Route 42 Design Document

Table of Contents

Team

Team Members and Roles

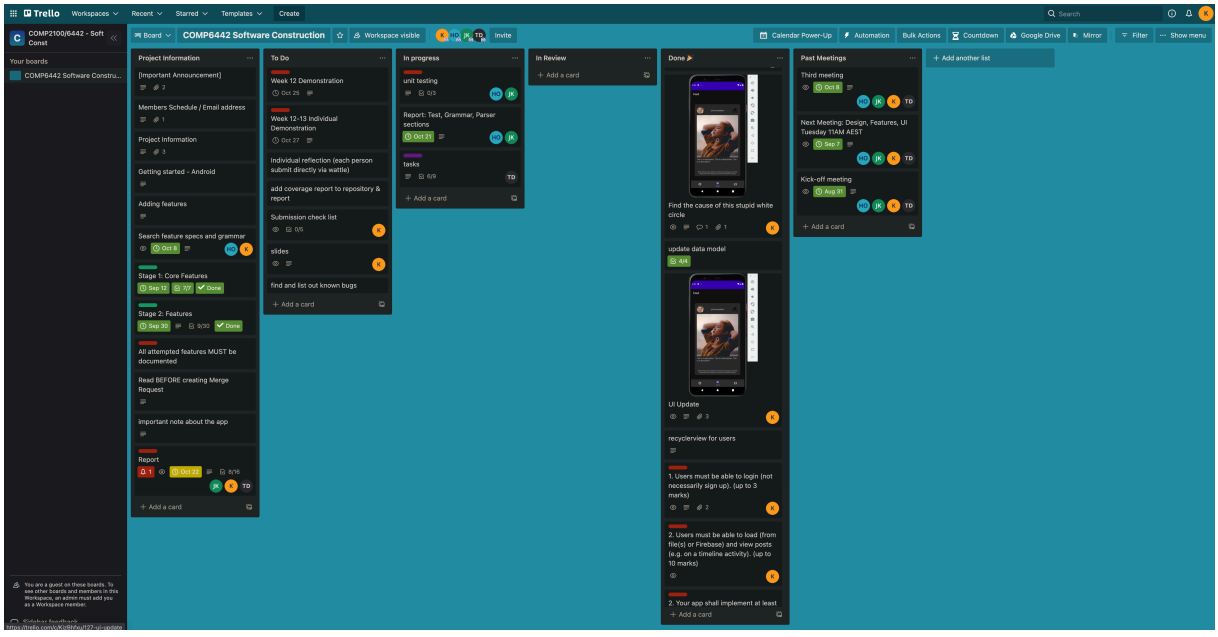
UID	Name	Role
u7233149	Kai Hirota	Full-Stack
u7269158	John (Min Jae) Kim	Data Structure, Feature Testing
u7234659	Honggic Oh	Search, Feature Testing
u7199021	Theo Darmawan	Full-Stack

Meeting minutes

- Meeting 1 - 31st August
- Meeting 2 - 7th September
- Meeting 3 - 8th October

Conflict Resolution Protocol

- Conflicts will be resolved through civil discussion and democratic voting process involving all parties interested in the matter.
    - For example, if someone wants to change the direction or the concept of the app, everyone must be involved in the decision-making. If someone wants to change a small class in the project, then that can be done either through voting, or by mutual agreement upon directly discussing with the person who created the class.
- Task assignments: Trello Kanban board



Application Description

Targets Users: Workout Enthusiasts

Route42 is a social networking app for athletes of various levels. With Route42, users can:

1. Record workouts, including walking, running, and cycling.
2. Track performance metrics and see the recorded workouts in an interactive map.
3. Follow other users, and view and like other people's workouts.
4. Search for posts by username, hashtags, and proximity to the user's location

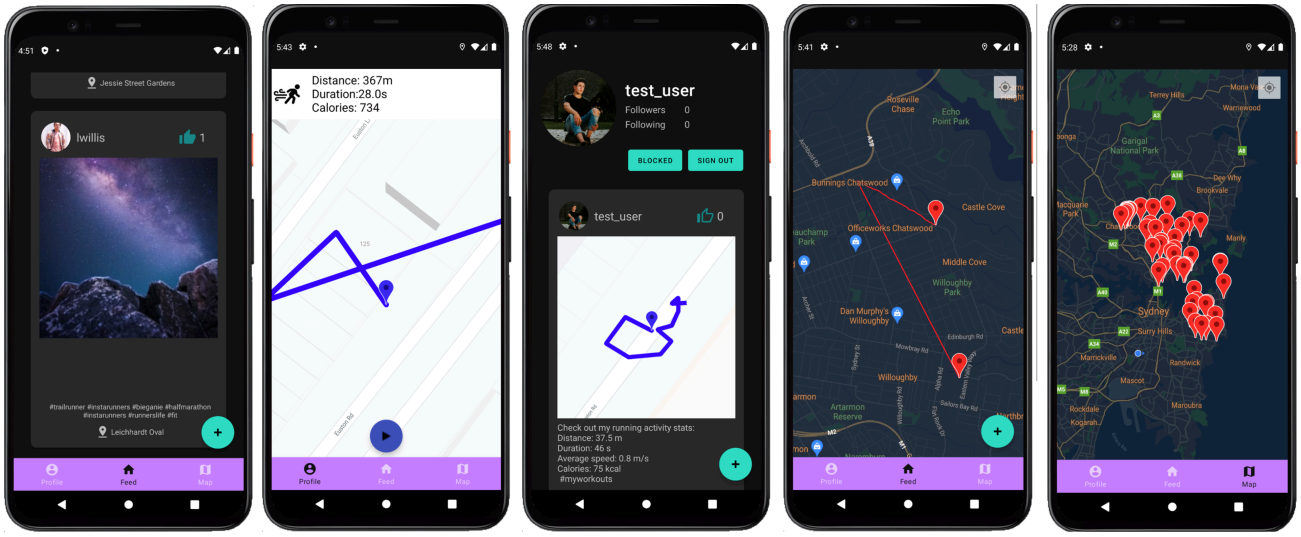
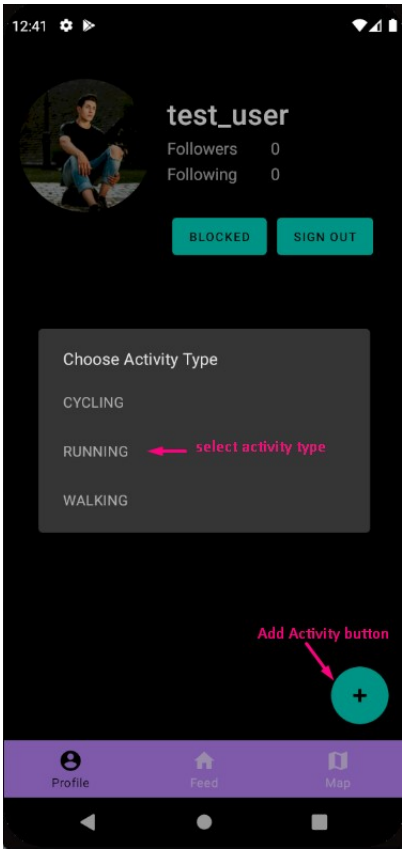


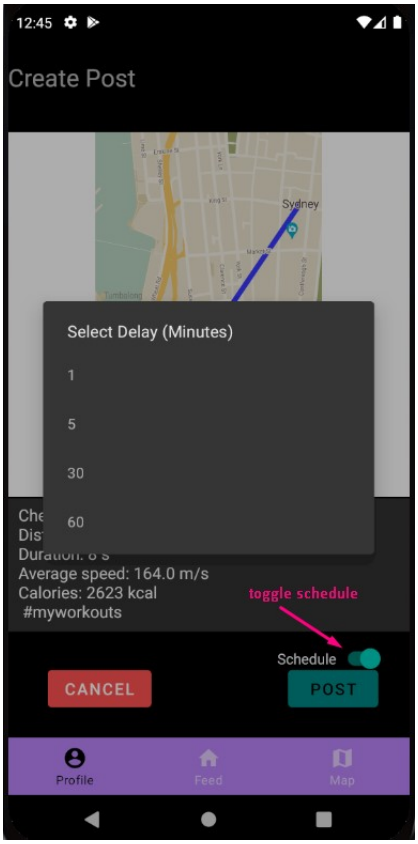
Image: Feed, Activity logging, Profile, Route, and Nearest Neighbor Search screens (Left to Right)

Use Case Example

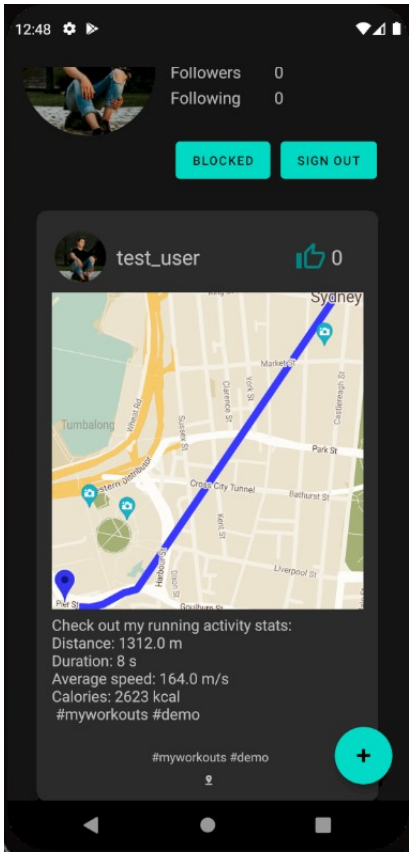
1. Athletic Activity Tracking and Sharing
    1. Michael runs at ANU running club, and wants to record and share his daily runs.
    2. Once ready, he starts the `run` activity on the app



3. The app tracks Michael's location and route, and display it on a map. Performance metrics are displayed in real time.
4. After finishing his run, Michael ends the `run` activity on the app.
5. The app will display a post template for sharing the completed activity.



6. Michael may write a description and add hashtags like `#ANUrunning` before sharing it. The app will extract the hashtags and tag the post for you.
7. The created post can immediately be viewed by other users on their feeds.



8. Michael can also select his past posts or posts created by others and see an interactive map of the route associated with the post.

2. Social networking and searching
    1. Emily is an avid runner who recently started competing in marathons. Emily wants to connect with other aspiring athletes.
    2. Emily can search for posts on Route42 app by username and hashtags, and look at other athletes and their workouts and routes.
    3. Emily can also search for posts by geographical proximity, and the app will visualize the places where others logged their workouts on an interactive map.

Scheduled Actions

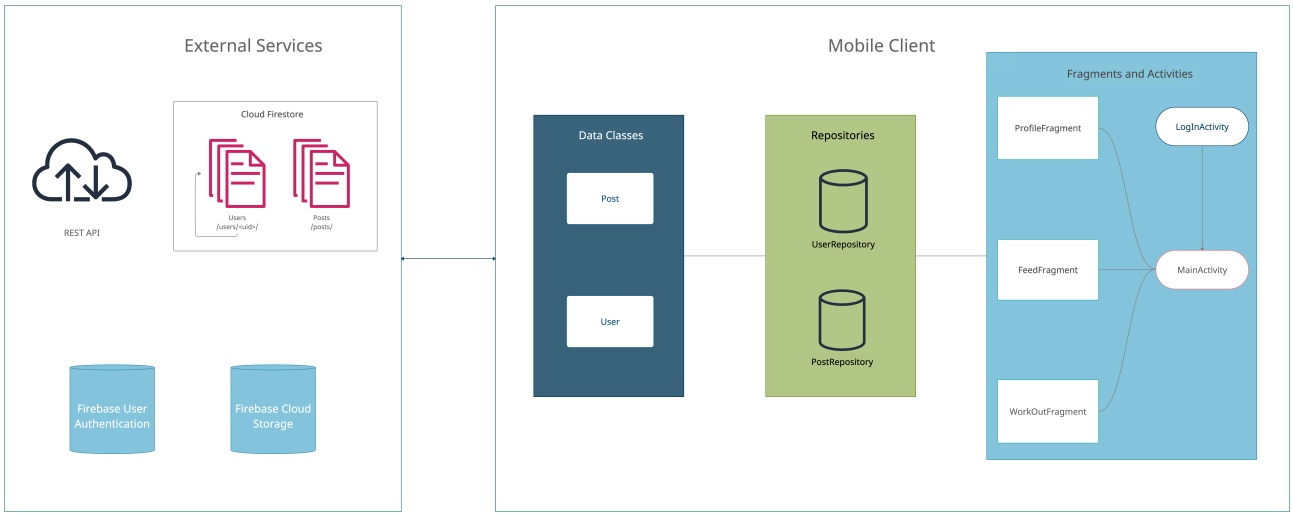
If the user does not have an active internet connection, the app allows scheduling of posts and likes. To schedule a post, the user checks the `schedule` button and selects the time delay. To schedule a like, the user long-clicks the like button and selects the time delay.

Pausing a workout

If the user needs to pause the workout, they can manually do so. Otherwise, navigating away from the `Activity` screen will automatically pause it for them.

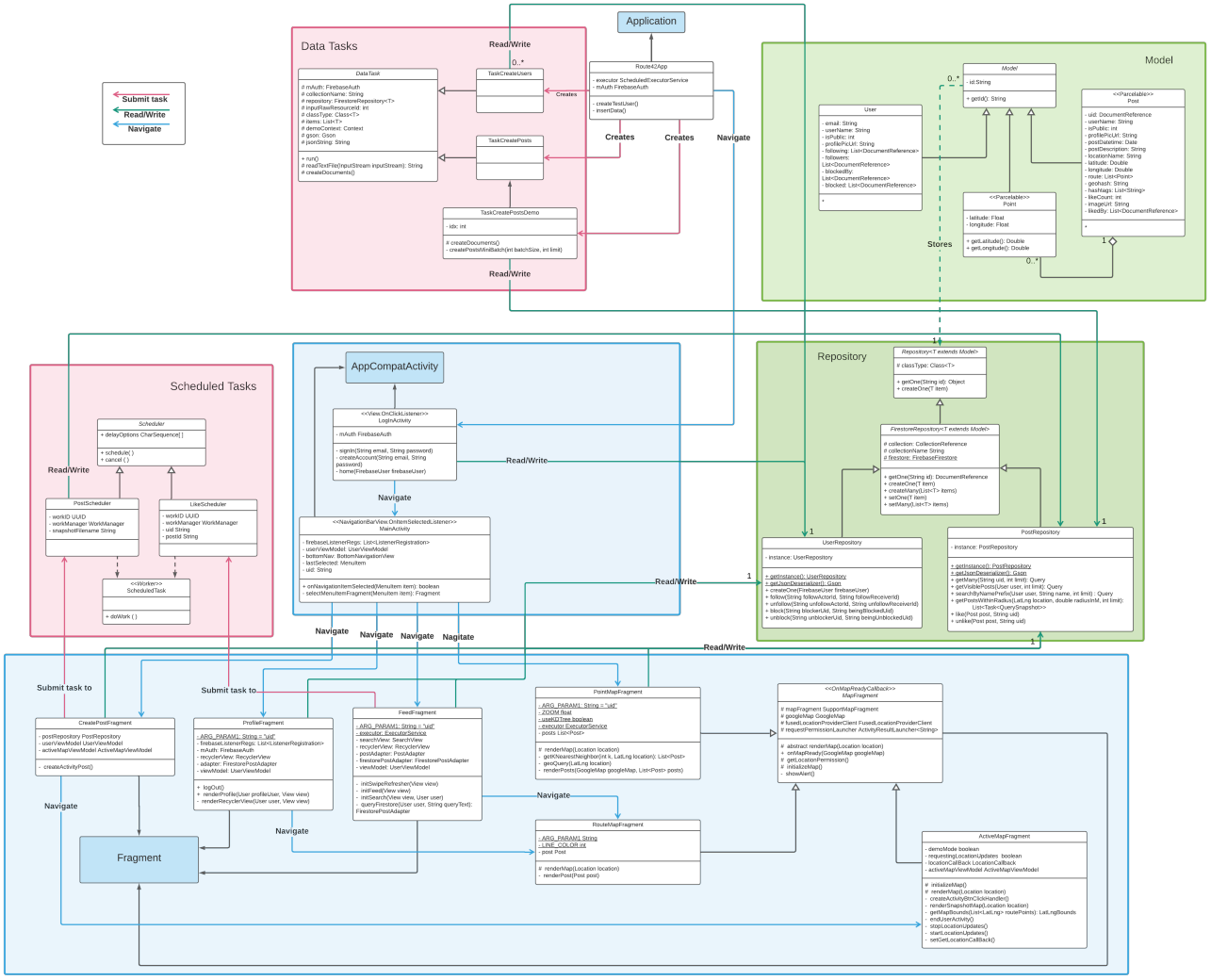
Diagrams

Architecture



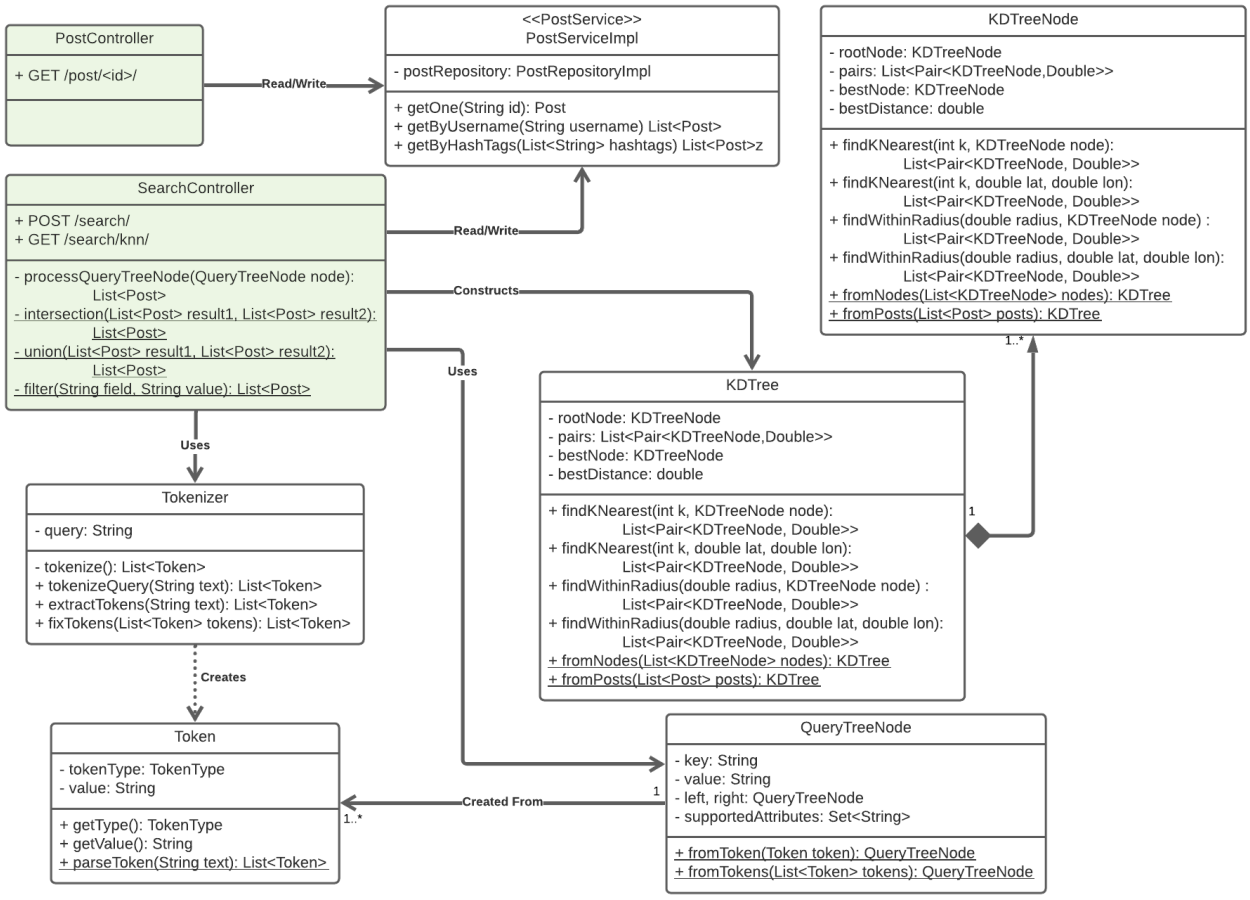
Link

Mobile App



[Link]

REST API



Link

Design Decisions

Data Structures

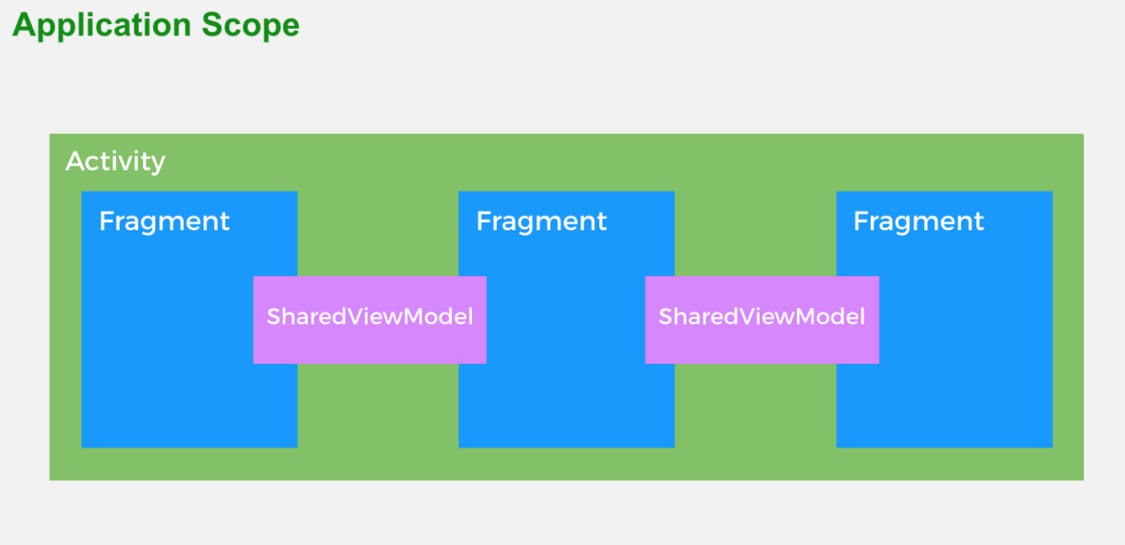
- KD Tree
    - Where: REST API `GET /search/knn` with `k`, `lon`, `lat` parameters.
    - Why: KD Tree (K-dimensional tree) is used to store and search 2-D data of location (longitude, latitude). KD tree is useful for finding nearest neighbors and performing range search based on multiple dimensions of data - such as longitude and latitude.
- HashMap
    - Where: Used by the REST API for union and intersection operations between lists of `Posts`.
    - Why: It's the most efficient way of finding set union and intersections. This is used to chain the left and right results when executing commands in `QuerySyntaxTree`.
- Binary Tree
    - Where:
        - REST API `POST /search/` endpoint uses `QuerySyntaxTree` to process the query text sent by
          a client.
        - `QueryTreeNode` is used to extract the hierarchical structure of nodes, each representing

a binary operator and two expressions.

- Why:
  - It allows efficient parsing of tokens, and allows us to express various operations in a recursive manner, making the code easy to read and maintain.

Design Patterns

- Singleton & Repository
  - Where: `UserRepository` and `PostRepository` classes under `repository` submodule.
  - Why:
    - Singleton pattern prevents unnecessary creation of multiple instances of connections with the database. By using singleton, the program uses less memory, and managing the connection with the database is easier.
    - Repository pattern abstracts the database operations and allows decoupling of the database access logic from the application logic.
- Single-activity architecture



  - Source
  - What: Composition of Android application based on single or a couple activities, each managing one or more fragments.
  - Where: Entire application.
  - Why:
    - Route42 primarily has one activity called `MainActivity` which contains a fragment container view, which swaps between fragments based on user interactions.
    - Usage of this architecture reduces lines of code required for the whole app, while making it easier to prototype new features.
- REST API
  - Where: In the cloud (AWS EC2 instance)
  - Why:
    - When using Cloud Firestore Android SDK, we have some limitations.

        - Cannot perform partial text search - for example, we cannot query on substring of a text field.
        - Cannot use more than one `arrayContains` in a single query.
        - No support for boolean OR operation between multiple filters.
        - Every CRUD operation must be asynchronous in order to not freeze up the UI thread.
    - Using the REST API allows us to use the Firebase-admin SDK, which gives the REST API higher privilege and more capability than the mobile client.
    - Using REST API allows us to simplify database reads and writes. The downside is that we sacrifice Firestore's document listener feature, where we can listen to updates on documents of interest.

- ViewModel
    - What: An intermediate observable class which enables data to persist independent of fragment lifecycle and attaching listeners to data changes.
    - Where: `ActiveMapViewModel` and `UserViewModel`
    - Why: By storing data in a view model class, data is not deleted when views are destroyed (e.g. when the user navigates to another page, or when the phone is rotated). Also, by listening to changes to `LiveData` members of the view model, views can update directly to changes in persistent data stored in Firebase, through listening to the `LiveData` class. This improves separation of UI layer from the data layer, as the UI is not dependent on any repository classes.
- Multi-threading / background execution
    - Where: `PhotoMapFragment`, `ScheduleablePost` , `SchedulableLike`
    - Why: When making the REST API call to `search/knn`, the communication is handled by a background worker thread. This ensures the UI thread (the main thread) does not freeze and remains responsive. Scheduled actions involving IO operations and network calls are also handled in the background to minimize load on the UI thread.

Grammars

- `<Term>  ::=  <Expr> | <Term> + <Term> | <Term> + <Operator>`
- `<Expr>  ::= <Keyword> | <bracket>`
- `<Operator>  ::=  <and> | <or>`

Advantage

- Parser Tree is a binary tree as opposed to being a n-ary tree, making it easier to construct the Parser Tree.

Disadvantage

- When multiple AND / OR operations are used in the query (i.e. `"hashtag: #running AND hashtag: #jogging"`), Parser Tree does not make optimizations. While Firestore supports `.arrayContains()` operation, our Parser Tree represent each `AND/OR` as a single node. In other words, `"hashtag: #running AND hashtag: #jogging"` could be represented as a single node in an n-ary tree, but

Tokenizer and Parsers

- Every token either contains an operator and two expressions, or a key and value.
- Tokens are extracted by prioritizing parenthesis, and then extracting from left to right.
- For example, if a query consists of 10 hashtags chained by OR, then the resulting `QuerySyntaxTree` will be equivalent to a linked list, where each node only has a right child.

Examples

```
1. "test test2" → {hashtags: ["test", "test2"]} →
Node(
    Node(null, "hashtags:test", null),
    OR,
    Node(null, "hashtags:test2", null)
)

2. "username: xxx AND hashtags: #hashtag1 #android #app" →
{OR: [
    {userName: "xxx"},
    {hashtags: ["#hashtag1", "#android", "#app"]}
  ]
}
Node(
    Node(null, "username:xxx", null),
    AND,
    Node(
        Node(
            null,
            "hashtags: #hashtag1",
            null
        ),
        OR, ,
        Node(
            Node(null, "hashtags: #android", null),
            OR,
            Node(null, "hashtags: #app", null)
        )
    )
)
```

Summary of Known Errors and Bugs

1. Base assumption of the app is that location data permission will be given, as a lot of core features depend on location data. As such, we did not place too much consideration on the case where the user declines location permission.

List all the known errors and bugs here. If we find bugs/errors that your team do not know of, it shows that your testing is not through.

Testing

Espresso UI Test

- For UI test was done on the test module `ui`. Following UI functionalities were tested :
    - Logging in
    - Changing the feed page
    - Creating different types of posts
    - Canceling posts
    - Creating scheduled post
    - Searching with specific hashtag
    - Clicking like, block, follow button for checking like/unlike, block/unblock, follow/unfollow
- Part of methods for UI Tests below methods which are based on uses references:
    - `MyViewAction`
        - Activates an item (e.g. like button, follow button etc) of specific post in recyclerview
    - `RecyclerViewMatcher`
        - Detects status of an item of specific post in recyclerview
        - Returns false if no item and input id is matched
    - `setChecked`
        - Sets status of an item

Unit Test

[all classes]

Overall Coverage Summary

Package	Class, %	Method, %	Line, %
all classes	93.3% (14/ 15)	87.7% (114/ 130)	74.5% (321/ 431)

Coverage Breakdown

Package	Class, %	Method, %	Line, %
com.comp6442.route42.api	100% (4/ 4)	100% (11/ 11)	76.9% (40/ 52)
com.comp6442.route42.data.model	90% (9/ 10)	86.6% (97/ 112)	73.1% (258/ 353)
com.comp6442.route42.utils	100% (1/ 1)	85.7% (6/ 7)	88.5% (23/ 26)

generated on 2021-10-22 11:30

`Data Model`

[all classes] [com.comp6442.route42.data.model]

Coverage Summary for Package: com.comp6442.route42.data.model

Package	Class, %	Method, %	Line, %
com.comp6442.route42.data.model	90% (9/ 10)	86.6% (97/ 112)	73.1% (258/ 353)

Class	Class, %	Method, %	Line, %
Activity	100% (1/ 1)	42.9% (3/ 7)	44.4% (8/ 18)
BaseActivity	100% (1/ 1)	100% (8/ 8)	100% (29/ 29)
Model	100% (1/ 1)	100% (2/ 2)	100% (4/ 4)
Point	100% (2/ 2)	75% (9/ 12)	61.3% (19/ 31)
Post	100% (2/ 2)	86.4% (38/ 44)	69.4% (109/ 157)
Schedulable	0% (0/ 1)	0% (0/ 1)	0% (0/ 1)
SchedulablePost	100% (1/ 1)	100% (11/ 11)	63.8% (30/ 47)
User	100% (1/ 1)	96.3% (26/ 27)	89.4% (59/ 66)

generated on 2021-10-22 11:30

`Api`

[all classes] [com.comp6442.route42.api]

Coverage Summary for Package: com.comp6442.route42.api

Package	Class, %	Method, %	Line, %
com.comp6442.route42.api	100% (4/ 4)	100% (11/ 11)	76.9% (40/ 52)

Class	Class, %	Method, %	Line, %
KNearestNeighbourService	100% (1/ 1)	100% (3/ 3)	64.7% (11/ 17)
QueryString	100% (1/ 1)	100% (4/ 4)	100% (8/ 8)
RestApiService	100% (1/ 1)	100% (1/ 1)	100% (12/ 12)
SearchService	100% (1/ 1)	100% (3/ 3)	60% (9/ 15)

generated on 2021-10-22 11:30

`Utils`

[all classes] [com.comp6442.route42.utils]

Coverage Summary for Package: com.comp6442.route42.utils

Package	Class, %	Method, %	Line, %
com.comp6442.route42.utils	100% (1/ 1)	85.7% (6/ 7)	88.5% (23/ 26)

Class	Class, %	Method, %	Line, %
Crypto	100% (1/ 1)	85.7% (6/ 7)	88.5% (23/ 26)

generated on 2021-10-22 11:30

Not covered in unit test:

- `ui.fragments` and `ui.activity` submodules are covered by Espresso UI tests instead.
- All classes `data/repository` module and `FirebaseAuthLiveData`
    - Testing repository class requires mocking `FirebaseFirestore`.
- `utils`
    - `MockLocation` requires to retrieve live location data
    - Submodules `tasks` and `xmlresource` requires mocking Firebase Firestore or a repository class.

Implemented Features

- Easy: 6
- Medium: 5
- Hard: 1
- Very Hard: 1

Improved Search

1. Search functionality can handle partially valid and invalid search queries. (medium)

UI Design and Testing

1. UI tests using espresso or similar. Please note that your tests must be of reasonable quality. (For UI testing, you may use something such as espresso) (hard)

Greater Data Usage, Handling and Sophistication

1. Read data instances from multiple local files in different formats (JSON, XML or Bespoken). (easy)
2. User profile activity containing a media file (image, animation (e.g. gif), video). (easy)
3. Use GPS information. (easy)

User Interactivity

1. The ability to micro-interact with 'posts' (e.g. like, report, etc.) [stored in-memory]. (easy)
2. The ability for users to ‘follow’ other users. There must be an adjustment to either the user’s timeline in relation to their following users or a section specifically dedicated to posts by followed users. [stored in-memory] (medium)
3. Scheduled actions. At least two different types of actions must be schedulable. For example, a user can schedule a post, a like, a follow, a comment, etc. (medium)

User Privacy

1. Privacy II: A user can only see a profile that is Public (consider that there are at least two types of profiles: public and private). (easy)

Peer-to-Peer Messaging

1. Privacy I: provide users with the ability to ‘block’ users. Preventing them from directly messaging them. (medium)

Firebase Integration

1. Use Firebase to implement user Authentication/Authorisation. (easy)
2. Use Firebase to persist all data used in your app (this item replace the requirement to retrieve data from a local file) (medium)
3. Using Firebase or another remote database to store user posts and having a user’s timeline update as the remote database is updated without restarting the application. E.g. User A makes a post, user B on a separate instance of the application sees user A’s post appear on their timeline without restarting their application. (very hard)

Appendix

Third-party libraries used

Android App

```
implementation 'androidx.annotation:annotation:1.2.0'
implementation 'androidx.activity:activity:1.2.0'
implementation 'androidx.fragment:fragment:1.3.0'
implementation 'androidx.appcompat:appcompat:1.3.1'
implementation 'androidx.constraintlayout:constraintlayout:2.1.0'
implementation "androidx.lifecycle:lifecycle-common-java8:2.3.1"
implementation 'androidx.lifecycle:lifecycle-livedata-ktx:2.3.1'
implementation 'androidx.lifecycle:lifecycle-viewmodel-ktx:2.3.1'
implementation 'androidx.multidex:multidex:2.0.1'
implementation 'androidx.legacy:legacy-support-v4:1.0.0'
implementation "androidx.work:work-runtime:2.7.0"

// ——— utils ———
implementation 'com.google.code.gson:gson:2.8.8'
implementation 'com.google.android.material:material:1.4.0'
implementation 'com.jakewharton.timber:timber:5.0.1'

// ——— Navigation Component ———
def nav_version = "2.3.5"
androidTestImplementation "androidx.navigation:navigation-testing:$nav_version"
implementation "androidx.navigation:navigation-compose:2.4.0-alpha08"
implementation "androidx.navigation:navigation-dynamic-features-fragment:$nav_version"
implementation "androidx.navigation:navigation-fragment:$nav_version"
implementation "androidx.navigation:navigation-ui:$nav_version"

// ———————— Glide ————————————
implementation 'com.github.bumptech.glide:glide:4.12.0'
annotationProcessor 'com.github.bumptech.glide:compiler:4.12.0'

// ———————— Google Maps ————————
implementation 'com.google.android.gms:play-services-location:18.0.0'
implementation 'com.google.android.gms:play-services-maps:17.0.1'

// ———————— REST API ————————————
implementation 'com.squareup.retrofit2:retrofit:2.8.0'
implementation 'com.squareup.retrofit2:converter-gson:2.8.0'
implementation 'com.squareup.okhttp3:logging-interceptor:3.12.7'

// ———————— Firebase ————————————
implementation platform('com.google.firebase:firebase-bom:28.4.0')
```

```
implementation 'com.google.firebase:firebase-analytics'
implementation 'com.google.firebase:firebase-auth'
implementation 'com.google.firebase:firebase-firestore:23.0.3'
implementation 'com.firebaseui:firebase-ui-firestore:6.2.1'
implementation 'com.google.firebase:firebase-storage'
implementation 'com.firebaseui:firebase-ui-storage:7.2.0'
implementation 'com.firebase:geofire-android-common:3.1.0'

// ———————— Tests ————————————
testImplementation 'junit:junit:4.+'
androidTestImplementation 'androidx.test.ext:junit:1.1.3'
androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
androidTestImplementation 'androidx.test.espresso:espresso-intents:3.4.0'
androidTestImplementation 'androidx.test:runner:1.4.0'
androidTestImplementation 'androidx.test:rules:1.4.0'
implementation "androidx.profileinstaller:profileinstaller:1.1.0-alpha04"```
```

REST API

```
implementation 'com.google.firebase:firebase-admin:8.1.0'
implementation 'org.springframework.boot:spring-boot-starter-web:2.5.5'
implementation 'org.springframework.cloud:spring-cloud-gcp-starter-
firestore:1.2.8.RELEASE'
developmentOnly 'org.springframework.boot:spring-boot-devtools:2.5.5'
```