

UNIVERSITATEA “ALEXANDRU IOAN CUZA” DIN IAȘI
FACULTATEA DE INFORMATICĂ



LUCRARE DE LICENȚĂ

**GoAnywhere - Planificator de rute ale mijloacelor de transport
în comun**

propusă de

Ştefan-Cătălin Chimu

Sesiunea: *iunie/ iulie, 2022*

Coordonator științific
Lect. Dr. Cristian Vidraşcu

UNIVERSITATEA “ALEXANDRU IOAN CUZA” DIN IAŞI

FACULTATEA DE INFORMATICĂ

**GoAnywhere - Planificator de rute ale mijloacelor de transport
în comun**

Ştefan-Cătălin Chimu

Sesiunea: *iunie/ iulie, 2022*

Coordonator științific

Lect. Dr. Cristian Vidraşcu

Cuprins

Introducere	7
Motivația alegерii	7
Obiective	7
Grad de noutate	7
Contribuții	8
Descrierea aplicației GoAnywhere	9
1.1. Problema adresată	9
1.2. Soluția propusă	10
Aplicații similare	10
2.1. Moovit[3]	10
2.2. Tranzzy[4]	11
2.3. Hereltis[5]	11
Estetica Aplicației	12
3.1. Diagrama use case	12
3.2. Paginile interfeței, funcționalități și componente principale	13
3.2.1. Bara de navigare	13
3.2.2. Permisii de localizare	15
3.2.3. Pagina Home	16
3.2.4. Pagina Lines	17
3.2.5. Pagina Transit Map	18
Implementarea Aplicației	19
4.1. Arhitectura Aplicației și tehnologii folosite	19
4.1.1. Arhitectura generală	19
4.1.2. Tehnologii folosite	20
4.1.3. Kotlin[18]	20
4.1.4. Room Database[9]	20
4.1.5. Dagger - Hilt[11]	21
4.2. Arhitectura API-ului și tehnologii folosite	21
4.2.1. Arhitectura generală	21
4.2.2. Tehnologii folosite	22
4.2.3. Python[21]	22
4.2.4. http-server[13]	22
4.2.5. Threading[23]	23
4.3. Modul de funcționare a API-ului	23
4.3.1. Location Grabber	23
4.3.2. Route Grabber	24
4.3.3. Mapper	25
4.3.4. Route Provider	27
4.3.5. Motivul creării API-ului	32
4.4. Modul de funcționare a aplicației GoAnywhere	34
4.4.1. Permisii de localizare	34

4.4.2. Main ViewModel	35
4.4.3. API Requests	36
4.4.5. RecyclerView	37
Concluzii și viitoare îmbunătățiri	38

Avizat,
Îndrumător Lucrare de Licență
Lect. Dr. Vidrașcu Cristian
Data 23.06.2022 Semnătura _____

DECLARAȚIE privind originalitatea conținutului lucrării de licență

Subsemnatul Chimu Stefan-Cătălin, cu domiciliul în Iași, județul Iași, născut la data de 27.10.1999, identificat prin CNP 1991027226703, absolvent al Universității „Alexandru Ioan Cuza” din Iași, Facultatea de Informatică specializarea Informatică în limba română, promoția 2022, declar pe propria răspundere, cunoscând consecințele falsului în declarații în sensul art. 326 din Noul Cod Penal și dispozițiile Legii Educației Naționale nr. 1/2011 art.143 al. 4 și 5 referitoare la plagiat, că lucrarea de licență cu titlul: GoAnywhere - planificator de rute ale mijloacelor de transport în comun elaborată sub îndrumarea dl. Vidrașcu Cristian, pe care urmează să o susțin în fața comisiei este originală, îmi aparține și îmi asum conținutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licență să fie verificată prin orice modalitate legală pentru confirmarea originalității, consumând inclusiv la introducerea conținutului său într-o bază de date în acest scop.

Am luat la cunoștință despre faptul că este interzisă comercializarea de lucrări științifice în vederea facilitării falsificării de către cumpărător a calității de autor al unei lucrări de licență, de diplomă sau de disertație și în acest sens, declar pe proprie răspundere că lucrarea de față nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Dată azi, 23.06.2022

Semnătură student



DECLARAȚIE DE CONSUMÂMÂNT

Prin prezența declar că sunt de acord ca Lucrarea de licență cu titlul „GoAnywhere - planificator de rute ale mijloacelor de transport în comun”, codul sursă al programelor și celealte conținuturi (grafice, multimedia, date de test etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea „Alexandru Ioan Cuza” din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Iași, data 23.06.2022

Absolvent:
Chimu Ștefan-Cătălin
(semnatura în original)



Introducere

Motivația alegерii

Având în vedere faptul că, de-a lungul perioadei în care am fost student obișnuiam să folosesc ca mod principal de a călători sistemul de transport în comun, am observat în tot acest timp că aplicațiile disponibile la momentul actual pentru a oferi informații despre trasee și timpii de așteptare, tind să ofere date eronate în anumite perioade ale zilei, de pildă, atunci când traficul este congestionat. Acest aspect poate constitui o problemă majoră pentru mulți dintre noi, deoarece, nu am dori să întârziem la facultate, spre exemplu. Motivul pentru care are loc acest fenomen este dat de unitatea de măsură folosită cu scopul de a informa utilizatorul atunci când urmează ca, în stația în care acesta se află, să sosească un autobuz sau un tramvai care are inclus în ruta sa destinația selectată de către utilizator. Folosirea timpului ca unitatea principală de măsură în aplicațiile de profil este cauzată de lipsa formatului standard GTFS[1] folosit la nivel global, însă care nu este disponibil în orașul Iași. Astfel că, ideea unei aplicații care să ofere date despre mijloacele de transport în comun, ce se folosește de coordonatele geografice puse la dispoziție prin intermediul platformei CTP Open Data[2] pentru a indica utilizatorului poziția vehiculului selectat în timp real, s-a dovedit a fi soluția perfectă la problema relatată mai sus.

Obiective

Având în vedere faptul că trăim într-o eră în care “*Dezvoltare continuă și rapidă*” este sintagma definitorie, majoritatea aplicațiilor tind să devină din ce în ce mai complexe. Astfel, scopul meu este să creez o aplicație simplă și ușor de folosit pentru publicul larg, indiferent de profesia sau de vârstă utilizatorului, pentru ca acesta să aibă acces rapid și facil la informația dorită.

Grad de noutate

Aplicația “*GoAnywhere*” are ca scop principal oferirea unei soluții de planificare a rutelor folosite de mijloacele de transport din Iași, și punte la dispoziția utilizatorului date concrete, în timp real, referitoare la vehiculele selectate de aceștia. Având în vedere faptul că o mare parte din populație folosește sistemul de transport în comun, există pe piață aplicațiilor mobile alternative precum Moovit[3], Tranzy[4], HereItIs[5]. Toate aceste aplicații au un

aspect în comun între ele, dar și o diferență față de aplicația prezentată: acestea comunică utilizatorului ora la care va ajunge autobuzul sau tramvaiul/ autobuzul în stație, pe când aplicația “*GoAnywhere*” oferă locația vehiculului în timp real. De asemenea, informațiile oferite de “*GoAnywhere*” vor fi mult mai precise față de aplicațiile enumerate mai sus, deoarece, procesul de asociere dintre datele oferite de platforma CTP Open Data[2] și cele oferite de WINK Public API[6] este conceput să ofere date de ieșire similare formatului standard GTFS[1].

Contribuții

Depinzând constant de sistemul de transport în comun din orașul Iași, și dorind informații mai precise decât cele oferite la momentul actual de aplicațiile de profil, am ajuns la concluzia că, prin crearea acestei aplicații, nu numai că îmi voi satisface nevoia proprie, ci voi avea posibilitatea să îi ajut și pe restul locuitorilor care obișnuesc să folosească acest mod de transport alternativ.

Pentru partea de arhitectură a aplicației “*GoAnywhere*”, în urma studiului mai multor resurse, am ajuns la concluzia că arhitectura MVVM[7] este cea care se pretează cel mai bine nevoilor mele, deoarece aceasta decouplează componentele aplicației, conducând la separarea interfeței grafice de logica de business. Acest aspect este foarte important, deoarece oferă modularitate aplicației, ceea ce va duce la o mențenanță mai ușoară a acesteia.

În continuare se poate regăsi o listă care cuprinde câteva librării folosite în cadrul implementării:

- *Material Design*¹[8], librărie folosită pentru realizarea interfeței.
- *Room Database*²[9], bază de date locală care persistă informațiile despre mijloacele de transport în comun primite de la API.
- *Navigation Component*³[10], librărie folosită pentru crearea grafului de navigare, scopul final fiind ca utilizatorul să poată trece de la un fragment al aplicației la altul.
- *Dagger - Hilt*⁴[11], librării folosite pentru Dependency Injection, astfel putând afișa datele din baza de date locală în interfața aplicației.
- *Haversine*⁵[12], librărie folosită pe partea de API care ajută la calcularea exactă a distanței dintre două coordonate GPS (latitudine și longitudine).

¹ <https://material.io>

² <https://developer.android.com/jetpack/androidx/releases/room>

³ <https://developer.android.com/guide/navigation/navigation-getting-started>

⁴ <https://dagger.dev/hilt/>

⁵ <https://pypi.org/project/haversine/>

- *http-server*⁶[13], librărie folosită la crearea API-ului care servește date aplicației GoAnywhere, acesta ocupându-se de obținerea, formatarea și asocierea datelor asociate mijloacelor de transport în comun.
- *Retrofit*⁷[14], librărie folosită pentru a efectua request-uri către API ce vor oferi datele necesare aplicației “*GoAnywhere*” în format JSON. Acestea vor fi procesate în interiorul aplicației pentru a putea crea modelele ce vor furniza datele din interfață.

1. Descrierea aplicației GoAnywhere

În acest capitol voi detalia impactul pe care îl are aplicația “*GoAnywhere*” asupra utilizatorilor, ce probleme de natură cotidiană au dus la crearea acestei aplicații și soluția propusă pentru rezolvarea acestora.

1.1. Problema adresată

Luând în considerare faptul că tot mai mulți oameni aleg să folosească mijloacele de transport în comun ca variantă principală de călătorie, lucru cauzat de o infrastructură locală subdezvoltată (lipsa locurilor de parcare fiind factorul principal), dar, totodată și costurile pe care le implică deținerea unui autoturism personal, aplicațiile de tip *planificator de rute destinate mijloacelor de transport în comun* au început să devină din ce în ce mai relevante.

Însă, aspectul care joacă un rol deosebit de important în această tranziție este precizia datelor primite de utilizator prin intermediul aplicației, care lipsește cu desăvârșire în situațiile critice. Luăm un exemplu clasic: un utilizator trebuie să ajungă într-un interval de timp din punctul A în punctul B. Acesta se deplasează către cea mai apropiată stație de transport în comun, selectează un mijloc de transport, primește implicit o rută și i se oferă timpul de așteptare până la sosirea vehiculului respectiv. Utilizatorul concluzionează că în 5 minute va urca în autobuz și va ajunge la timp. Însă, având în vedere că la fiecare pas există un mic grad de incertitudine, autobuzul selectat de utilizator suferă o defecțiune tehnică iar un alt vehicul, care are inclus în ruta sa destinația selectată de utilizator, va ajunge cu mult peste ora comunicată prin intermediul aplicației, utilizatorul neavând posibilitatea de a ști cu exactitate distanța reală dintre stația în care acesta așteaptă și mijlocul de transport în cauză. Scenariile de acest fel duc la întârzieri neașteptate și pot cauza un număr de neplăceri pentru utilizatori, deoarece noțiunea de punctualitate va fi anulată de astfel de factori externi.

⁶ <https://pypi.org/project/httpserver/>

⁷ <https://square.github.io/retrofit/>

1.2. Soluția propusă

În acest subcapitol voi reaminti problema adresată mai sus și totodată, voi detalia soluția pe care aplicația “*GoAnywhere*” o aduce. De asemenea, voi menționa și modul în care această aplicație a fost concepută dar și cum poate fi folosită aceasta în viața de zi cu zi.

Ideea principală a acestei aplicații constă în transparența datelor oferite către utilizator, pentru ca acesta să evite cu brio întârzierile nedorite. Totodată, simplitatea interfeței aduce un beneficiu major în adoptarea la scară largă a acestei soluții. Atunci când utilizatorul se află la primul contact cu aplicația “*GoAnywhere*”, va fi întâmpinat de o interfață simplă și intuitivă, construită prin intermediul celor mai noi practici, cu ajutorul librăriei Material Design[8].

Prima pagină, care este intitulată “*Home*”, oferă posibilitatea de a introduce destinația dorită și, folosind locația curentă obținută în urma acceptării permisiunilor de localizare, este determinată cea mai apropiată stație prin intermediul coordonatelor geografice (latitudine, longitudine), care va fi folosită ulterior în planificarea rutelor. De asemenea, utilizatorul are la dispoziție și posibilitatea de a introduce locația de plecare și cea de sosire, pentru a putea vedea în avans ce mijloc de transport poate folosi la următoarea călătorie, fără a fi nevoie ca acesta să se afle la locația de plecare introdusă. Acest ultim aspect lipsește cu desăvârsire din lista de funcționalități oferite de restul aplicațiilor din acest domeniu.

Pagina intitulată “*Lines*” oferă detalii despre toate rutele atunci când utilizatorul nu a introdus o destinație la primul pas. În caz contrar, va afișa componente de tip *Card View* cu toate rutele pe care utilizatorul le are la dispoziție în funcție de locația introdusă. Aceste componente conțin informații despre toate mijloacele de transport în comun, sunt generate dinamic și se pot expanda pentru a putea vedea toate stațiile în care vehiculul selectat se oprește.

Ultima pagină, intitulată “*Transit Map*”, permite vizualizarea locației în timp real a unui autobuz sau a unui tramvai, astfel, utilizatorul având parte de transparență în vizualizarea datelor primite de la API.

2. Aplicații similare

2.1. Moovit⁸[3]

Moovit este cea mai cunoscută aplicație de tip *planificator de rute a mijloacelor de transport* care oferă posibilitatea utilizatorilor să introducă o locație ce reprezintă destinația și folosindu-se locația curentă a acestuia ca punct de plecare, sunt oferite rutele disponibile.

⁸ <https://moovit.com>

Utilizatorul este informat prin intermediul unei ore estimative când urmează ca vehiculul selectat să ajungă în stație. Map View-ul unde pot fi observate rutele vehiculelor are la bază Google Maps SDK⁹[15].

2.2. Tranzy¹⁰[4]

Tranzy este o aplicație care funcționează în aceeași manieră ca și Moovit[3], doar că, aceasta procură datele legate de mijloacele de transport în comun prin intermediul platformei CTP Open Data¹¹[2] pentru a fi folosite ulterior la estimarea unei posibile ore la care va ajunge în stația curentă vehiculul selectat. Tranzy oferă de asemenea utilizatorului posibilitatea de a introduce o locație și pe baza acesteia, vor fi afișate rutele disponibile. Si în acest caz Google Maps SDK[15] este folosit este folosit când vine vorba de Map View.

2.3. HereItIs¹²[5]

HereItIs are o abordare puțin diferită față de Moovit[3] și Tranzy[4] deoarece interfața aplicației aduce aminte de Waze¹³[16], având toate mijloacele de transport plasate pe hartă concomitent, ceea ce duce la o vizualizare aglomerată. De asemenea, datele care pun aplicația în funcțiune sunt cele furnizate de serviciul CTP Open Data[2], unitatea de măsură prin care utilizatorul este informat când urmează ca vehiculul selectat de acesta să ajungă în stație fiind în continuare tot timpul. Pe partea de Map View, în continuare, s-a rămas la Google Maps SDK[15].

⁹ <https://developers.google.com/maps/documentation/android-sdk>

¹⁰ <https://tranzy.ro>

¹¹ <https://www.sctpiasi.ro/servicii/opendata>

¹² <https://start-up.ro/hereitis-waze-ul-romanesc-pentru-transport-in-comun-pornit-din-iasi/>

¹³ <https://www.waze.com>

1. Estetica Aplicației

3.1. Diagrama use case

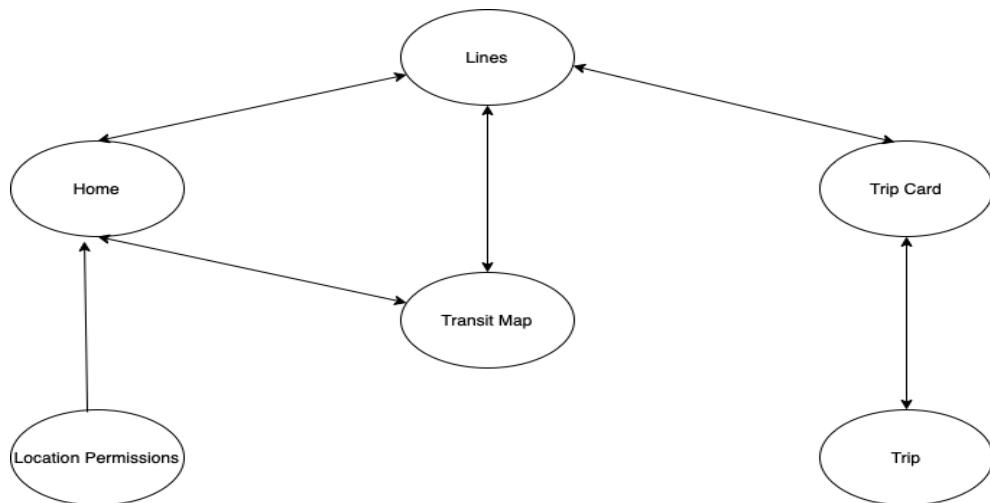


Figura 1.1. Diagrama use care care prezintă modul de funcționare al aplicației GoAnywhere

După cum se poate observa în Figura 1.1, primul fragment care apare când aplicația este lansată poartă numele de Location Permissions, care îi va cere utilizatorului acordul pentru folosirea informațiilor referitoare la locația curentă pentru ca aplicația să funcționeze în parametrii optimi. Ulterior, acesta este redirecționat către pagina principală, intitulată Home, de unde poate alege dacă dorește să folosească locația curentă ca punct de start prin bifarea opțiunii “*Use my current location instead*” sau preferă să introducă o altă locație de plecare prin intermediul căsuței “*Start Location*”. De asemenea, la acest pas este necesară completarea căsuței “*Destination*” cu locația în care utilizatorul dorește să ajungă.

Din fragmentul Home, utilizatorul poate să viziteze fragmentele Lines și Transit Map prin intermediul barei de navigare poziționată în partea de jos. Bara de navigare are o structură de graf în spate, ceea ce permite formarea de legături între fragmente, acest lucru permitând utilizatorului să navegheze în oricare dintre paginile prezente în bară. Fiecare pagină din bara de navigare vine însotită de o reprezentare grafică sugestivă, cu scopul de a ajuta utilizatorul în a recunoaște rapid ce fragment urmează să acceseze. Pictogramele folosite la acest pas fac parte din libraria Material Design[8].

Fragmentul Lines are la rândul său două moduri de funcționare:

- Dacă utilizatorul accesează acest fragment din bara de navigare, fără a oferi datele cerute în fragmentul Home, acesta va putea vedea toate mijloacele de transport în comun, reprezentate sub formă de carduri.
- Dacă utilizatorul completează datele cerute în fragmentul Home, fragmentul Lines va afișa sub formă de carduri doar vehiculele ce au incluse în ruta lor locația și destinația obținute la pasul anterior.

Fragmentul Trip Card, care reprezintă un card cu informații succinte referitoare la un vehicul, este un element vizual inclus în fragmentul Lines. La expandarea componenței, informații precum: numărul de stații dintre locația de plecare și cea de sosire; distanța față de user; ruta completă; locația mijlocului de transport selectat în timp real însotită de ora la care s-a dat acel status; timpul necesar ajungerii vehiculului selectat de la stația de plecare către stația de sosire vor fi afișate. Varianta expandată a componenței va constitui fragmentul Trip, pentru a avea o viziune mai clară asupra datelor disponibile.

Ultimul fragment, Transit Map, are ca scop localizarea pe hartă prin intermediul unui marcator de tip pin a unui vehicul selectat de utilizator, prin folosirea căsuței de search.

3.2. Paginile interfeței, funcționalități și componente principale

3.2.1. Bara de navigare

Bara de navigare, care ajută utilizatorul să se deplaseze printre paginile pe care aplicația “GoAnywhere” le conține face parte din librăria Material Design[8] și găzduiește trei butoane asociate fiecare câte unei pagini. Acestea sunt însotite de iconițe sugestive, care, de asemenea, fac și acestea parte din librăria mai sus menționată, pentru a oferi o experiență cât mai intuitivă posibil. Bara de navigare poate fi observată în *Figura 1.2*.

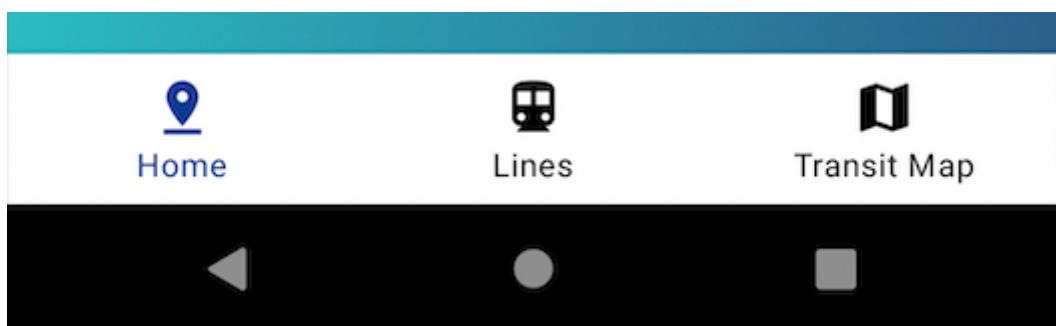


Figura 1.2. Bara de navigare care conține paginile principale ale aplicației

Opțiunile propriu-zise, adică Home, Lines și Transit Map au fost integrate cu bara de navigare prin crearea unui obiect de tip meniu, ce poate fi observat în *Figura 1.3*, aici fiind atribuite numele paginilor, referințele către acestea cât și pictogramele.

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <menu xmlns:android="http://schemas.android.com/apk/res/android">
3
4     <item android:id="@+id/directionsFragment"
5         android:title="Home"
6         android:icon="@drawable/ic_baseline_pin_drop_24"
7         />
8     <item android:id="@+id/routesFragment"
9         android:title="Lines"
10        android:icon="@drawable/ic_baseline_directions_transit_24"
11        />
12     <item android:id="@+id/mapFragment"
13         android:title="Transit Map"
14         android:icon="@drawable/ic_baseline_map_24"
15         />
16 </menu>

```

Figura 1.3. Obiectul meniu ce ține elementele de design cât și referințele către fragmente

Pentru a face posibilă navigarea de la o componentă la cealaltă prin intermediul barei de navigare, am apelat la o structură de tip graf, care conține toate fragmentele și tranzițiile dintre acestea. Graful de navigare poate fi observat în *Figura 1.4*.

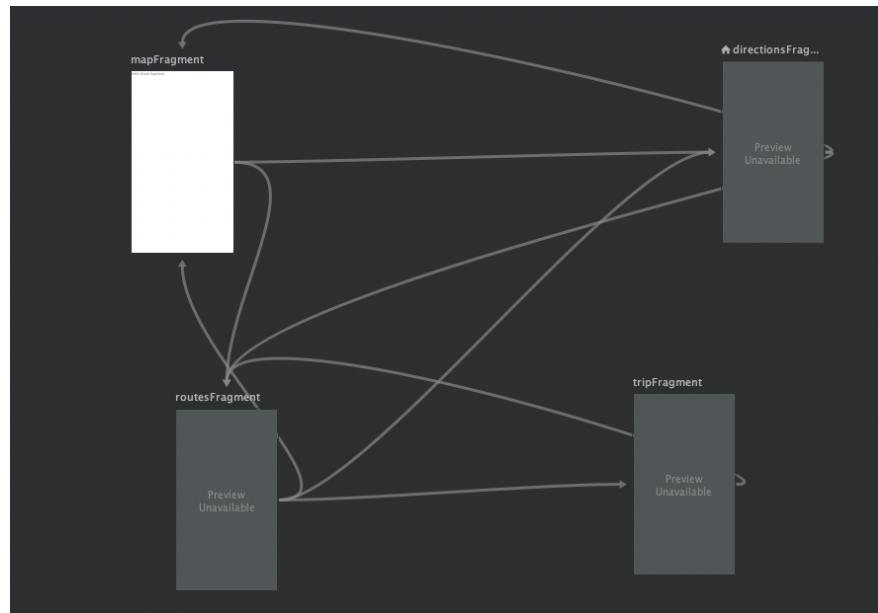


Figura 1.4. Graficul de navigare al aplicației “GoAnywhere”

Acest graf este disponibil global la nivel de aplicație, putând fi invocat, spre exemplu, atunci când un buton înregistrează o apăsare. Spre exemplu, în *Figura 1.5*, butonului din fragmentul Home îi este asociată tranziția către fragmentul Lines atunci când este apăsat de către utilizator în urma completării datelor cerute la acest pas pentru a putea vedea rutele disponibile.

```
_binding!!.button.setOnClickListener { it: View!  
    findNavController().navigate(R.id.action_directionsFragment_to_routesFragment)  
}
```

Figura 1.5. Tranziție de la fragmentul directions (Home) către routes (Lines) asociată unui buton

3.2.2. Permișii de localizare

Atunci când utilizatorul deschide pentru prima dată aplicația “*GoAnywhere*”, va fi rugat să accepte un set de permisiuni de localizare pentru a putea avea acces la locația acestuia în timp real prin intermediul unei ferestre de tip *pop-up*, fapt care poate fi regăsit în *Figura 1.6*. Scopul principal al acestei aplicații este de a efectua planificări de rute cât mai precise pentru a putea ajuta utilizatorii în activitățile lor cotidiene. Pentru ca acest lucru să poată avea loc, utilizatorul ar trebui să ofere acces la locația sa precisă astfel încât erorile ce pot apărea la calcularea distanțelor față de stațiile de plecare cât și de sosire să fie minime.

Sistemul de cerere și înregistrare a permisiunilor de localizare este preluat de pe StackOverflow¹⁴ și adaptat nevoilor apărute în procesul de dezvoltare al aplicației. Pentru această funcționalitate se folosește noul Activity Result API¹⁵ pentru că, odată cu Android Q, mai exact API level 29, pe lângă permisiunea numită “*ACCESS_FINE_LOCATION*” este necesară și permisiunea “*ACCESS_BACKGROUND_LOCATION*” pentru a avea parte de date cât mai corecte.

După cum am menționat și mai sus, la prima lansare a aplicației, utilizatorului îi va apărea o fereastră de tip *pop-up* unde poate să accepte sau să refuze folosirea datelor despre locația sa curentă în timp real. Dacă utilizatorul optează pentru opțiunea “*Don't allow*”, va avea în continuare acces la funcționalitățile oferite de aplicație, însă, la următoarea sesiune de utilizare, acesta va fi întărit că este indicat să pornească serviciile de localizare pentru ca aplicația să funcționeze în parametrii optimi, fiind redirecționat către pagina de setări ale aplicației, aspect care poate fi observat în *Figura 1.7*.

¹⁴ <https://stackoverflow.com/questions/40142331/how-to-request-location-permission-at-runtime>

¹⁵ <https://developer.android.com/training/basics/intents/result>

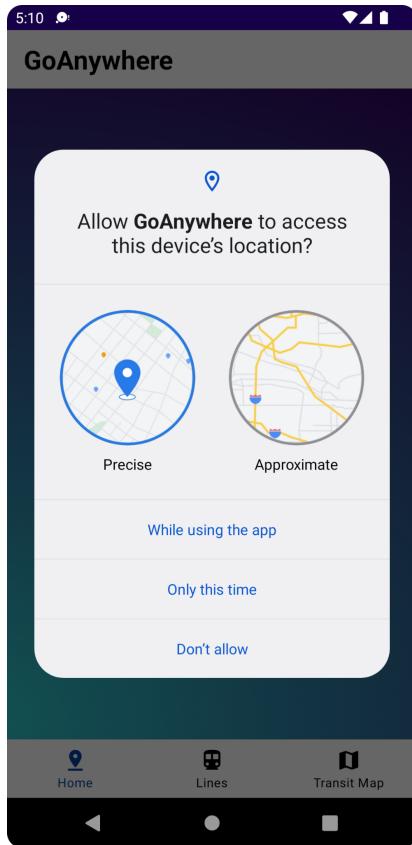


Figura 1.6. Cererea permisiunilor de localizare

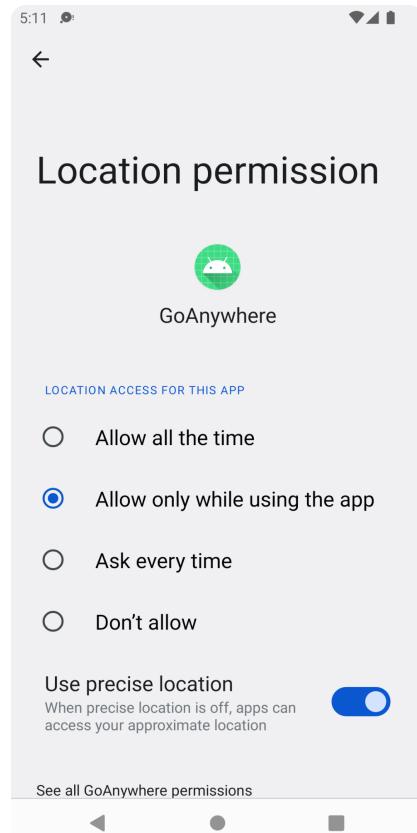


Figura 1.7. Pagina de setări a aplicației

3.2.3. Pagina Home

După ce utilizatorul a acceptat sau a respins permisiunile de localizare, acesta este întâmpinat de pagina Home asociată fragmentului Directions, unde urmează să ofere datele ce vor fi folosite pentru a se realiza planificarea rutelor mijloacelor de transport în comun. Astfel, în această secțiune sunt puse la dispoziție două modalități de a oferi date:

- Dacă utilizatorul alege să bifeze bulina asociată funcționalității “Use my current location”, aplicația va considera ca punct de plecare locația curentă a acestuia, urmând ca secțiunea “Current Location” să fie dezactivată.
- Dacă utilizatorul alege să introducă manual o locație de plecare pentru a efectua o viitoare planificare a unui traseu, va fi necesare doar un cuvânt cheie pentru ca adresa completă să fie găsită și folosită în mod automat. Spre exemplu, pentru locația de plecare “Zimbru” introdusă în căsuța “Current Location”, se va genera în mod automat adresa completă însorită de coordonatele geografice atunci când este făcut request-ul către API.

Cele două modalități prin care se obține o locație de plecare sunt reprezentate în *Figura 1.8*. După acest pas, utilizatorul are posibilitatea de a introduce o destinație prin intermediul căsuței “*Destination*”, urmând ca procesul de obținere a datelor în vederea generării rutelor să fie încheiat prin intermediul apăsării butonului “*Take me where I want*”, reprezentat în *Figura 1.9*. Imaginea de fundal folosită în aplicație poate fi găsită aici¹⁶.

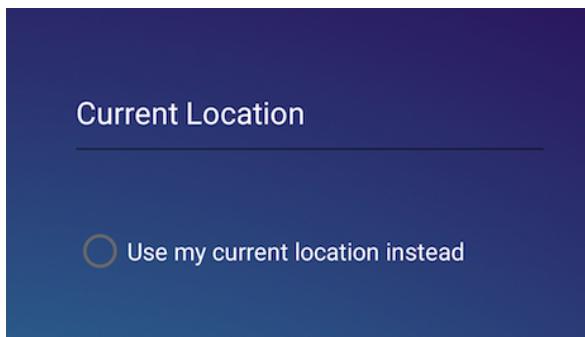


Figura 1.8. Selectarea locației de plecare start

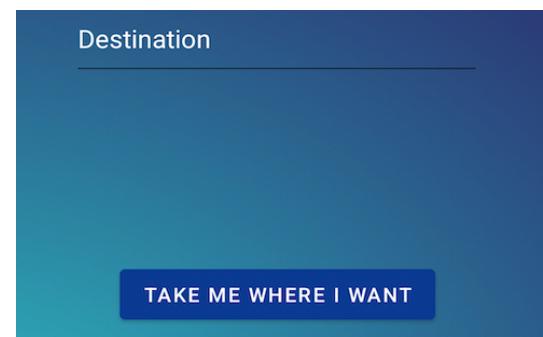


Figura 1.9. Secțiunea Destinație și butonul de

3.2.4. Pagina Lines

În urma obținerii locației de plecare și a destinației de la pasul anterior, un request către API va fi făcut prin intermediul librăriei Retrofit[14] pe ruta “/getroutes” pentru a i se oferi utilizatorului toate rutele pe care acesta le are la dispoziție în funcție de datele primite. Fiecare rută este încadrată într-un card separat, aspect reprezentat în *Figura 1.10*, în care este afișat tipul mijlocului de transport în comun (autobuz sau tramvai) și numărul acestuia, pentru a putea fi reperat cât mai ușor atunci când vehiculul ajunge în stație.

De asemenea, fiecare card dispune și de un buton intitulat “More”, ce are ca scop oferirea de detalii mai amănunțite referitoare la ruta aleasă, precum se poate observa în *Figura 1.11*. Datele afișate în secțiunea “More” sunt obținute în urma unui GET request către API prin intermediul librăriei Retrofit[14] la ruta “/inforoute”. Detaliile cele mai relevante ce vor constitui interes pentru majoritatea utilizatorilor sunt: locația în timp real a mijlocului de transport selectat, distanța dintre stația curentă și acesta cât și durata călătoriei.

Însă, această pagină poate fi accesată și din bara de navigare, nu numai prin intermediul introducerii datelor legate de traseu și apăsarea butonului “*Take me where I want*”. Dacă utilizatorul accesează această pagină fără să ofere detalii despre locația curentă și destinație, acesta va avea acces la toate liniile disponibile în Iași însotite de o scurtă descriere a traseului, afișate în aceeași manieră ca și rutele propuse în urma oferirii datelor necesare planificării.

¹⁶ <https://unsplash.com/photos/3rWagdKBF7U>



Figura 1.10. Rutele sugerate în funcție de datele obținute

3.2.5. Pagina Transit Map

Pagina Transit Map oferă posibilitatea de a repera pe hartă mijlocul de transport dorit care este cel mai apropiat de locația curentă a utilizatorului. Prin simpla căutare după tipul mijlocului de transport și număr, spre exemplu “Tramvaiul 11”, ultima locație exprimată în coordonate geografice a celui mai apropiat vehicul de locația curentă a utilizatorului va fi marcat pe un Map View pus la dispoziție de Google Maps SDK[15] printr-un marcator de tip “pin”.

4. Implementarea Aplicației

4.1. Arhitectura Aplicației și tehnologii folosite

4.1.1. Arhitectura generală

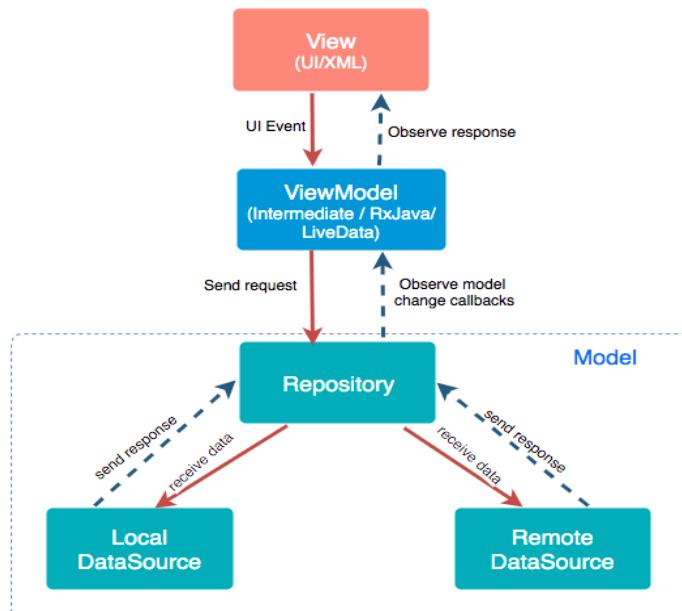


Figura 2.1. Arhitectura generală a aplicației GoAnywhere¹⁷

După cum se poate observa în Figura 2.1, arhitectura generală a aplicației “GoAnywhere” este bazată pe arhitectura MVVM[7], care este formată din trei componente principale:

- **Model**, care cuprinde subcomponenta Repository, rolul acesteia fiind de a face legătura între aplicație și API-ul care are ca scop servirea datelor. Repository primește date de la sursa de date remote, în cazul nostru API-ul și le trimit ulterior către sursa de date locală, în cazul nostru Room Database[9] pentru a fi persistate și utilizate la nevoie.
- **Views**, constă în totalitatea fragmentelor prezentate la capitolul 3, subcapitolul 3.1, acestea folosind limbajul de markup XML¹⁸[17] și formând totodată partea vizuală a aplicației. Acestea trimit acțiunile utilizatorului către ViewModel și primesc datele necesare prin intermediul unor observabile la care acestea se înscriu.

¹⁷ <https://blog.mindorks.com/mvvm-architecture-android-tutorial-for-beginners-step-by-step-guide>

¹⁸ https://developer.mozilla.org/en-US/docs/Web/XML/XML_introduction

- **ViewModel** este componenta care face legătura dintre *View* și *Model*, furnizând datele necesare pentru interfață prin intermediul expunerii unor observabile.

4.1.2. Tehnologii folosite

În procesul de realizare al aplicației “*GoAnywhere*” au fost folosite mai multe tehnologii noi pentru a putea oferi utilizatorilor o experiență cât mai plăcută încă de la primul contact. În continuare, voi menționa aceste tehnologii, unde voi oferi explica motivul pentru care am ales să le folosesc.

4.1.3. Kotlin¹⁹[18]

Kotlin[18] este limbajul de programare pe care l-am folosit în procesul de dezvoltare al aplicației “*GoAnywhere*”. Acesta este bazat pe Java²⁰[19], reușind să îmbine cu brio avantajele unui limbaj de o asemenea anvergură cu un set de funcționalități bine venite și utile în dezvoltarea de aplicații mobile. Principalele avantaje pe care le aduce acest limbaj sunt enumerate mai jos:

- *Sintaxa ușor de înțeles* permite dezvoltatorilor de aplicații să se focuseze pe aducerea de funcționalități noi produsului în loc ca aceștia să se focuseze pe compactarea și eficientizarea codului scris.
- *Mențenanță facilă* asupra codului scris, acest aspect fiind datorat sintaxei ușor de folosit și înțeles, dar și faptului că lungimea codului scris în Kotlin[18] este observabil mai mică decât lungimea aceluiași cod scris în Java[19].
- *Codul este complet interoperabil* cu Java[19], punct foarte important deoarece, în caz că există anumite module în aplicație care sunt scrise în Java, tranzitia către Kotlin[18] va fi una simplă și cu un grad ridicat de siguranță. De asemenea, acest principiu se aplică și viceversa.

4.1.4. Roon Database[9]

Room Database[9] este o bază de date *in memory* folosită cu scopul de a persista datele primite de la API pentru a putea fi folosite ulterior în aplicație, acestea fiind afișate în fragmente. Cele două mari avantaje pe care le aduce această metodă față de SQLite[20] spre exemplu sunt:

¹⁹ <https://kotlinlang.org>

²⁰ <https://dev.java>

- *Eliminarea codului de tip boilerplate*, scris de dezvoltatorii de aplicații pentru crearea și managementul bazei de date.
- *Validare la compilare a query-urilor SQL*, acest aspect permitând dezvoltatorului să vadă dacă query-urile SQL sunt scrise corect.

4.1.5. Dagger - Hilt[11]

Dagger - Hilt[11] este o librărie care ușurează procesul de dependency injection în procesul de dezvoltare a unei aplicații mobile, generând clasele și dependințele necesare în mod automat prin intermediul unui set de adnotări predefinite. Avantajele principale în folosirea acestei librării sunt:

- *Eliminarea codului boilerplate*, absolut necesar în cazul în care procesul de dependency injection se face în mod manual.
- *Refactorizare facilă*, dat fiind faptul că sunt eliminate blocurile de cod inutile.

4.2. Arhitectura API-ului și tehnologii folosite

4.2.1. Arhitectura generală

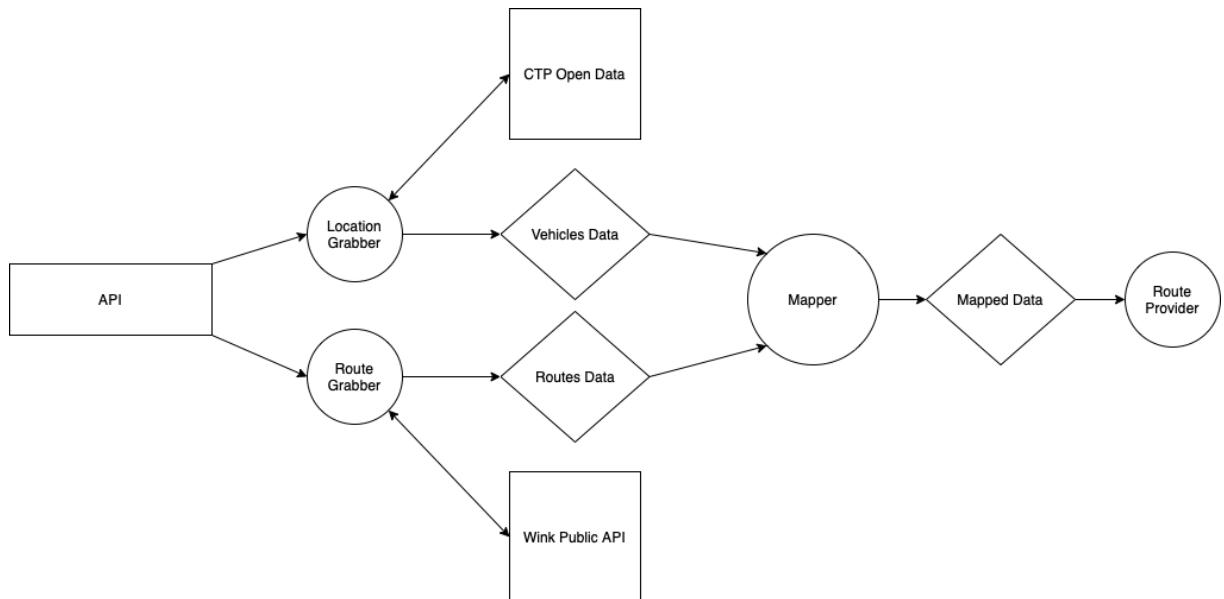


Figura 3.1. Arhitectura generală a API-ului care se ocupă de servirea datelor către aplicația GoAnywhere

În continuare, voi enumera tehnologiile folosite de către API-ul care servește date aplicației “*GoAnywhere*”, reprezentat în *Figura 3.1*, urmând apoi să fie descris și modul de funcționare a acestuia.

4.2.2. Tehnologii folosite

În decursul dezvoltării API-ului care se ocupă de servirea de date către aplicație, am folosit mai multe tehnologii, pe care urmează să le enumăr mai jos.

4.2.3. Python²¹[21]

Python[21] este limbajul pe care l-am ales în procesul de dezvoltare al acestui API, deoarece vine cu un set de avantaje care se pretează pe nevoile apărute pe parcursul proiectării. În continuare, voi enumera funcționalitățile care au dus la alegerea acestui limbaj:

- *Sintaxa simplă și ușor de înțeles*, aspect care a ajutat la concentrarea pe partea funcțională a codului. Totodată, acest beneficiu vine la pachet și cu ușurință în implementarea modularității, factor cheie în dezvoltarea unui proiect complex, dar și în procesul de mențenanță.
- *Portabilitatea*, aceasta permitând rularea API-ului pe orice sistem de operare în aceeași manieră și ajutând la procesul de deploy.
- *Disponibilitatea modulelor third-party* extinde capabilitățile acestui limbaj în procesul de dezvoltare. Modulele pot fi obținute prin intermediul managerului de pachete pip²²[22], care ușurează procesul de instalare, cât și cel de aducere la zi a acestora.

4.2.4. http-server[13]

Librăria http-server[13] a fost folosită în procesul de dezvoltare a API-ului pentru a putea defini rutele pe care le poate accesa aplicația “*GoAnywhere*” în vederea obținerii datelor necesare despre rutele mijloacelor de transport în comun. Procesul de inițializare simplu a fost factorul cheie care a dus la folosirea acestui modul.

²¹ <https://www.python.org>

²² <https://pypi.org/project/pip/>

4.2.5. Threading²³[23]

Threading[23] a fost folosit în dezvoltarea API-ului cu scopul de a rula concomitent mai multe procese de tip I/O (input/ output) pentru a nu apărea conflicte între acestea. Având în vedere faptul că datele sunt procurate din două surse, folosirea acestui modul a fost absolut necesară în procesul de prelucrare a datelor.

4.3. Modul de funcționare a API-ului

În acest subcapitol, voi prezenta modul de funcționare a API-ului, descriind structura și modul de funcționare a fiecărei componente reprezentate în *Figura 3.1*.

4.3.1. Location Grabber

Componenta *Grabber*, asociată thread-ului denumit *grabberThread*, efectuează un request de tip GET către platforma CTP Open Data[2] pentru a prelua date în timp real despre toate mijloacele de transport în comun disponibile în Iași. Datele oferite de această platformă sunt disponibile în format JSON[24] și conțin informații precum: *vehicleName* (reprezentat de un ID intern, unic, asociat unui mijloc de transport), locația curentă exprimată în coordonate geografice (latitudine și longitudine) și ora, însotită de data la care a fost oferit ultimul status (un astfel de exemplu poate fi observat la *Figura 4.1* de mai jos).

Datele sunt apoi parsate, creându-se câte un fișier separat pentru fiecare ID unic disponibil, adăugându-se în acestea ultima locație și ultima oră obținute la fiecare update al platformei. Un exemplu pentru un astfel de output poate fi observat în *Figura 4.2*.

Acest thread pornește odată cu serverul și rulează constant în fundal pentru a putea prelua date în timp real.

```
{  
  "vehicleName": "812",  
  "vehicleLat": "47.1647484",  
  "vehicleLong": "27.5738195",  
  "vehicleDate": "2022-06-18 20:51:34"  
},
```

```
≡ .812.txt ×  
mapping > data > ≡ .812.txt  
1 47.1592402,27.5874738;21:09:05  
2 47.1579271,27.5851914;21:10:07  
3 47.1548857,27.5861814;21:11:04  
4 47.1535688,27.5875621;21:12:05
```

Figura 4.1 Date distribuite de CTP Open Data[2]

Figura 4.2. Output-ul componentei Grabber după parsare

²³ <https://docs.python.org/3/library/threading.html>

```

def json_parser():
    jsonBlob = js.loads(data_downloader(const.ctpURL))
    for i in jsonBlob:
        # check if the vehicle is moving by the timestamp
        if (i['vehicleDate'] != const.disfunc):
            # create/ append to specific file for every bus/ tram named after the vehicle id from json
            fileName = const.filePath + "." + i['vehicleName'].replace(" ", "") + ".txt"
            if (path.exists(fileName) == False):
                f = open(fileName, "x")
            else:
                f = open(fileName, "a")
            output = i['vehicleLat'] + "," + i['vehicleLong'] + ";" + str(i["vehicleDate"]).split(" ")[1]
            f.write(output + "\n")
            f.close()

```

Figura 4.3. Funcția care parsează datele obținute de pe platforma CTP Open Data[2]

4.3.2. Route Grabber

Componenta *Route Grabber* efectuează un request de tip GET către WINK Public API²⁴[6], de unde obține informații despre fiecare autobuz sau tramvai disponibil în Iași, precum: numărul acestuia (informație care lipsește din setul de date oferit de CTP Open Data[2]), stațiile în care acesta se oprește însotite de coordonatele lor geografice, shape (lista tuturor locațiilor, acestea reprezentând un sumar al coordonatelor geografice puse la dispoziție de CTP Open Data[2]), sumarul rutei (informație afișată și pe mijloacele de transport în comun) și distanța totală parcursă.

Având în vedere că și WINK Public API[6] folosește ID-uri pentru a accesa ruta unui autobuz sau a unui tramvai (<https://m-go-iasi.wink.ro/apiPublic/route/byId/ID> fiind URL-ul care oferă acces către setul de date necesar), am descoperit, în timpul proiectării API-ului, faptul că ID-ul este cuprins între valorile 1 și 100 prin execuția unei bucle *for*. Acest pas a găsit toate cele 33 de rute disponibile în Iași, dintre care 9 sunt destinate tramvaielor și restul de 24 autobuzelor. Prin urmare, am format un fișier care conține asocierea dintre fiecare ID și ruta aferentă pentru a avea un acces mai facil la date. Folosindu-mă de acest aspect, am creat două funcții:

- *stops*, care iterează prin intermediul funcției *iterator* printre rutele disponibile și oferă un array care cuprinde totalitatea stațiilor, însotite de coordonatele geografice aferente, și a mijloacelor de transport care vizitează fiecare stație. Structura funcției *stops* poate fi observată în *Figura 4.4*.
- *shapes*, care, de asemenea, iterează prin intermediul funcției *iterator* printre toate rutele disponibile și oferă pentru fiecare mijloc de transport, colecția tuturor coordonatelor geografice disponibile. Funcția *shapes* este reprezentată în *Figura 4.5*.

²⁴ <https://m-go-iasi.wink.ro/apiPublic/route/byId/1>

```

# Appends to dataArr array all the stops without duplicates for one vehicle #
def stops(jsonBlob, vehicleID, dataArr):
    for i in jsonBlob["data"]["routeWaypoints"]:
        if (i["name"] != "" and i["name"] != None):
            outLine = (i["name"], float(i["lat"]), float(i["lng"]), vehicleID)
            appended = False
            for iter in range(0, len(dataArr)):
                temp = list(dataArr[iter])
                if (temp[1] == outLine[1] and temp[2] == outLine[2]):
                    appendedAlready = vehicleID in str(temp[3]).replace(" ", "").split(",")
                    if (appendedAlready == False):
                        temp[3] = temp[3] + ", " + vehicleID
                        appended = True
                        dataArr[iter] = tuple(temp)
            if (appended == False):
                dataArr.append(outLine)

```

Figura 4.4. Funcția *stops*, care asociază fiecare stație cu mijloacele de transport ce o au inclusă în ruta lor

```

# Appends to dataArr array all the coordinates that are checked in by a vehicle
# It is used on mapping process for relevant results #
def shapes(jsonBlob, vehicleID, dataArr):
    vehArr = [(str(vehicleID).split("_")[0], str(vehicleID).split("_")[1])]
    for i in jsonBlob["data"]["routeWayCoordinates"]:
        vehArr.append((float(i["lat"]), float(i["lng"])))
    dataArr.append(vehArr)

```

Figura 4.5. Funcția *shapes* care oferă totalitatea coordonatelor asociate fiecărui mijloc de transport în comun

Folosindu-mă de datele oferite de aceste două funcții, am reușit să efectuez asocierea dintre datele oferite de CTP Open Data[2] și WINK Public API[6] prin intermediul componentei intitulate *Mapper*.

4.3.3. Mapper

Componenta *Mapper*, asociată thread-ului *mapperThread*, se ocupă de efectuarea asocierii datelor provenite de la CTP Open Data[2] și WINK Public API[6]. Aceasta este cel mai important pas din întregul parcurs de funcționare a API-ului deoarece permite oferirea de date în timp real despre fiecare mijloc de transport.

Thread-ul asociat componentei *Mapper* este inițializat în *Location Grabber*, acesta fiind invocat la fiecare 4 ore din momentul în care serverul este pornit. Folosind fișierele cu date colectate de la platforma CTP Open Data[2] și array-ul obținut în urma apelării funcției *shapes* din componenta *Route Grabber*, funcția *mapper* reprezentată în Figura 4.6 iterează concomitent prin cele două surse de date și apelează funcția *match_ratio* căreia îi oferă ca parametrii un obiect din array și un fișier cu totalitate coordonatelor geografice obținute din CTP Open Data[2] până în acel moment. În continuare, unui element din array-ului obținut de

funcția *shapes* i se va atribui denumirea de şablon, pentru a avea o viziune mai clară asupra evenimentelor care au loc la pașii următori.

În funcția *match_ratio*, reprezentată în Figura 4.7, cele două surse de date sunt parcuse linie cu linie și se calculează o rație de potrivire între acestea, prin verificarea distanței dintre coordonate geografice aparținând seturilor de date disponibile.

Distanța dintre perechea latitudine, longitudine prezente în fișierul cu date obținute de către componenta *Location Grabber* și şablon este calculată prin intermediul librăriei *Haversine*[12], care are ca scop aplicarea formulei Haversine²⁵ pentru a obține distanța precisă dintre cele două coordonate.

În acest proces se consideră posibilă o eroare de maxim 10 metrii deoarece setul de coordonate oferite de CTP Open Data[2] se poate schimba de la o zi la alta. În final, se returnează un array care cuprinde ID-ul intern prezent în setul de date obținut de pe platforma CTP Open Data[2] care a fost verificat cu şablonul curent și rația de potrivire, data de rezultatul împărțirii dintre numărul de coordonate geografice comune găsite și numărul total de coordonate geografice aflate în şablon.

Acest rezultat este trimis înapoi către funcția *mapper*, pentru a fi comparat ulterior cu celelalte rezultate obținute anterior. Astfel, şablonului curent i se va asocia ID-ul intern cu rația de potrivire maximă. Această asociere este salvată ulterior în fișierul denumit *mapped_vehicles*, pentru ca datele din acesta să fie folosite în componenta *Route Provider*.

```
def mapper():
    dataFiles = os.listdir(f.elim_noise(dataPath))
    fileName = "mapped_vehicles.txt"
    dataArr = rg.iterator("shapes", "mapping/id_mapping.txt")
    if (path.exists(outputPath + fileName) == False):
        outFile = open(outputPath + fileName, "x")
    else:
        outFile = open(outputPath + fileName, "w")
    print("[mapper] Started mapping!")
    for iter in dataArr:
        max = [str(iter[0][0]) + " " + str(iter[0][1]), "", 0]
        iter.pop(0)
        print("[mapper] Mapping " + max[0])
        for d in dataFiles:
            f.elim_duplicates(d, filteredPath)
            retVal = match_ratio(filteredPath + d, iter)
            if (retVal[1] > max[2]):
                max[1] = retVal[0]
                max[2] = retVal[1]
        outFile.write(str(max) + "\n")
    outFile.close()
    print("[mapper] Done mapping!")
```

Figura 4.6. Structura funcției *mapper*; care oferă date de intrare funcției *match_ratio*

²⁵ <https://www.movable-type.co.uk/scripts/latlong.html>

```

def match_ratio(dataFile, patternArr):
    matchCounter = 0
    with open(dataFile) as data:
        dataLines = data.readlines()
        for iter in patternArr:
            for dataLine in dataLines:
                splittedDataLine = dataLine.split(",")
                dataCoord = (float(splittedDataLine[0]), float(splittedDataLine[1]))
                if (distance(iter, dataCoord) < 11.0):
                    matchCounter = matchCounter + 1
                    break
            if (matchCounter == 0):
                return [str(dataFile).split("//")[-1], 0]
        else:
            return [str(dataFile).split("//")[-1], matchCounter / len(patternArr)]

```

Figura 4.7. Structura funcției *match_ratio*, care compară un şablon cu un fișier de obținut de Location Grabber

```

['bus 3b', '1086.txt', 0.8898305084745762]
['tram 3', '458.txt', 0.8571428571428571]
['bus 18', '2022.txt', 0.7717391304347826]

```

Figura 4.8. Exemplu de rezultat obținut în urma rulării funcției Mapper

4.3.4. Route Provider

Componenta *Route Provider* conține o colecție de funcții care furnizează datele necesare aplicației “*GoAnywhere*”. În continuare, voi enumera funcțiile prin referințe la rutele expuse de API unde acestea sunt apelate, urmând ca apoi să le descriu modul de funcționare.

- */providecoordinates* instanțiază thread-ul *provideCoordinatesThread*, acesta urmând să efectueze un apel către funcția *provide_coordinates*.
- */getroutes* instanțiază thread-ul *getRoutesThread*, care efectuează un apel la funcția *get_routes*.
- */inforoute* instanțiază thread-ul *infoRouteThread*, acesta apelând la rândul său funcția *provide_info_route*.

Funcția *provide_coordinates*, reprezentată în Figura 4.9, are ca scop preluarea destinației introdusă de utilizator în căsuța “*Destination*” dar și a locației de start din căsuța “Start Location” în cazul în care s-a optat pentru opțiunea de a se introduce o locație de pornire în detrimentul folosirii locației curentă. Astfel, este efectuat un GET request către WINK Nominatim²⁶[25], la care se adaugă datele obținute mai sus plus un formator pentru a obține datele dorite. Ceea ce obținem în urma apelării funcției este o autocompletare a adresei și datele returnate constă în coordonatele acesteia din urmă pentru a dispune de o localizare mai precisă.

²⁶ <https://m-nominatim.wink.ro/search/>

```

def provide_coordinates(dest):
    destURL = const.addrURL + dest + const.addrFormatter
    destContent = js.loads(data_downloader(destURL))
    destCoord = (destContent[0]["lat"], destContent[0]["lon"])
    return destCoord

```

Figura 4.9. Reprezentarea funcției *provide_coordinates*

Funcția *get_routes*, reprezentată în Figura 4.10 are scopul de a sugera rute utilizatorului pe baza coordonatelor geografice ale locației de start și a destinației obținute la pasul anterior, urmând ca aceste date să fie trimise către aplicația “GoAnywhere” și să fie prezentate utilizatorului prin intermediul fragmentului “Lines”.

Datele despre rute sunt obținut în urma unui GET request către WINK Public API[6] la care se adaugă calea “*routing/routes/*” urmată de perechile de coordonate geografice ale locației de plecare și cele ale destinației. Rezultatul oferit de către această funcție este inclus într-un array și este trimis ca răspuns către aplicația “GoAnywhere”, unde fiecare rută sugerată este afișată într-un card cu informații expandabil.

```

# Function that provides all available routes from nearest station to destination #
def get_routes(current, dest):
    routeURL = const.apiURL + "/routing/routes/" + str(current[0]) + "/" + str(current[1]) + "/" + str(dest[0]) + "/" + str(dest[1])
    availableRoutes = []
    routes = js.loads(data_downloader(routeURL))
    for i in routes["data"]:
        availableRoutes.append(str(len(availableRoutes)) + " - " + str(i["routes"][0]['routeName']).replace("5 TRAMVAI", "Tramvai 5"))
    return [f.to_json(availableRoutes), routes["data"]]

```

Figura 4.10. Funcția *routes*, care oferă toate rutele posibile în funcție de locația de plecare și de destinație

Funcția *provide_info_route* are ca scop oferirea de informații despre o rută selectată de utilizator din cele puse la dispoziție de către funcția *get_routes*. În continuare, voi descrie funcția *provide_info_route* pe secvențe.

În primul rând, se vor căuta stațiile cele mai apropiate de locația de start cât și de destinație prin intermediul coordonatelor geografice puse la dispoziție de către funcția *provide_coordinates*, luând în considerare ruta aleasă de către utilizator de la pasul anterior. Această posibilitate este oferită de funcția *provide_nearest_station*, care este reprezentată în Figura 4.11.

```

# Provide nearest station from the user's current location #
def provide_nearest_station(currLocation, destLocation, vehicle):
    stopsArr = get_all_stops()
    minDistanceCurr = 10000.0
    minDistanceDest = 10000.0
    nearestStationCurr = ("", 0.0, 0.0, "")
    nearestStationDest = ("", 0.0, 0.0, "")
    for iter in stopsArr:
        isVehiclePresent = vehicle in iter[3]
        if (isVehiclePresent == True):
            iterCoord = (iter[1], iter[2])
            currentDistanceCurr = distance((currLocation[0], currLocation[1]), iterCoord)
            currentDistanceDest = distance((destLocation[0], destLocation[1]), iterCoord)
            if (currentDistanceCurr < minDistanceCurr):
                minDistanceCurr = currentDistanceCurr
                nearestStationCurr = iter
            if (currentDistanceDest < minDistanceDest):
                minDistanceDest = currentDistanceDest
                nearestStationDest = iter
    return [nearestStationCurr, nearestStationDest]

```

Figura 4.11. Funcția `provide_nearest_station` care oferă coordonatele geografice a celor mai apropiate stații

Funcția `provide_nearest_station` apelează la rândul ei funcția `get_all_stops`, care returnează un array care cuprinde totalitatea stațiilor din oraș însotite de mijloacele de transport în comun care au incluse în ruta lor acea stație. Acest array este obținut cu ajutorul funcției `stops` din componenta *Route Grabber*, pe care am descris-o în subcapitolul 4.3.2.

După ce au fost obținute toate stațiile, vor fi inițializate două variabile care vor reprezenta cele mai apropiate stații de locația curentă și de destinație. Ulterior, se va efectua o iterație printre toate stațiile obținute și se vor calcula valorile minime în funcție de distanță dintre perechile de coordonatele geografice primite ca parametrii și stația curentă din timpul iterației. Această distanță este calculată folosind librăria Haversine[12] pentru a dispune de un rezultat precis. Rezultatul obținut constă într-un tuplu care conține cele mai apropiate două stații.

Revenind la `provide_info_route`, în urma apelării funcției `get_all_stops` este obținut array-ul cu totalitatea informațiilor despre o rută. Astfel, în următoarea secvență se va itera prin linia array-ului care reprezintă ruta aleasă de utilizator. Aici se crează o listă cu toate stațiile mijlocului de transport selectat, unde stația de plecare cât și cea de sosire sunt marcate cu indicativele “stație de plecare” și “stație de sosire”. Aceasta secvență poate fi observată în Figura 4.12.

```

counter = 1
for i in data[routeNumber]["routes"][0]["routeWaypoints"]:
    if (i["name"] == data[routeNumber]["routes"][0]["statiilePlecareNume"]):
        output.append("statiile " + str(counter) + " (plecare) - " + f.elim_diacritics(i["name"]))
    elif (i["name"] == data[routeNumber]["routes"][0]["statiileSosireNume"]):
        output.append("statiile " + str(counter) + " (sosire) - " + f.elim_diacritics(i["name"]))
    else: output.append("statiile " + str(counter) + " - " + f.elim_diacritics(i["name"]))
    counter = counter + 1
output.append("~")

```

Figura 4.12. Secvența din funcția `provide_info_route` care obține lista stațiilor unui vehicul selectat

Pentru a obține date în timp real pentru mijlocul de transport selectat, este necesară selectarea ID-ului corespondent rutei selectate din colecția de date pusă la dispoziție de către platforma CTP Open Data[2]. După cum am menționat în subcapitolul 4.3.1 la componenta “*Location Grabber*”, ID-urile sunt grupate ca fișiere diferite care conțin coordonatele geografice a locației în timp real a unui mijloc de transport. La acest pas, se caută fișierul care conține date despre localizare pentru a se putea afla ultima locație a mijlocului de transport dorit. Aceasta secvență este reprezentată în Figura 4.13.

```

with open(mappedPath + mappedFileName) as map:
    lines = map.readlines()
    for iter in lines:
        iterSplitted = iter.replace("[", "").replace("]", "").replace("'", "").split(",")
        if(iterSplitted[0] == routeName.replace("_", " ")):
            dataFile = str(iterSplitted[1]).replace(" ", "")
            break
    map.close()

```

Figura 4.13. Secvența care se ocupă de localizarea fișierului care conține ultima locație a unui vehicul

După ce este selectat fișierul asociat mijlocului de transport dorit, se va returna ultima linie a acestuia, astfel afișându-se ultima locație a acestuia în timp real prin intermediul funcției `provide_last_location_bus_tram`, reprezentată în Figura 4.14, care este proiectată într-un mod în care să returneze eficient ultima linie a unui fișier cu un număr considerabil de linii. Având în vedere că la fiecare minut platforma CTP Open Data[2] oferă câte o nouă pereche de coordonate geografice, eficiența în manipularea unor astfel de fișiere este foarte importantă.

```

# Provide last location of a bus/ tram using data files generated from CTP Open Data after mapping #
def provide_last_location_bus_tram(filename):
    with open(filename, 'rb') as data:
        try:
            data.seek(-2, os.SEEK_END)
            while data.read(1) != b'\n':
                data.seek(-2, os.SEEK_CUR)
        except OSError:
            data.seek(0)
        last_line = data.readline().decode()
    data.close()
    return last_line

```

Figura 4.14. Funcția care are ca scop returnarea ultimei linii a unui fișier dat ca parametru într-un mod eficient

Datele obținute în final de către funcția *provide_info_route*, care urmează să fie trimise către aplicația GoAnywhere pentru a putea popula componentele fragmentului “Lines” sunt următoarele:

- Stația cea mai apropiată de locația curentă.
- Numărul autobuzului sau al tramvaiului.
- Ruta pe care acesta o parcurge, cu stațiile de plecare și de sosire evidențiate.
- Ultima locație în care mijlocul de transport a fost prezent.
- Ora la care a fost înregistrată ultima locație.
- Distanța dintre stația de plecare și vehicul.
- Numărul de stații parcurse până la destinație.
- Durata în medie a cursei exprimată în minute.

Datele obținute de funcțiile *get_routes* și *provide_info_route* sunt trimise către funcția *to_json*, acesta reprezentând un convertor către formatul JSON[24] create de mine pentru a putea servi date către aplicația “*GoAnywhere*” într-o manieră organizată și facilă. Funcția care se ocupă de formatarea datelor este reprezentată în *Figura 4.15*.

```
# Function that formats an array as a json #
def to_json(array):
    jsonOutput = "" + array[0]
    array.pop(0)
    for iter in array:
        if (type(iter) == tuple):
            splitted1 = iter[0].split(" - ")
            splitted2 = iter[1].split(" - ")
            formatedJsonLine = "{" + "\"" + str(splitted1[0]) + "\"" + " : " + "\"" + str(splitted1[1]) + "\"" + ", "
            formatedJsonLine = formatedJsonLine + " \\" + str(splitted2[0]) + "\"" + " : " + " \\" + str(splitted2[1]) + "\"" + ","
            jsonOutput = jsonOutput + formatedJsonLine
        else:
            separatorCheck = " - " in iter
            endCheck = "~" in iter
            if(endCheck == True):
                formatedJsonLine = "],""
                jsonOutput = jsonOutput[:-1] + formatedJsonLine
            else:
                if (separatorCheck == False):
                    formatedJsonLine = "\\" + iter + "\"" + " : ["
                    jsonOutput = jsonOutput + formatedJsonLine
                else:
                    splitted = iter.split(" - ")
                    formatedJsonLine = "\\" + splitted[0] + "\"" + ":" + "\\" + splitted[1] + "\"" + ", "
                    jsonOutput = jsonOutput + formatedJsonLine
        if (jsonOutput[0] == "{"):
            jsonOutput = jsonOutput[:-1] + "}"
        else:
            jsonOutput = jsonOutput[:-1] + "]"
    return jsonOutput
```

Figura 4.15. Funcția to_json care se ocupă de convertirea datelor la formatul JSON[24]

Pentru că am făcut o mențiune la adresa datelor oferite de către funcțiile *get_routes* și *provide_info_route* mai sus, în *Figura 4.16* și *Figura 4.17* poate fi observat rezultatul request-urilor către rutele */getroutes* și */inforoute*.

```
[{"nearestStation": "('Zimbru', 47.166828288229745, 27.55598545074463, 'bus_28')", "vehicul": "bus_28", " ruta": [{}], {"id": "2", "route": "Autobuzul 36"}, {"id": "3", "route": "Autobuzul 28"}, {"id": "4", "route": "Autobuzul 44"}, {"id": "5", "route": "Autobuzul 44B"}, {"nearestStation": "('Minerva (sosire)', 47.166677, 27.555176)", "ora": "17:28:09", "distanță": "63.462971327354936", "nrStări": "1", "arrivalTime": "3 min"}]
```

Figura 4.16. Rutele disponibile pentru traseul Zimbru - Minerva

Figura 4.17. Ruta autobuzului 28 extinsă în urma selectării variantei bus 28 din colecția pusă la dispoziție de /getroutes

4.3.5. Motivul creării API-ului

Motivul pentru care acest API a fost creat a fost dorința de a avea la dispoziție un format standard al datelor pe care o aplicație de tip *planificator de rute a mijloacelor de transport în comun*, precum este și aplicația “*GoAnywhere*”, să îl poată folosi. Inițial, în căutarea mea de informații ce m-ar putea ajuta la dezvoltarea aplicației, am observat faptul că Google pune la dispoziția un set de API-uri denumit Transit APIs²⁷[26], ce au ca rol standardizarea datelor oferite de către agențiile de transport în comun și punerea acestora la dispoziție în timp real dezvoltatorilor de aplicații.

Plecând cu această idee în minte, am început să mă documentez cu privire la datele oferite de compania de transport în comun locală, mai exact CTP²⁸[27] Iași. Datele pe care aceștia le oferă sunt în timp real, însă, nu sunt suficiente pentru a le putea folosi în contextul discuției, anume în construirea unui *planificator de rute a mijloacelor de transport în comun*, acest aspect putând fi văzut în *Figura 4.1* de la subcapitolul 4.3.1.

Problema majoră la acest set de date este lipsa numărului pe care un vehicul îl are și pe care un utilizator al aplicației îl poate recunoaște (spre exemplu, autobuzul 28). În schimb, ne este pus la dispozitie un ID unic, ce reprezintă un identificator al vehiculului la nivel de flotă,

²⁷ <https://developers.google.com/transit>

²⁸ <https://www.sctpiasi.ro/>

aspect care este inutil din motiv că nu este publică asocierea dintre numărul unui vehicul și ID-ul de flotă.

Următorul pas a fost să mă documentez pe tema formatului GTFS²⁹[28] (General Transit Feed Specification) știind că un astfel de format poate pune la dispoziție toate informațiile necesare care trebuie trimise către aplicație. În căutarea efectuată, am dat peste un proiect intitulat “*Iasi-GTFS-Exporter*”³⁰[29]³⁰, unde autorul afirma că a creat un API ce ar trebui să asocieze datele primite de la CTP Open Data[2] cu datele primite de la WINK Public API[6]. Însă, proiectul a fost abandonat și folosea tehnici învechite în procesul de asociere.

Așadar, am început să analizez ambele seturi de date primite de la cele două platforme și am ajuns la această soluție, pe care se bazează API-ul construit de mine. Asocierea dintre coordonatele geografice primite în timp real de la CTP Open Data[2] și formele rutelor constituite de asemenea din coordonate geografice care sunt înregistrate în WINK Public API[6] duc la un proces de asociere ce oferă date precise oricând, deoarece, numerele autobuzelor se pot schimba de la o zi la alta (spre exemplu, dacă în ziua curentă vehiculul cu ID-ul de flotă *1990* este pus în circulație ca fiind *autobuzul 47*, există șanse ca în următoarea zi, vehiculul cu același ID de flotă să devină *tramvaiul 6*). Acest caz este acoperit de procesul de *mapping* care este efectuat constant.

În concluzie, datele oferite de API-ul construit de mine nu sunt reprezentate în formatul GTFS[28], deoarece acest lucru presupune ca formatul fișierelor să fie de tip protobuf³¹[30], însă, maniera în care acestea sunt colectate, prelucrate și asociate constituie o adaptare a acestui format, construite special pentru acoperirea nevoii mele de date necesare aplicației “*GoAnywhere*”. Însă, acest API are un potențial semnificativ deoarece poate revoluționa industria mijloacelor de transport în comun prin convertirea sistemului de date al transportului în comun Ieșean către un formatul standard folosit la nivel global.

²⁹ <https://gtfs.org/>

³⁰ <https://github.com/FlashWebIT/Iasi-GTFS-Exporter>

³¹ <https://developers.google.com/transit/gtfs-realtime/gtfs-realtime-proto>

4.4. Modul de funcționare a aplicației GoAnywhere

4.4.1. Permisii de localizare

După cum deja am menționat în subcapitolul 3.2.2, procesul de cerere și primire a permisiunilor de localizare la runtime este realizat cu ajutorul Activity Result API[31], acesta permitând dezvoltatorilor de aplicații să folosească cele mai bune practici în tratarea diverselor cazuri de accept sau refuz a permisiunilor. În cazul de față, pentru ca aplicația “GoAnywhere” să funcționeze în parametrii optimi, a fost nevoie de două astfel de permisiuni, mai exact “*FINE_LOCATION*” și ”*BACKGROUND_LOCATION*” în forma lor prescurtată, cea din urmă fiind necesară dispozitivelor ce rulează minim versiune de Android Q. După cum se poate vedea în Figura 5.1 și Figura 5.2, cele două permisiuni au propria funcție prin care sunt invocate. Acest lucru permite o compatibilitate la fel de bună și cu dispozitivele ce rulează o versiune mai veche de Android decât Q.

```
private fun requestLocationPermission() {
    ActivityCompat.requestPermissions(
        requireActivity(),
        arrayOf(
            Manifest.permission.ACCESS_FINE_LOCATION,
        ),
        MY_PERMISSIONS_REQUEST_LOCATION
    )
}
```

Figura 5.1. *FINE_LOCATION*, necesară indiferent de versiunea de android a dispozitivului

```
private fun requestBackgroundLocationPermission() {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.Q) {
        ActivityCompat.requestPermissions(
            requireActivity(),
            arrayOf(
                Manifest.permission.ACCESS_BACKGROUND_LOCATION
            ),
            MY_PERMISSIONS_REQUEST_BACKGROUND_LOCATION
        )
    } else {
        ActivityCompat.requestPermissions(
            requireActivity(),
            arrayOf(Manifest.permission.ACCESS_FINE_LOCATION),
            MY_PERMISSIONS_REQUEST_LOCATION
        )
    }
}
```

Figura 5.2. *BACKGROUND_LOCATION*, necesară de la Android Q/ API Level 29 și mai sus

4.4.2. Main ViewModel

Main ViewModel este ViewModel-ul pe care se bazează toate fragmentele, acest aspect presupunând că layout-ul pus la dispoziție de acesta va fi folosit de toate fragmentele. Acesta este un avantaj major deoarece se păstrează o consistență în aspect pe tot parcursul navigării printre paginile aplicației. Însă, acesta nu e singurul motiv, deoarece acest ViewModel este responsabil de aducerea de date în View-uri, mai exact în fragmentele pe care le avem la dispoziție. Interfața acestor ViewModels și Fragmente este reprezentată în format XML[17] și în cazul Main ViewModel, poate fi observată în *Figura 5.3* și *Figura 5.4*. Cele 2 figuri enumerate mai sus formează bara de sus ce poartă numele aplicației, *Figura 5.5*.

```
<com.google.android.material.appbar.AppBarLayout
    android:id="@+id/appBarLayout"
    android:layout_width="match_parent"
    android:layout_height="?android:attr/actionBarSize"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.5"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent">

    <com.google.android.material.appbar.MaterialToolbar
        android:background="@color/white"
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
```

```
<com.google.android.material.textview.MaterialTextView
    android:id="@+id/ToolbarTitle"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:text="GoAnywhere"
    android:textSize="25sp"
    android:textStyle="bold"
    android:gravity="center_vertical"
    android:textColor="@android:color/black"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.5"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

</com.google.android.material.appbar.MaterialToolbar>
</com.google.android.material.appbar.AppBarLayout>
```

Figura 5.3. Bara ce conține numele aplicației

Figura 5.4. Text View-ul cu numele aplicației



Figura 5.5. Bara de titlu a aplicației

Încadrarea fragmentelor în layout-ul unui ViewModel se face prin containerul “FrameLayout”, care găzduiește designul unui fragment, de asemenea în format XML[17] și el, și permite navigarea prin intermediul barei de navigare. Un astfel de “FrameLayout” poate fi observat la *Figura 5.6*. În cazul meu, pentru tag-ul “fragment” m-am folosit de bara de navigare pentru a ști ce fragment urmează să fie încărcat prin intermediul tranzițiilor definite în graful de navigare, din subcapitolul 3.2.1.

```

<FrameLayout
    android:id="@+id/flFragment"
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:background="@drawable/background"
    app:layout_constraintBottom_toTopOf="@+id/bottomNavigationView"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/appBarLayout">

    <fragment
        android:id="@+id/nav_host_fragment"
        android:name="androidx.navigation.fragment.NavHostFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:defaultNavHost="true"
        app:navGraph="@navigation/nav_graph"
        tools:layout="@layout/fragment_directions" />

</FrameLayout>

```

Figura 5.6. Reprezentare containerului unde sunt încărcate fragmentele

4.4.3. API Requests

Obținerea de date din partea API-ului se face prin intermediul librăriei Retrofit[14], care efectuează request-uri către API-uri externe. În acest proces, json-urile obținute de la API sunt transformate în POJO³²[32] și ulterior persistate în Room Database[9], în cazul unui obiect obținut printr-un request către “/getroutes” sau trimise direct către un View și afișate în interfață, ca în cazul rutelor obținute în pagina Lines.

Un request către un API este reprezentat în *Figura 5.7*, acesta fiind făcut către“/getroutes”:

```

interface DataAPI {

    @GET("value: "/getroutes")
    suspend fun getRoutes(@Query("value: "currentlat") q1: String,
                         @Query("value: "currentlon") q2: String,
                         @Query("value: "destlat") q3: String,
                         @Query("value: "destlon") q4: String): Response<List<RouteModel>>
}

```

Figura 5.7. Get request către /getroutes prin intermediul Retrofit[14]

³² <https://www.baeldung.com/java-pojo-class>

4.4.5. RecyclerView

Pentru a putea afișa datele obținute de la API în mod dinamic, am decis să folosesc RecyclerView³³[33] împreună cu Coroutines³⁴[34] pentru a nu bloca întreg firul de execuție al aplicației, având în vedere că “GoAnywhere” este o aplicație de tip “single task”. Pentru a discuta asupra unui exemplu concret, vom considera secvența de preluare a datelor disponibile în urma unui GET request către “/getroutes”.

```
override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): RouteViewHolder {
    return RouteViewHolder(ItemTripBinding.inflate(
        LayoutInflater.from(parent.context),
        parent,
        attachToParent: false
    ))
}

@SuppressLint("SetTextI18n")
override fun onBindViewHolder(holder: RouteViewHolder, position: Int) {
    holder.binding.apply { this: ItemTripBinding
        val routes = routes[position]
        tvTitle.text = routes.id + ": " + routes.route
    }
}
```

Figura 5.8. Crearea modelului ce va fi folosit în afișarea dinamică a datelor

În primul rând, în clasa *RouteAdapter* îmi creez un template ce va fi folosit la afișarea unei liste cu o lungime nedeterminată. Având în vedere că am reușit să convertesc json-ul primit de la API într-un POJO[32], am posibilitatea să mă folosesc de proprietățile acestuia pentru a completa layout-ul ce va fi folosit de RecyclerView[33].

Ulterior, după cum se poate observa în Figura 5.9, în interiorul unui “lifecyclescope”, este realizat request-ul către API și răspunsul este verificat înainte de a fi procesat.

³³ <https://developer.android.com/guide/topics/ui/layout/recyclerview>

³⁴ <https://kotlinlang.org/docs/coroutines-overview.html>

```
lifecycleScope.launchWhenCreated { this: CoroutineScope
    _binding!!.progressBar.isVisible = true
    val response = try {
        // test api call
        RetrofitInstance.api.getRoutes(q1: "47.1665182", q2: "27.5579386", q3: "47.1631825", q4: "27.5645637")
    } catch (e : IOException){
        Timber.v( message: "IOException: No internet connection!")
        Timber.v( message: "$e")
        _binding!!.progressBar.isVisible = false
        return@launchWhenCreated
    } catch (e:HttpException) {
        Timber.v( message: "HttpException: Unexpected response!")
        _binding!!.progressBar.isVisible = false
        return@launchWhenCreated
    }
    if(response.isSuccessful && response.body() != null){
        routesAdapter.routes = response.body()!!
    } else {
        Timber.v( message: "Err: Response not successfully!")
    }
    _binding!!.progressBar.isVisible = false
}
```

Figura 5.9. Request către API și procesare răspuns

Concluzii și viitoare îmbunătățiri

Pe parcursul dezvoltării aplicației “GoAnywhere”, pot spune că am evoluat ca programator, având ocazia să lucrez cu cele mai noi tehnologii în materie de dezvoltare software pe platforme mobile. Mai presus de faptul că am avut posibilitatea să studiez tehnologiile folosite în dezvoltarea acestei aplicații, este înțelegerea ideii de creare unui proiect de la zero, aceasta fiind o experiență care poate pune o cărămidă solidă la baza de cunoștințe pe care un programator o deține. Focusul meu principal de la începutul dezvoltării acestui proiect a fost ideea de a standardiza datele pe care un programator le poate obține de la o agenție de transport în comun, astfel contribuind la o ușurare a procesului de dezvoltare a unui planificator de rute destinate mijloacelor de transport în comun. Totodată, prin intermediul aplicației, am intenționat să aduc o soluție ușor de folosit de către oricine, oferind date ce sunt de interes principal pentru majoritatea potențialilor utilizatori.

Viitoare îmbunătățiri pe care aş vrea să le implementez în această aplicație sunt:

- Notificări push oferite atunci când stația destinație este următorul punct de oprire.
- Indicații de deplasare live către cea mai apropiată stație de către utilizator.
- Notificări push atunci când utilizatorul se apropie de un punct cultural.
- Un nou design uniform, care să se potrivească cu restul elementelor de interfață din Android.