VRIJE
UNIVERSITEIT
BRUSSEL

MULTIMEDIA

App: MIVB Contol

Seppe Goossens, Sam Dilmaghanian

21 december 2022

Academie jaar 2022-2023

Industriële Wetenschappen ELO-ICT

1 Intro

Onze app: MIVB Control, is een gekend concept in Brussel. In 2012 was er al een applicatie die de gebruikers informeerde over de controles van het Brussels openbaar vervoersnetwerk MIVB-STIB.
Uit het Nieuwsblad (2012)[1] *"De mobiele applicatie 'Contrôle STIB', die sinds december 2012 gebruikers informatie laat uitwisselen over controles op het Brussels openbaar vervoer, is al meer dan 30.000 keer gedownload. Oprichter Olivier Kaisin merkt net als vorig jaar een piek bij het begin van het nieuwe schooljaar."*

De applicatie telde al gauw 120.000 gebruikers (Nieuwsblad, 2012) wat stabiel uitgroeide naar meer dan 300.000 in 2016 (Bruzz, 2016)[2].
Het concept was simpel. De gebruikers gaven controles van de MIVB zelf aan. Dit werd dan opgeslagen en door gecommuniceerd naar de andere gebruikers. Een app om zwart te rijden.
Uit de lokale krant Bruzz (2016): *" Gemiddeld raadplegen wel 3.500 mensen per dag de applicatie om te checken aan welke bus- of metrohaltes van de MIVB er controles plaatsvinden. En dat brengt extra kosten met zich mee. 'Onze community groeit dagelijks. Het geld dat we nu bij elkaar zoeken dient om onze servers te versterken. Ook het onderhoud van de app kost ons gemiddeld 200 euro per maand en dat betaalden we tot nu toe uit eigen zak', legt Emile-Victor uit."*

We vinden online geen nieuwe informatie over *"Contrôle STIB"*. De app bestaat niet meer en is nu vervangen door onofficiële FaceBook groepen en Twitter accounts. De tegenpartij, de MIVB, vermeldde dat ze principieel niet tegen de app waren (Bruzz, 2016). Maar toch is de app magisch verdwenen.

Daarom dat het tijd is voor een nieuwe app. Gebouwd op de laatste versie van Android, rekening houdend met Android's Material 2 huistijl. We profileren ons volledig op onze dappere voorgangers. Alhouwel we op de schouders staan van de vorige applicatie, gebruiken we de tools van de MIVB tegen henzelf. Hun openbare API was de start van de backend en versnelde het ontwikkelingsproces.
Het minimum valabel product is het kunnen opslagen en live vertonen van MIVB controles. Dit is meer dan gelukt.

[1] App om MIVB-controles te ontwijken enorm populair. (5 september 2014. Het Nieuwsblad, Verkregen 18 december 2022 van https://www.nieuwsblad.be/cnt/dmf20140905_01252657

[2] "Contrôle Stib" vernieuwt verklikapplicatie in september. (24 augustus 2016). Bruzz. Verkregen 18 december 2022, van https://www.bruzz.be/videoreeks/bruzz-24-02082016/video-controle-stib-vernieuwt-verklikapplicatie-september

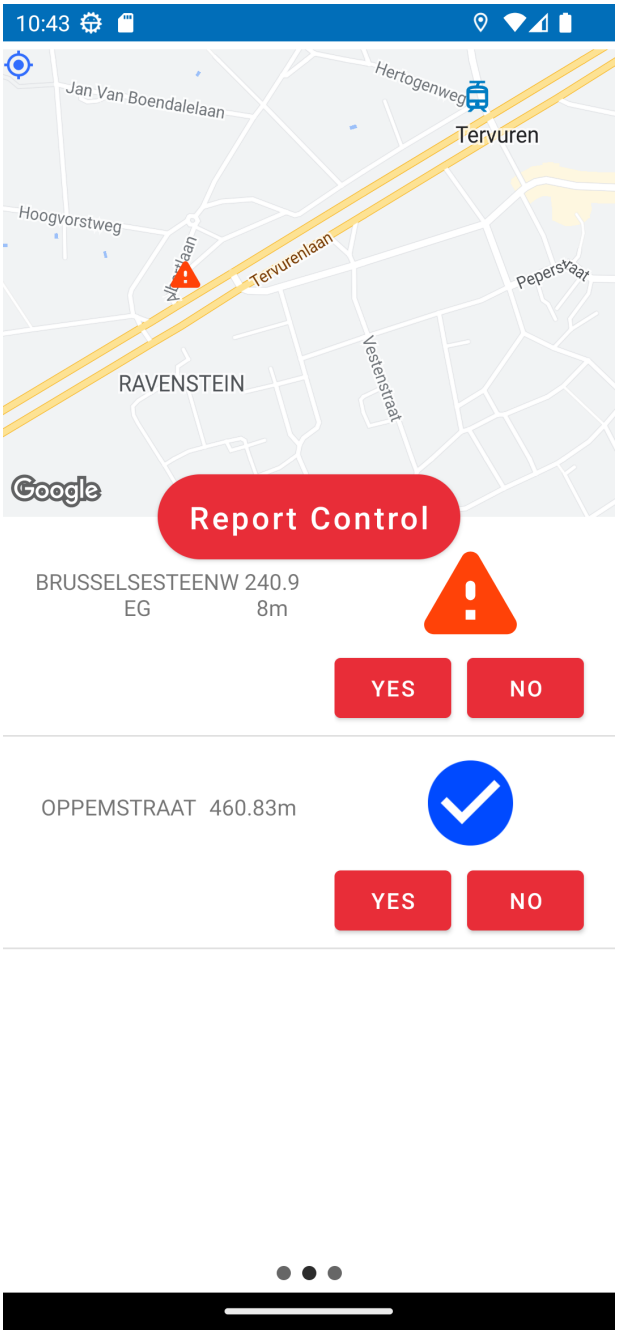

Figure 1: Main Screen App

2 Functionaliteit

De applicatie heeft 3 functionaliteiten:

- De dichts bijzijnde haltes die berekend worden a.d.h.v. de huidige locatie van de gebruiker
- Een manier om controles in te voeren en te verifiëren
- Interactieve Google Maps extensie die gebruikt kan worden om over heel Brussel de haltes en controles te bekijken

2.1 Dichts bijzijnde haltes

Om de dichts bijzijnde haltes te verkrijgen heb 2 dingen nodig. Enerzijds een lijst met alle haltes die de MIVB bediend in Brussel en anderzijds de huidige locatie van de gebruiker. Alle haltes die de MIVB bediend zijn beschikbaar op de Open Data website van de MIVB[3]. Initieel verkregen we deze data via een API Call maar vermits we deze data eigenlijk maar 1 keer moeten inladen (bij het opstarten van de app) leek ons het een beter idee om de data als .csv bestand op te slaan en zo eenmalig in te lezen. Dit gebeurt dus ook in de `readStationData()` functie waar we de data toevoegen aan `stationData`. Dit is een zelf gedefinieerde object lijst van het type `StationSample`. Deze klas bevat de naam van het station, longitude, latitude, afstand en of er een controle plaatsvindt of niet. Deze lijst is de rode draad doorheen dit project en wordt in elke activity opnieuw gebruikt. Dit zorgt ervoor dat we maar 1 klas moeten gebruiken die algemeen is over het hele project en ook de data coherent is in elke activity.

Voor de huidige locatie van de gebruiker te verkrijgen maken we gebruik van de `LocationRequest` en `FusedLocationProviderClient` class. De `LocationRequest` wordt gebruikt om de eigenschappen van de `FusedLocationProviderClient` aan te passen. Zo kunnen we instellen wat het update interval moet zijn als ook welke power mode we willen gebruiken. Om de locatie te gebruiken moeten we ook toestemming vragen aan de gebruiker. Dit wordt gedaan bij het starten van de app. Als de gebruiker toestemming geeft dan starten we de `updateGPS()` functie. Deze functie wordt opgeroepen elke keer dat we een nieuwe locatie binnen krijgen. Deze functie wordt uit de `Location location` object de huidige longitude en latitude verkregen. Deze wijzen we dan toe aan globaal gedefinieerde variabelen zodat we deze ook in andere functies kunnen gebruiken. Hierna wordt de `updateStationDistance()` functie opgeroepen en wordt het `location` object meegegeven. In deze functie gaan we over elk element van `stationData` (dus elke halte) en berekenen we de afstand tussen de huidige locatie en die halte. Met deze informatie updaten we elk element in stationData met de berekende afstand.

[3] Open Data MIVB-STIB https://data.stib-mivb.brussels/explore/?sort=modified:

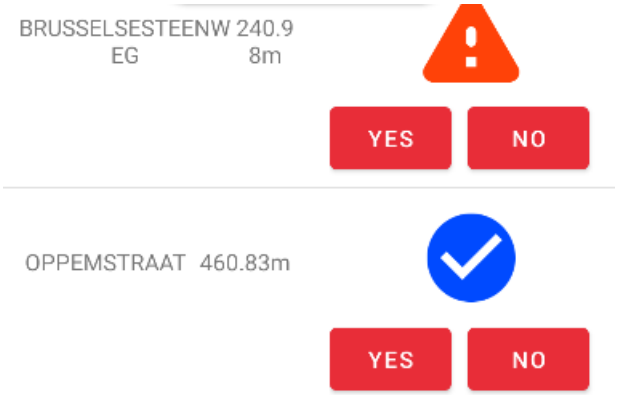

Figure 2: Main List View

Om deze data weer te geven hebben we gebruik gemaakt van een `ListView`. Om dit te gebruiken moet je een `customAdapter` maken. In `MainStationDataAdapter` wordt er voor elk element van de meegegeven lijst een `View` gemaakt. Hierin vullen we de naam van het station, afstand tot huidige locatie en of er al dan niet een controle is in. Als er een controle is wordt een waarschuwing symbool weergegeven. Ook zijn er 2 buttons aanwezig waar de gebruiker kan aangeven of er al dan niet een controle is aan het station. Deze `MainStationAdapter` wordt aangemaakt in de `updateMainLV()` functie. Enkel voor de haltes in `stationData` waarbij de afstand kleiner is dan een vooraf gedefinieerde afstand (bijvoorbeeld: 500m) zal tevoorschijn komen in de `ListView`.

2.2 Report control

De app heeft ook de mogelijkheid om een halte in te voegen waar momenteel een controle plaatsvindt. Dit kan op 2 manieren. De eerst manier brengt je naar een aparte Activity om een halte in te geven. Hier geraak je door oftewel door op de "Report Control" knop te duwen of door naar rechts te swipen. Hiermee wordt de `ReportControlActivity()` opgeroepen. In het tekstveld kan de gebruiker dan een naam van een station ingeven. Wanneer de gebruiker letters ingeeft zal er een autocomplete tevoorschijn komen dat de een lijst met de overeenkomende haltes weergeeft. Via de "Submit" knop kan het station geregistreerd worden.

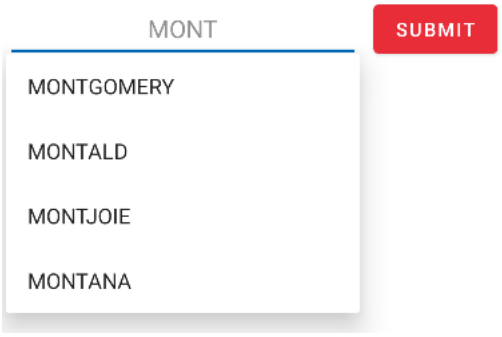

Figure 3: Auto Complete

Om ervoor te zorgen dat de lijst met haltes waar een controle plaatsvindt geüpdatet wordt vertrouwen we op de eerlijkheid van de gebruikers. Dit is een bewezen concept, zo lang het aantal gebruikers groot genoeg is. Op het startscherm heb je de mogelijkheid om bij de dichts bijzijnde haltes aan te duiden of er al dan niet een controle plaatsvindt aan die halte. Stel dat er op de app staat dat er geen controle plaatsvindt bij een dichts bijzijnde halte maar er toch een controle plaatsvindt kan je dit hier ook ingeven. Het omgekeerde geldt ook. Stel dat een controle afgelopen is bij een dichts bijzijnde halte maar dit wel nog aangeduid staat dan kan je dit veranderen door op de "No" knop te duwen. Hierdoor zal de halte verwijderd worden uit de lijst met controle haltes. Naar mate de werkelijke lancering van de applicatie moet dit nu eenvoudig algoritme aangepast worden zodat de controles correct blijven aangevuld.

2.3 Google Maps

Om de app wat interactiever en gebruiksvriendelijker te maken hebben we ook een Google Maps extensie toegevoegd. Deze bevindt zich in een fragment op het hoofdscherm. Op de map worden alle haltes aangeduid als ook de haltes waar er een controle plaatsvindt. De haltes met een controle worden aangeduid met een waarschuwingssymbool. De map wordt aangemaakt bij het opstarten van de app. Eerst wordt de huidige locatie en `stopData` in een `Bundle` gestoken en deze wordt meegegeven met de `mapsFragment`. Verder wordt de `mapsFragment` bij elke `updateGPS()` geupdate met de `updateMapsFragment()` waarin er telkens een nieuwe `Bundle` zal aangemaakt worden met de nieuwe huidige locatie en `stopData` zodat de `mapsFragment` altijd up to date is.

Figure 4: Google Maps Fragment

De `mapsFragment` klas haalt bij het oproepen van de klas eerst de data van de `Bundle` binnen. Vervolgens wordt er een `View` en een `SupportMapFragment` object gecreëerd. Wanneer de map klaar is kunnen we onze data toevoegen aan de map. Eerst wordt de huidige locatie toegevoegd als ook een knop om terug te keren naar de huidige locatie. Vervolgens wordt er over elk element van `stationData` geïtereerd en kijken we wat de controle status van deze halte is. Als er een controle is dan wordt er een waarschuwingssymbool geplaatst op de corresponderende longitude en latitude en anders een gewoon halte symbool.

2.4 All stations

Als je naar links swiped dan krijg je een ListView te zien met alle stations en hun controle status.

2.5 Firestore database

Om ervoor te zorgen dat de lijst van controle stations coherent is over meerdere devices hebben we gebruik gemaakt van een Google Firestore database. Dit is een non-SQL database. In deze database houden we alle stations bij waar momenteel een controle plaatsvindt. Elke keer dat een gebruiker een wijziging aanbrengt zal er eerst gecommuniceerd worden met de database en nagekeken worden of de ingegeven halte reeds aanwezig is in de database of niet.

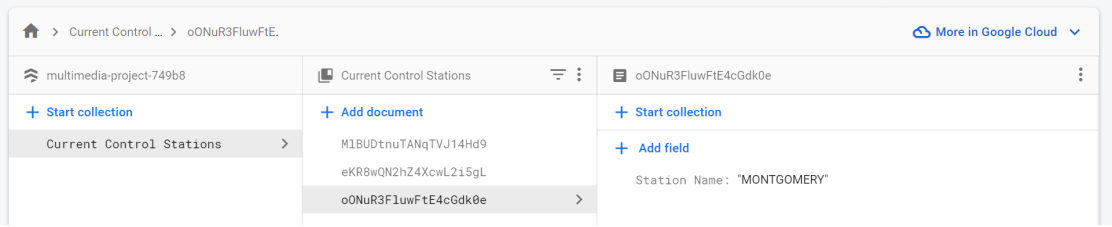

Figure 5: Firestore Database Monitor

Bij het aanmaken van de Firestore database op de Google Firebase website moet je een aantal stappen volgen om de database toe te voegen aan je project. Vervolgens verkrijg je een API-key waarmee je je database calls can doen. Al de database functionaliteiten zijn samengevat in 3 functies: `addToDatabase()`, `removeFromDatabase()` en `retrieveFromDatabase()`. Deze 3 functies werken op een gelijkaardige manier. Het gedefineerde `FirebaseFirestore` object zal opgeroepen worden met een functie en hieraan zal een `onCompleteListener/onSuccesListener` aan toegevoegd worden. Binnen deze functie krijgen we dan een callback met de data van de database of in het geval van het toevoegen/verwijderen van een element, een succes of failure bericht.

2.6 Frontend

Onze voorganger "STIB-Contrôle" was een applicatie met meer dan 300.000 gebruikers. Dit is onmogelijk zonder een gebruiksvriendelijke applicatie. Het doel van onze frontend is het duidelijk vertonen van de data van onze backend. Het is belangrijk dat deze goed met elkaar werken. In Android Studio valt dit goed mee door de redelijk vaste implementatie van de .xml bestanden. Zo lang je Android Studio's structuur blijft volgen zullen er geen grote verassingen opkomen. Een voorbeeld hiervan zijn de theme of style bestanden en het gebruik van logische ID's.

De huisstijl is overgenomen van de gewone MIVB app (zie fig. 6). De kleuren en onze MainActivity zijn quasi gekopieerd van moderne populaire app van onze overheid. De verschillen vinden we in de navigatie. Het is mogelijk om via "swipe-gestures" door de volledige app de surfen. Alternatief kan de gebruiker op de navigatie bolletjes drukken (geïnspireerd op IOS 2, Apple 2008 zie fig. 7). Dit werd geïmplementeerd via transparante knoppen.

We kiezen logisch voor de implementatie van een fragment of een activity. De 3 "views" van de applicatie zijn Activities terwijl de Google Maps implementatie werd geïmplementeerd in een aparte recycleerbare Fragment. We hebben niet geopteerd voor de standaard "navigatiebalk". Ook is er geen klassieke sidebar. Deze keuzes werden gemaakt om de app zo simpel mogelijk te houden en voor zowel grote als kleine schermen bruikbaar te blijven.

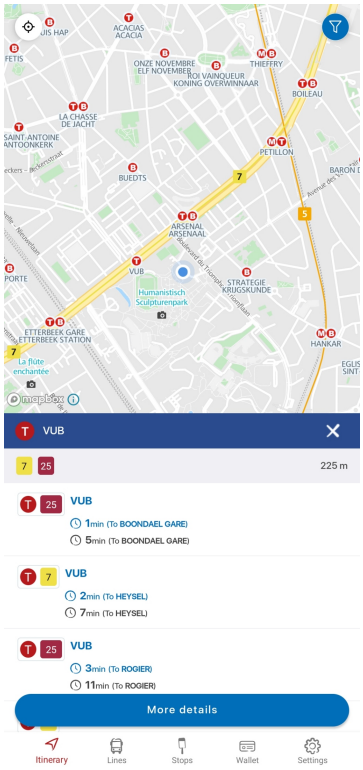

Figure 6: Officiële MIVB Applicatie (versie 2.5.0)

Figure 7: Apple IOS 2.0 (2008)

3 Ondervonden problemen

Android ontwikkeling is opzoeken en repareren. Een programmeur begint op zichzelf met een toepassing implementeerd ze en moet dan verschillende grote aanpassingen maken aan de rest van de code. Hiermee was het lastig om samen te werken aan de code. Voor de oplossing hebben we een klassieke Web-Ontwikkeling techniek toegepast. Het splitsen van het werk in zowel de frontend als de backend. Seppe G. werkte voornamelijk op de backend en Sam D. focuste zich op de gebruiksvriendelijkheid en opmaak van de applicatie. Dit laatste maakte onze applicatie meer dan een schoolproject. Met een rigide huisstijl en implementatie van enkele moderen UX technieken.
Erg grote problemen hebben we niet ondervonden tijdens het maken van de app. Natuurlijk wel lastige bugs zoals bijvoorbeeld bij het maken van de Google Maps extensie. Deze gaf onvoorziene willekeurige stopzettingen van de applicatie. Blijkbaar stond er een recourse file voor het tekenen van de markers op de map in een foute folder (andere versie nummer). Dit zijn lastige bugs die vaak verstopt zitten in een esoterisch detail. Met als enige hoop de verrassend nuttige LogCat ingbouwd in Android Studio. Door dit te gebruiken in combinatie met `Log.d()` statements konden we goed de flow van het programma volgen. Verder was het ook nieuw om te werken met een Firestore database. Het proberen uitwisselen van data tussen verschillende activities zijn we veel Null Reference errors tegengekomen maar uiteindelijk hebben we een goeie methode gevonden door `Bundle` te gebruiken. Maar zoals eerder vermeld hebben we geen grote problemen ondervonden en de applicatie succesvol voltooid.

4 Flowcharts

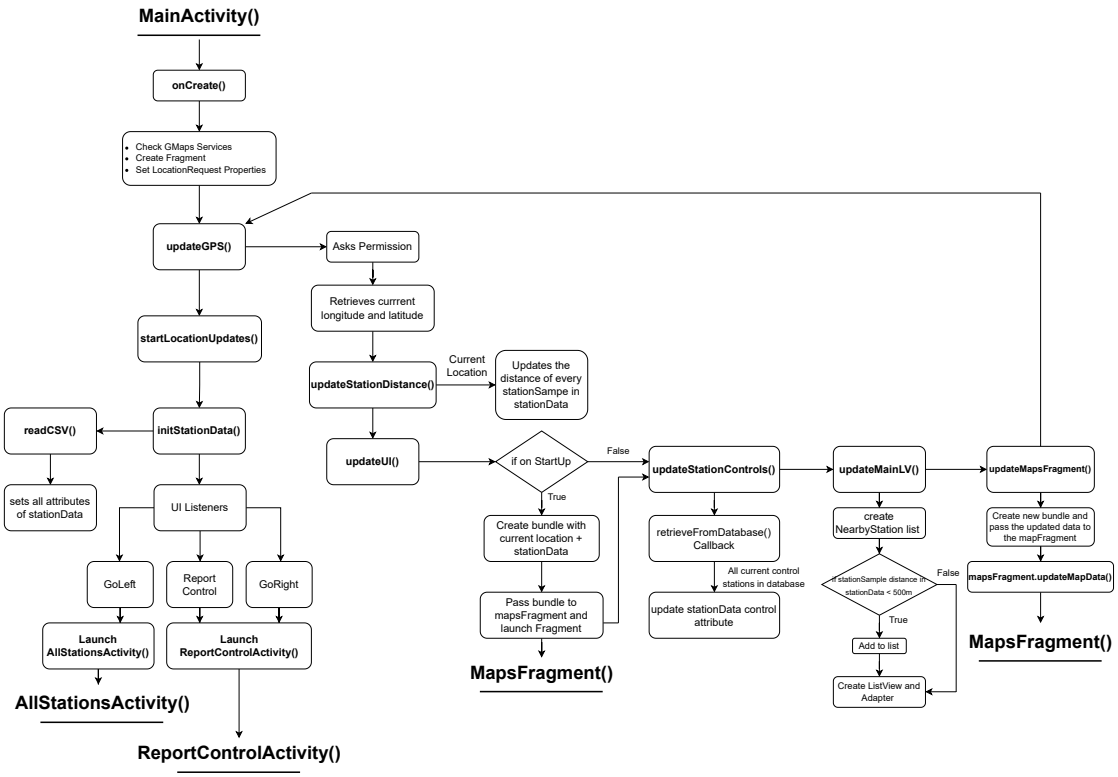

Figure 8: Main Activity

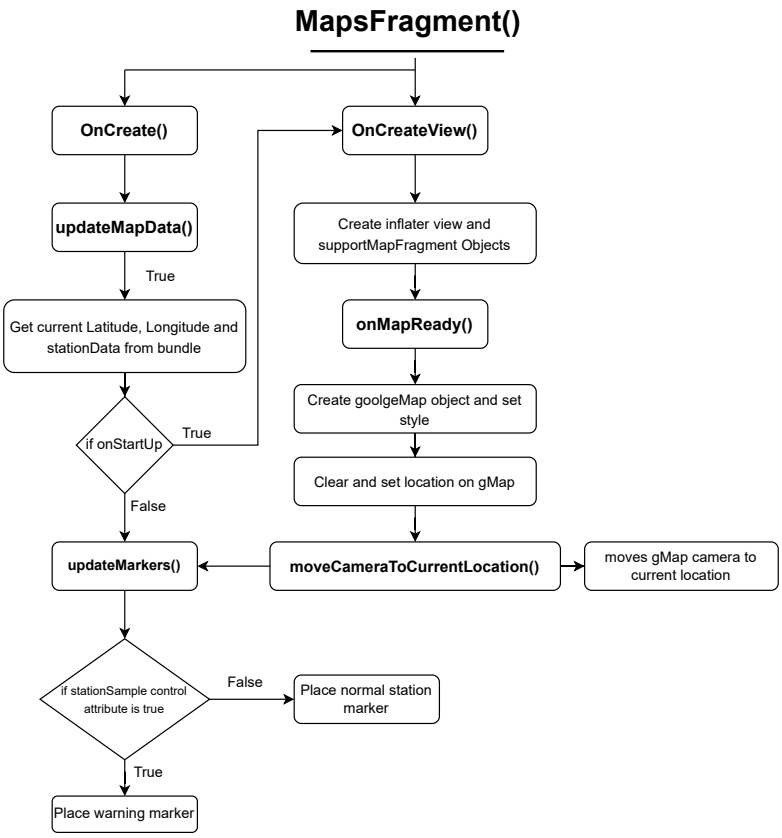

Figure 9: Maps Fragment

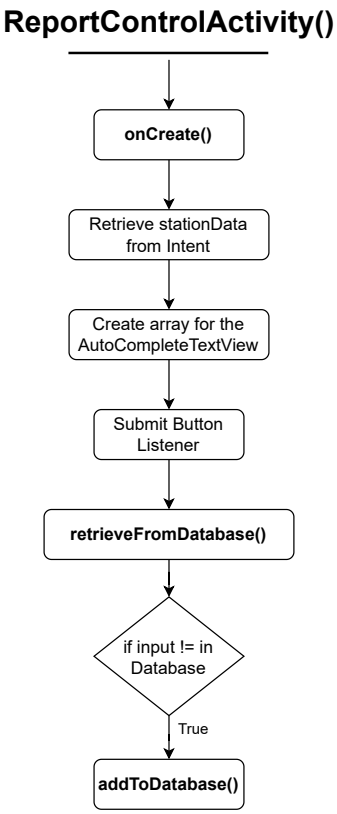

Figure 10: Report Control Activity

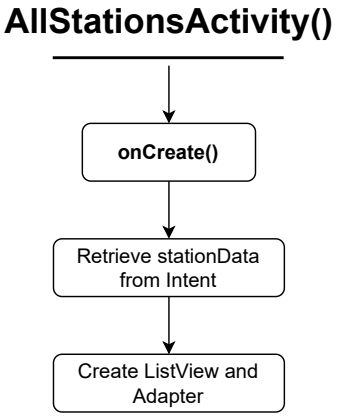

Figure 11: All Stations Activity