List of challenges with visuals.

1.Introduction

a. Group Members

   i. Melody B. (Prarin Behdarvandian), Jardi Martinez Jordan, and Breanna Powell

b. Pitch

   i. Did you know that 6 out of 10 people with dementia will wander off at some point in their life? Do you have that one friend in your circle that likes to wander off? Do you suffer from a lack of sense of direction? *MeetMe* is the app for you! Designed to help you and your loved ones, dependants, or associates to never lose each other! *MeetMe,* stay together.

c. Target Audience

   i. Initially the goal was to have the target audience be for friend groups and conference groups going on trips together or for caretakers to be able to track the location of their patients in case they are not aware of where they are. With feedback being given by others, the target audience has potential to expand to child care, long distance trips, and for the safety of solo travelers to be tracked by relatives.

   ii. **P0 - Caretakers who have dependents / groups of dependents**

      1. Why? Although there are already apps for families and friends to use to find each other, there are not as many targeted toward people who are caretakers or babysitters who are not necessarily close friends / family and do not necessarily want to track someone all day. Plus, if the user is a caretaker for someone with dementia, then calling the person to find out where they are may not be the best idea – it may startle the person unnecessarily or the patient might not be able to answer.

      2. How? We could partner with a website like Care.com or APlaceForMom.com for advertising.

   iii. **P0 - Conference / Convention / Comicon / Concert attendees**

1. Why? People traveling in groups on business or for pleasure to a conference or convention want to stick together. They don't want to have to call each other and risk disturbing other conference attendees while trying to locate each other and they might not be able to hear each other if they were to call.
2. How? We could seek out large conferences or conventions and see if they would help us advertise our app for their in person attendees. For concert goers, we could advertise on music sites, with concert venues, and/or with music artists.

iv. **P1 - Vacationers / Travelers / Tourist groups: People who want to stick together and not get lost while traveling and touring new and unfamiliar places.**

1. Why? Many people travel in tourist groups with companions who are not necessarily friends/family, and there is a great chance of getting lost while traveling. It might be a large group of 10-20 people. If they are far out on their phone plan, they may not want to call everyone.
2. How? We could advertise with tourism companies that offer group tours.

v. **P2 - Other potential future demographics: educators/chaperones who take their students on field trips, clubs, nature hiking groups, snowboarders!**

d. Primary Goals

i. Pair Tracking

ii. Caretaking tracking

iii. Future Goal: Group tracking

✅Does it accomplish the stated objective? (10 points)

- It's on its way to the stated objective. It does track the main user successfully. The framework to have two devices communicate is there in the database and API, but with

the time that we had we were not able to successfully implement the HTTP communication. All we need to do is add the 2nd person's location once we get cronet to work.

e. Value Prop

i. People get lost and scared, especially when they are in new locations or huge crowds or alone, and they need a way to be able to find each other.

✅**Does it deliver on the "value proposition"? (10 points)**

- Again, it's almost there. It does track one person.
- It easily allows two people to connect across a great distance though the initial handshake using intents and deep links.
- It has an easy and simple interface so that people will not stress out even more if they are lost.
- People are now very familiar with sending links to each other via any type of social app. We are tapping into a flow that people are already familiar with. For example, for a Zoom meeting, all you have to do is send a deep link to the other person. Our app works the same way.

f. Success Criteria

i. Accuracy of tracking - main complaint in a lot of tracking apps was that the there was a lack of accuracy in their apps and it was very late on the updates in terms of the current location of the device

ii. Public Good

g. Happy Path

Our happy path is from 2 perspectives: the Finder and the Wanderer.
Any Wanderer can be a Finder and vice versa.

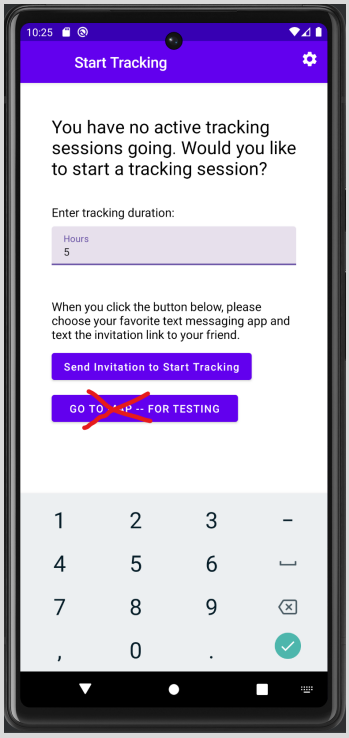
States to save: not tracking, waiting for consent, tracking in progress, giving consent

States to not save: tracking is ended

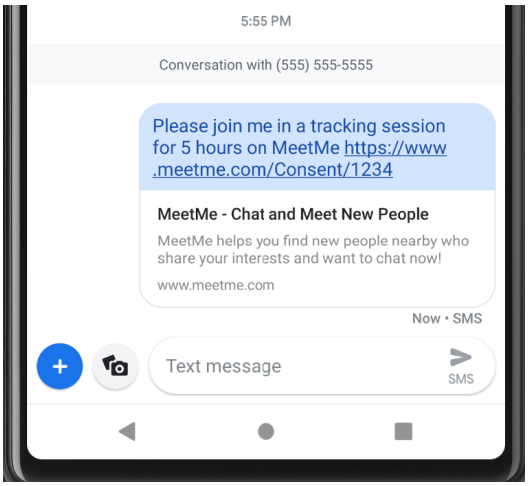
(***We don't need to save this because we can just return to the start screen – like a "Game Over" page)

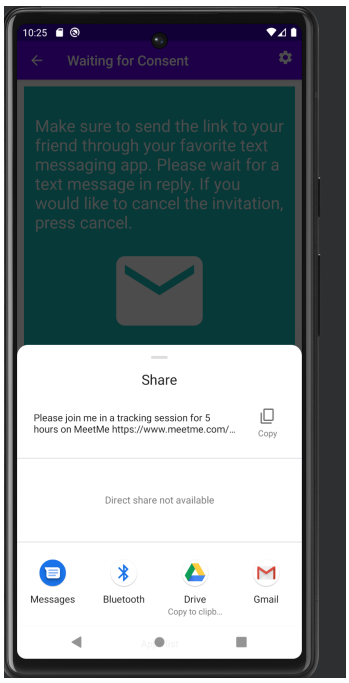
The Finder's perspective:

1. Start tracking session / **NOT TRACKING**
    a. Screen says "You have no active tracking sessions going. Would you like to start a tracking session?"
    b. Screen has an area to input the duration of the tracking.
2. The Finder inputs the duration of the tracking session
3. (***In the future, we would also like to display the number of FREE hours remaining, since we are using a Freemium model. If the finder exceeds FREE limit, the Finder sees the Bill amount)
4. The Finder clicks on "Send Invitation to Start Tracking"

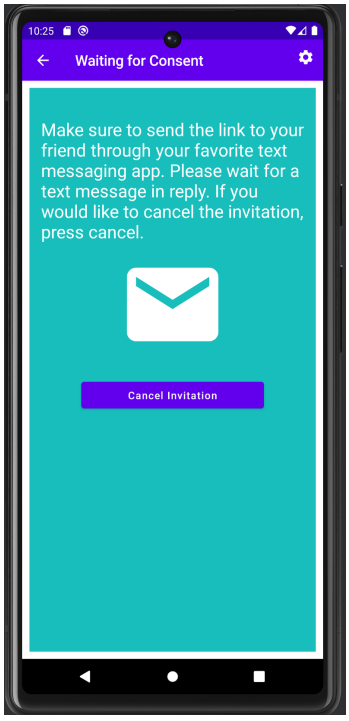


5. The user's phone opens up a chooser element to be able to choose an app through which to send the link
    a. The user chooses an app to use
        i. If sending through SMS, the user will choose a conversation from their text messaging app and send the link.

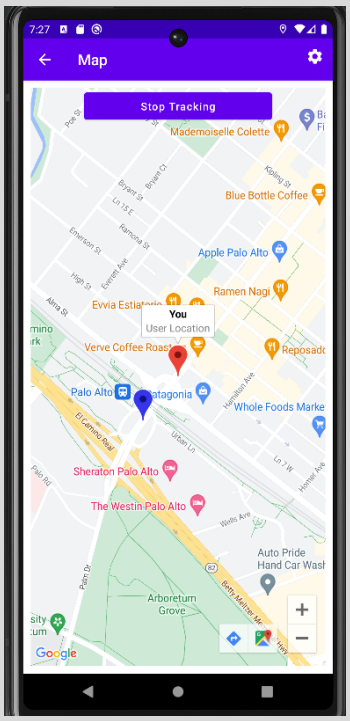




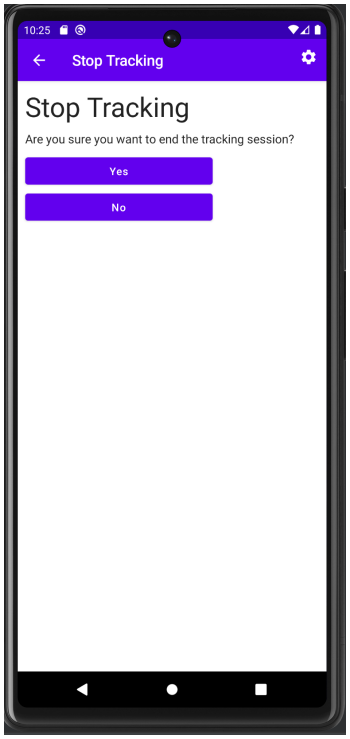
6. Waiting page / **WAITING FOR CONSENT**
    a. The Finder can see a message that says "Make sure to send the link to your friend through your favorite text messaging app. Please wait for a text message in reply. If you would like to cancel the invitation, press cancel."
    b. There's an option to return to the previous screen and/or cancel the invitation



7. Map page / **TRACKING IN PROGRESS**:
    a. The Finder gets a text message invitation with a link to the map. They open the link.
    b. The tracking session has started
    c. The Finder can see a map and a pin for their location
    d. (***In the future, 2 pins:
    one for the Finder and one for the Wanderer
        i. Displays color-coded map markers
        ii. Displays distance from their location to the location of the Wanderer they are trying to find.
        iii. Displays the route between the 2 pins.
        iv. Displays the “time remaining” at the top.)
    e. Displays a “stop tracking” area to end the session early.
    f. The Finder ends the tracking session.

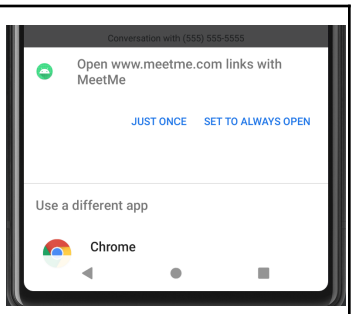


8. Stop tracking screen / **TRACKING IS ENDED**
    a. Two possibilities: Time is up or Finder or Wanderer ended the session prematurely
        i. **Ended prematurely:** A pop up window asks “Are you sure you want to end the tracking session? Yes / No.
            1. Yes: The tracking session is ended
                a. Return to Start Tracking Session
            2. No: The tracking session continues
                a. Return to the Maps screen
        ii. (***In the future,
            1. **Time is up:** The timer ran out. A pop up window says “There is no time remaining. The tracking session has ended.”
            2. Return to the Start Tracking Session
        iii. **Payment Confirmation Screen - not implemented yet**
            1. **Displays the bill)**
        iv. The app returns to the Start Tracking screen



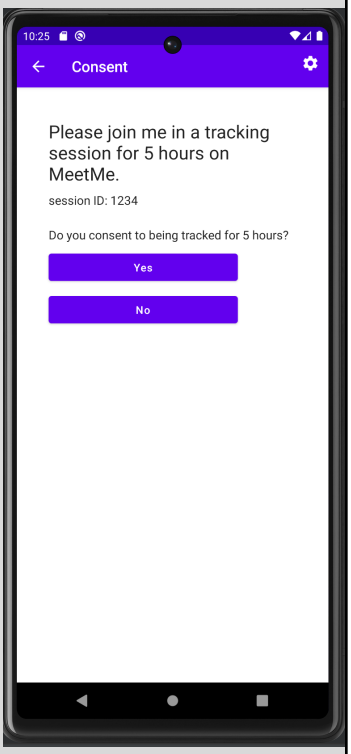
The Wanderer's Perspective:

1. The Wanderer gets a text message from the other person. They see the link. When they click the link the first time, the following screen pops up →
2. The link takes them to the Consent page.



1. Consent page / CONSENT TO BEING TRACKED
2. A notification pops up that says "Your friend %s has invited you to a tracking session. Do you agree? Yes / No."
3. The Wanderer agrees OR disagrees
    a. **Agrees**: The Wanderer is taken to the Map Screen.
        i. Map Page: The Wanderer sees the maps page with the same view as the Finder. They may choose to end the session at any time.
    b. **Disagrees**: The Wanderer returns to the Start Tracking Screen since no tracking sessions are in progress.

(The rest of the Wanderer's views will be the same as the Finder's perspective).



User Experience

Does it have a good user experience? (5 points)

- Yes, because the design is built on intuitiveness and familiarity of the map. The map is the google map, which is an already established and well known visual layout. Bonus is that additionally thanks to google maps, we have other indicators and markers that will help the user identify their location better.

- One concern at the moment is that we do not own www.meetme.com, so when a user opens our app with any device above API level 31 will resolve a web intent using the web browser rather than allowing the user to open our app. It takes them to a dating app that is being beta tested!!! We will need a new domain / app name. We had to use the Pixel 6 at API level 29 to get it to work.

> **Note:** Starting in Android 12 (API level 31), a generic web intent resolves to an activity in your app only if your app is approved for the specific domain contained in that web intent. If your app isn't approved for the domain, the web intent resolves to the user's default browser app instead.

- Just to make a note of testing for this app - when running the app with android studio, the location of the device is pre-selected by the emulator and will show the user that location.

2. Logistics

a. Platform - Android

   i. We went with Android because there were fewer friend tracking apps available. All the competitors we found had flaws with accuracy and with consent management, so we felt we could address both of those problems with our app.

b. Language - Kotlin with Jetpack Compose UI

   i. We wanted to try Jetpack Compose for our UI because it seems so versatile and is the current trend in the industry. Compose keeps track of state by implementing a back stack and also by only re-composing when a change has occurred.

✅**Does it have proper app lifecycle/state management? (10 points)**

- Yes.
- Instances in which our app does a proper use of lifecycle/state management are:
  - Keeping track of the user location
  - Maintaining the current state (No Tracking, Waiting, Tracking) in between switching app screens and after app termination.
  - When a user partially enters inputs and during input validation.

c. Manifests and Gradle Additions

i. This section of the code contains accesses and permissions needed in order for the app to work and again permission to use applications on android devices like the GPS.

d. App Navigation

i. Introduction: Jetpack Compose has its own navigation.

ii. How it works/what was done: The navigation requires a NavHost function to handle all of the navigation.

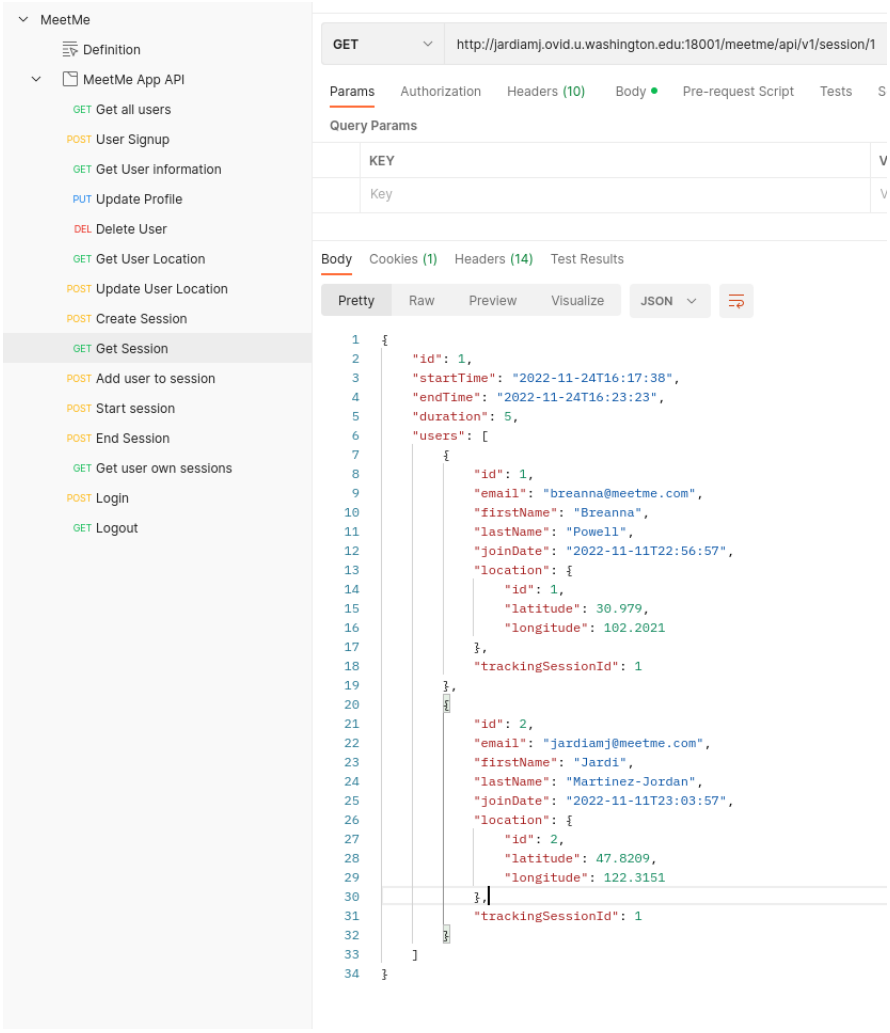
iii. Challenges: We had to watch a bunch of tutorials to learn how Jetpack Compose works and how to set up the navigation. See the Deep Links section for more challenges involved.

e. API

i. Introduction: We needed to allow users to communicate their location with each other, but we couldn't find a tool or platform that provided something we could leverage. Therefore, we decided to develop our own API for this purpose.

ii. How it works/what was done: The API is a set of end-points that accept HTTP requests to Create, Retrieve, Update, and Delete (CRUD) resources in a database. It was designed to handle group sessions where a user can have ownership over multiple sessions. Also, multiple users can be part of a tracking session they have

been invited to. The following is a screenshot of one end-point as it was being tested in postman.com:



iii. Challenges: The main challenges when developing the API were hosting it and the **security** considerations when storing sensitive data such as user location. The user password is saved in the Database in an encrypted form using a one way encryption protocol (BCrypt) as a security measure. All the provided end-points were evaluated from a security perspective, and we put measures in place to ensure that users can only access and manipulate data they either own or have rights to (like being part of the same tracking session).

f. Intents

i. Introduction: We decided that we do not want to require tons of permissions for our app to run, so using an Intent to open up a text messaging or other social app to share a link with another person seemed the least intrusive way to go. An Implicit Intent opens an external app (such as a text messaging app). An Explicit Intent opens up our app.

ii. How it works/what was done: The Implicit Intent needs the context from our app passed to it. The context is collected in the MainActivity.kt file and then passed to MeetMeScreen.kt as a parameter for the MeetMeApp function (where all navigation lives). Inside the MeetMeApp function, we call a custom function named "sendIntent". The sendIntent function takes in the context and a message (which contains the link). It creates an Intent object and applies ACTION_SEND (which enables it to send through any app that can handle the request) and putExtra (which attaches the message to the Intent. It also permits reading a URI and sends the message as plain text. Then there is a createChooser for the intent that allows the user to be able to pick an app to use to send the message. After that, startActivity will launch the Intent.

```
366  fun sendIntent(context: Context, message: String){
367      val sendIntent: Intent = Intent().apply {  this: Intent
368          action = Intent.ACTION_SEND  //Limit to the SMS apps
369          putExtra(Intent.EXTRA_TEXT, message) //Works
370          flags = Intent.FLAG_GRANT_READ_URI_PERMISSION //Works
371          type = "text/plain"
372      }
373
374      val shareIntent = Intent.createChooser(sendIntent,  title: null)
375      startActivity(context, shareIntent,  options: null)
```

iii. Challenges: Originally, we wanted this to open only on SMS (not any other type of app) so we attempted to limit and not use a "chooser" element. We also limited it to sending to only one phone number. However, putExtras was not working with that configuration, so the text message would not be pre-populated with our

message with the link! Therefore, we pivoted and focused on making sure that the link could be sent. The link had to be sent without specifying a specific phone number and a specific app. This problem was a blessing in disguise, because the way we have it arranged now will be more scalable. The user can open up whatever app they want to send the message. The user can also send the link to however many people they want!

✅Does it respect user privacy/is it secure? (10 points)

- We only access permissions for the location, not for contacts or for displaying push notifications.
- Everything is consent-based. Both parties need to agree to be tracked in order for the session to take place.

g. Deep Linking

i. Introduction: A deep link will allow another user to go directly to a certain page in our app. We first send a deep link to User 2 for the Consent Page. Next, we have User 2 send a link back that goes to the Map Page.

ii. How it works/what was done: The Deep Linking requires an Explicit Intent to be handled by our app. First, the app needs to advertise to the device that it can handle any link that contains our URI with the host of www.meetme.com and the scheme with either http or https. This advertising happens by adding an intent-filter within the AndroidManifest.xml file. Next, we need to have each of the composable functions within our NavHost that need to be specific to the user's particular session, like the Consent Page and the Maps Page, to parse the uriPattern and send the user to the page that is specific to their session.

```
composable(route = MeetMeScreen.Consent.name,
            arguments = listOf(navArgument( name: "sessionID") { type = NavType.IntType}),
            deepLinks = listOf(navDeepLink { uriPattern = "$uri/Consent/{sessionID}" })
) {  it: NavBackStackEntry
```

iii. Challenges: Learning about the whole process and how to create a link in the first place was hard. Also, figuring out how to pass the arguments to the composable navigation was difficult. At one point, the app navigation broke because it did not want to open up a composable that had arguments attached.

h. Mapping

i. Introduction: It was decided that in order to keep the goal of having the app be user friendly, to implement and use Google map for the map layout since that would be more familiar to users and would remove the need to create a map from scratch. There were two potential options to use for the google map which were the fusedlocationproviderclient and the location manager, the former is a google play API that allows access to the GPS via google play while the location manager gets it directly from the android device. A main complaint that the fusedlocationproviderclient received was that it would be slow in terms of updating the map, and given that a lot of location apps made the same complaint, the decision was to use the location manager instead.

ii. How it works/what was done: The location manager is created in a view model The location is saved as a mutable state of LatLng in order to allow the latitude and longitude of the GPS location to be taken in. At the initial start the function getLocation will be called to start the location manager and grab the last known location of the device as the initial location before it begins to calibrate to the actual location of the device, after it has been initialize, a location listener called onLocationChanged will be constantly checking the location of the device and updating it as it sees fit.

```
@SuppressLint("MissingPermission")
fun getLocation(){
    try {
        var locationManager = context.getSystemService(LOCATION_SERVICE) as LocationManager
        locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, minTimeMs: 5, minDistanceM: 1f, listener: this)
        location = locationManager.getLastKnownLocation(LocationManager.GPS_PROVIDER)!!
        onLocationChanged(location)
    } catch (e: Exception) {
        e.printStackTrace()
    }
}
override fun onLocationChanged(location: Location) {
    _latLong.value = LatLng(location.latitude, location.longitude)
    _latLong2.value = LatLng( latitude: location.latitude-.001, longitude: location.longitude-.001);
}
```

iii. The actual map itself is in a compose file which will call on the view model to initialize the location manager. From there, the location of the device will be taken and set a marker on the map to be viewed. Currently the map has two markers to indicate the main device and the companion device on the same map to show that there can be more than one marker on our map. Should this project evolve from a pair tracking to group tracking, we can effectively add more markers to accommodate any group size.

```
fun GoogleMapView (
    location: LatLng, friendLocation:LatLng
) {
    val labelState = MarkerState(position = location)
    val friendState = MarkerState(position = friendLocation)
    val cameraPositionState = rememberCameraPositionState {  this: CameraPositionState
        position = CameraPosition.fromLatLngZoom(location,  zoom: 17f)
    }
    GoogleMap(
        modifier = Modifier.fillMaxSize(),
        cameraPositionState = cameraPositionState
    ) {
        Marker(
            state = labelState,
            title = "You",
            snippet = "User Location"
        )
        Marker(
            state = friendState,
            title = "Friend",
            snippet = "Friend Location",
            icon = BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_BL
        )
    }
}
```

iv. Challenges: The biggest challenges for the map were lack of versatile documentation and setting up gradle and framework properly in order for the map to show on the app and run. With setting up the gradle and framework, the biggest issue was if the metadata was not inputting in the correct spot, the app would just crash and the debugger just would not catch the actual error as you would step into the function. This caused the project to almost be scrapped and restarted until the problem was discovered due to rebuilding the map on a new project.

```
<!-- MAP META DATA -->
<meta-data
    android:name="com.google.android.geo.API_KEY"
    android:value="${MAPS_API_KEY}" />

<!-- Apache HTTP Legacy library for map-->
<uses-library
    android:name="org.apache.http.legacy"
    android:required="false" />
```

v. The main problem with the documentation was that a lot of the documentation for the map building online was focussed on creating the map as a main activity rather than a composable. The overall design of both Kotlin and Compose tutorials is set to be used by users who have a long history of using it, which causes this expectation that anyone looking for any guides online, to have a firm grasp of kotlin.

i. Cronet

i. Introduction: To communicate between devices, we are going to make use of our home brewed API developed as a microservice and hosted in a remote server. Each device must make the appropriate HTTP requests to create, update, retrieve information. Cronet is a networking library provided by android with HTTP3 support, which we are using to interface with the aforementioned API.

ii. How it works/what was done: The CronetEngine builder will be initialized in the map view model in order to send HTTP requests to and from the database in order to retrieve the other device's location and update the current device's location on the database. The request will be done in JSON strings where the location and the user ID are the main required pieces of data in order to update the database table.

1. Current steps needed to get cronet to work: there needs to be a call back structure that will have all of the basic calls to the url request and this will just be like a lifecycle design of each request. Then build a cronet engine which will be connected to the database that will take in each request as a thread via an executor which is tracking the progress of each request.

iii. Challenges: The current challenge is getting the HTTP requests to work. A main issue has been that online resources are either difficult to come by or deprecated. For newer people it's a little bit harder to understand because the tutorials don't go into depth of what

3. How to run the app

a. Settings

i. Emulator: Pixel 6 API 29 for correct results otherwise you will be taken to a dating website thanks to the web intent settings. We need to come up with a new domain name.

1. The default location in the emulator is preset to whatever location is predefined on your chosen emulator. The group as a whole had the location in California as the default location.
2. Also without the API key, the map will not work.

ii. Note: Keep in mind that currently the HTTP requests do not work but we are able to show more than one marker on the map

✅Does it address at least three additional challenges (other than privacy and state management) unique to mobile app development? (15 points)

- Security
- Device Communication

- Location Sensor/Permissions
- Framework/Dependencies
- Scalability