Faculty of Engineering
Department of Electrical and Computer Engineering

NeAR Me
Detailed Design and Timeline
2023.12

Alice Ye, 20700916
Michael Sawyer, 20765348
Shizhen Li, 20785413
Aidan Foster, 20721410

Pouya Mehrannia

June 27, 2022

Table of Contents

1. Project Description

1.1 Motivation

Nowadays, with about 84% of the world population owning a smartphone [1], people rely on digital devices to learn about and connect to the surrounding world more than ever. Statistics show an increasing trend that people continue to rely on the map and navigation applications on their mobile phones [2]. Google data from March 2022 demonstrates that the “open now near me” searches have grown globally by over 400% year-over-year [3]. This means that people are no longer using digital maps to just look up directions, but are using them more and more to learn about interesting places near them while they are on the move. The problem is that, with the existing map and navigation mobile applications, exploring the surroundings is not a very smooth experience when one is on the move; they need to stop, search for what’s happening near them (possibly in another application), and then pick a destination to go to. Why can’t we make this experience smoother? As mobile augmented reality (AR) gains its popularity worldwide [4], it is a perfect option to present location-based information in a convenient way while on the move. We will use AR to enrich people’s experience while they are on the move by rendering information about their surroundings in real-time, right in front of their camera.

1.2 Project Objective

The objective of NeAR Me is to eliminate the need for people to look up information about their surroundings while they are on the go. Users can subscribe to, or publish location-based information to, topics of interest. As users move around, information about places or things near them, from topics they are subscribed to, will be displayed in AR, in real-time.

1.3 Block Diagram

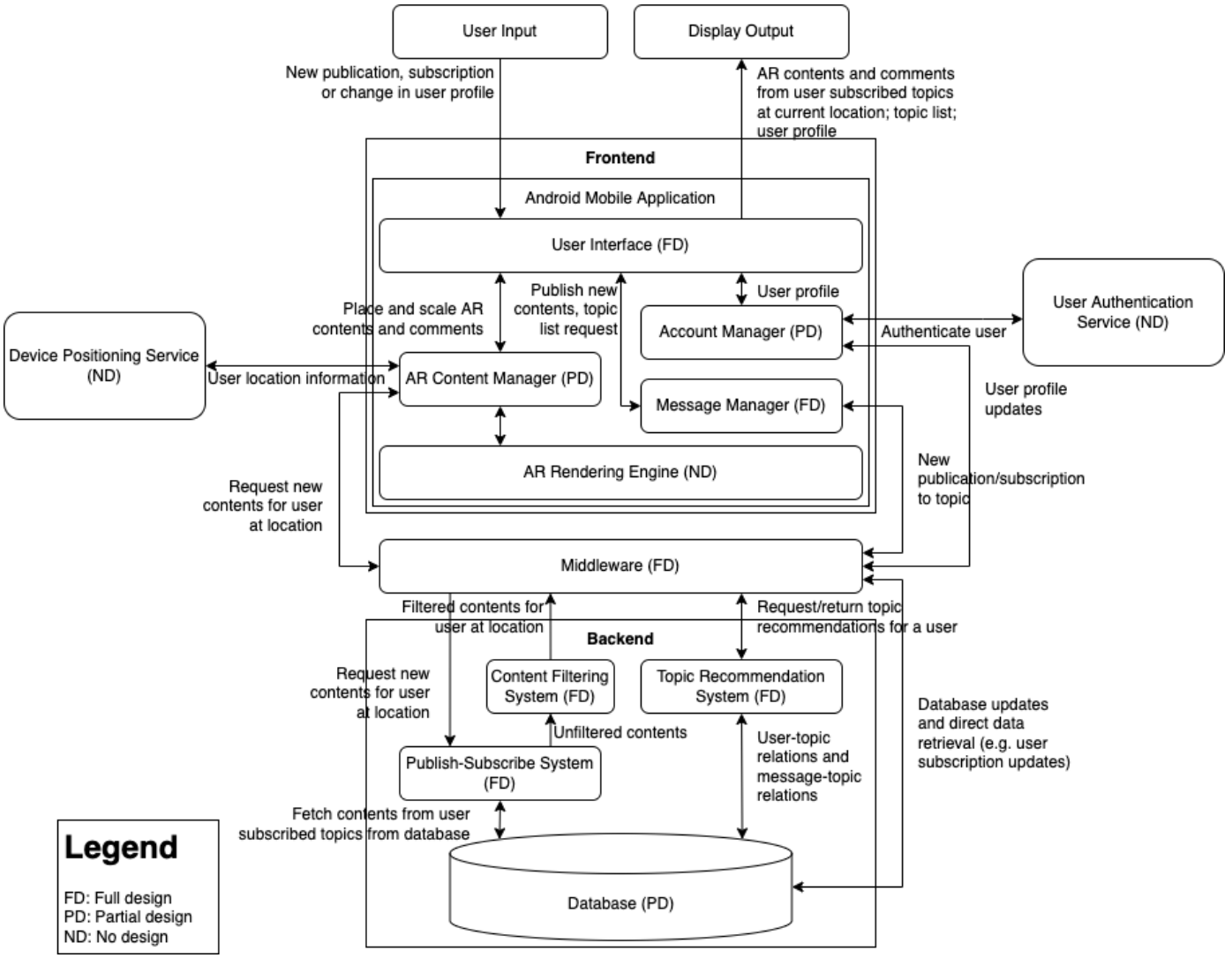

Figure 1: Block diagram of the NeAR Me subsystems.

As Figure 1 depicts, the project is divided into two parts: the frontend and the backend. The frontend and the backend are connected through a middleware layer. There are also third-party services that the frontend interacts with. The user provides input to, and receives output from, the frontend.

The frontend is an Android mobile application that consists of a user interface, AR content manager, account manager, and AR rendering engine. The user interacts with the frontend via the user interface. The user can create an account, login, publish/subscribe to topics, and view AR content from topics. The AR content manager controls what, where, and how much AR content is displayed on the user's screen. It makes requests to the backend for content from topics that the user is subscribed to, and it interacts with the AR rendering engine to display the content on the user's screen. The device position service is used to ensure that the content is placed at the correct location relative to the user. The account manager manages

the login session, subscriptions, and other account information. The user authentication service stores usernames and passwords, and provides a secure way for users to authenticate.

The backend consists of a publish-subscribe system, content filtering system, topic recommendation system, and database. The database stores every topic, the contents of every topic, users' subscription lists, and users' account information, and it provides a way to query this information. The publish-subscribe system receives requests for AR content near a user, from topics that the user is subscribed to. It queries the database for this content and passes it through the content filtering system. The content filtering system ensures that only a certain amount of content is returned to the frontend by filtering out content that is packed too densely and/or content that the user will likely not be interested in. The topic recommendation system will recommend new topics to a user based on the topics they are subscribed to and most interested in.

The middleware connects the frontend to the backend. It handles dispatching requests from the frontend to different components in the backend, balancing the load among replicated components in the backend, routing information between distributed components in the backend, and returning responses from the backend to the frontend. It also protects the backend from denial-of-service attacks by limiting the number of requests made by the frontend.

2. Project Specifications

2.1 Functional Specifications

Table 1: NeAR Me functional specifications.

ID	Subsystem	Description	Necessity
FS1	UI, Account Manager, User Authentication Service	Users must be able to create a new account with a username and password.	Essential
FS2	UI, Account Manager, User Authentication Service	Users must be able to login with the correct username and password.	Essential
FS3	UI, Account Manager, Publish-Subscribe System	Users must be able to subscribe to topics from a set of default topics (e.g. #food, #shopping, #music).	Essential
FS4	User Interface, Account Manager, Publish Subscribe System, Database	Users must be able to create and publish/subscribe to custom topics.	Essential

FS5	UI, Topic Recommendation System	Users must be able to receive topic recommendations for topics they are not subscribed to.	Non-Essential
FS6	UI, Message Manager, Database	Users must be able to become authorized to publish to topics at a specific location.	Essential
FS7	Publish-Subscribe System, Database	Users must be able to publish to topics only at locations they are authorized to publish at.	Essential
FS8	UI, Account Manager, Publish-Subscribe System, Database	Users must be able to comment on any topic at any location.	Essential
FS9	Publish-Subscribe System	Users must be able to see only information from topics that they are subscribed to.	Essential
FS10	UI, AR Content Manager	Users must be able to read comments.	Essential
FS11	AR Content Manager, Content Filtering System	Users must be able to change the number of entries rendered in AR to between 1 and 100 entries.	Essential
FS12	AR Content Manager, Publish-Subscribe System	Users must be able to see content in AR that is up to 100 m away.	Essential
FS13	AR Content Manager, Middleware, Publish-Subscribe System, Content Filtering System	Users must be able to see new AR content appear within 1 s of them moving.	Essential
FS14	UI, Message Manager, Database	Users must be able to like/dislike the displayed contents.	Non-Essential
FS15	UI, Account Manager, Database	Users must be able to change their profile avatar, username, and description.	Non-Essential
FS16	AR Content Manager, Database	Users must be able to adjust the AR content's GPS location and size before publishing it to a topic.	Non-Essential
FS17	Account Manager, User Authentication Service	Users must have to login only once.	Non-Essential

FS18	Content Filtering System	Users can filter the displayed contents by preference (i.e. preference on topics, sort by newest, most-liked, etc.)	Non-Essential
FS19	AR Content Manager	Users must not see AR content if it is behind a physical object.	Essential
FS20	AR Content Manager	Users must be able to publish entries as large as 2.5 m by 2.5 m.	Essential
FS21	UI	Users must be able to insert text input for publication with an upper bound of 140 characters.	Essential
FS22	UI	Users must be able to insert image input for publication with a 5 MB upper bound on image size.	Non-Essential

2.2 Non-functional Specifications

Table 2: NeAR Me non-functional specifications.

ID	Description	Essential/Non-Essential
NFS1	The backend must be scalable across multiple servers, at least one per region served, to ensure service availability.	Essential
NFS2	Users can be served by the app in the case of a single server failure.	Essential
NFS3	The mobile application must be compatible with most AR-capable models of Android devices running Android 7 or newer.	Essential
NFS4	The total cost of the project should not exceed 600 CAD.	Non-Essential
NFS5	The user interface should provide a smooth user experience; easy to navigate.	Non-Essential
NFS6	Users must be able to change the font size for the UI text display.	Non-Essential
NFS7	Users must be able to publish up to 1 topic entry per minute.	Non-Essential
NFS8	Users must be able to see AR content within 5 m of its actual location.	Essential

3. Detailed Design

3.1 Frontend

3.1.1. User Interface

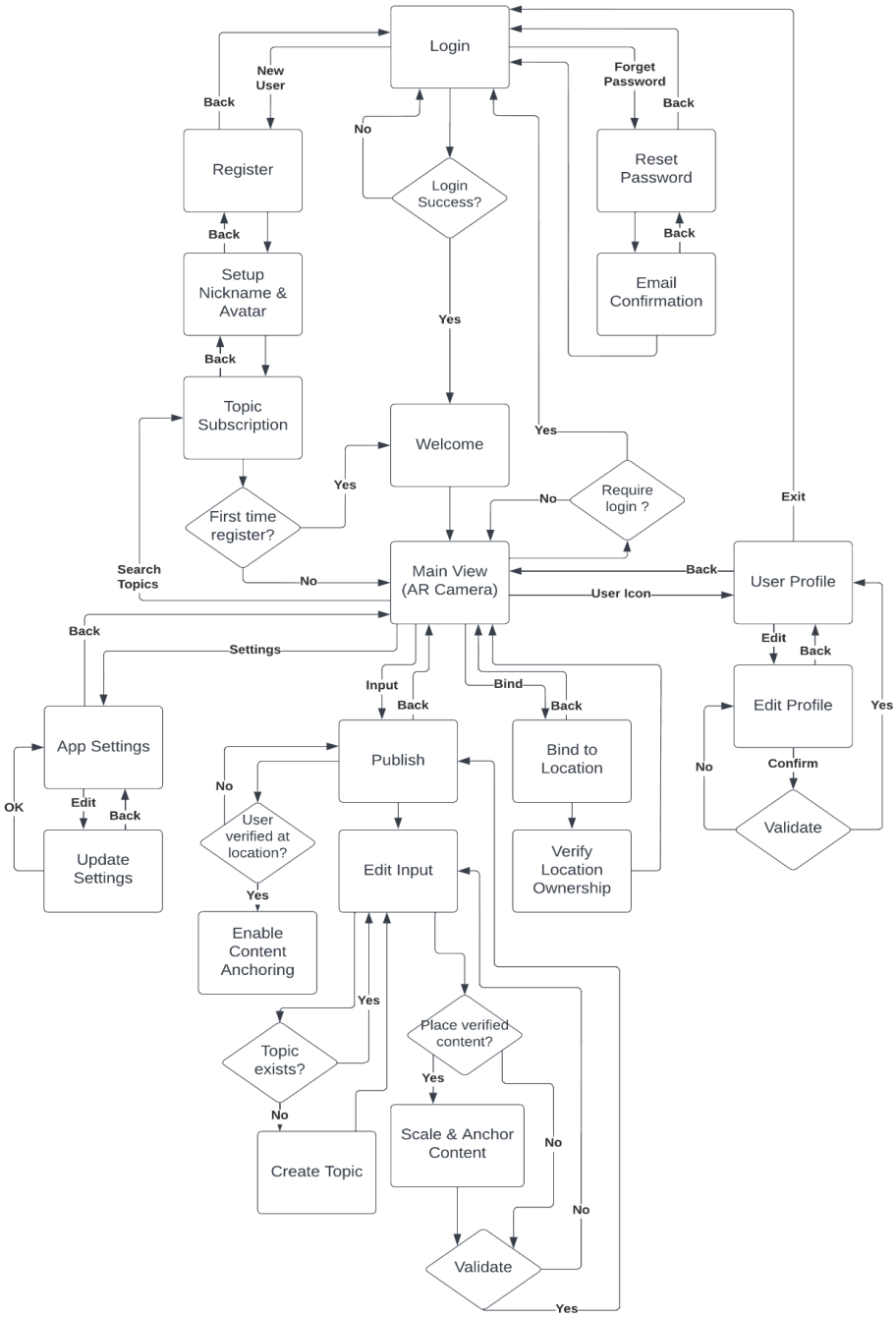

Figure 2: NeAR Me UI flow.

The user interface design consists of the following parts:

- User authentication (login, register, reset password)
- Main view (AR camera)
- Publish and subscribe to topics of interest
- User profile
- App settings

And the state transitions are described by Figure 2 above.

NeAR Me uses email and password user authentication and keeps the login session for 30 days after successful login (FS17). When a user forgets password at login, a link will be sent to their email address for confirmation and password reset. Correct implementation of the user authentication UI flow are building blocks to FS1 and FS2.

Figure 3 shows the main view of the app – an AR camera street view. Here, real-time location-based contents are selected and displayed to users from their subscribed topic. Two types of display: (1) verified contents are published by verified location/property owner(s) and are anchored to the location, and (2) transient comments (non-verified contents) are not anchored to a physical property, but instead randomly pops up from user's subscribed topic when user is near the location where the comments are sent. When no network is detected, an alert is prompted and the AR camera view is paused. Users can force reload the AR view when they experience lags.

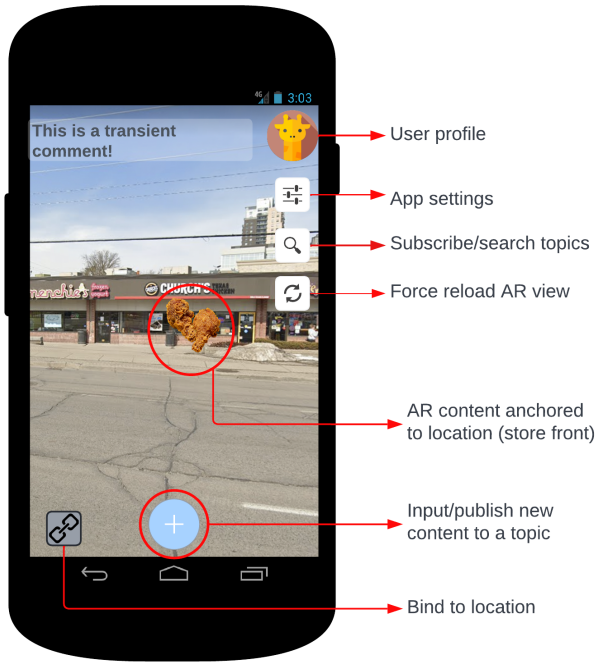

Figure 3: NeAR Me app main view prototype.

When a user taps on a transient comment, they pause the comment (FS10) and can like or dislike the comment (FS13). The comment details (topic, date of publication, nickname of the sender) are shown when the comment is paused.

When a user taps on the input button in the main view, a keyboard shows up to allow text input. If the user is verified at the location, they will have extra options to choose if they want to anchor this new publication and also if they want to upload an image as content (FS7). For all text input, the font, font size, font color, and font shadow can be altered (FS15). All publications must be sent to a topic. If users enter a topic that does not exist, they can create this new topic by publishing new content to it. This UI flow satisfies the create and publish aspect of FS4.

When a user taps on the magnifying glass icon in the main view, they get redirected to a list view of topics which allows them to search and subscribe to topics. The list view is linked to a topic recommendation system and can suggest new interesting topics for the user to subscribe to (FS5).

In the user profile page, users can access and perform the following:

- Personal info: update
- Bound location(s): add, delete, update
- Subscriptions: unsubscribe
- Publications: delete

In the app settings page, users are allowed to modify the following:

- Content display density
- Comment display speed
- Comment display font size

An iterative design approach is taken to design the user interface:

1. Prototype layout and flow in UI design tool
2. Compare with the specifications in section 2. If a specification is not satisfied, go back to step 1.
3. Compare UI layout with apps/designs that have similar functionalities. Go back to step 1 if adjustment is necessary.
4. Prototype UI in development environment (validate feasibility). If not feasible, go back to step 1.

3.1.2. AR Content Manager

The cache is a Bkd-tree consisting of multiple k-d trees. The Bkd-tree consists of one "in-memory" k-d tree of size M, and a set of K k-d trees where the ith k-d tree is of size $2^i \cdot M,\ 0 \leq i < K$. When a **Node** is inserted into the Bkd-tree, it is inserted into the in-memory k-d tree. If this tree becomes full, then the following occurs:

(1) Find the first empty k-d tree at index k;
(2) If there are no empty k-d trees, then perform a region search on each k-d tree in the Bkd-tree; any **Node** within 100 m of the user's location is inserted into a new Bkd-tree; go to (6);
(3) For each valid **Node** in the old in-memory buffer and in all k-d trees with index $0 \leq i < k$, create a new **Node**, copy the **Metadata** and **GameObject** pointers to the new **Node**, and insert it into two singly linked lists (using **Node.left** and **Node.right** as the "next pointer" in the respective lists);
(4) Sort both linked lists using merge sort;

(5) Perform a normal k-d tree insertion into the kth k-d tree using the two linked lists; the new Bkd-tree consists of the old $k \leq i < K$ k-d trees; the others are now empty;
(6) Change the Bkd-tree pointer to the new Bkd-tree; the old one is garbage collected.

Each **Node** in the Bkd-tree is defined as follows:

```
struct Node {
   Node *left;
   Node *right;
   Metadata *metadata;
   GameObject *content;
};

struct Metadata {
   uint8_t topic[20 * 4];
   uint8_t id[12];
   double lat;
   double long;
   uint8_t valid;
};
```

During application initialization, or after the cache has been reset, the Bkd-tree is empty and a request is made for AR content within a 110 m radius of the user's location. A **Node**, **Metadata**, and **GameObject** are created for each entry in the response, and each entry is inserted into the Bkd-tree.

In their settings, the user can set the number of AR content that they want rendered; the content that is currently being rendered is called "active content". An array is used to store pointers to the active content. Each frame, the content in this array is deactivated, and the array is emptied. Then, the user's location is obtained and a nearest neighbor search is performed on the Bkd-tree to find the next set of valid content to become the active content. The nearest neighbor search finds the closest entries within 100 m of the user, so FS11 and FS12 are satisfied.

In a separate thread, a request is made every 500 ms for new AR Content with 110 m of the user. The user can see entries up to 100 m away and entries up to 110 m away are cached; when the response time is 500 ms, the request and response take 1 s; a user moving at 36 km/h will move 10 m in 1s; thus, FS13 is satisfied. The response can contain new content, updated content, and/or deleted content. Any new content in the response is inserted into the Bkd-tree. Any updated content is updated in the Bkd-tree. Any deleted content is marked as invalid.

When the user unsubscribes from a topic, any **Node.metadata.topic** in the Bkd-tree matching that topic is marked as invalid.

The centroid of the user's location over the last 5 seconds is maintained. If the user is within 10 m of that centroid, then a request is made for comments in that circle. The comments are queued up and displayed one at a time, 5 seconds each until the user moves out of that circle. This satisfies FS10.

The user can interact with a **GameObject** by tapping the screen. The game engine transforms the screen coordinates of the touch into a raycast from the camera to the point that was touched. If this raycast intersects AR content, then a menu is displayed that allows the user to like the content. When publishing content, the user can touch the screen with two fingers; if the distance between the fingers increases/decreases with time, then the **GameObject**'s size is scaled up/down, and is capped at 2.5 m by 2.5 m. This satisfies FS14, FS16, and FS20.

The game engine is capable of detecting planes in the physical world and representing these planes internally. The depth of a plane in the game engine can be used to occlude other **GameObject**s that are behind that plane. This satisfies FS19.

Table 3: comparison of data structures used for storing geographic data.

Algorithm	**Sorted, Doubly Linked List**	**K-d Tree [16]**
Insert	$O(1)$ to append to the end of the list.	$O(log\ n)$ to insert into a BST.
K Nearest Neighbor Search	$O(k)$ since the first k elements in the list are closest to the user.	$O(k\ log\ n)$ search for each neighbor.
Exact Search	$O(n)$ for a linear search.	$O(log\ n)$ for a binary search.
Region Search	$O(k)$ to remove the farthest k elements from the user.	$O(n)$ when the entire tree is traversed.
Delete	$O(n)$ search and $O(1)$ mark for deletion.	$O(log\ n)$ search and $O(1)$ mark for deletion.
Maintenance	$O(n)$ distance update and $O(n\ log\ n)$ merge sort.	$O(n\ log\ n)$ merge sort on lists and $O(n\ log\ n)$ tree construction.

Every frame, the sorted, doubly linked list needs to run the maintenance and K nearest neighbor search algorithms, which results in a complexity of $O(n\ log\ n)$ each frame. The k-d tree needs to run the K nearest neighbor search, which results in a complexity of $O(k\ log\ n)$ each frame. Each update, the sorted, doubly linked list may need to run the insert, exact search, and delete algorithms, which results in a complexity of $O(n)$ each update. Each update, the k-d tree may need to run the same algorithms, which results in a complexity of $O(log\ n)$ each update. When it fills up, the sorted, doubly linked list needs to do a region search to remove the farthest **Node**s, which results in a complexity of $O(k)$. The k-d tree needs to run the region search and maintenance algorithms, which results in a complexity of $O(n\ log\ n)$ each time it fills up. The k-d tree is less efficient in algorithms that are run infrequently, and more efficient in algorithms that run frequently, so it is chosen over the sorted, doubly linked list.

The problem with the k-d tree is that it is a static data structure. After only a few insertions the k-d tree can become unbalanced, resulting in poor K nearest neighbor search performance. To keep the k-d tree balanced, it needs to be rebuilt frequently. The Bkd-tree is a dynamic version of the k-d tree that rebuilds a large portion of the data infrequently, and smaller portions more frequently [17]. Also, portions of it can be rebuilt while answering queries. Thus, the Bkd-tree is chosen over the k-d tree.

3.1.3. Message Manager

The message manager is responsible for communicating with the backend when users publish or subscribe to a topic. This implementation is related to the realization of FS4, FS5, FS6 and FS14. Inside Input Manager's message validation block, the upper bound of the user input size is checked (FS21, FS22).

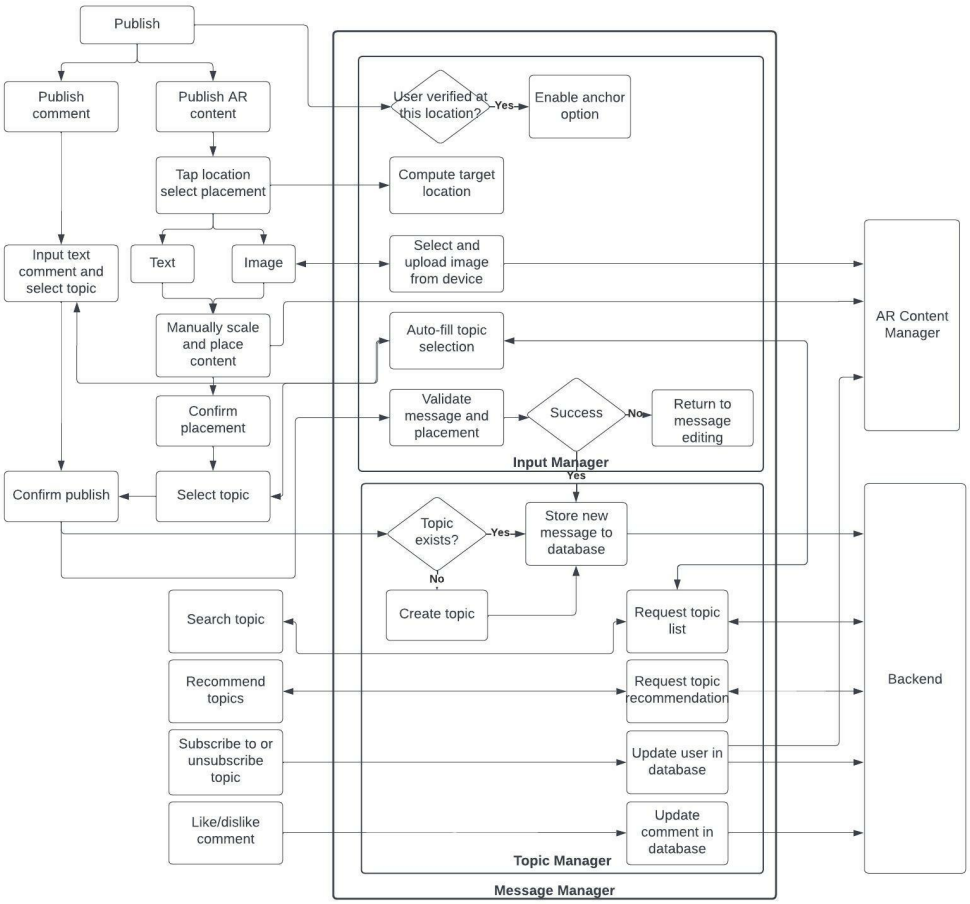

Figure 4: Message manager components and integration.

3.1.4. Account Manager

The account manager is responsible for the user authentication and management.

To increase the speed of our deployment and focus on the core logic implementation, we decide to use third-party service to provide secure user authentication. We compare the two most popular authentication services for mobile apps in table 4, and using that information, we rank the options in a decision matrix in table 5. We select **Firebase** as our authenticator service because it is well-documented, easy to integrate, and free to use.

Table 4: Comparison of user authentication approaches. [5] [6]

	Firebase	**Auth0**	**Custom Implementation**
Pros	Good documentation; many sample projects.	Easy to integrate.	Independent of 3rd party; full control of data flow and data privacy.
	High availability.	Good MFA support.	No cost.
	Easy to integrate.	Data integrity.	Optimizable query.
	No cost (free tier usage).		
	Data integrity.		
Cons	Dependent on 3rd party (Google).	Expensive.	Encryption complexity; protect our own data.
	Limited support for IOS.	Fast changing APIs.	Extra implementation effort on the backend.
	Limited querying capabilities.	Dependent on 3rd party.	Difficult to add-on MFA.
	Difficult to predict future cost.	Poor developer support.	Complexity in data syncing.

Table 5: Decision matrix for the user authentication approaches.

	Firebase	**Auth0**	**Custom Implementation**
Ease to implement	2	1	0
Cost (cheap = high score)	1	0	1
Security	2	2	1
Total	**5**	3	2

The account manager fits into our system as follows:

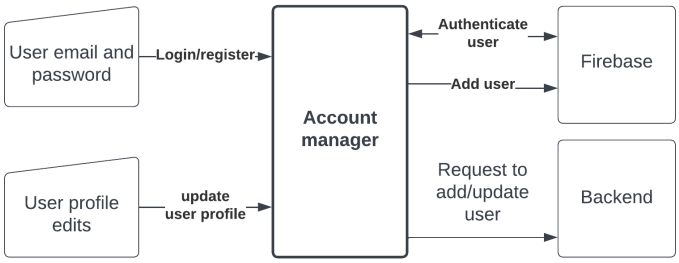

Figure 5: Account manager integration.

When the user attempts to login, the account manager uses the Firebase APIs to authenticate the user (FS2) and to maintain a persistent login session (FS17). When a new user is registered, the account manager adds the user to the Firebase database. When the user profile is modified from the UI, the account manager makes an API request to the app backend to update our backend database.

3.2 Backend

3.2.1 Framework

The backend's purpose is to process and answer user requests that require complex computation and database communication. Under the assumption that the backend must handle a large number of users requests from all over the work and response with near-real-time latency, the backend framework must provide support for speed optimization (high concurrency and database access speed), scalability and reliability. Given the time constraint, we favor the backend framework which is easy to interface, integrate, and implement.

Having compared two most popular backend frameworks and considered a serverless cloud deployment approach (i.e. deploy backend functions and database as cloud functions, not managing a server ourselves), we decide to use **Java Spring Boot** as our backend framework.

Table 6: Decision matrix for the backend framework.

	Java Spring Boot	**Python Django**	**Serverless Cloud Deployment**
Response speed	3	2	2
Ease of use	2	2	3
Scalability	2	1	3
Reliability	3	2	1
Total	**10**	7	9

We first compare Java Spring Boot and Python Django. Spring and Django are both easy-to-use frameworks with good scalability; however, compared to the high concurrency (multithreading) support in Java Spring Boot, Python Django is slower in terms of processing and can only handle one HTTP request at a time. Java Spring Boot is more scalable and reliable than Python Django because Spring is more opinionated and modularized than Django [8]. Django templates by default errors quietly, and the REST framework can only handle one HTTP request at a time [9]. Next we compare the serverless cloud deployment options with Java Spring Boot and Python Django, based on team member experience (Google/AWS cloud). The serverless option scores the highest in terms of ease of use and scalability because we do not need to manage our own server and infrastructure if we deploy our backend computation as cloud functions, and the computing resource will be allocated and charged per request. However, in exchange for the ease of use and automated scaling, the response time might not be guaranteed because the resource allocation is dependent on a third party. For this same reason, and also with the concern of unexpected cost, the serverless option scores the lowest in terms of reliability.

3.2.2 Database

For the database of our project, we determined the following aspects would be important to our system:

Table 7: Aspects, Weights, and Reasons for Weights in Database Decision Matrix

Aspect	**Weight**	**Reason**
Query by GPS speed	3	Positioning objects in AR requires accurate position data, which necessitates that position-based queries are done quickly, and since that is the focus of the app, this aspect should have the most weight
Query by Username speed	2	In order to expediently deliver subscribed-to content to the user, the system needs to be able to quickly retrieve the list of the user's subscriptions
Query by Topic Speed	2	In order to filter content to display to the user, the system will need to quickly return contents pertaining to a specific topic
Scalability to multiple servers	1	In order to maintain fast query speeds as user counts increase, the requests may have to be spread across multiple servers, but this is not the most vital to operation (NFS1)
Comment post/query speed	1	Since comments cannot all be viewed at the same time, the speed at which a comment can be posted/queried is not the greatest concern in choosing a database design

We have thought about the following possible database designs/choices:

- **Design 1**: Separate databases for each topic, or databases indexed by topic
- **Design 2**: Separate databases per geographic region, indexed by location
- **Design 3**: Single monolithic database for all backend data

Table 8: Decision Matrix for Three Database Designs (scores based on educated speculation)

Aspect (Weight)	**Design 1**	**Design 2**	**Design 3**
Query by GPS speed (3)	1	3	2
Query by Username speed (2)	1	3	2
Query by Topic Speed (2)	3	2	2
Scalability to multiple servers (1)	2	3	1
Comment post/view speed (1)	2	2	1
Total Score	15	**25**	16

From the above table, Design 2, where databases are separated by geographical area, and content is indexed by geographic area, appears to be the best option, so that is what we have chosen.

Table 9: General Comparison of Database Platform Options.

	MongoDB	**Redis**	**PostgreSQL**
Pros	Built-in support for sharding, easy to scale [19] (NFS1)	Embedded pub-sub with pattern matching	Have experience with relational databases already
	Free “sandbox” for design/testing purposes, with 100 databases, 500 entries [18]	Dedicated servers with replication relatively cheap, starting at $7 USD/month [21]	Database program is free
	Replication and auto-failover [20] (NFS2)	GUI for development purposes, instead of CLI	Can be run on any server that we would like
	“Serverless” option available, pay per read [18]	Free tier available [21]	
	Database input, output and querying done in JSON format [23]	Database input, output and querying done on key/value pairs [23]. Values can be JSONs, lists, sets, and more	
	Allegedly faster than Redis for very large databases [23]	Supports “master-master” replication, in addition to “master-slave” [20]	
Cons	No prior experience with NoSQL databases	No prior experience with non-relational databases	Servers need to be purchased and set up separately

	Paying for dedicated servers is relatively expensive (at least $0.08 USD/hr) [18]	Deployment on a seperate will require more work to set up replication, likely manual	Sharding, replication, and auto-failover not built in, separate additions [19]
		Free tier only includes a single database [21], meaning that testing will probably require a paid plan	SQL less flexible, harder to use than JSON [22]
		Indexing and searching JSONs is more complicated than MongoDB (from personal testing)	
		Documentation for API implementation in certain languages is poor, and/or a work-in-progress in some circumstances (from personal testing)	

MongoDB vs Redis Performance Benchmarks:
To further compare between MongoDB and Redis, a benchmark script was developed in Python that randomly generates 10000 AR content database entries in JSON format (more entries would be more desirable, but the Redis free tier has only 30MB of storage [21]), connects to a free-tier database for each option, uses a specified number of threads (each representing a user) to write all of those entries in each database, averages the time taken for each database transaction to get the write speed, then filters the databases for text-based AR content entries (see entry spec below) to get the filter speed. The following data was collected with 25 threads (so each thread sequentially makes 400 writes to each database):

Table 10: Speed of MongoDB vs Redis for Database Access.

	MongoDB	**Redis**
Average Write Speed (ms)	53.9	53.4
Filter Speed (ms)	~0.0	2.4

It must be noted that the filter speeds cannot be considered in isolation, only as part of a comparison, as the fraction of entries that are text-based is randomly determined before the tests (both databases have identical data sets). Only 25 threads were used because the free tier of Redis has a limit where up to only 30 connections can access the database concurrently [21].

While MongoDB is ever so slightly slower than Redis for writing single entries to the database, most of the database interactions are going to be reads, as a user is almost constantly reading entries for content regarding their surroundings, but will write comments/content to the database comparatively infrequently. The increased need for fast reads makes MongoDB the better option, as in the benchmarking, searching the (admittedly small) database for a particular string was practically instant with MongoDB, whereas with Redis the same read took multiple milliseconds.

Therefore, the decision has been made to use **MongoDB** for the backend database.

Database Entry Specification
AR_Content:

- **_id** : Unique 12-byte hexadecimal ID (autogenerated by MongoDB when entry is added to database), eg. “7d35eb5113e0d511b00e57f3”
- **Geomesh** : Unsigned integer to be computed by middleware layer from GPS coordinates, to be used as secondary index in MongoDB for faster querying by GPS, eg. 7841564
- **Location** : 2-element list of double-precision floats for GPS coordinates, eg. [20.1245, 15.5131]
- **Type** : String to indicate whether content is text or image, ie. “text” or “image”
- **Anchored** : String representing a boolean to indicate if this entry is anchored in AR or is a comment, if the entry is text
- **Content** : A string with two uses. If Type is “text”, this element is that text, otherwise if Type is “image”, this element is a URL to the image for the middleware layer to deliver to the user
- **Topic** : 12-byte hexadecimal ID of the topic this content was posted under
- **User** : 12-byte hexadecimal ID of the user that posted this content
- **Timestamp** : Signed 32-bit integer Unix timestamp
- **Deleted** : String representing a boolean to indicate if this entry has been deleted by the user (but not yet permanently deleted on the backend), ie. “T” or “F”
- **Likes** : Signed integer number of times the post has been liked, minus the times it has been disliked (FS14)

User:

- **_id** : Unique 12-byte hexadecimal ID (autogenerated by MongoDB)
- **AuthLocations** : List of coordinates (2-element list of doubles) representing locations where the user is authorized to post content, eg. [[20.1245, 15.5131], [41.2515, 23.1355]]
- **Subscriptions**: List of IDs of topics subscribed to by the user, eg. [“ef88ea4dae9d22a65d60cad6”, “ab707742c991ab81e176f724”]
- **Publications** : List of IDs of AR content entries posted by the user

Topic:

- **_id** : Unique 12-byte hexadecimal ID (autogenerated by MongoDB), eg. “ab707742c991ab81e176f724”
- **Name** : String name of topic, eg. “Restaurants”

3.2.3 Publish-Subscribe System

The purpose of this subsystem is to provide the end-user with real-time information and contents surrounding the user with regards to FS11, FS12 and FS13.

Our initial design features a general bus with all connected frontend (representing end-user) and the database, where all the connections to user devices are kept open and active at the same time. Its workflow can be described by the following figure:

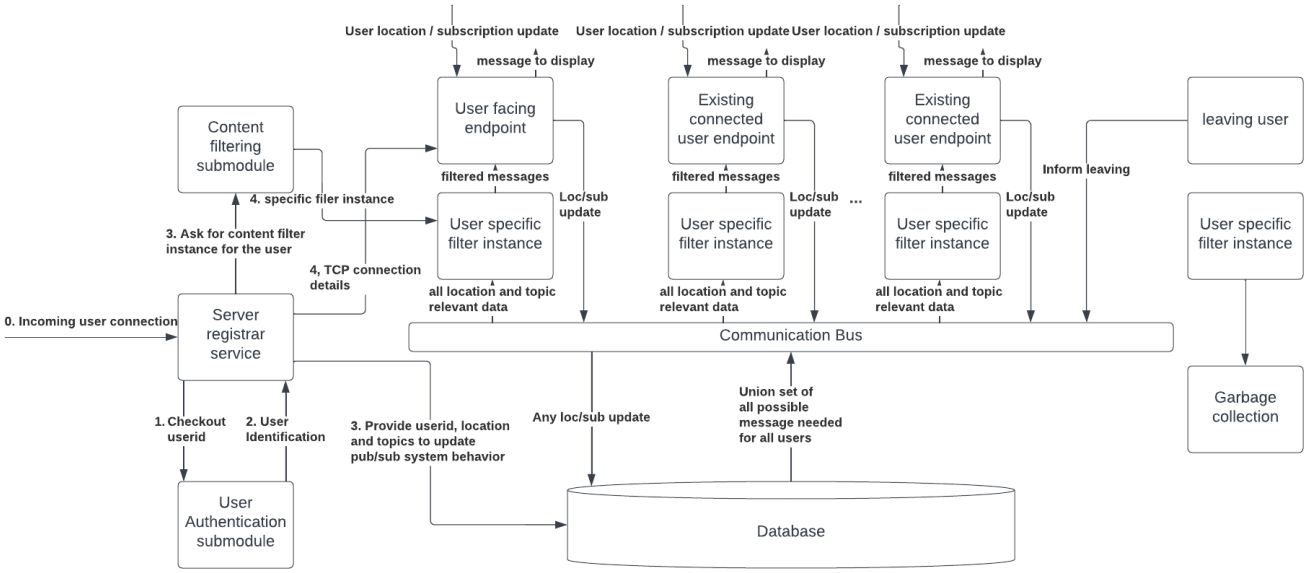

Figure 6: Initial design of the pub-sub system.

This prototype has the following major problems:

- This design will cause massive data transactions and a great amount of system resources for each additional user. The database is required to push the union of all the subscribed topics of current connecting user and union of all connected users' locations. A new content filter instance needs to be created for every additional connected user. All these situations make the service efficiency low.
- The status of the system is very unstable. As the user is constantly moving, the locations where the database should fetch messages from is constantly changing; as a user opens and closes the app unpredictably, the whole system needs to update accordingly. This creates lots of overhead doing bus setup modifications in the runtime.

To address these outstanding difficulties, a second design is created where this system will only be responding upon request. The messages specifically needed by one client are pushed out from the database upon that client's request, presumably triggered by user movement or timeout. It's workflow can be described by the following figure

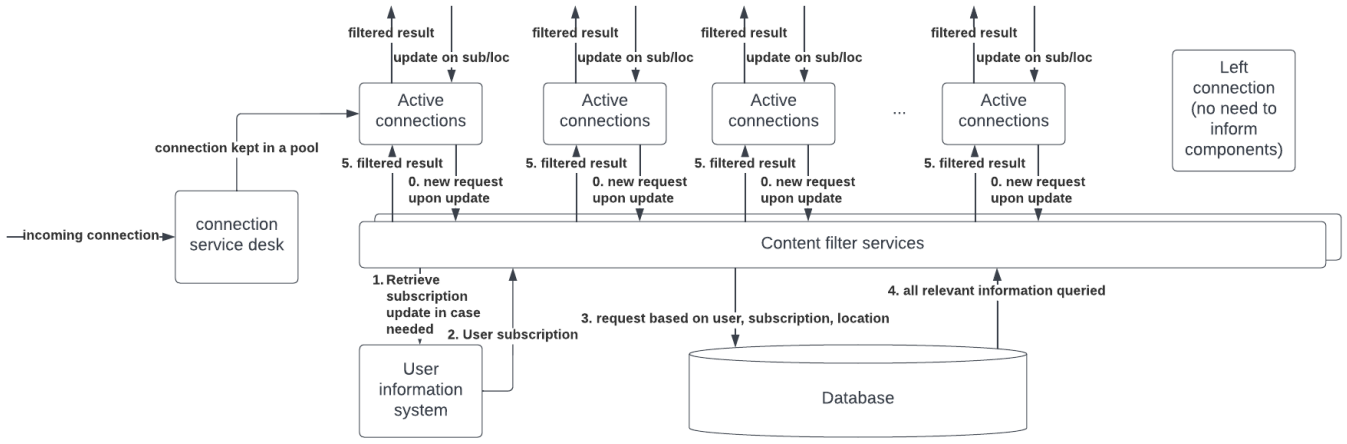

Figure 7: Second design of the pub-sub system.

This design has some issues that could cause difficulties in other components of the system:

- With this subsystem only knowing the current position of a specific user, it has to prepare all data around the user, which might already exist on the user screen. As the use case of our application is mainly mobile, any duplicated data transmission between frontend and backend is undesirable.
- As our application resides on mobile phones and is used when the user moves around, the IP address of the user could be frequently changing. A stable TCP connection usually can't be kept.

To address the above two problems, we made some changes to the design with the help of other subsystems and achieved the following final workflow design as shown in the figure below:

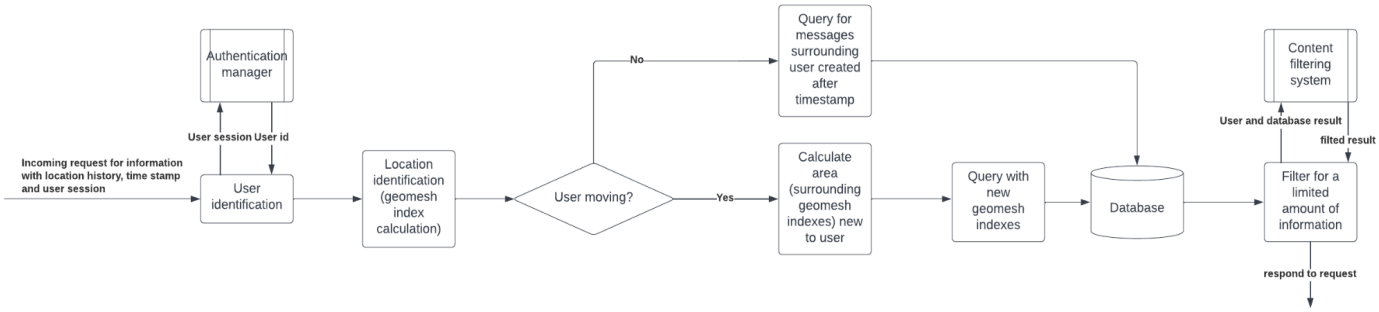

Figure 8: Final design of the pub-sub system.

In this design, the user session from the authentication submodule is used to uniquely identify users on the move. This means new connections need to be set up for each request, but it is not a big issue as managing frequently changing connections is not any better. Instead of one GPS position the client provides to the backend, a list of history is also provided so that the backend can find only additional regions the client hasn't cached or need update on.

Besides, the system also used an area discretization method to assist both database and content filter submodules in their operations. In this method, the maps are divided into small blocks called geo-mesh with a unique index, which can be quickly acquired. With this discretization method, the database can easily group and store messages within the same geo-mesh block, and the query will be using discrete integers instead of floating points, which causes harder manipulation and thus additional delays. It also helps the content filter system in scattering filtered results, as the geo-mesh block itself is a proper division of the map.

3.2.4 Content Filtering System

The content filtering system is responsible for reducing the number of entries contained in a response to the frontend. The number of entries per response must be limited because a given user can have an unbounded number of subscriptions, but the number of entries contained in a response must be bounded. It is not clear which entries should be rejected and which should be selected, so we have decided to give the choice to the user. The content filtering system will filter entries using one of the following conditions: most-liked, most-recent, and random. This satisfies FS18.

The content filtering system receives a set of entries from the publish-subscribe system. Suppose the number of entries in the input is n. The content filtering system selects $k \leq n$ entries that satisfy the

given filtering condition. For the most-liked condition, each of the k entries' number of likes is greater than or equal to each of the other $n - k$ entries' number of likes. Similarly, for the most-recent condition, each of the k entries' timestamps is greater than or equal to each of the other $n - k$ entries' timestamps. For the random condition, the k entries are randomly chosen from the n entries.

The most-liked and most-recent conditions can be implemented using a min-heap as follows:

(1) If the set of n entries is empty, then go to (4); otherwise, remove the next entry;
(2) If the size of the min-heap is less than k, then insert the entry; otherwise, if the entry's number of likes (or timestamp) is greater than the root of the min-heap, remove the root of the min-heap and insert the entry;
(3) Go to (1);
(4) Return the min-heap as the set of k entries.

The random condition can be implemented as follows:

(1) If the set of n entries is empty, then go to (4); otherwise, generate $r \ mod \ n$ and remove the entry at that index;
(2) Append the entry into a list;
(3) If the size of the list is not k, then go to (1);
(4) Return the list as the set of k entries.

3.2.5 Topic Recommendation System

The topic recommendation system is responsible for suggesting new topics for users to subscribe to. The recommender algorithm can suggest based on topic similarity, user similarity, and topic popularity (FS5).

Referring to Figure 9, we design the topic recommendation system with hybrid methods (collaborative filtering + content-based filtering). We do not use the content-based methods alone because we do not have many usable features from the entities (user, topic) to model the relation, and we do not use the collaborative filtering approach alone because it can suffer cold start (newer topics not getting recommended).

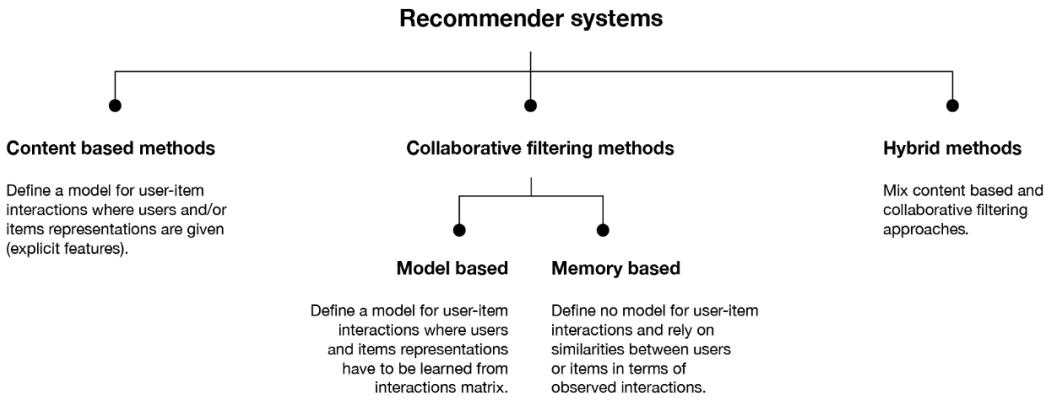

Figure 9: Types of recommender systems. [7]

The recommendation system works as follows:

1. Before the minimum data thresholds T1 and T2 are reached in the system, recommend topics to users based on popularity only (i.e. total number of messages in the topics).
2. When the minimum threshold T1 is reached (total subscriptions > 1000), train a model M1 for the user-user similarity based on the topic subscriptions.
3. When the minimum threshold T2 is reached (total messages > 5000), train a model M2 to classify the custom topics by randomly selecting 100 contents from each topic for comparison.
4. After every 5% increase in the total number of subscriptions, retrain M1.
5. After every 10% increase in the total number of messages, retrain M2.
6. If topic recommendation is requested for users with less than 3 subscriptions, recommend based on topic popularity (and topic similarity); otherwise, recommend with models m1 and m2 using the hybrid approach defined below.

For a user U with more than 3 subscriptions, the topic recommender will:

1. Find the first 20 users similar to U with M1.
2. Get all the topics subscribed by the similar users and store this set of topics as S1.
3. For the topics in S1, find the topics that have not yet been subscribed by U, store this as set S2.
4. Find the first 20 topics similar to U's subscribed topics with M2 and store this as set S3.
5. Combine set S2 and S3, remove duplicates and store this as set S4.
6. If S4 has size less than 5, try to fill S4 with popular topics (no duplicates).
7. Return S4 as the topic recommendations to user U.

For the user-user similarity computation, suppose the input to the computation is all user-topic relations in the database in the form of binary matrix with user_id as row index and topic_id as column labels, we compare model-based and memory-based collaborative filtering methods at finding similar users based on the binary matrix. In general, model-based methods (e.g. clustering, SVD) are slower to train but have better prediction speed (on larger dataset especially) and accuracy compared to memory-based approaches (e.g. kNN, RandomForest) [12]. Neural Collaborative Filtering (NCF) is also an option, but studies have shown that although NCF produces more reliable predictions than the traditional filtering methods [13], it

also requires a lot more time for model training and is complex to tune [14]. Comparing two most widely used CF methods on similarity finding, SVD (Single Value Decomposition) tends to have better performance than kNN (k Nearest Neighbours)[15]. Thus, the user-based CF for our recommendation system implements SVD.

For the custom topic classification, we randomly select min(100, existing) messages from the topic, encode the message text to vector using BERT (Bidirectional Encoder Representations from Transformers), and then classify the topics with the SVM (Support Vector Machine) algorithm [11].

Python is the most suited tool for building the topic recommendation system because of its rich machine learning libraries. We create the recommendation system as a stand-alone python module, containerized to run on cloud and with APIs exposed to communicate with our Java backend.

3.3 Middleware

3.3.1 Exposed API

The exposed API is the application programming interface that our frontend uses to communicate with the backend, internal to the NeAR Me application. The major aspects of design consideration are:

- Reduction of total amount of data transferred over the network, as we expect users to be using connections charged on data mostly outdoor.
- Reduction of computation stress on the end-user device, as mobile phones have weaker computation ability and are already performing a lot of computations with AR content manager.

Table 11: NeAR Me internal APIs.

API endpoint	**Usage**
nearme/api/getContents	Get new/updated/deleted AR contents for user subscribed topics at the current location.
nearme/api/getComments	Get comments for user subscribed topics at the current location.
nearme/api/addMessage	Add/publish new message (comment/AR content) to a topic.
nearme/api/deleteMessage	Delete the message from the database.
nearme/api/isVerified	Check if the user is verified at the location.
nearme/api/createTopic	Create a new topic.
nearme/api/subscribeTopic	Subscribe user to a topic.
nearme/api/unsbscribeTopic	Unsubscribe user from a topic.
neame/api/getTopicList	Get the list of topics available to the current location.

nearme/api/getRecommendedTopicList	Get the list of recommended topics for the user.
nearme/api/addLocation	Add a bound/verified location to the user.
nearme/api/getLocations	Get the list of verified locations for the user.
nearme/api/deleteLocation	Delete (unbound) the location for the user.
nearme/api/getSubscriptions	Get the list of subscribed topics for the user.
nearme/api/getPublications	Get the list of publications for the user.

3.3.2 Load Balancer

Under the assumption that our backend server must handle a vast amount of external HTTP(S) requests from users world-wide and respond with small latency (FS13 and NFS1), the requests to our backend must be load-balanced. With the choice of backend framework (Java Spring Boot) in section 3.2.1, we compare three different load balancing approaches that are applicable to the Java backend.

Table 12: Decision matrix for the load balancing approaches.

	Java Spring Cloud LoadBalancer	**Google Cloud Load Balancer**	**Nginx (open source)**
Ease to implement	2	2	2
Performance & Reliability	1	2	3
Cost (cheaper = higher score)	1	0	1
Total	4	4	**6**

The three compared options have similar implementation complexity; all have well-documented configuration steps for an out-of-the-box solution. In terms of load balancing performance, Nginx performs a lot better than Java Spring Cloud LoadBalancer [10]. Google cloud load balancer is a cloud service and can also provide very good performance not bound by hardware limitations, but since the service is managed by a third party (vendor lock-in), this option scores lower than Nginx in terms of reliability. In terms of cost, Java Spring Cloud LoadBalancer and Nginx (open source) are free to use, therefore score higher than Google cloud load balancer, which incurs cost per uptime.

When scaling our backend service, we deploy multiple server instances on cloud and use **Nginx**'s Round-Robin as our load balancing policy.

4. Discussion and Project Timeline

4.1 Evaluation of Final Design

The design meets all essential functional specifications listed in section 2.1 as specified in each detailed design subsection. The non-essential functional specifications are mostly satisfied, except for FS18 (user preference based content sorting), which is addressed but is likely subject to changes in the design refinement phase. The final design also theoretically satisfies all non-functional specifications, although the performance related specifications (e.g. NFS2, FS13) must be further verified in the testing phase. Since all subsystems/modules of the design incurs no cost in the implementation phase and little cost in the deployment phase, NFS4 (cost constraint) is satisfied as well.

4.2 Use of Advanced Knowledge

The recommendation system design applies the knowledge from ECE457A with regards to AI algorithms. The backend server design employs the Java programming and project structuring knowledge from ECE351. The splitting of frontend and backend and the decision to add load balancer to the backend design is an application of knowledge from ECE454, distributed computing. The database and data structure design uses concepts from ECE356.

4.3 Creativity, Novelty, Elegance

The AR content manager module is the focus of our creativity. The module allows real-time content anchoring and scaling with regards to proximity, and also location-based content/comment feed based on user preference. Different from basic AR object rendering, our system allows custom placement and adjustment of AR contents, syncs regional content updates for all users, and displays only contents relevant to both user and location.

Another novelty is the use of geo-mesh in our backend design. When updating the display at user's displacement, we divide the user's surroundings into geo-mesh to allow geographically distributed content rendering around the user. This design prevents the frontend from rendering crowded and repetitive contents in one area, thus improves the elegance of display.

4.4 Student Hours

The total working hours each team member has put in so far is:

Alice	Michael	Aidan	Sean
70	110	45	40

4.5 Potential Safety Hazards

This project is purely software and does not introduce any safety hazard.

4.6 Project Timeline

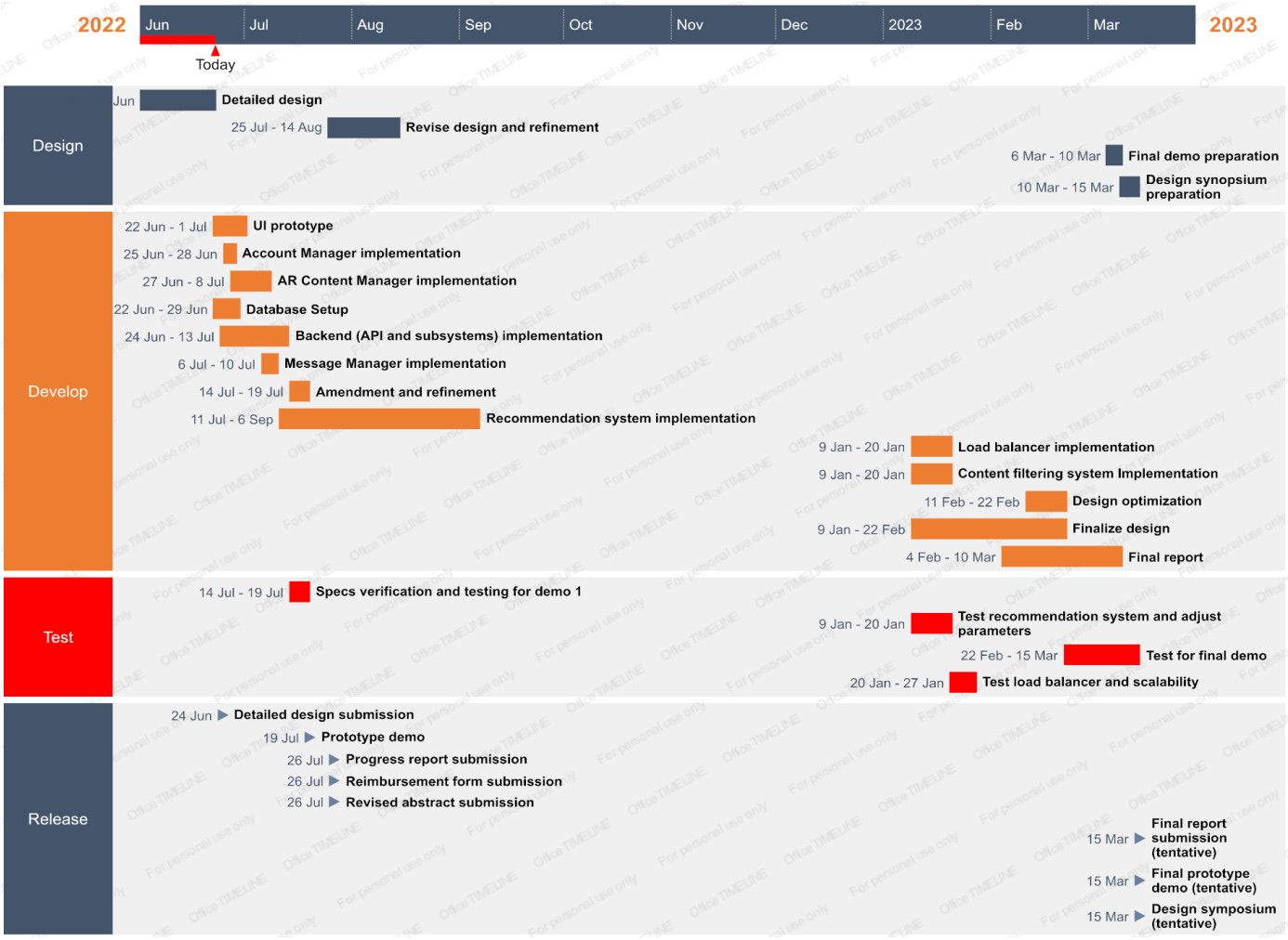

Figure 10: Project timeline.

References

1. Turner, A. (2022). *HOW MANY SMARTPHONES ARE IN THE WORLD?* Bankmycell. Retrieved May 23, 2022, from https://www.bankmycell.com/blog/how-many-phones-are-in-the-world
2. He, A. (2019, July 18). *People Continue to Rely on Maps and Navigational Apps*. eMarketer. Retrieved May 23, 2022, from https://www.emarketer.com/content/people-continue-to-rely-on-maps-and-navigational-apps-emarketer-forecasts-show
3. Ready, B. (2022, March). *2022 Retail Marketing Guide: Drive foot traffic and in-store sales*. Google. Retrieved May 23, 2022, from https://www.thinkwithgoogle.com/consumer-insights/consumer-journey/increase-foot-traffic-and-in-store-sales/
4. Alsop, T. (2021, November 29). *Number of mobile augmented reality (AR) active users worldwide from 2019 to 2024*. Statista. Retrieved May 23, 2022, from

https://www.statista.com/statistics/1098630/global-mobile-augmented-reality-ar-users/#:~:text=Global%20mobile%20augmented%20reality%20(AR)%20users%202019%2D2024&text=As%20per%20recent%20data%2C%20by,billion%20mobile%20AR%20users%20worldwide

5. Anonymous. *What are some alternatives to firebase authentication?* Stackshare. Retrieved June 9, 2022, from https://stackshare.io/firebase-authentication/alternatives
6. Isichko, D. (2021, October 19). *Downsides of firebase: Limitations to be aware of*. Medium. Retrieved June 9, 2022, from https://medium.com/moqod-software-company/downsides-of-firebase-limitations-to-be-aware-of-886ade5ae5a2
7. Rocca, B. (2019, June 2). *Introduction to recommender systems*. Towards Data Science. Retrieved June 8, 2022, from https://towardsdatascience.com/introduction-to-recommender-systems-6c66cf15ada
8. Rodreguaze, S. (2020, December 11). *Spring vs Django: Know the difference between the two*. eduwyre. Retrieved June 20, 2022, from https://eduwyre.com/article/spring-vs-django-differences-and-similarities
9. Bakshi, N. et al. (2020, June). *Spring Framework vs Django Framework: A Comparative Study.* IRJET. Retrieved June 20, 2022, from https://www.irjet.net/archives/V7/i6/IRJET-V7I61162.pdf
10. Turgay, Ç. (2017, December 11). *Comparing API Gateway Performances: NGINX vs. ZUUL vs. Spring Cloud Gateway vs. Linkerd*. Opsgenie Engineering. Retrieved June 20, 2022, from https://engineering.opsgenie.com/comparing-api-gateway-performances-nginx-vs-zuul-vs-spring-cloud-gateway-vs-linkerd-b2cc59c65369
11. Paialunga, P. (2022, January 16). *Hands-on content based recommender system using Python*. Medium. Retrieved June 22, 2022, from https://towardsdatascience.com/hands-on-content-based-recommender-system-using-python-1d643bf314e4
12. Aditya, P. H., Budi, I., & Munajat, Q. (2016, October). A comparative analysis of memory-based and model-based collaborative filtering on the implementation of recommender system for E-commerce in Indonesia: A case study PT X. In *2016 International Conference on Advanced Computer Science and Information Systems (ICACSIS)* (pp. 303-308). IEEE.
13. He, X., Liao, L., Zhang, H., Nie, L., Hu, X., & Chua, T. S. (2017, April). Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web* (pp. 173-182).
14. Rendle, S., Krichene, W., Zhang, L., & Anderson, J. (2020, September). Neural collaborative filtering vs. matrix factorization revisited. In *Fourteenth ACM conference on recommender systems* (pp. 240-248).
15. Hssina, B., Grota, A., & Erritali, M. Building Recommendation Systems Using the Algorithms KNN and SVD.
16. Bentley, J.L. Multidimensional binary search trees used for associative searching. *Commun. ACM 18*, 9 (Sept. 1975), 509-517.
17. Robinson, J.T. The K-D-B-tree: a search structure for large multidimensional dynamic indexes. In *Proceedings ACM SIGMOD Conference on Management of Data* (Boston, June 1984). ACM, New York, 1984, 10-18.
18. MongoDB. (2022). *MongoDB Pricing*. Retrieved June 15, 2022, from https://www.mongodb.com/pricing
19. Crowell, W. (2021, June 24). *PostgreSQL vs. MongoDB: Features and Benefits Comparison*. OpenLogic. Retrieved June 25, 2022, from https://www.openlogic.com/blog/postgresql-vs-mongodb
20. Sharma, A. (2021, December 3). *Redis vs MongoDB: 10 Critical Differences*. Hevo. Retrieved June 15, 2022, from https://hevodata.com/learn/redis-vs-mongodb/#rs
21. Redis. (2022). *Redis Pricing*. Retrieved June 15, 2022, from https://redis.com/redis-enterprise-cloud/pricing/
22. Geekboots. (2019, June 20). *Advantages and disadvantages of JSON over SQL*. Retrieved June 25, 2022, from https://www.geekboots.com/story/advantages-and-disadvantages-of-json-over-sql
23. SF AppWorks. (2022, January 25). *Redis vs. MongoDB: Which One Should You Choose?* Retrieved June 25, 2022, from https://www.sfappworks.com/blogs/redis-vs-mongodb-which-one-should-you-choose