Date

October 21, 2024

Coffee Spot

Team 3 Members

Miguel Antonio Logarta - Lead |

Alan Yu - Scrum Master | Diane Bilse - Front End |

Halia Tavares - Front End | John Bagwell - Back End |

Su Tun (Emily Su) - Git Master | Timmy Tram - Back End |

Milestone 2

Architecture & UI,

Mock-ups and

Vertical SW

Prototype

SW Engineering CSC648/848 - Section 01

History	Date
V0.1	10.11.2024
V0.2	**10.16.2024**

1. Data Definitions V2

Below are some of our core data definitions that will be used in our app. Some are database tables, while others are API responses and data structures.

<table>
<tr><th colspan="2">User</th></tr>
<tr><td colspan="2">Description: representation of a user who uses our web app.</td></tr>
<tr><td>Members</td><td>Purpose</td></tr>
<tr><td><b>id</b>: ObjectId</td><td>A unique Mongodb identifier used to distinguish each user.</td></tr>
<tr><td><b>username</b>: string</td><td>A unique username assigned to each user. The username is the identity shown to every other user on the website’s platform.</td></tr>
<tr><td><b>email</b>: string, optional</td><td>User’s email address. It is optional for those who would like an option to recover their account using their email address. It is also used by Coffee Spot to send promotions, recommendations, and general notifications.</td></tr>
<tr><td><b>password</b>: string</td><td>Encrypted and hashed; the primary method for users to authenticate their identity.</td></tr>
<tr><td><b>role</b>: enum | bitfield</td><td>Defines user permissions. Regular users can post reviews, while Admin users have the additional ability to moderate content.

Defines user permissions. Certain roles grant certain users specific types of privileges. Here are the list of roles:

Customer: Ability to search for locations and leave reviews on those locations.
BusinessOwner: Ability to edit information of their own businesses and view analytics.
Moderator: Ability to edit and delete content that violates user guidelines.
Admin: Full control of the website.</td></tr>
<tr><td><b>creationDate</b>: DateTime</td><td>The date and time when the user account was created.</td></tr>
<tr><td><b>preferences:</b> Object[]</td><td>User preferences in regards to what type of third space</td></tr>
</table>

	they typically prefer.
settings: UserSettings[]	User settings in regards to how they would like their app to function. Includes color preferences, and notification settings.

Location	
Description: a physical place, business, or establishment.	
Members	Purpose
id: ObjectId	A unique Mongodb identifier used to distinguish each location.
name: string	The name of the location/business.
address: string	The address of the location/business. Validation will be needed before setting the address to make sure that it actually exists.
phoneNumber: string, optional	An optional phone number to contact the business. Currently we don't know if we should allow only one number or multiple numbers.
hasWifi: boolean	Check if the location has available wifi. In the future, we might want to change it to enum for values: PUBLIC | PASSWORD_PROTECTED | PRIVATE wifi
seatingCapacity: integer, optional	The amount of seats available at a location. This is useful if the user wants to sit down and study.
category: enum	The category of the location. Current categories include: cafe, library, park
rating: float	The average rating of a location.
busynessStatus: float	The current average busyness of a location. Values that determine busyness has not been decided yet.

	Busyness status types: NOT BUSY AVERAGE A LITTLE BUSY BUSY In the future we might implement a different structure that stores the average busyness throughout the day while also keeping track of the current busyness.
imageWebLink: string	Thumbnail image link to the business (.png, .jpg, jpeg)
locationWebsiteLink: string, optional	Link to the business's website
animalFriendliness: boolean	Determines if pets are allowed in or not into the establishment.
reviews: Review[]	Reviews from users about the location
operatingHours: OperatingHours[]	Operating hours of an establishment. Array structure stores days of the week with the OperatingHours storing the day and hours the business is open.
bookmarks: Bookmark[]	Bookmarks that point to this location. Bookmarks are necessary so that when a location is deleted, the bookmarks to that location are also deleted.

Bookmark	
Description: A user's saved location. A user can have many bookmarks. A bookmark corresponds to one location.	
Members	Purpose
id: ObjectId	A unique Mongodb identifier used to distinguish each bookmark.
locationId: ObjectId	The id of the location associated with the bookmark
userId: ObjectId	The bookmark owned by the user

creationDate: DateTime	Time when user bookmarked the location

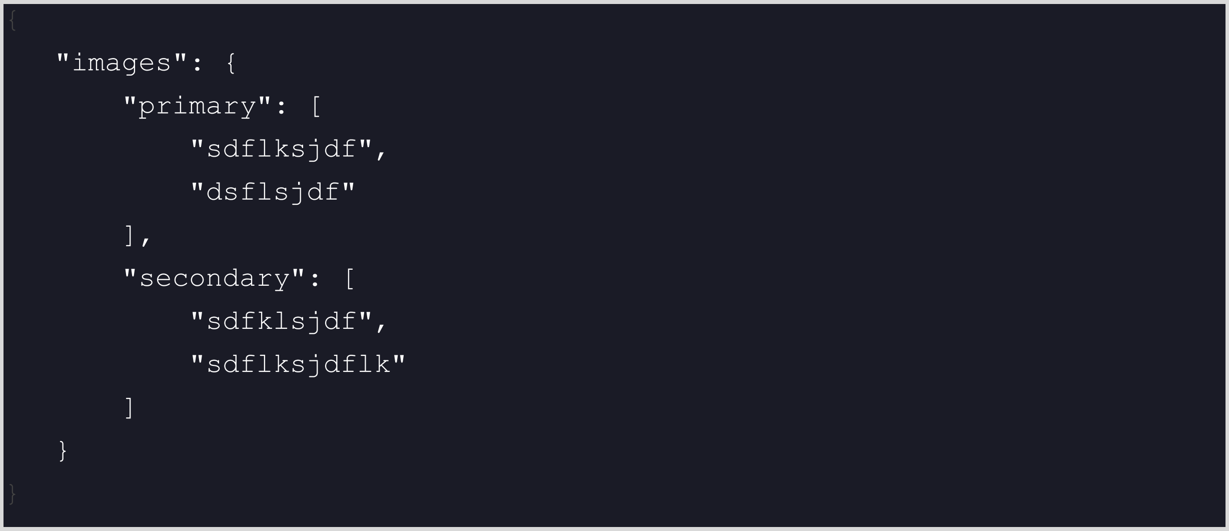
Time Slot (Old) -> Operating Hours	
Description: The operating hours of a business. A business can have at most 7 operating hours since that is the amount of days available in a week.	
Members	Purpose
id: ObjectID	Id of the operating hours object
day: string	Corresponding day
openingTime: string	Opening time of the business for that day.
closingTime: string	Closing time of the business for that day
locationId: ObjectID	Location/business that owns these operating hours.

Review	
Description: Representation of review detailing a user’s thoughts.	
Members	Purpose
id: ObjectId	A unique Mongodb identifier used to distinguish each Review.
locationId: ObjectId	The location associated with the Review
userId: ObjectId	The user who created the Review.
rating: float	The rating of the review. Range is from 0 to 5 inclusive. Prior checks will have to be implemented to make sure that the rating stays in this range.
content: string, optional	A more detailed review of the location. Contains what the user thinks of the place which can be read by other users.
images: string[]	A list of links to images that is included in the review. (.png, .jpg, jpeg)

creationDate: DateTime	Date of when the review was created.

Location Gallery

Description: Images related to Location. Images provided to the location can either be from the business (primary) or from the users (secondary). In the future, we can change the names into separate categories.



```
{
   "images": {
       "primary": [
           "sdflksjdf",
           "dsflsjdf"
       ],
       "secondary": [
           "sdfklsjdf",
           "sdflksjdflk"
       ]
   }
}
```

If we want to display images provided from the users, we may take those from the Reviews document and extract image links from each review.

Members	Purpose
id: ObjectId	A unique Mongodb identifier used to distinguish each bookmark.
locationId: ObjectId	The location associated with the images
images: Json	Images submitted by the business and its users. (.png, .jpg, jpeg)

Map

Description:

Members	Purpose
id: ObjectId	A unique Mongodb identifier used to distinguish each bookmark.

<table>
<tr><th colspan="2">UserSettings</th></tr>
<tr><td colspan="2">Description: 1 to 1 Relationships between User and UserSettings meaning each User gets one UserSettings dedicated to them.

Note: Technically possible to store UserSettings on client side in localstorage.</td></tr>
<tr><th>Members</th><th>Purpose</th></tr>
<tr><td>id: ObjectId</td><td>A unique Mongodb identifier used to distinguish each UserSetting</td></tr>
<tr><td>darkMode: Boolean</td><td>By default it is set to lightmode, user must manually set it to darkmode</td></tr>
<tr><td>notifications: boolean</td><td>By Default it is set off, the user must manually set it on.</td></tr>
</table>

<table>
<tr><th colspan="2">Filter</th></tr>
<tr><td colspan="2">Description: Parameters that will be passed to the backend to filter out map data. The user creates a filter to search up locations that they desire</td></tr>
<tr><th>Members</th><th>Purpose</th></tr>
<tr><td>Radius: integer</td><td>The desired search radius. Every location within this radius is included.</td></tr>
<tr><td>hasWifi: boolean</td><td>Filter out locations that have may or may not have wifi</td></tr>
<tr><td>Busyness: integer</td><td>Filter out locations based on how busy they are at the moment.</td></tr>
</table>

Permissions

- For our authentication system, we’re planning on going for a role-based system, however, not enough research has gone into our authentication architecture so the structure is still unknown

Recommendation Algorithm:

- We want to implement a recommendation algorithm. However, not enough research has gone into this topic for our team so the structure is still unknown.

2. Functional Requirements V2

#	Category	Feature Category	Feature Description	Priority & Tier
1	User Functions	User Registration	Ability for users to register with necessary information	Must Have (1)
2		User Login/Logout	User login and logout with password	Must Have (1)
3		Leave Reviews	Ability for users to leave reviews on locations	Must Have (1)
4		Add Location Status	Ability for users to add an instance of location status (e.g., busyness)	Desired (2)
5		Change Account Info	Ability for users to change account information such as password or email	Must Have (1)

6	Location Owner Functions	Set Location Images	Ability for location owners to set images for their location	Opportunistic (3)
7		Set Operating Hours	Ability to set operating hours (e.g., days open, opening and closing times)	Desired (2)
8		View Analytics	Ability to see analytics for location (clicks, busiest times, reviews)	Opportunistic (3)
9	Admin Functions	Add/Delete Locations	Admins can add or delete locations from the database	Must Have (1)
10		Moderation Content	Ability to delete and add content on the website for moderation	Opportunistic (3)
11		Promoted Locations	Ability to add locations to a 'promoted/sponsored' position in the list system	Opportunistic (3)

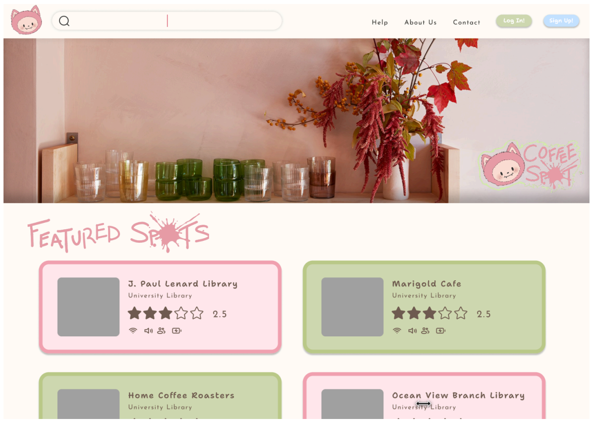
12	Map Functions	Map Pins	Ability to see locations on a map with clickable pins leading to the location's information	Must Have (1)
13		Location List View	Locations can also be displayed as a list, in addition to the map view	Must Have (1)
14	Location Information	See Ratings	Ability to see location ratings	Desired (2)
15		See Address	Ability to see the address of a location	Must Have (1)
16		Clickable Address	Clicking the address takes users to their preferred map application	Desired (2)
17		Accessibility Info	Ability to see general accessibility info (amenities, busyness, disabled-friendly, etc.)	Desired (2)
18	Search/Filter Function	Search by Name	Ability for users to search for a specific location by name	Desired (2)

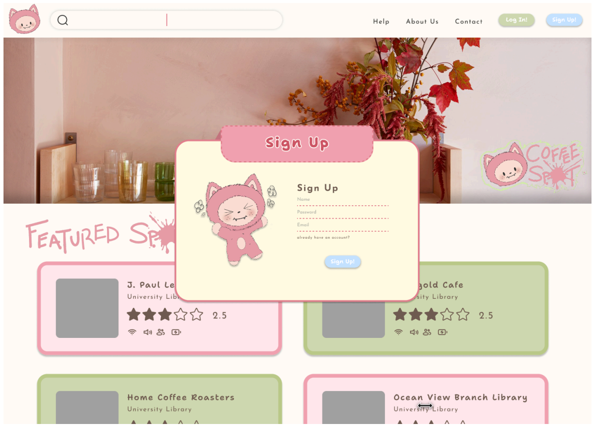
19		Search by Category	Ability for users to search/filter using a specific category	Desired (2)
20		Search by Attributes	Ability for users to search/filter by attributes like Wi-Fi	Desired (2)
21		Search by Any Attribute	OPTIONAL: Ability to search using any location attribute	Opportunistic (3)
22	Reservation Function	Reserve Spots	If allowed, users can reserve spots (e.g., study rooms)	Opportunistic (3)
23		External Links	Links that lead to the location's website for further information and features	Desired (2)
24	AWS Instance	File Storage	Ability to store large files (e.g., images, videos) on an S3 instance	Must Have (1)
25	Reviews	Leave Reviews	Ability for users to leave reviews on locations	Opportunistic (3)

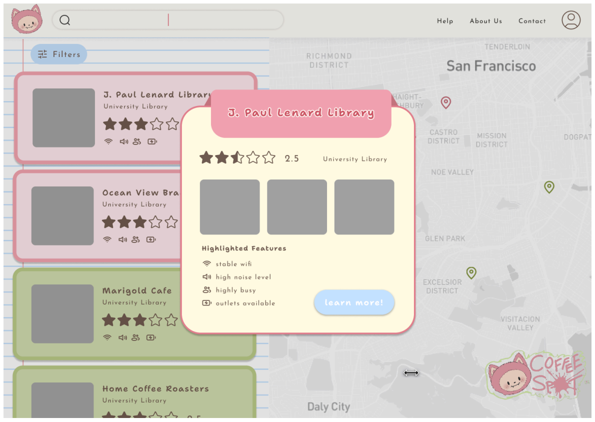
26	Recommendation Algorithm	Recommendation Algorithm	Recommendation algorithm for personalized suggestions	Desired (2)

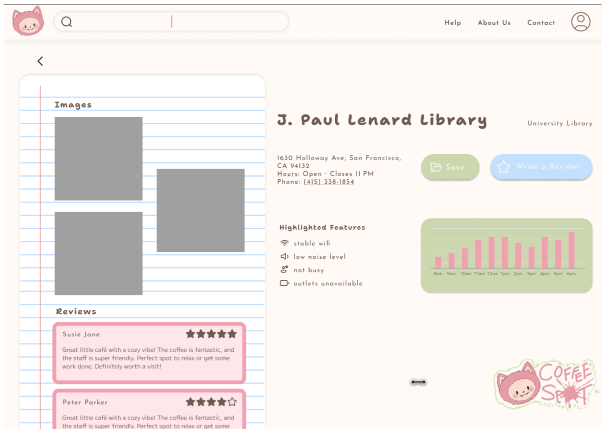
3. UI Mockups and UX Flows

Here are some of our designs:





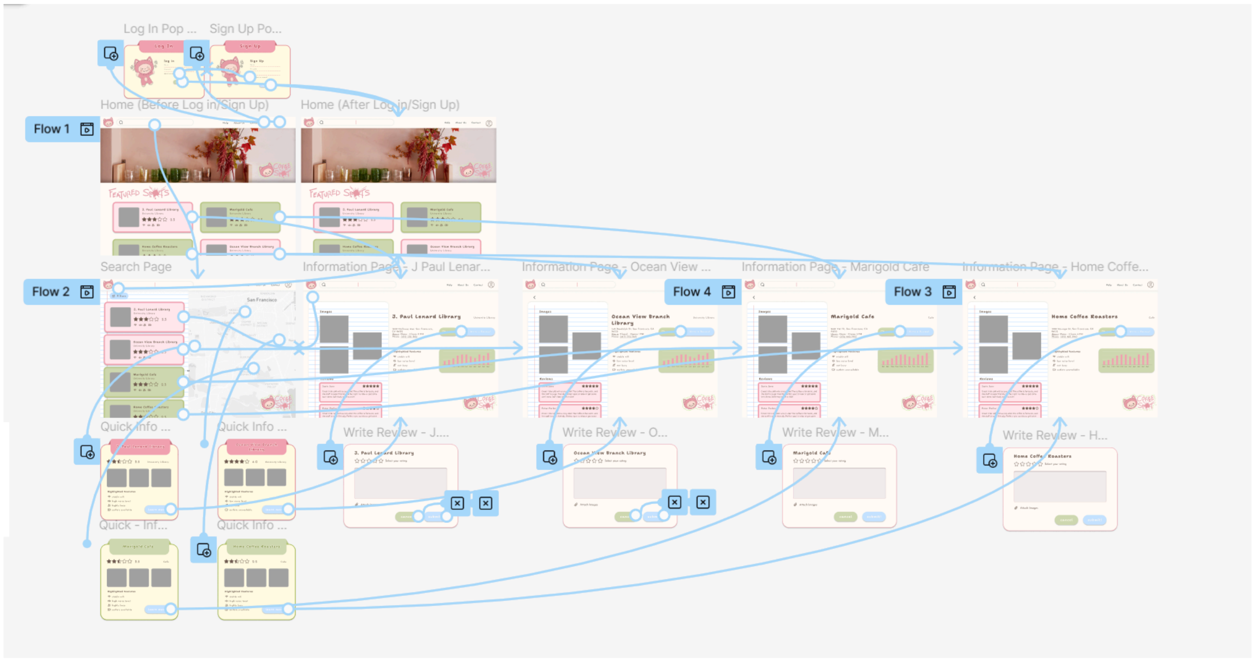




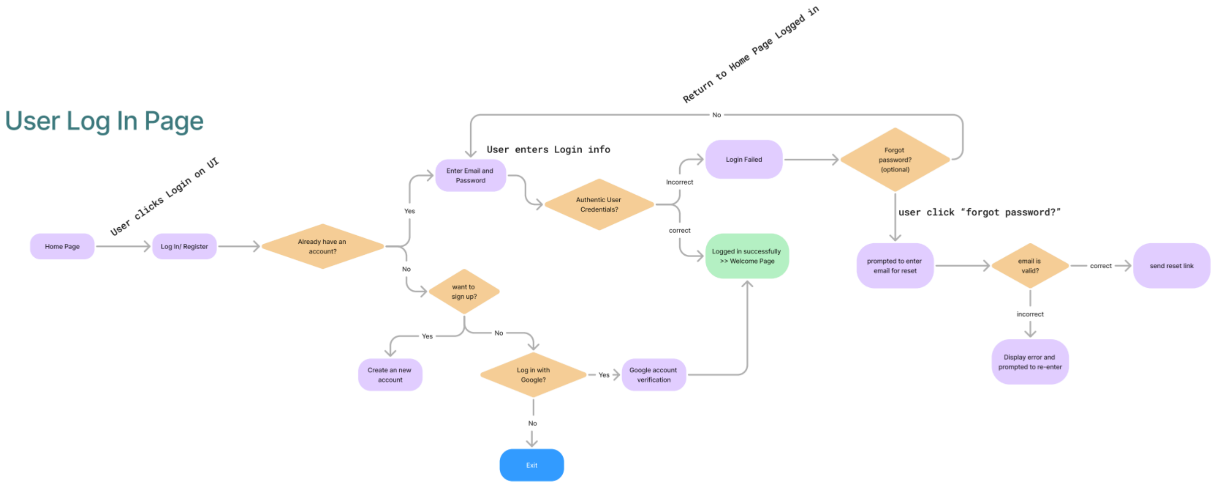
Great little café with a cozy vibe! The coffee is fantastic, and the staff is super friendly. Perfect spot to relax or get some work done. Definitely worth a visit!

Great little café with a cozy vibe! The coffee is fantastic, and the staff is super friendly. Perfect spot to relax or get some

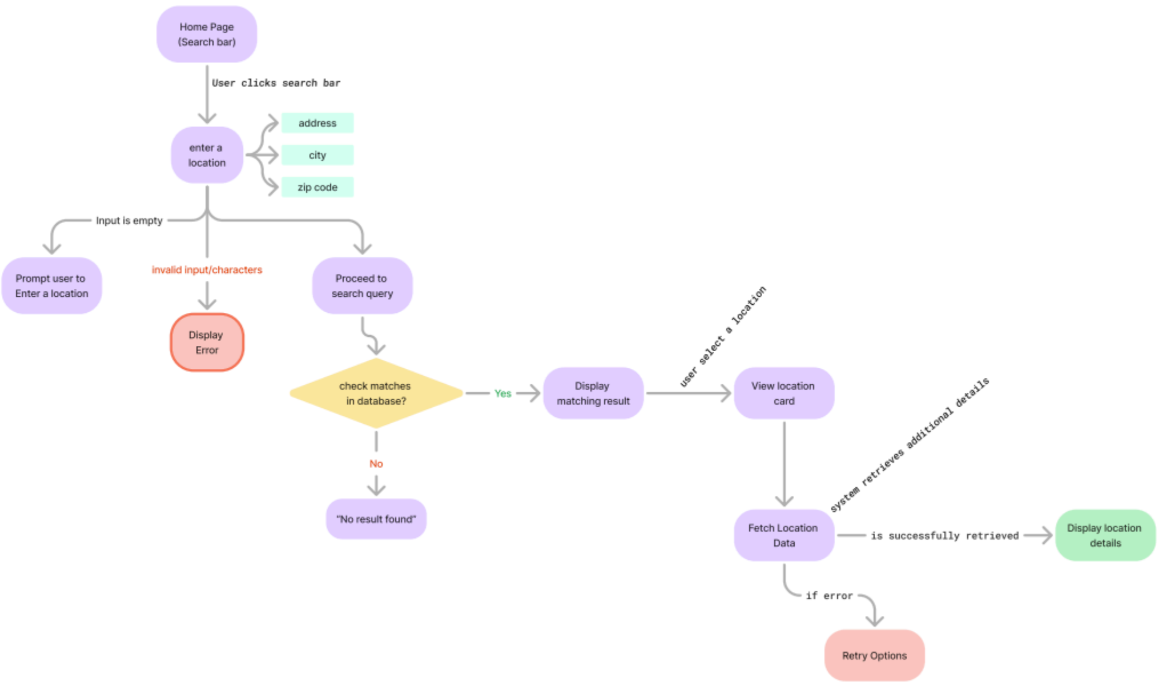
https://www.figma.com/proto/W6lwO3ys03oNp1Qqy3sTug/648-Flows-and-UI?node-id=198-1048&node-type=canvas&t=YwaAn3grgsUbNjbj-1&scaling=min-zoom&content-scaling=fixed&page-id=0%3A1&starting-point-node-id=1%3A17



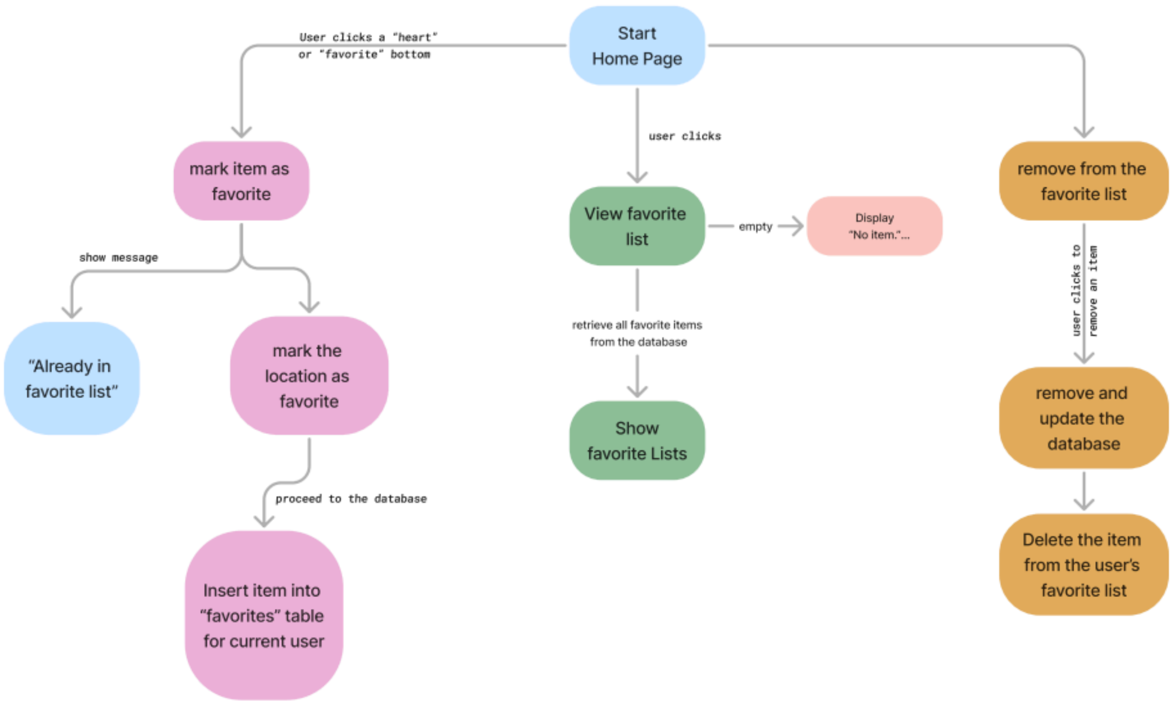
Here are our UX flows:



Location Search UX Flow



User Favorites List Flow



4. High Level Architecture, Database Organization

<table>
<tr><th colspan="2">User</th></tr>
<tr><td colspan="2"><b><u>Operations Permitted:</u></b>
CREATE User
PATCH User
DELETE User
SEARCH User
DISPLAY User</td></tr>
<tr><td>Fields</td><td>Data Types in Prisma ORM</td></tr>
<tr><td>id</td><td>String @id @default(auto()) @map("_id") @db.ObjectId</td></tr>
<tr><td>username</td><td>String @unique</td></tr>
</table>

email	String? @unique @default(“N/A”)
password	String
role	Role @default(CUSTOMER) // this is an enum
creationDate	DateTime @default(now())
preferences	LocationType[] // contains locations user prefers
reviews	Review[] // Prisma relationship
bookmarks	Bookmark[] // Prisma relationship
settings	UserSettings? @relation(fields: [settingsId], references: [id]) // Prisma relationship
settingsId	String? @unique @db.ObjectId // Prisma relationship

UserSettings

Operations Permitted:
PATCH UserSettings
DISPLAY UserSettings
Note:
(UserSettings with default values are created when a User is created.)

Fields	Data Types in Prisma ORM
id	String @id @default(auto()) @map("_id") @db.ObjectId
darkMode	Boolean @default(false)
notifications	Boolean @default(false)
user	User? @relation // Prisma relationship

Location

Operations Permitted:
CREATE Location
PATCH Location
DISPLAY Location
SEARCH Location
Note: seatingCapacity = -1 means we don’t know how many seats there are, frontend should handle displaying this properly.

Fields	Data Types in Prisma ORM
id	String @id @default(auto()) @map("_id") @db.ObjectId
name	String
address	String @unique
phoneNumber	String? @default(“N/A”)
hasWifi	Boolean
seatingCapacity	Int? @default(-1)
category	LocationType // this is an enum
rating	Float @default(0)
busynessStatus	Float @default(0)
imageWebLink	String? @default(“N/A”)
locationWebsiteLink	String? @default(“N/A”)
animalFriendliness	Boolean
reviews	Review[] // Prisma relationship
operatingHours	OperatingHours[] // Prisma relationship
bookmarks	Bookmark[] // Prisma relationship
gallery	LocationGallery?

OperatingHours

Operations Permitted:
DISPLAY OperatingHours
PATCH OperatingHours
Note: OperatingHours is created when a location is created.

Fields	Data Types in Prisma ORM
id	String @id @default(auto()) @map("_id") @db.ObjectId
day	DayOfWeek // this is an enum
openTime	String
closeTime	String
location	Location @relation(fields: [locationId], references: [id], onDelete: Cascade) // Prisma relationship
locationId	String @db.ObjectId // Prisma relationship

Reviews

Operations Permitted:
CREATE a Review
PATCH a Review
SEARCH a Review
DISPLAY a Review
DELETE a Review

Fields	Data Types in Prisma ORM
id	String @id @default(auto()) @map("_id") @db.ObjectId
rating	Float
content	String? @default(“”)
images	String[]
creationDate	DateTime(now())
location	Location @relation(fields: [locationId], references: [id], onDelete: Cascade) // Prisma relationship

locationId	String @unique @db.ObjectId // Prisma relationship
user	User @relation(fields: [userId], references: [id], onDelete: Cascade) // Prisma relationship
userId	String @unique @db.ObjectId // Prisma relationship

Bookmarks	
Operations Permitted: CREATE a Bookmark DELETE a Bookmark SEARCH a Bookmark DISPLAY a Bookmark	
Fields	Data Types in Prisma ORM
id	String @id @default(auto()) @map("_id") @db.ObjectId
userId	String @db.ObjectId // Prisma relationship
user	User @relation(fields: [userId], references: [id], onDelete: Cascade) // Prisma relationship
locationId	String @db.ObjectId // Prisma relationship
location	Location @relation(fields: [locationId], references: [id], onDelete: Cascade) // Prisma relationship
creationDate	DateTime @default(now())

LocationGallery	
Operations Permitted: ADD a Location Image DISPLAY a Location Image DELETE a Location Image	
Fields	Data Types in Prisma ORM

id	String @id @default(auto()) @map("_id") @db.ObjectId
images	Json?
locationId	String @unique @db.ObjectId // Prisma relationship
location	Location @relation(fields: [locationId], references: [id], onDelete: Cascade) // Prisma relationship

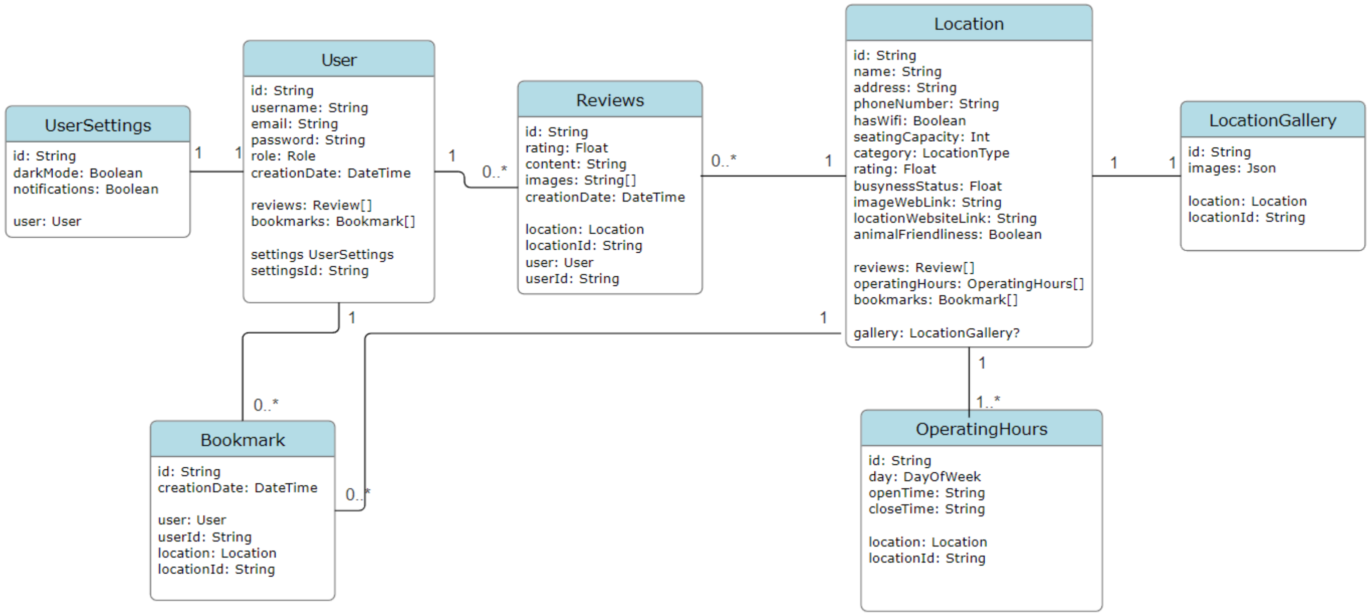
APIs

3rd Party API(s) include Google Maps API

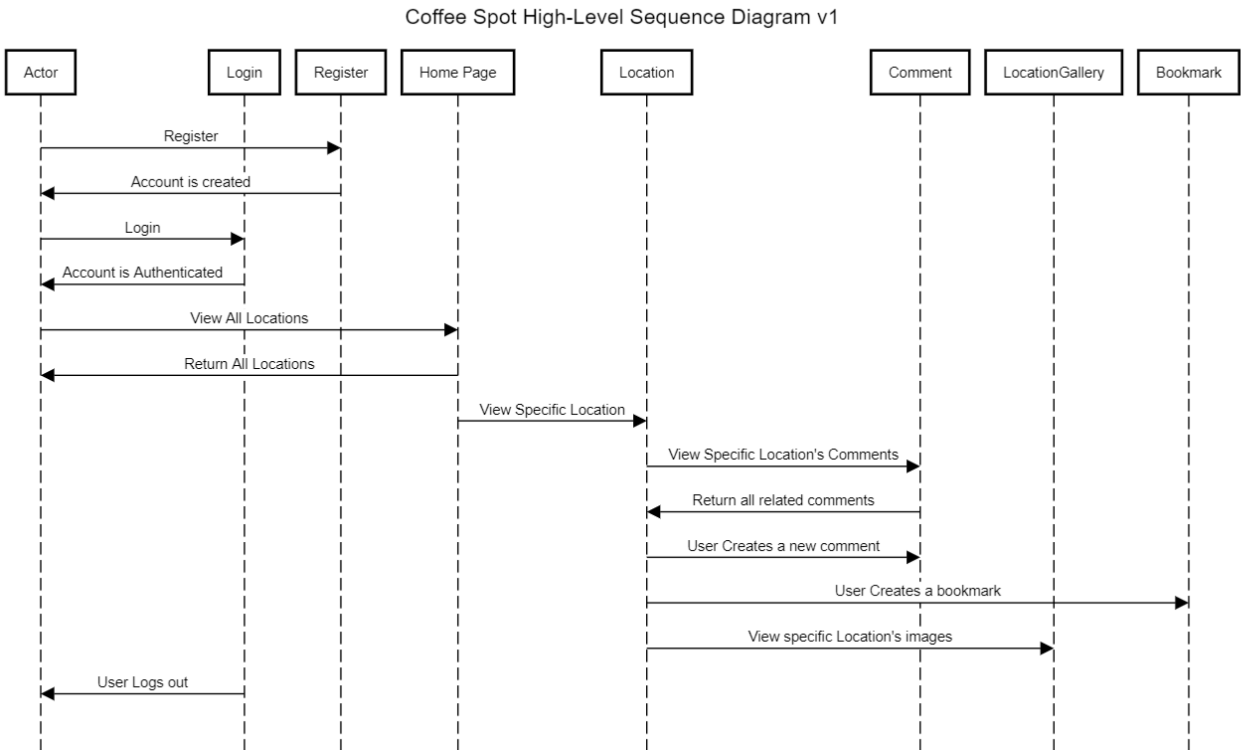
API Endpoints	
Backend Endpoints are defined as the url of the website appended with /api/	
Major Endpoints	Usage
/api/users	Route handles data regarding User and UserSettings
/api/locations	Route handles data regarding Location and Image Uploads
/api/reviews	Route handles data regarding Reviews and multiple Image Uploads
/api/bookmarks	Route handles data regarding Bookmarks
/api/auth	Route from nextauth to handle user authentication and authorization. Research is still needed to be done and implemented in a later milestone.

5. High Level UML Diagrams

High Level UML Class Diagrams



High Level UML Sequence Diagrams



6. Key Risks

Skills Risks:

Risk: Some team members lack technical knowledge on several technologies required for the project.

Mitigation: We have a study plan as well as discord channels with resources found by other team members to allow self study opportunities. We are also paired up in sub teams to accelerate skill development.

Schedule Risks:
Risk: Delays due to unexpected events

Mitigation: We have an agreed upon time with a flexible backup time selected if too many members cannot meet on the agreed upon day.

Risk: Shifting priorities some members have Jobs that unfortunately shift their focus from the project

Mitigation: Meetings are generally scheduled at a time that is unobtrusive to our working team members. With backup dates and times determined.

Teamwork Risks:
Risk: Inconsistent attendance in meetings and uneven task progress

Mitigation: The Team Lead is open to holding review/makeup sessions to keep team members that are falling behind up to date

Legal/Content Risks:
Risk: We are web scraping there is an opportunity for us to collect copyrighted data

Mitigation: Data that gets entered into the database should generally risk free. We have designated a team member to verify any images are not in some way restricted.

7. Project Management

Our team has been managing our tasks for Milestone 2 (M2) with a structured and collaborative approach. We have been using Trello as our primary tool for task management. Each team member has been assigned Trello Cards. We also have an open pool of cards with tasks for team members to assign to themselves as time permits. This platform enables us to monitor progress and receive real-time updates, while also allowing us to break the project into smaller, more manageable tasks.

During our scrum meetings, team members give a brief update on their assigned tasks, outlining what they have completed, what they are working on and any blocks they face. These updates are shared with the entire team allowing all of us to be aware of individual progress and make adjustments if needed. Additionally since the team was broken into smaller sub teams, each sub team created PowerPoint slides that compile and highlight the assigned tasks for that team. This simplifies communication and allows the Team Lead and Scrum Master to track progress.