Team 21 - Aryan Jumani, Annsh Navle, Pranav Bansal, Shriyan Bachigari

ANCHOR

Purpose

Location-specific information plays an important role in everyday life. However, people often share context-dependent details through generic platforms such as cloud drives or social media that are not aware of where the information is actually relevant. This leads to clutter, poor discoverability, and security risks when sensitive information such as Wi-Fi passwords, instructions, or local announcements is shared too broadly. There comes a need for a system that ensures information is only accessible at the right place and time.

The purpose of this project is to develop a location-based information sharing application called *Anchor*. Existing platforms, such as Google Maps, QR codes, cloud documents, or social media, allow for information distribution but do not enforce spatial or access restrictions. Anchor provides secure, location-based unlocking combined with group-based permissions, time-based expiration, and encrypted content storage. By verifying geographic proximity before granting access, Anchor ensures that digital content remains context-aware, controlled, and discoverable only to the intended audience.

Functional Requirements

1. **Authentication and Account Security**
    a. As a user, I would like to sign up and log in using email or OAuth so that my Anchors and permissions are securely associated with my identity.
    b. As a user, I would like to reset my password via email so that I can recover my account if I lose access.
    c. As a user, I would like to login using existing safe credentials on my device (e.g., FaceID, Touch ID, etc.). (if time allows)
    d. As a user, I would like to log out and revoke sessions so that my account remains secure across devices.
    e. Authentication tokens shall expire and be revocable.
2. **Identity and Profile**
    a. As a user, I would like to manage my identity by updating my username, email, avatar, and bio so that I am recognizable within Circles.
3. **Anchor Creation and Configuration**
    a. As a user, I would like to create an Anchor by dropping a pin on a map so that content is tied to a precise physical location.
    b. As a user, I would like to attach different content types (text, links, images, files) to an Anchor so that information is flexible and useful.
    c. As a user, I would like to configure an unlock radius (e.g., 10–100 meters) so that Anchors only unlock when users are physically nearby.

   d. As a user, I would like to set an Anchor's visibility to public, private, or Circle-based so that sensitive content is protected.
   e. As a user, I would like to set expiration parameters (time-based expiration or maximum "unlock count") so that the Anchor automatically disappears after a specific duration or once the maximum number of users have successfully unlocked it.
   f. As a user, I would like to edit or delete the Anchors I created so that outdated or incorrect information can be removed.
   g. As a user, I would like to preview how an Anchor will appear to others before publishing it so that I can verify correctness.
   h. As a user, I would like to create tags for Anchors so that people can get a brief idea of what the Anchor is about before clicking on it.
   i. As a user, I would like Anchors to activate only during specific time windows so that they align with real-world events.
   j. As a user, I would like to create unsavable Anchors (special tag) so that sensitive, personal information cannot be saved forever.

4. **Anchor Discovery**
   a. As a user, I would like to view nearby Anchors on a map so that I can discover relevant information around me.
   b. As a user, I would like Anchors to automatically unlock when I enter their radius so that I do not need to refresh or search manually.
   c. As a user, I would like Anchors outside my radius to remain hidden or blurred so that location enforcement is preserved.
   d. As a user, I would like to receive notifications when new Anchors appear nearby so that I do not miss relevant updates.
   e. As a user, I would like to filter based on tags, content type, expiration status, etc, so that the map view remains legible in dense areas.
   f. As a user, I would like to view a "List View" of the nearby Anchors so that I can browse content without solely relying on the map. (*if time allows).*
5. **Saving and Personal Library**
   a. As a user, I would like to save Anchors to a personal library so that I can reference that information forever without being in the same location and worrying about an expiration date.
   b. As a user, I should be able to differentiate between Expired and Non-Expired Saved Anchors in my personal library.
6. **Privacy Controls**
   a. As a user, I want to toggle "Ghost Mode" that stops tracking a user's location so that privacy can be maintained if wanted.
   b. As a user, I would like to block users so that I do not see their Anchors. (if time allows)
7. **Circles and Access Control**
   a. As a user, I would like to create Circles (groups) so that I can share Anchors with specific people only.
   b. As a user, I would like to invite or remove members from a Circle so that access control remains accurate.

   c. As a user, I want to see a “Global Search” for public Circles, so that a user can discover public groups.

8. **Social and Engagement**
   a. As a user, I would like to upvote/downvote anchors to provide a signal for the credibility/usefulness of an anchor. (if time allows)
   b. As a user, I want to see how many people have unlocked my Anchor so that social proof is established.
9. **Scavenger and Hunt Games**
   a. As a user, I would like to create “Scavenger hunts” by linking Anchors in a sequential process where unlocking one Anchor reveals the next clue. (if time allows)
   b. As a user, I would like to track my progress through a Scavenger hunt so that I know how many steps remain. (if time allows)
   c. As a user, I would like to view a leaderboard for hunts so that participation is engaging and competitive. (if time allows)
10. **Moderation and Administration**
    a. As a user, I would like to report inappropriate Anchors so that the platform remains safe.
    b. As an admin, I would like to review reported Anchors so that I can take moderation actions.
    c. As an admin, I would like to disable or remove abusive users so that misuse is prevented.
    d. As a system, I would like to maintain an audit log of critical actions so that abuse investigations are possible.
11. **AR Mode (Optional Enhancement)**
    a. As a user, I would like to view Anchors in an AR camera mode so that I can see distance and direction visually. (if time allows)
    b. As a user, I would like AR Anchors to update in real time as I move so that navigation feels accurate. (if time allows)

Non-Functional Requirements

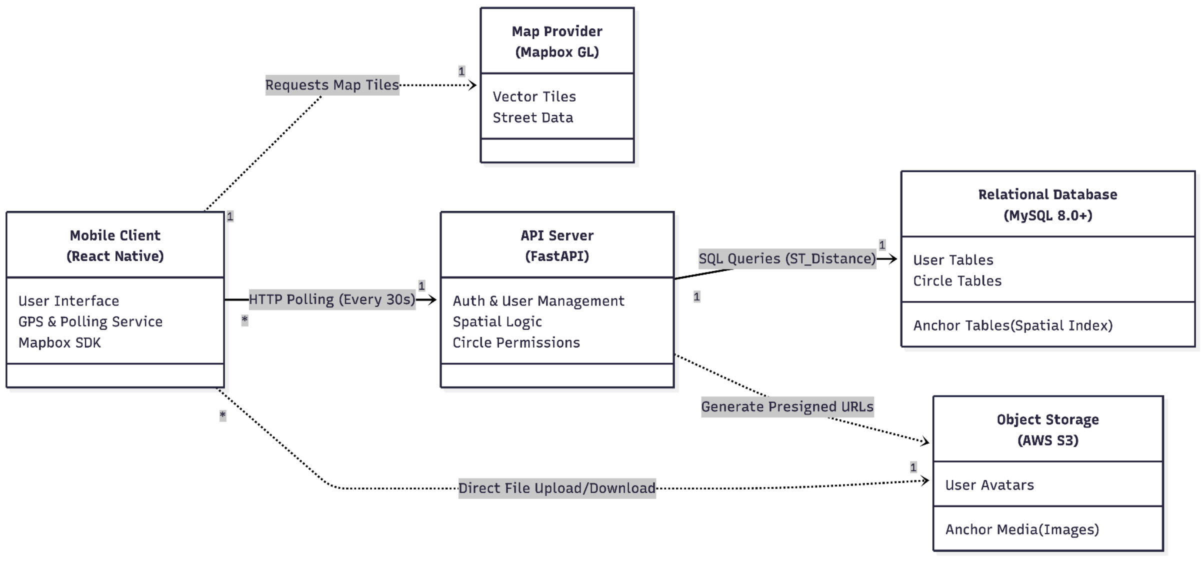
1. **Performance and Latency**
   a. The system shall unlock Anchors within 500 ms after a user enters the configured radius.
   b. Map and nearby Anchor queries shall respond within 1 second under normal load.
2. **Scalability and Load Handling**
   a. The backend shall support ~10,000 concurrent users without degradation.
   b. The system shall be horizontally scalable to handle increasing Anchor density in urban areas.
   c. The spatial backend shall support efficient geospatial queries using indexing.

3. **Availability and Reliability**
    a. The service shall be available 24/7 with at least 99% uptime.
    b. Anchors and permissions data shall be persisted reliably with automated backups.
4. **Security and Privacy**
    a. All Anchor content shall be encrypted at rest and in transit.
    b. Unauthorized users shall never receive decrypted Anchor content.
    c. Authentication tokens shall expire and be revocable.
    d. Location spoofing attempts shall be mitigated using consistency checks.
5. **Abuse Prevention**
    a. The system should rate-limit Anchor creation so that spam is minimized. (if time allows)
6. **Usability and UX**
    a. New users shall be able to create their first Anchor within ~2 minutes of onboarding.
    b. The UI shall be consistent across Android and iOS platforms.
    c. The app shall provide clear visual feedback for locked vs. unlocked Anchors.
7. **Architecture and Extensibility**
    a. The backend shall follow a modular FastAPI architecture.
    b. The system shall support future extensions such as AR HUDs and new Anchor types without major refactors.
8. **Offline and Caching**
    a. The app shall cache metadata for nearby Anchors to allow discovery in areas with poor connectivity. (if time allows)

Design Outline

High-Level Overview

Our project will be a mobile application that follows a Client-Server architecture designed to handle geospatial data and queries in real-time. The server will act as an intermediary between the clients and the data storage layers, enforcing location validation before allowing the client to view any content.

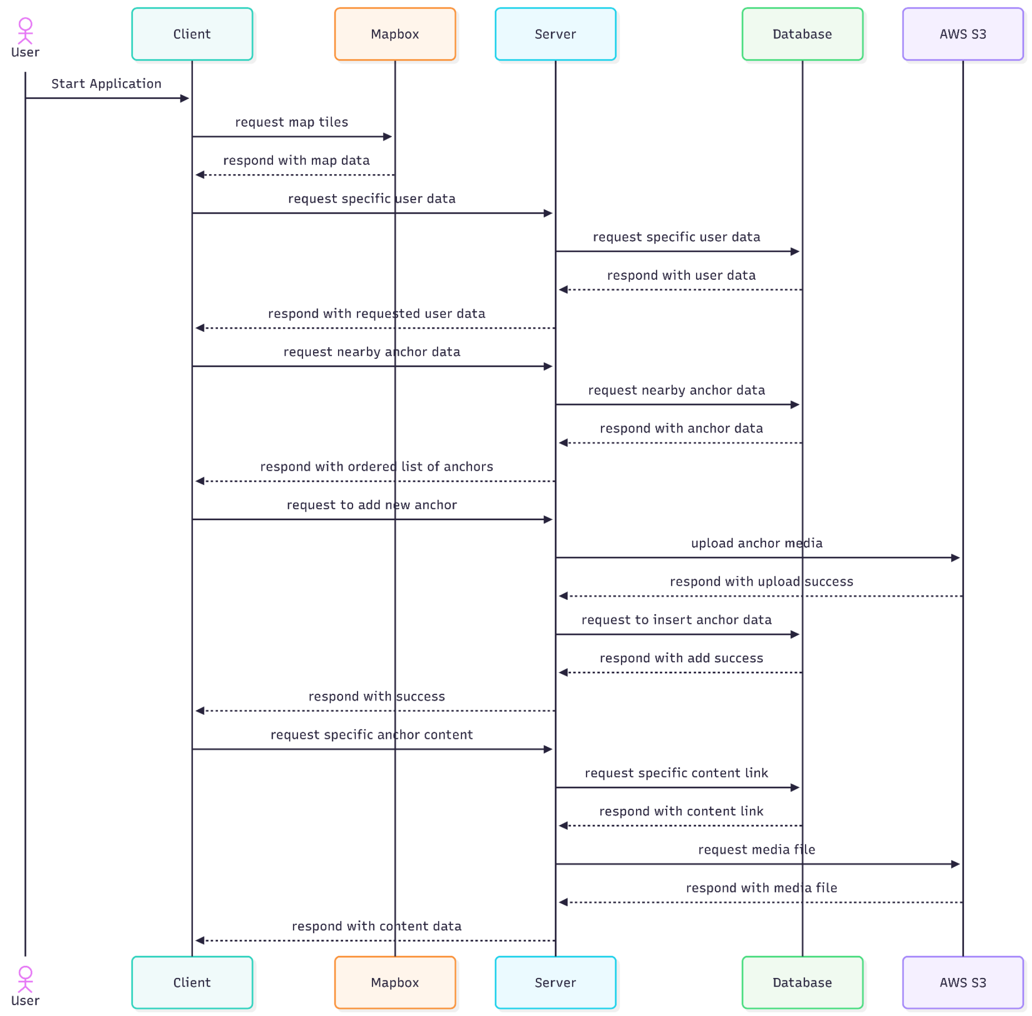


1. Mobile Client
    a. The client provides the mobile interface for the system (Android/iOS).
    b. The client captures the user's real-time GPS location and altitude and sends HTTP requests to the server to discover and unlock Anchors.
    c. The client interprets JSON responses from the server to render Anchors on a map and display unlocked media content to the user.
2. Server
    a. The server receives and processes API requests from the clients, such as creating a new Anchor, unlocking an Anchor, changing information, etc.
    b. The server executes the logic, calculates the distance between the Anchor and User, and validates the user.
    c. The server queries the database for metadata and generates access links for files stored in the Object Storage, sending this information back to the client using FastAPI.
3. Database
    a. A MySQL database stores all the structural data, including user information, Anchor metadata, and Circles.
    b. Stores references to the binary media located in the Object Storage,
    c. The database responds to queries from the server to retrieve the list of relevant Anchors for a specific geographic area.
4. Map Provider
    a. Mapbox provides the underlying map data and rendering engine for the mobile client.

   b. It displays the Anchors as interactive pins based on their coordinates.
5. Object Storage
   a. AWS S3 acts as the object storage system for the actual binary files.
   b. Delivers content to the Mobile Client via presigned URLs using the server.

Brief Sequence of Events

The sequence diagram below shows the typical interaction among the User, Client, Server, Database, Mapbox, and AWS S3. The sequence is initiated by a user starting the application, which triggers the client to request map tiles from Mapbox and user profile data from the server. As the user explores, the client sends requests to the server to discover nearby anchors. The server handles these requests by querying the database and returning an ordered list of available anchors to the client. Requests to create a new anchor will be sent to the server, which handles uploading the binary media to AWS S3 and inserting the metadata into the database. Upon success, the server returns a confirmation to the client. Requests by the client to unlock and view a specific anchor will be sent to the server. The server retrieves the specific media link from the database and AWS S3 and returns the content data to the client for display.


User
Client
Mapbox
Server
Database
AWS S3
Start Application
request map tiles
respond with map data
request specific user data
request specific user data
respond with user data
respond with requested user data
request nearby anchor data
request nearby anchor data
respond with anchor data
respond with ordered list of anchors
request to add new anchor
upload anchor media
respond with upload success
request to insert anchor data
respond with add success
respond with success
request specific anchor content
request specific content link
respond with content link
request media file
respond with media file
respond with content data
User
Client
Mapbox
Server
Database
AWS S3

Design Issues
Functional

1. Anchor discovery must be fast as users move; choice affects network usage, latency, and backend scalability.
    a. Option 1: Server query per map view/populate call- Client asks backend: "What anchors near X,Y?" on map open or user pan. Simple REST flow, scales well if backed by indexed geospatial queries, but periodic refresh required for movement.
    b. Option 2: Continued low-frequency polling - App sends periodic location updates to backend (e.g., every 30–60s) and server responds with new anchors. Better continuity than manual refresh but uses more network.
    c. Option 3: Server-push notifications of nearby anchors - Backend determines when user should see new anchors and pushes them (e.g., via push services or persistent connection). Reduces polling and latency but requires persistent connection logic.

    Best Choice: Continued low-frequency polling. Better continuity, should not cause network problems on our scale. Ensures seamless user experience while maintaining continuity without battery drain of constant, every second, GPS tracking.

2. Notification alerts must be timely but not overwhelming, balancing relevance and user retention
    a. Option 1: Immediate push at unlock radius entry - Send a push immediately when user enters an anchor's radius. Most relevant but can create alert fatigue if anchors are dense.
    b. Option 2: Scheduled summary alerts - Batch notifications into periodic summaries (e.g., every 10min). Reduces interruptions but sacrifices immediacy.
    c. Option 3: In-app badge / silent alert - Instead of push, update in-app badges or notification center without interruptive alerts. Good for heavy users; risks being unnoticed by casual users.

    Best Choice: Scheduled summary alerts. Don't bombard the user with multiple notifications. Also reduces alert fatigue by preventing multiple notifications in dense areas.

3. Saved Anchor semantics - what "save forever" actually means
    a. Option 1: Save a decrypted snapshot - When a user saves, store the full content in their library and allow offline access later. Best UX, but it conflicts with the spirit of "unsavable" unless you implement strict client controls and accept screenshot risk.
    b. Option 2: Save an encrypted snapshot bound to the user - Store encrypted content and only allow decryption for that user inside the app. Stronger control

than plaintext storage, but still cannot fully prevent exfiltration (screenshots, screen recording).

c. Option 3: Save a reference only (recheck permission on view) - Saved item is a pointer that requires server authorization at view time, and content can disappear if the anchor expires, is deleted, or access is revoked. Cleanest for "unsavable" and revocation semantics, but users will dislike "I saved it but it is gone."

Best Choice: Save a reference only. Lazily delete if the original Anchor doesn't exist. Also prevents the "ghost data" problem by lazily purging pointers from user libraries once the Anchor no longer exists.

4. Reporting and blocking, who can moderate
    a. Option 1: Report anchor only, admin reviews - Simple, but users still need a personal safety mechanism. App Store and Play policies commonly require reporting and blocking in UGC contexts.
    b. Option 2: Report anchor and block creator - Adds immediate personal protection: the reporter stops seeing content from that user right away, while admin reviews. This matches typical "report vs block" expectations described in platform guidance for UGC safety tools.
    c. Option 3: Circle-level moderators - Allow trusted circle owners or moderators to remove anchors or members inside that circle. Faster response for private communities, but increases abuse potential by moderators and adds governance complexity.

Best Choice: Report anchor only, admin review. Safer for admin to review, easy to review on our current scale. Ensures high quality content standards are maintained by us rather than the community. Minimizes risk of power abuse of "mods" or spam reports.

5. Pre-unlock visibility (what users can see before they are allowed to unlock)
    a. Option 1: Hidden until eligible - Anchors outside the user's allowed radius do not appear at all. Strong privacy and fewer "bait" pins, but weaker exploration and lower perceived density.
    b. Option 2: Blurred pins with minimal metadata - Show approximate pin location plus tag and type, but blur title and creator and block content. Good discovery while still enforcing access boundaries, but increases "this exists here" leakage.
    c. Option 3: Density heatmap only - Show an area heatmap of "anchor activity" without discrete pins. This preserves privacy better than pins, but is less actionable and can frustrate users trying to find the exact spot.

Best Choice: Density Heatmap. Faster than getting metadata for each anchor, still shows user Anchor density for farther locations.

Non-Functional

1. Efficient Geospatial Storage and Queries - we need to find all anchors discoverable by the user
    a. Option 1: MySQL 8.0+
    b. Option 2: MongoDB & GeoJSON
    c. Option 3: Spatial database (PostGIS)
    d. Option 4: Redis

Best Choice: MySQL 8.0+. This is the simplest to use and we have experience using it before. It is also cheaper. MySQL 8.0 supports native spatial indexes (e.g., R-trees on geometry columns), which are sufficient for radius-based geospatial queries at our expected scale without introducing architectural complexity.

2. File storage - storing images and files for users
    a. Option 1: AWS S3
    b. Option 2: Google Cloud Storage
    c. Option 3: Store in MySQL/PostgreSQL database as binary data itself

Best Choice: AWS S3. Storing binary data could be insecure and we have used S3 buckets before. S3 provides high durability (11 9's), built-in scalability, and secure pre-signed URLs, which cleanly separates file storage from application servers.

3. Mobile Development Framework - consistent UI across Android and iOS
    a. Option 1: Flutter
    b. Option 2: React Native

Best Choice: React Native. We have more experience in using React Native over Flutter. React Native allows shared JavaScript logic and a single codebase while still enabling native modules for AR and geolocation features when needed.

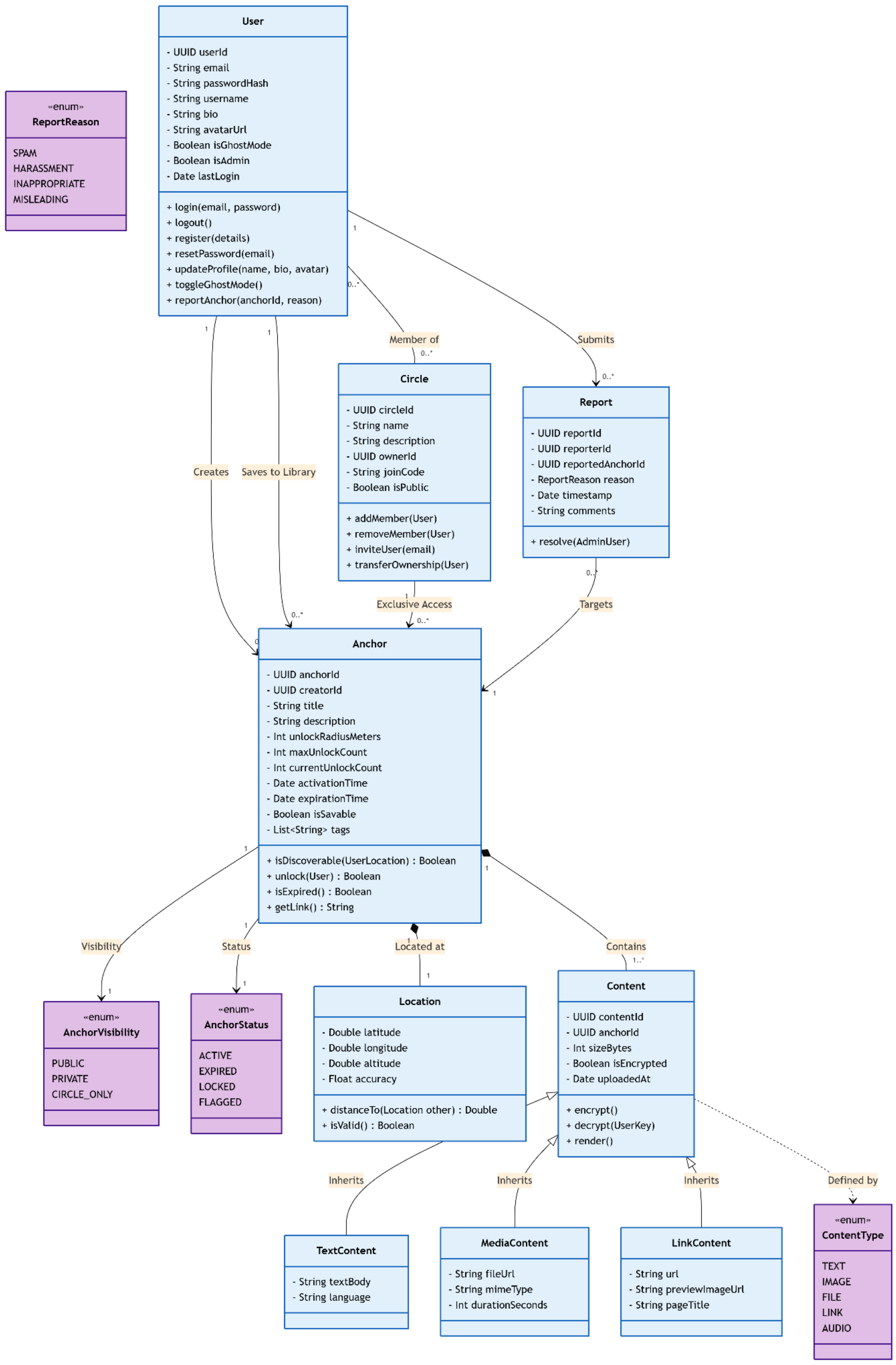
4. Map Data Provided - we need detailed street-level maps
    a. Option 1: Google Maps SDK
    b. Option 2: Mapbox GL
    c. Option 3: OSM

Best Choice: Mapbox GL. Customizable with good accuracy and cheaper to use than Google Maps. Mapbox provides fine-grained styling control and efficient vector tile rendering, which is important for displaying dynamic Anchor overlays at scale.

5. Hosting infrastructure - scalable and available 24/7
    a. Option 1: AWS EC2
    b. Option 2: Heroku
    c. Option 3: Render
    d. Option 4: AWS Lambda - serverless, infinite scaling, cheap, slow
    e. Option 5: GCP - serverless, infinite scaling, cheap, slow

Best Choice: AWS EC2/Render. Simplest to use, prior experience. EC2 or Render provide predictable performance, easier debugging, and long-running backend support, which is better suited for geospatial queries and real-time unlock logic than cold-start-prone serverless functions.

Class Level Design



Flow Design

The class design represents the entities required to enforce location-based access control and group management

1. User

    Creation: Represents a registered identity in the system. Created when a user signs up via email or OAuth.
    Attributes: Stores details like authentication (`email, passwordHash`), profile information (`username, bio, avatarURL`), and privacy settings (`isGhostMode`).
    Interactions: Can create Anchors, join Circles, save Anchors to their library, and submit reports.
    Ghost Mode: The `isGhostMode` boolean determines if the user's location is broadcasted to others or processed just for discovery.

2. Location

    Creation: Instantiated when a User's device reports position or when an Anchor is placed on the map.

    Attributes: Stores geospatial information like the latitude, longitude, altitude, and accuracy.

    Key Logic: The `distanceTo()` method function calculates the distance between the User's current location and the Anchor's target location to determine if the User has access to the Anchor or not.

3. Anchor

    Creation: Created when a User drops an Anchor to attach some form of content to a physical location.

    Attributes: Stores the rules under which an Anchor unlocks itself, including `unlockRadiusMeters, maxUnlockCount, activationTime`, and `expirationTime`.

    Status & Visibility: Uses enums like `AnchorVisibility` and `AnchorStatus` to determine if it is discoverable and if it is currently valid.

    Interactions: It aggregates Location and Content. The `unlock()` method validates User proximity and permissions before releasing decrypted content.

4. Content

    Creation: Created immediately after an Anchor is defined.

    Attributes: Metadata regarding file size, upload time, and encryption status.

Security: Handles the `encrypt()` and `decrypt()` logic to make sure data is unreadable by any malicious user until the Anchor is successfully unlocked by a verified user.

This is a base class extended by specific content types:

i. TextContent: Stores raw strings and language metadata.
ii. MediaContent: Stores URLs to images, audios, and other file formats within our storage, as well as duration.
iii. LinkContent: Uses external URLs.

5. Circle

Creation: Created by a User to form a group for private sharing.

Attributes: Contains identity details (name, description) and access controls (`joinCode, isPublic`).

Interactions: Permissions for Anchor usage. If an Anchor's visibility is set to CIRCLE_ONLY, only Users linked to this Circle object can unlock it. If an Anchor's visibility is set to PRIVATE, only that User can unlock it. If an Anchor's visibility is set to, everyone can access it.

6. Report

Creation: Generated when a User flags an Anchor for containing potentially harmful content.

Attributes: Links the reporter, the target Anchor, the `ReportReason` (Spam, Harassment, etc.), and optional comments.

Interactions: Reviewed by an Admin user via the `resolve()` method.

7. Enums

AnchorVisibility: Defines access scope for Anchors (PUBLIC, PRIVATE, CIRCLE_ONLY).

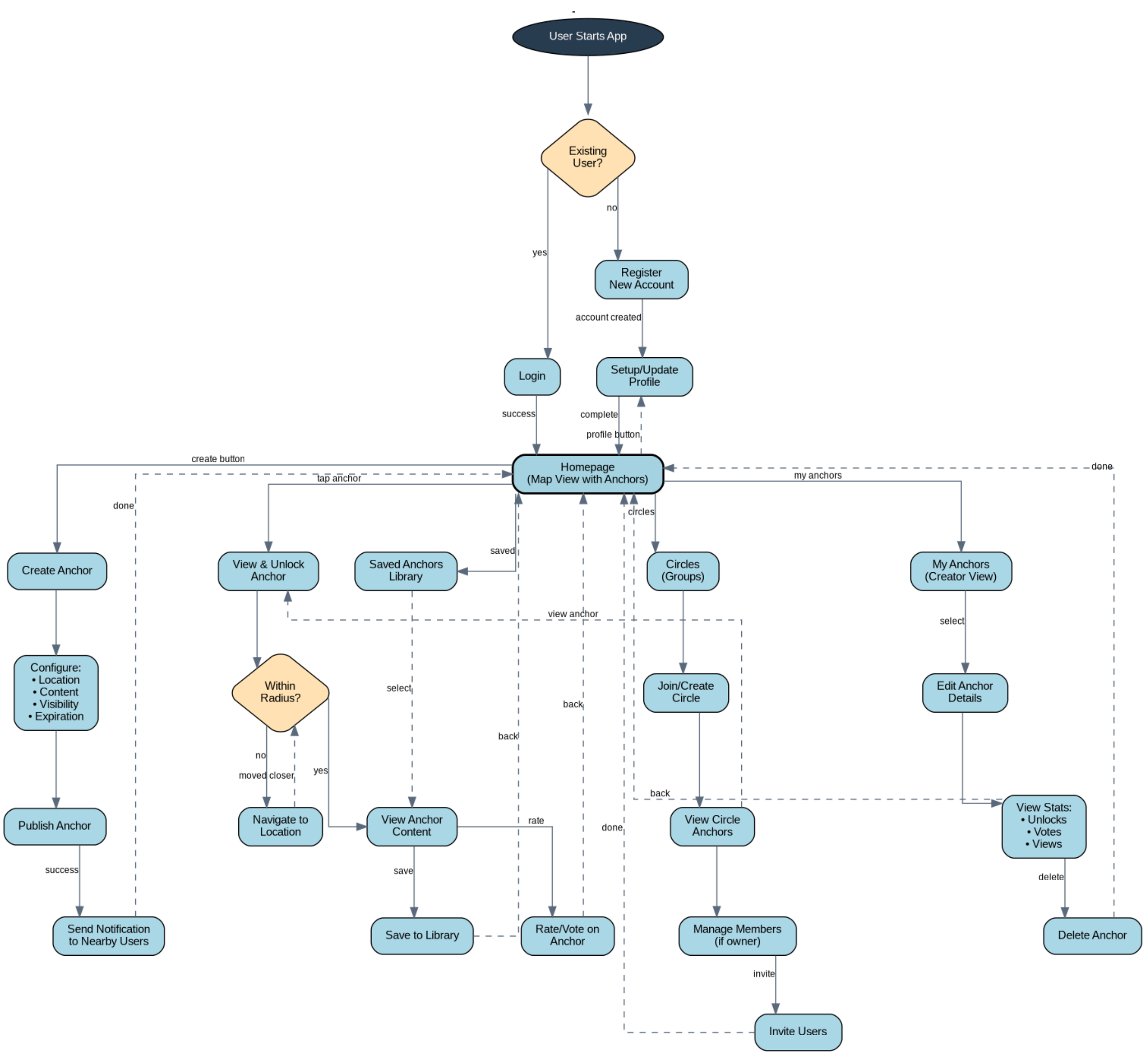
AnchorStatus: Manages the lifecycle for Anchors (ACTIVE, EXPIRED, LOCKED, FLAGGED).

ContentType: Categories for content attached to Anchors (TEXT, IMAGE, AUDIO, FILE, LINK).

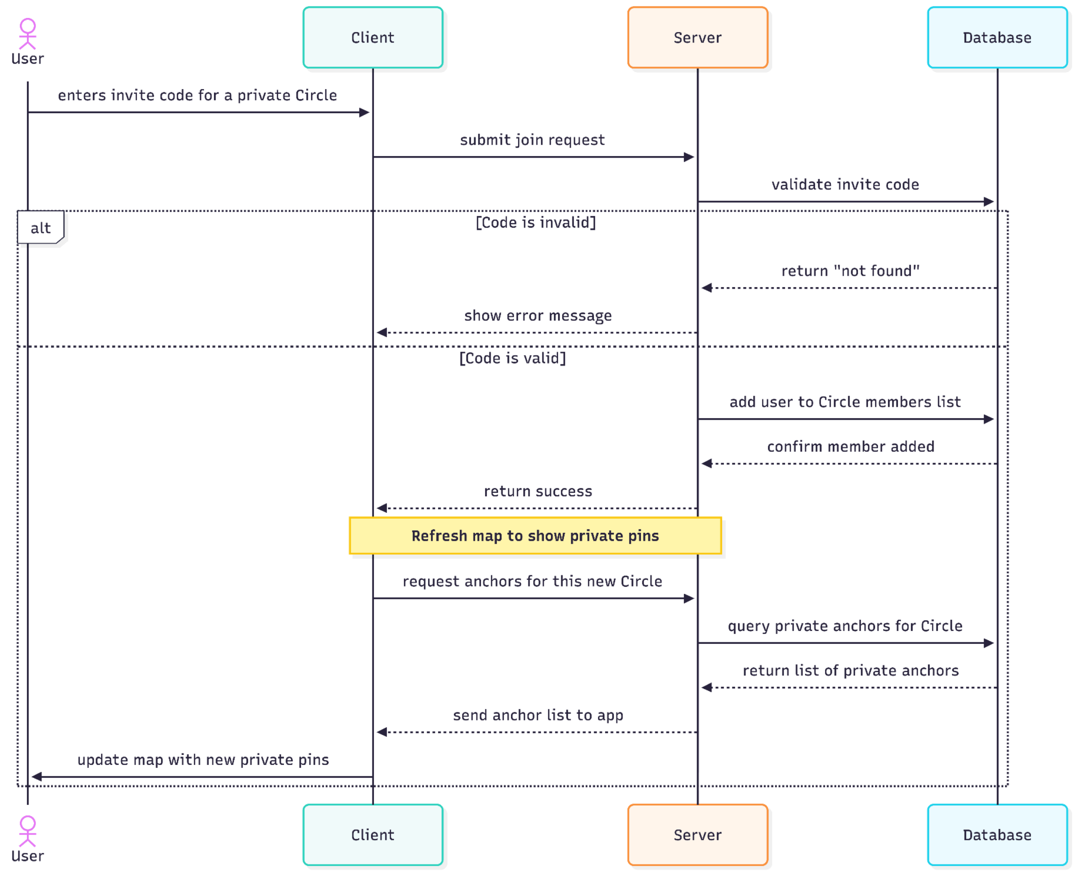
ReportReason: Categories for flagging Anchors (SPAM, HARASSMENT, INAPPROPRIATE, MISLEADING).

Team 21 - Aryan Jumani, Annsh Navle, Pranav Bansal, Shriyan Bachigari

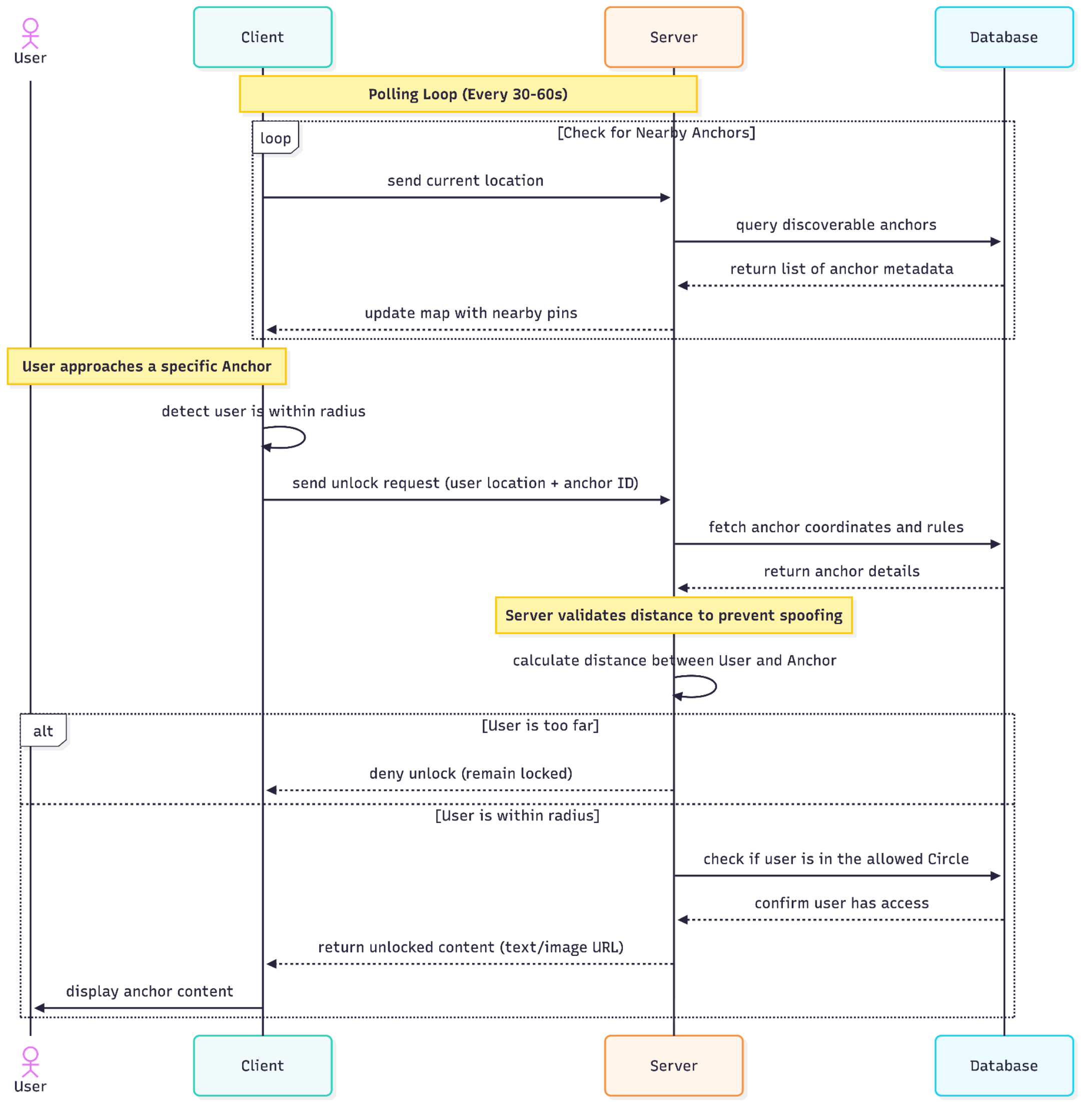
Navigation Flow



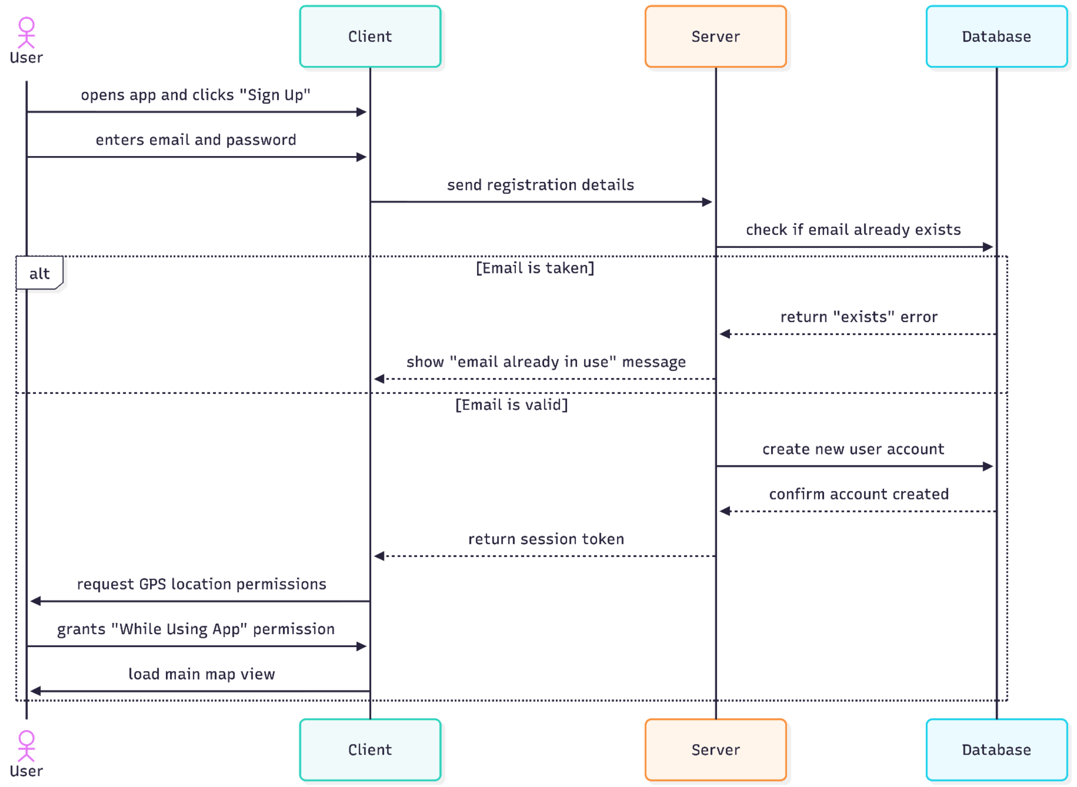
Sequence of events to join a circle


User
Client
Server
Database
enters invite code for a private Circle
submit join request
validate invite code
alt
[Code is invalid]
return "not found"
show error message
[Code is valid]
add user to Circle members list
confirm member added
return success
Refresh map to show private pins
request anchors for this new Circle
query private anchors for Circle
return list of private anchors
send anchor list to app
update map with new private pins
User
Client
Server
Database

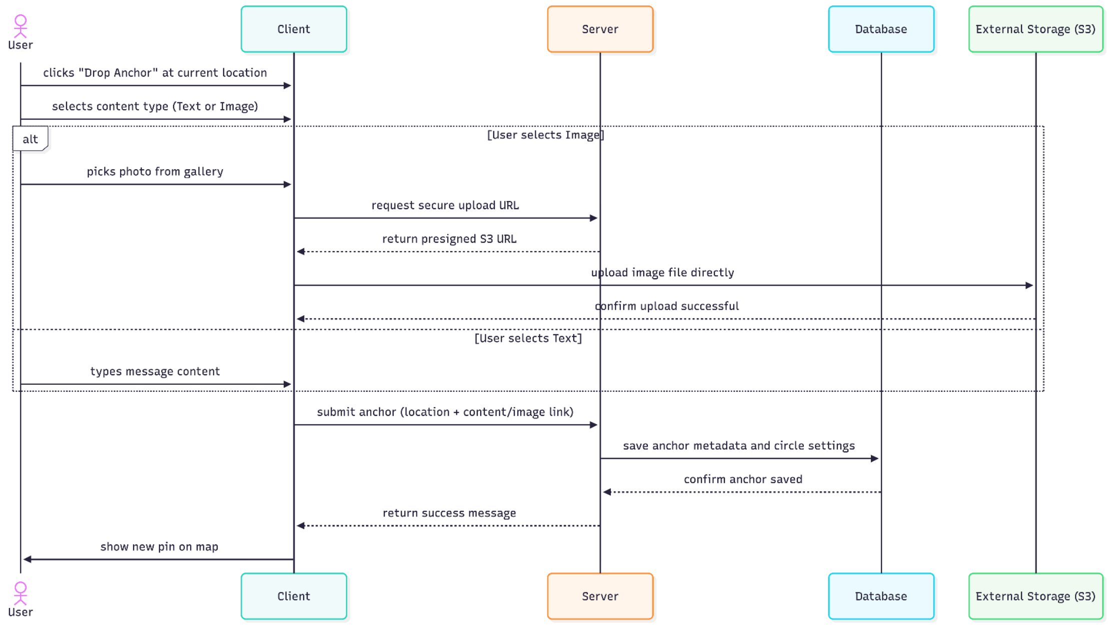
Sequence of events to discover and unlock the anchor


User
Client
Server
Database
Polling Loop (Every 30-60s)
loop
[Check for Nearby Anchors]
send current location
query discoverable anchors
return list of anchor metadata
update map with nearby pins
User approaches a specific Anchor
detect user is within radius
send unlock request (user location + anchor ID)
fetch anchor coordinates and rules
return anchor details
Server validates distance to prevent spoofing
calculate distance between User and Anchor
alt
[User is too far]
deny unlock (remain locked)
[User is within radius]
check if user is in the allowed Circle
confirm user has access
return unlocked content (text/image URL)
display anchor content
User
Client
Server
Database

Sequence of events for authentication and onboarding

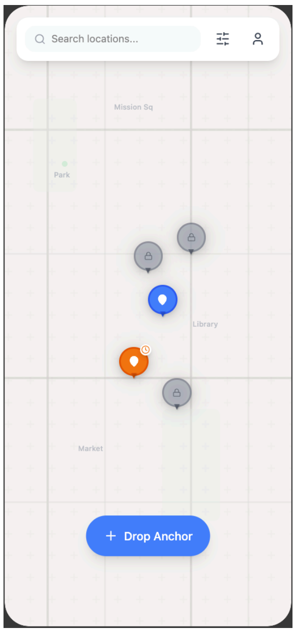

User
Client
Server
Database
opens app and clicks "Sign Up"
enters email and password
send registration details
check if email already exists
alt
[Email is taken]
return "exists" error
show "email already in use" message
[Email is valid]
create new user account
confirm account created
return session token
request GPS location permissions
grants "While Using App" permission
load main map view
User
Client
Server
Database

Sequence of events when user wants to create Anchor

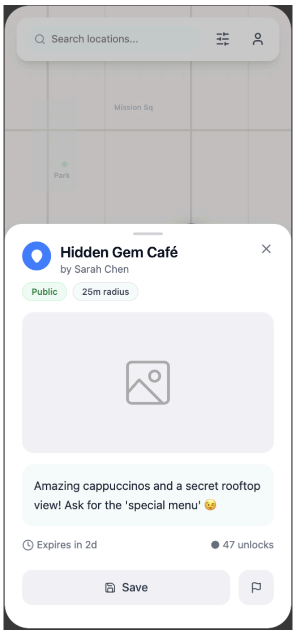


Team 21 - Aryan Jumani, Annsh Navle, Pranav Bansal, Shriyan Bachigari

UI Mockups

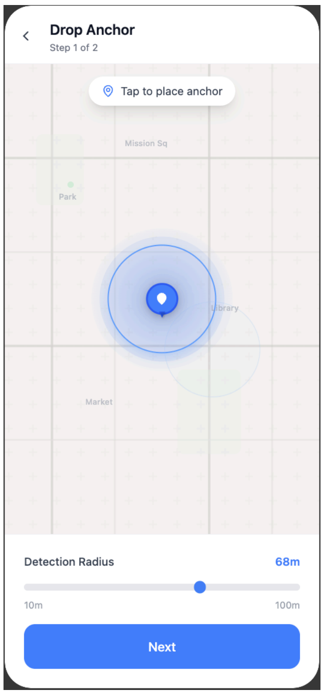


Screen showing nearby anchors

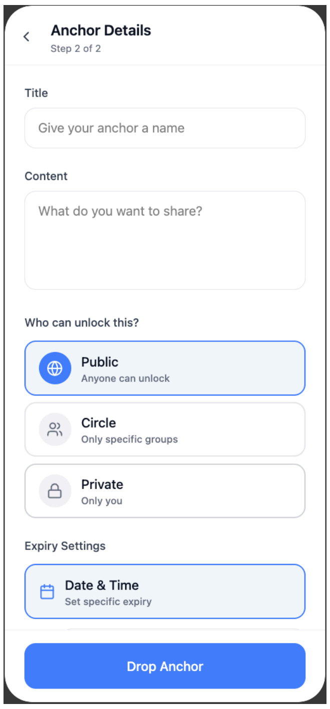


Amazing cappuccinos and a secret rooftop view! Ask for the 'special menu' 😉

Selected Anchor shows related data

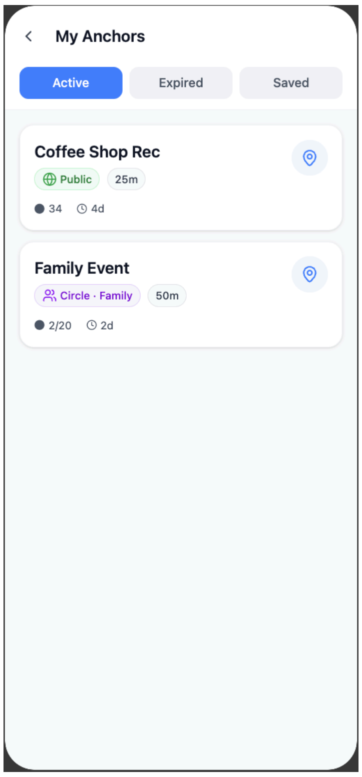


Creating an Anchor with custom unlock radius



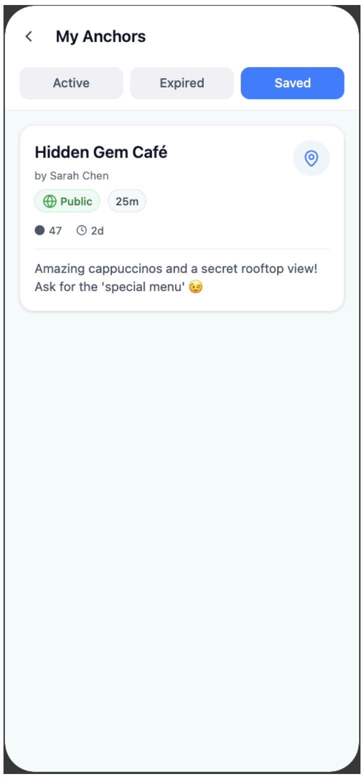
Fill Anchor details

Team 21 - Aryan Jumani, Annsh Navle, Pranav Bansal, Shriyan Bachigari



Coffee Shop Rec
Public 25m
● 34 4d

Family Event
Circle · Family 50m
● 2/20 2d



Hidden Gem Café
by Sarah Chen
Public 25m
● 47 2d

Amazing cappuccinos and a secret rooftop view! Ask for the 'special menu' 😉

View Active, Saved and Expired Anchors