "Sandra" followed his profile on the app. He sees her profile picture, and that's when it hits him. It was her! The woman on the beach! Brandon had shown her his profile on the BucketLyst app, and because of that action, she was able to find him by searching up his name username in the search bar, visiting his profile, and sending him a friend request. Right now, it would be seen as "Pending" on her screen with a yellow-colored button. Brandon clicks the notification and is greeted by a prompt of "Accept" or "Decline." He clicks the green "Accept" button, and the app opens a page to her profile. There among the list of places were all sorts of stops in the Bay Area. However, at the top of her favorites list was one he had never been to before, "Sutra Baths."

Later that day, Brandon thinks about the place he saw on Sandra's list and decides to go for a visit at sunset. When he arrives, he's greeted by a beautiful sight of the rocks and ocean. Then from behind him, he is greeted by that unmistakable voice from the beach.

"I knew that putting this place on my list would lead to us meeting again."


login
Click "Start"
Click "Filter"
Bucketlyst
Login
Make Account
Start
New list
My lists
Feed
My lists
Search
Filter
My lists
Pick "Users"
5. Type in Search bar
6. Follow User
Search
Lists
Users
My lists
Search
User B
Users
My lists
User Page
Users list's
Pic
Follow
Block
About

4. High level database architecture and organization

A. The Database Requirements (Business Rules)

Based on the competitive analysis we conducted in Milestone 1, we identified the ability for users to customize their lists as the primary unique feature of our application. This includes the ability to create, edit, add, delete, search within, and import from Google Maps. All functional requirements associated with these features are considered of equal importance (Priority 1). The requirements below:

1. PersonalUser
    i. A user must have a unique email address and username.
    ii. A user can have zero or more LocationLists.
    iii. A user can make zero or more locations.
    iv. A user can follow zero or more other users.
    v. A user can be followed by zero or more other users.
    vi. A user can be blocked by zero or more other users.
2. UserProfile
    i. A user can make at most one UserProfile.
3. LocationList
    i. A locationList must have a single owner user.
    ii. A LocationList can be public or private.
    iii. A LocationList can contain zero or more locations.
4. Location
    i. A Location must have GPS coordinates or a Google Maps link.
    ii. A Location can have a rating system.
    iii. A Location can have zero or more tags.
5. LocationDataObject
    i. A LocationDataObject must have GPS coordinates or a Google Maps link.
    ii. A LocationDataObject can be part of zero or more Locations.
6. Follower
    i. A Follower must have a user who is being followed and a user who is doing the following.

B. The Entities, Their Attributes, and Domains at the High Level

1. **PersonalUsers** (Strong): This table stores basic user information such as username, email, and password, as well as a flag indicating whether the user's profile is public or not.

    - user_id: primary key, numeric
    - username: strong key, alphanumeric
    - email: strong key , alphanumeric
    - password: weak key, alphanumeric
    - is_public: weak key, boolean

2. **UserProfile** (Strong): This table stores the information about the profile information of users.

    - profile_id: primary key, numeric
    - first_name: weak key, alphabet
    - last_name: weak key, alphabet
    - description: weak key, alphanumeric
    - date_of_birth: composite key, numeric
    - user_fk: foreing key, numeric

3. **Followers** (Weak): This table stores the information about the user's followers and followings.

    - id: primary key, numeric
    - follower_fk: foreign key referencing to PersonalUser entity, numeric
    - following_fk: foreign key referencing to PersonalUser entity, numeric

4. **LocationLists** (Strong): This table stores the information about the LocationLists such as the name, description, and creator of the list, as well as whether it is public or not.

    - id: primary key, numeric
    - name: weak key, alphanumeric
    - description: weak key, alphanumeric
    - location_name: weak key, alphanumeric
    - is_public: weak key, boolean
    - created_time: weak key, alphanumeric
    - owner_fk: foreign key referencing User entity, numeric
    - photo: unique key, alphanumeric

5. **Locations** (Strong): This table stores the information about the location of the lists.

    - location_id: primary key, numeric
    - name: weak key alphanumeric
    - description: weak key alphanumeric
    - tags: weak key alphanumeric
    - map_link: weak key alphanumeric
    - rating: weak key, numeric
    - created_time: weak key, alphanumeric
    - list_fk: foreign key referencing LocationList entity, numeric
    - location_data_fk: foreign key referencing to LocationDataObject entity, numeric

6. **LocationDataObject** (Strong):This table stores the actual location data, such as GPS coordinates and Google Maps links, and is referenced by the Location table.

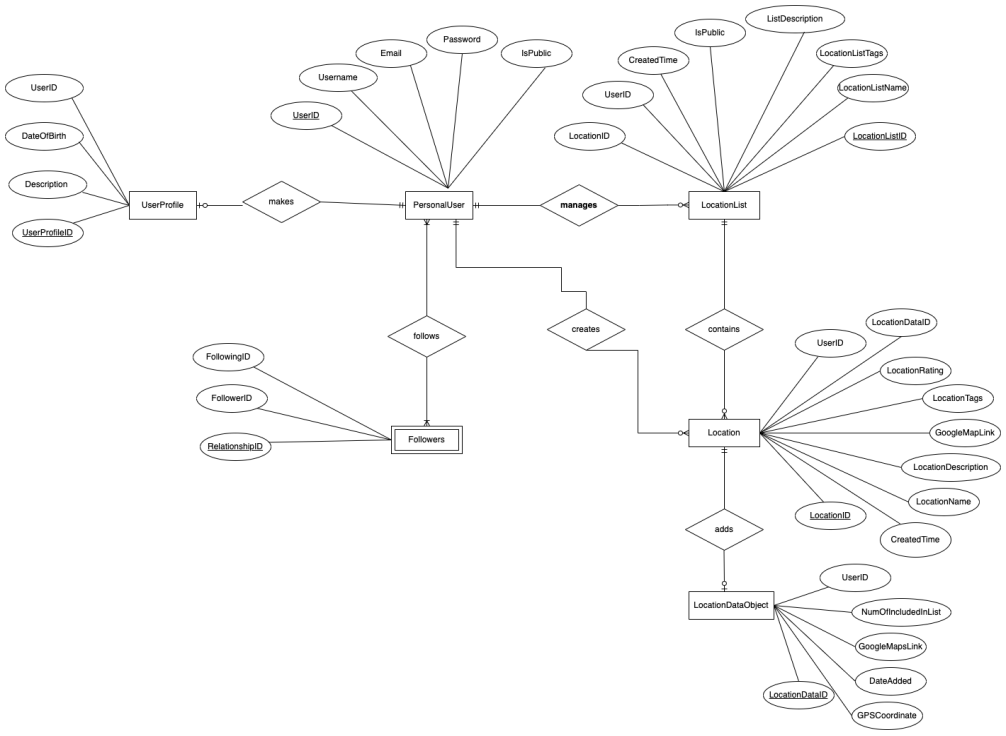
    - ldo_id: primary key, numeric
    - creation_date: weak key, alphanumeric
    - gps_info: weak key, alphanumeric
    - map_link: strong key, alphanumeric
    - included_list: weak key, numeric
    - location_fk: foreign key referencing Location entity, numeric

C. The Relationships Table

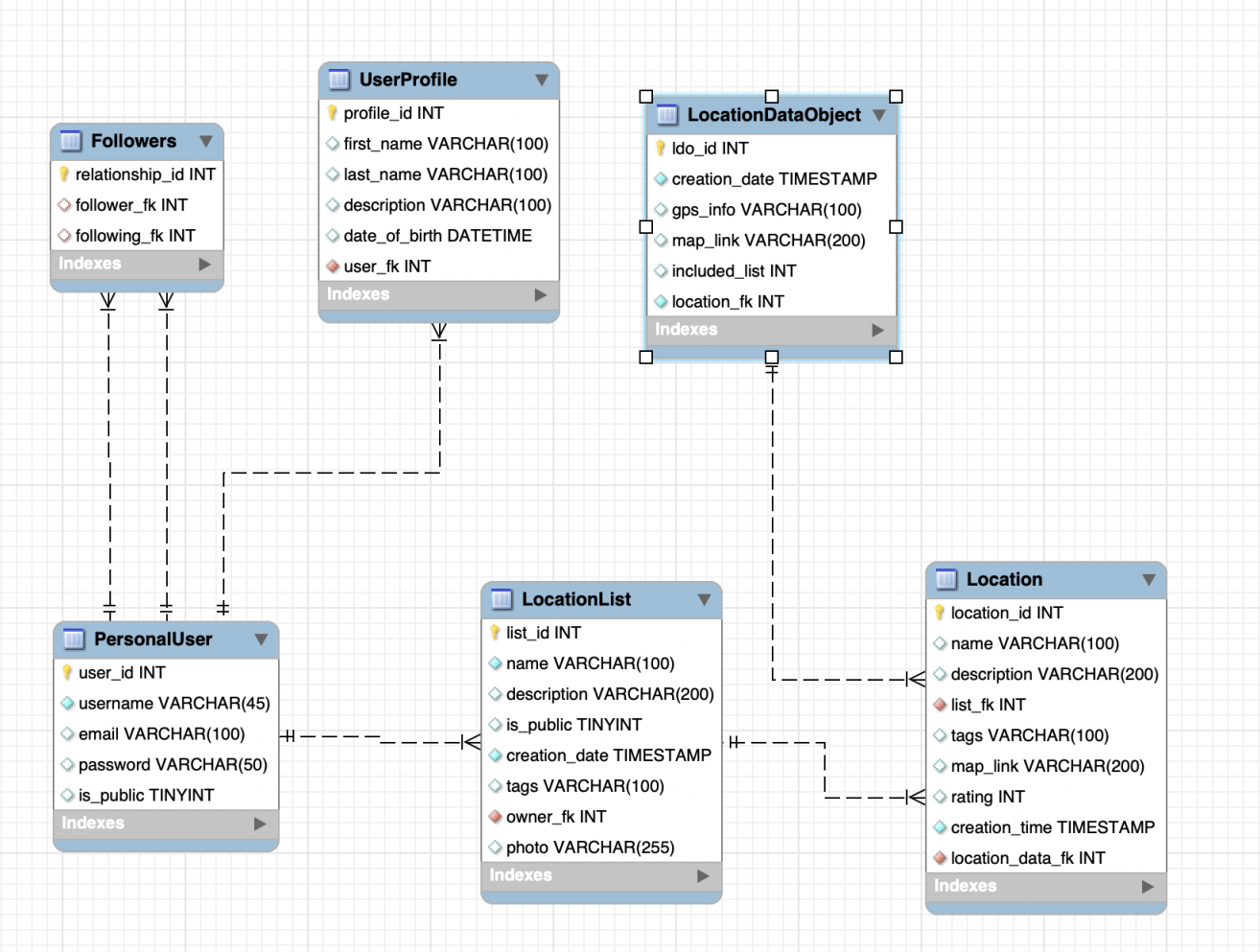
Rule	Entity-1	Entity-2	Relationship	Relationship Type	Nullable (NOT NULL-NN)
1	PersonalUser	LocationList	manages	1-0/M	NN
3	PersonalUser	Followers	follows	M-M	NN
4	PersonalUser	Profile	make	1-1	NN
5	LocationList	Location	contains	1-0/M	NN
6	Location	LocationDataObject	adds	1-0/1	NN
7	PersonalUser	Location	creates	1-0/M	NN

Note: 1-0/M→ one to optional many , 1-0/1—> one to optional 1 relationships

D. ER-Diagram Based on the above requirements



E. Database Model(EER)



F. Defining DBMS

We opted for MySQL as the DBMS to build our database because Amazon RDS offers a free tier for MySQL instances, which helps us save costs. Additionally, MySQL is a well-known and widely used open-source DBMS that supports relational databases.

G. Media Storage

The images and thumbnails in our application will be stored to AWS S3 cloud service and it will produce URL for the image and the URL will be stored in the database.

H. Serch & Filter Algorithem

Our search/filter architecture and implementation prioritize searching for users and location lists based on specific criteria. For the GET /users and POST /searchUser

endpoints, the search algorithm is a straightforward SELECT query with a WHERE clause that filters the results based on the user's name. As for the GET /LocationList endpoint, the search algorithm involves a SELECT query that filters the location lists based on both the LocationListName and LocationName attributes.

To facilitate the search, we created a LocationList entity in the database with the LocationName and LocationListName attributes. We use the %like operator in the SELECT query to match any characters before or after the search Location. The query selects all the lists from the LocationList table that have at least one list item with a LocationName that matches the search term LocationName.

Overall, the search/filter implementation in the code is based on query parameters and SQL queries with WHERE and LIKE clauses to filter the results based on username and LocationList information.

5. High Level APIs and Main Algorithms

1. We will be creating a **geolocation API** for our location-based service on our web app. The functions will be for displaying nearby points of interests and providing directions for our users.
    a. For the geolocation API we will be using third-party API, Google Maps free plan API, “Maps JavaScript API ".
    b. With Maps JavaScript API, the functions will be:
        i. Map display - Create and display maps on our web page.
        ii. Markers - Users will be asked to mark certain locations on the map. Users can add labels and information windows to markers to personalize them.
        iii. Info windows - When a marker is clicked, info windows open with extra details about the selected site. The API offers tools for designing info windows with text and links.
        iv. Directions - On the map, our API will show the routes from point A to point B.
        v. Geocoding - This is the procedure for converting a location or street address into latitude and longitude coordinates for use in mapping applications.

2. **Social media API** that will allow users to follow other user profiles, receive notifcations, post, like, comment, and share content.
    a. Functions for our social media API listed:
        i. Commenting- API adds data to desired location by the user
        ii. Search - Developers can look for particular users or content on our platform
        iii. Login - By validating their login information and granting access to application resources according to their role or permissions, the API enables users to log in to the application.
        iv. Logout - Users can log out of the application over the API to end their session.
        v. Register - Users can utilize the API's capabilities to register for new accounts by entering their name, email address, and password.

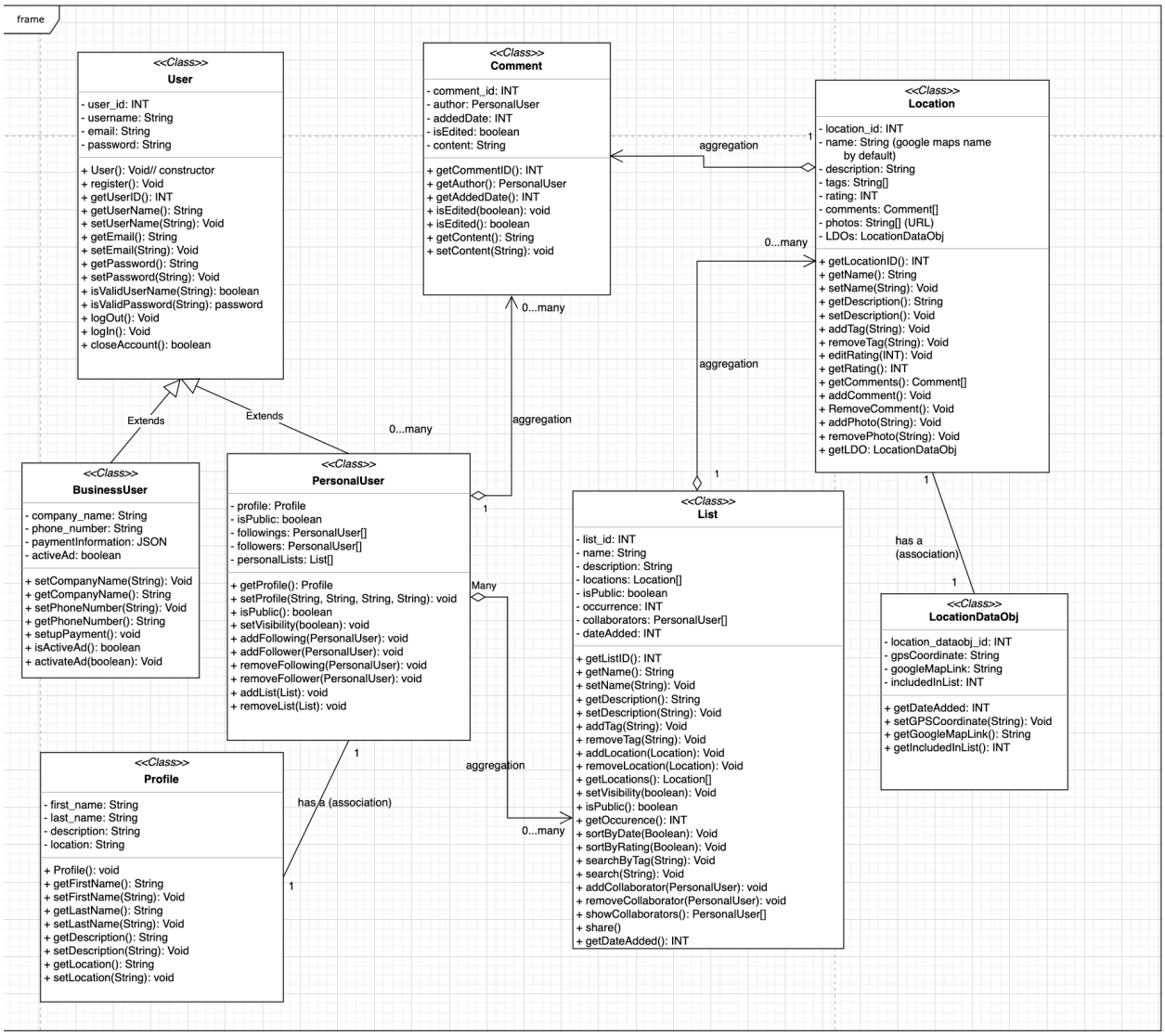
3. **Image processing API -** enables users to preview a location
    a. Functions:
        i. Posting - API will gather images relating to the location that is being previewed by the user

4. **Ad-management API -** Enables programmers to incorporate advertising features into their applications, enabling them to monetize their programs through the display of ads.
    a. Functions:
        i. Ad Creation - The API offers capabilities for defining ad budgets, ad targeting, and ad creatives as well as for managing ad campaigns.
        ii. Ad placement - The API offers resources for including native advertisements, interstitial ads, and banner ads within an application.

5. **Lists API -** this API allows users to create, update, and delete lists of locations. Users could add locations to their lists, remove locations, and modify their lists as needed.

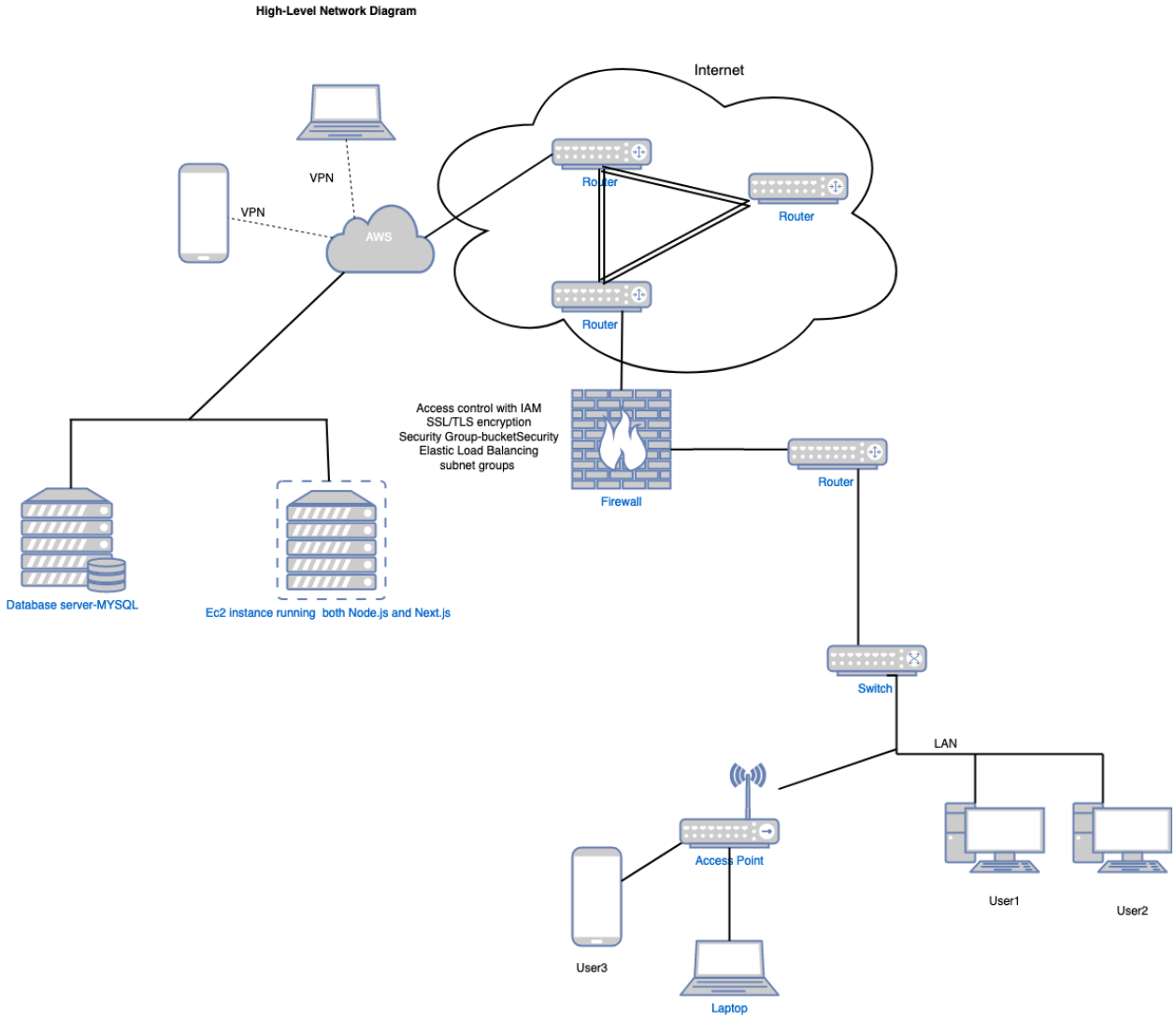
6. **Locations API-** The locations API would allow users to search for locations within their lists. Users could search for locations by name, address, or other criteria, and could view details about each location, such as its address, phone number, and website.

6. High Level UML Diagrams

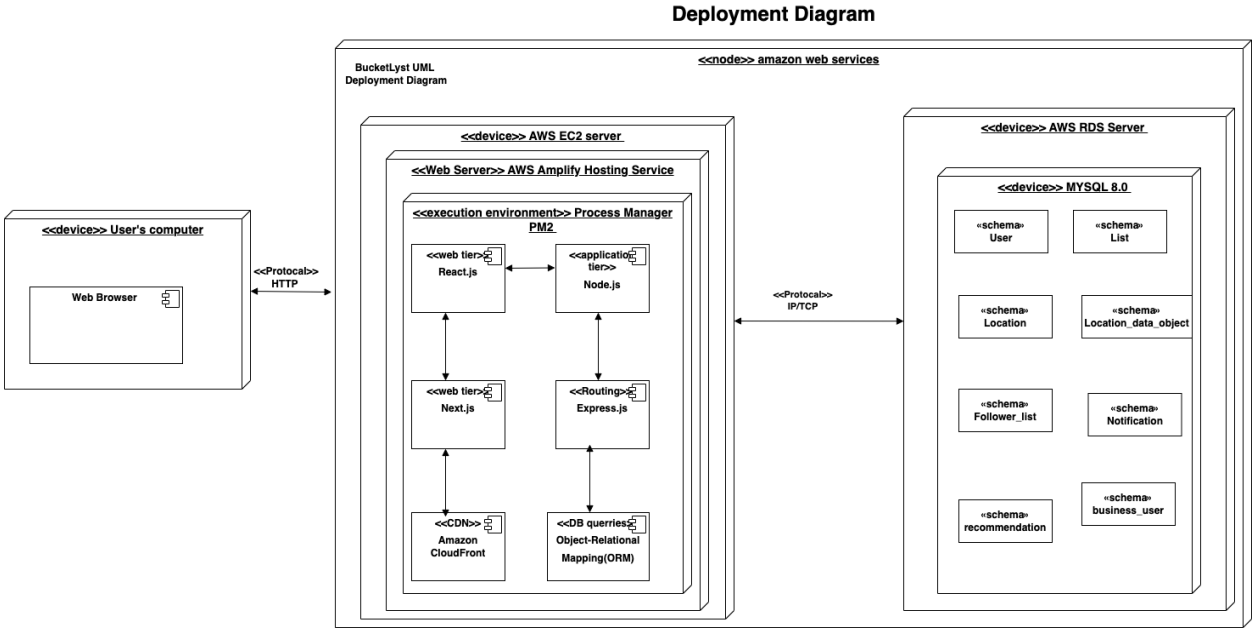


7. High Level Application Network and Deployment Diagrams

1. High-level Network Diagram



2. Deployment Diagram



8. Key Risks

1. **Skills**

   Risk : While everyone has some familiarity to their respective roles, there remains some doubt behind our technologies. AWS Amplify, which we use for web hosting, requires a lot of understanding on how to integrate the back end, front end, and database. React.js, Express.js, REST API, MySQL requires integration with Amplify and each other, and can cause delays with product deployment and CI/CD.

   Proposed Solution: The strongest resource available is the documentation provided by AWS Amplify and tutorials revolving integration. We can also mitigate delays by checking in not only within our own team, but across other teams.

2. **Schedule**

   Risk: We currently have a schedule risk. Halfway through March, I will be out of town until the 14th and for all of spring break, Aaron will be out of the country. This cuts into our front end team workforce.

   Proposed Solution: To address this issue, the front end team will work as much as we can before and after the spring break. In the meantime, we will make homework for ourselves to finish up during spring break and minimize communication with Aaron to only necessities.

3. **Technical**

   Risk 1 - As all our web hosting is through an outside company, we may run into the issue of an unexpected server outage. Also, after 12 months we would have to look into better priced alternatives.

   Proposed Solution 1 - If our product is impeded by any unexpected delay of service from our third party services, we can look into implementing backup systems in case of outage. We will also look into pricing and options as early as 6 months away from the server migration deadline.

   Risk 2 - The node-modules folder is a necessary component to allow a build to run locally on each of our individual machines. However, the folder itself takes up a large amount of space which if uploaded on the repo, increasing loading times when working with git, which is why I leave out the folder in a GIT IGNORE folder. However, the code will not run without this folder.

Proposed Solution 2 - When a group member pulls into their machine the code for our website for the first time, they should run ‘npm install’, before doing ‘npm start’ so the node-modules folder will be installed locally on the machines without it needing to be on Github.

4. **Teamwork**

Risk - Without a concrete grasp of the amount of work needed, the amount of time it would take, and the resources needed from different teams, it would be difficult to visualize and adhere to a concrete schedule. Delays in product deployment may be caused by slow communication and improper time management.

Proposed Solution - Atlassian Jira can indicate severity and urgency. Both tags can be used to visualize which team’s task might be backing up our workflow. Additionally, clear and constant lines of communication will reduce the amount of confusion caused by interteam communication.

5. **Legal/Content**

Risk - As we would produce all of our design and digital media, our issue of legality might lie in user-uploaded media and business user advertisements. Also, COPPA and other international laws require strict regulation of content.

Proposed Solution - We would require users to confirm that their age is above 13 and accept the Acceptable User Policy and implement a content moderation system for user uploaded images and created content. Also, we can dive deeper into this with expertise from a dedicated legal team.

9. Project management

We use Jira Board to manage our team's tasks, visually displaying individual and group tasks. Marie assigns tasks to our team, while Francis updates our Jira sprints. Each assignment contains direct links to its respective portion of the Milestone 2 document. In Jira, team members change the status of a task from "to do" to "in progress" and finally "done." In addition to individual tasks, team-based tasks are allocated by the team leads. The lead or assignee is responsible for adding tasks to the Jira board. As group assignments are not a free feature, we assign the task to one person and include other team members in the comment section. To improve our process in the future, we'll include team reminders to update Jira tasks and set internal deadlines for feedback and corrections if needed. Outside of Jira, we have mandatory weekly Zoom meetings, recorded in case anyone misses, and group meetings after lectures. We supplement these meetings with a weekly agenda for notes and action items. We use Discord, text messages, or phone calls for communication, reserving voting as a group for Google Forms or Google Sheets.

10. Team Contribution

Name	Role	Contribution
Arielle Riray 10/10	Frontend Lead	- Provided general layout for the UI mockups for frontend team to follow - Made a third of the Use Case Storyboard Mockups (8,9,10) - Helped assisted with revising the database design and initalization - Implemented the majority of the Front End for theVertical prototype - Imported Milestone 1 About Page for this Milestone - Created the hotbar and its respective routes using react-dom-router - Made/Designed full Log in/ Membership Form page - Mapped out list/user data items to be displayed on the home page - Created search bar with parameters for mapped items
Francis Quang 9/10	Backend Lead	- Finished first UML Diagram with Aaron - Created two API in amplify app - Connect amplify backend to frontend - Installed amplify cli into project - Finished High Level API with Kenny
Tommy Truong 10/10	Document Editor	- Wrote project management - Contributed to Network Diagram - Document editor - Made use case mockups - Contributed to front end web page
Kenneth Gee 8/10	Backend Engineer	- Reviewed Network Diagram and Deployment Diagram - Contributed to High Level API and Main Algorithms
Rabin Karki 10/10	Database Master	- Finished Network Diagram and Deployment Diagram. - Worked on DB design
Aaron Kuo 10/10	Github Master/ Frontend Engineer	- Worked on first draft of UML diagram with Francis - Worked on risks - Worked on search bar - Worked on vertical prototype backend - Implement multiple people approval system on github pull request