**DOCUMENTAȚIE PROIECT IS**

**Nume prenume Țigărean Lucian**

**Grupa 302310**

**Indrumator de proiect Mitrea Delia**

Contents

[1. Cerințe Funcționale 3](#_Toc30070052)

[2. Precizarea limbajului de programare ales, a sistemului de operare sub care se face implementarea, a cerințelor hardware 4](#_Toc30070053)

[3. Obiective 5](#_Toc30070054)

[3.1. Obiectiv Principal: 5](#_Toc30070055)

[3.2. Obective Secundare: 5](#_Toc30070056)

[4. Analiza Problemei 6](#_Toc30070057)

[5. Proiectare 6](#_Toc30070058)

[5.1. Structuri de date 6](#_Toc30070059)

[5.2. Diagrama de clasa 8](#_Toc30070060)

[5.3. Diagrama de use case 8](#_Toc30070061)

[5.4. Diagrama de secvență 9](#_Toc30070062)

[5.5. Diagrama de activitate 10](#_Toc30070063)

[5.6. Algoritmi 11](#_Toc30070064)

[6. Implementare 12](#_Toc30070065)

[6.1. Implementarea funcționalitații 12](#_Toc30070066)

[6.2. Design pattern 15](#_Toc30070067)

[7. Testare 17](#_Toc30070068)

[8. Concluzii si Dezvoltari Ulterioare 18](#_Toc30070069)

[9. Bibliografie 18](#_Toc30070070)

# Cerințe Funcționale

Dezvoltarea unei aplicații android cu scopul de a oferi posesorilor de autovehicule posibilitate de a-și monitoriza autovehiculul(de a salva locația unde este parcată), de a interacționa cu alți utilizatori in scopul de a oferi detalii despre observarea unei acțiuni de ridicare a mașinilor parcate in locuri nepermise.

Astfel cerințele funcționale ale aplicației android sunt:

* Oferirea unei interfețe de logare pentru utilizatori
* Posibiliatea de înregistrare a utilizatoriilor noi
* Accesarea server-ului Google Maps pentru a putea implementa afișarea unui hărți.
* Posibiliatea salvarii locației mașinii pe hartă
* Opțiunea de a trimite o alertă pentru alți utilizatori
* Primirea unei notificari de tip push prin care ești notificat asupra faptului ca este posibil sa ti se ridice mașina
* Opțiunea utilizatorului de a-și putea modifica datele contului

# Precizarea limbajului de programare ales, a sistemului de operare sub care se face implementarea, a cerințelor hardware

Aplicațiile Android pot fi scrise folosind limbi Kotlin, Java și C ++. Tool-ul Android SDK compileaza codul cu toate resursele si datele din folosite, într-un APK, un pachet Android, care este un fișier arhivă cu .apk. Un fișier APK conține tot conținutul unei aplicații Android și este fișierul pe care dispozitivele bazate pe Android le folosesc pentru instalarea aplicației.

Fiecare aplicație Android trăiește în propriul său sandbox de securitate, protejat de următoarele caracteristici de securitate Android:

* Sistemul de operare Android este un sistem Linux multi-utilizator în care fiecare aplicație este un utilizator diferit.
* În mod implicit, sistemul atribuie fiecărei aplicații un ID de utilizator Linux unic (ID-ul este folosit doar de sistem și nu este cunoscut pentru aplicație). Sistemul stabilește permisiunile pentru toate fișierele dintr-o aplicație, astfel încât numai ID-ul de utilizator atribuit aplicației respective să le poată accesa.
* Fiecare proces are propria sa mașină virtuală (VM), astfel încât codul unei aplicații rulează izolat de alte aplicații.
* În mod implicit, fiecare aplicație rulează în propriul proces Linux. Sistemul Android începe procesul când oricare dintre componentele aplicației trebuie să fie executate și apoi oprește procesul atunci când nu mai este nevoie sau când sistemul trebuie să recupereze memoria pentru alte aplicații.

Componentele unei aplicatii android

Componentele aplicației sunt elementele esențiale ale unei aplicații Android. Fiecare componentă este un punct de intrare prin care sistemul sau un utilizator pot intra în aplicația ta. Unele componente depind de altele.

Există patru tipuri diferite de componente pentru aplicații:

* Activities
* Services
* Broadcast receivers
* Content providers

Fiecare tip servește unui scop distinct și are un ciclu de viață distinct, care definește modul în care componenta este creată și distrusă.

Limbajul ales pentru implementarea aplicatiei este Java. Am ales acest limbaj deoarece este un limbaj orientat pe obiecte cu care sunt familiarizat iar folosirea acestui lui a fost mai simpla.

# Obiective

## Obiectiv Principal:

Obiectivele principale sunt de a implementa toate cerințele funcționale și dacă este posibil implementarea de fucționalitați noi, de a utiliza mediul de dezvoltare Android Studio pentru dezvoltarea aplicației, de a aprofunda cunștințele în android, de a utiliza design pattern pentru o mai buna implementare si pentru evitarea unor erori de programare.

## Obective Secundare:

|  |  |  |
| --- | --- | --- |
| **Obiectiv Secundar** | **Descriere** | **Capitol** |
| Dezvoltarea de use case-uri si scenarii | Sunt descrise use case-uri pentru efectuarea operatiilor specifice utilizatorului: logare, autentificare, actualizare date cont, salvare locație mașina, trimitere alerta catre alți utilizatori | 3 |
| Alegerea structurilor de date | Alegerea unei structuri de date convenabile atat din punct de vedere al eficienței cat si a memoriei ocupate | 4 |
| Impartirea pe clase | Imparțirea pe pachete si clase | 4 |
| Dezvoltarea algoritmilor | Dezvoltarea algoritmilor pentru rezolvarea problemei | 4 |
| Implementarea solutiei | Implementarea solultiei utilizand diferite strategii de rezolvarea pentru problema | 5 |
| Testare | Testarea realizata pe parcursul proiectarii | 6 |

# Analiza Problemei

Scenariul de utilizare pentru:

Logare

* Utilizatorul deschide aplicația
* Introduce email-ul si parola
* Apasa pe butonul de Log In

Crearea Cont nou:

* Utilizatorul deschide aplicatia
* Apasa pe butonul de Autentificare
* Introduce datele
* Apasa pe butonul de validare si creare cont

Trimitere alerta

* Utilizatorul deschide aplicația
* Selecteaza din bottom bar Send alert

Modificare informații cont:

* Utilizatorul deschide aplicația
* Selecteaza din bottom bar Account
* Modifica datele
* Apasa pe butonul de salvarea a modificarilor

# Proiectare

## Structuri de date

Structurile de date folosite sunt List si HashMap.

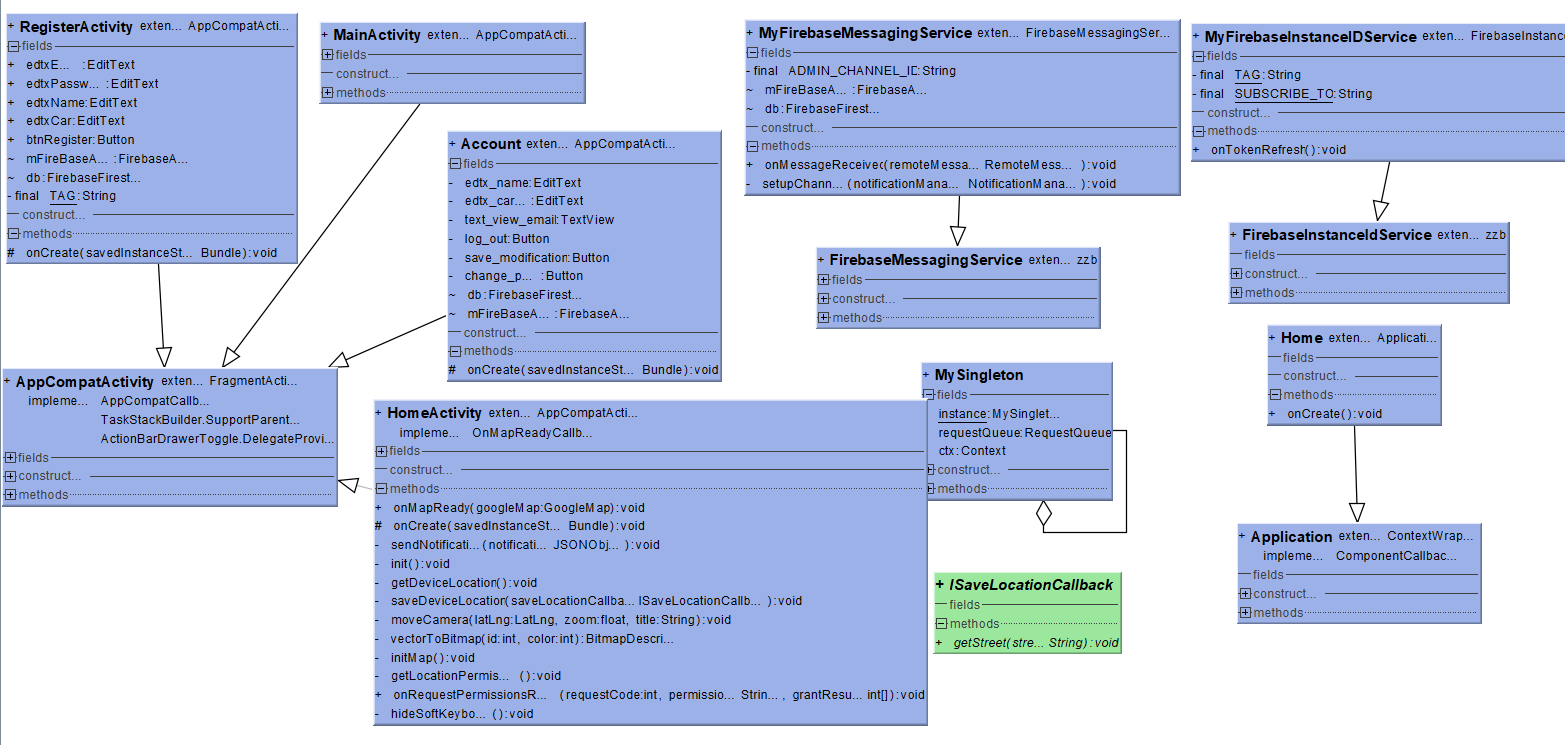
Structura de date HashMap impelmenteaza interfata Map din Java. Aceasta structura stocheaza o pereche de tipul (cheie, valoare). Pentru a accesta o valoare trebuie sa cunoastem cheia. HashMap este cunoscut ca HashMap deoarece utlizeaza o tehnica numita Hashing(„ converteste un string lung la un string mai mic pentru a creste viteza de cautare).

HashMap face parte din pachetul java.util package, implementeaza interfetele Clonable si Serializable. HashMap nu permite chei duplicate dar permite valori duplicate.

Clasa HashMap are metode implementate care ne permit sa lucram mai usor, de exemplu metoda clear() care sterge toate perechile cheie,valoare din hashmap, put(k key, v value) prin care introducem in hashmap un nou set de cheie,valoare, get(Object key) care returneaza valoarea pentru cheia data, keySet() returneaza un set de chei aflate in hashMap, entrySet() care returneaza un setul de cheie,valoare aflate in hashmap, remove(Object key) prin care stergem setul de cheie, valoare cu cheia key din hasMap.

Structura de date folosita este List din pachetul java.util. List implementeaza interfata Collection. Structura de date List contine liste(secvente) de elemente indexate. Listele pot contine duplicate si permit un control precis asupra pozitiei unui element prin intermediul indexului acelui element. Clasa List are metode implementate care ne permite sa lucram mai usor, de exemplu pentru a adauga un element in lista folosim metoda add(Object a), pentru accesare indexata folosim metoda get(int index), pentru a afla numarul de elemente din lista folosim metoda size(), pentru stergere putem folosi atat metoda remove(Object a) cat si remove(int index) care sterge obiectul dat, respectiv sterge obiectul de pe pozitia data.

## Diagrama de clasa

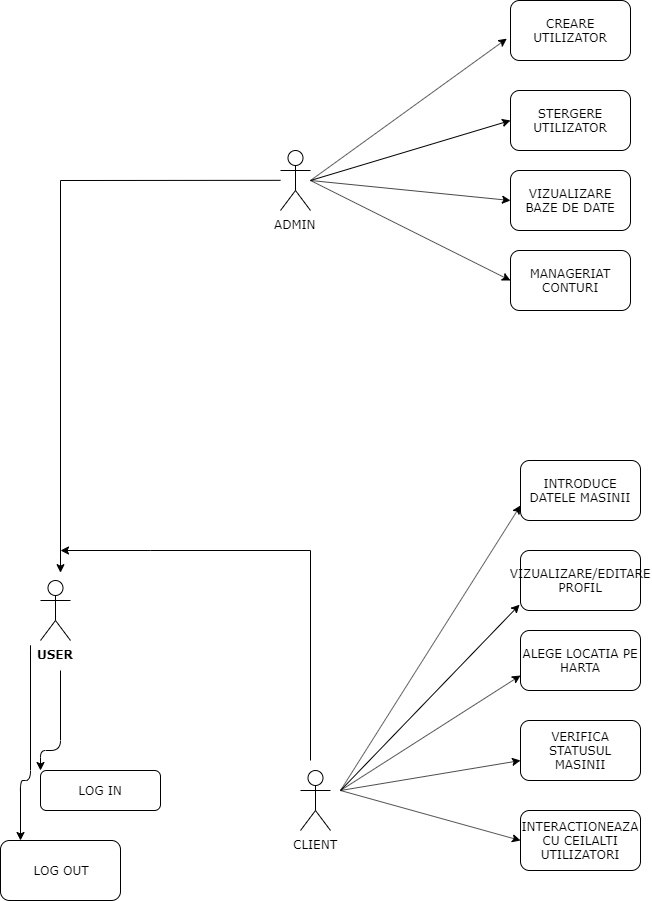


## Diagrama de use case

Diagramele usecase prezinta funcițonalitatea sistemului. O diagramă use case are urmatoarele elemente:cazuri de utilizare, actori, realtii.

In primul ecran al aplicatiei utilizatorul poate sa se logheze sau sa se înregistreze.

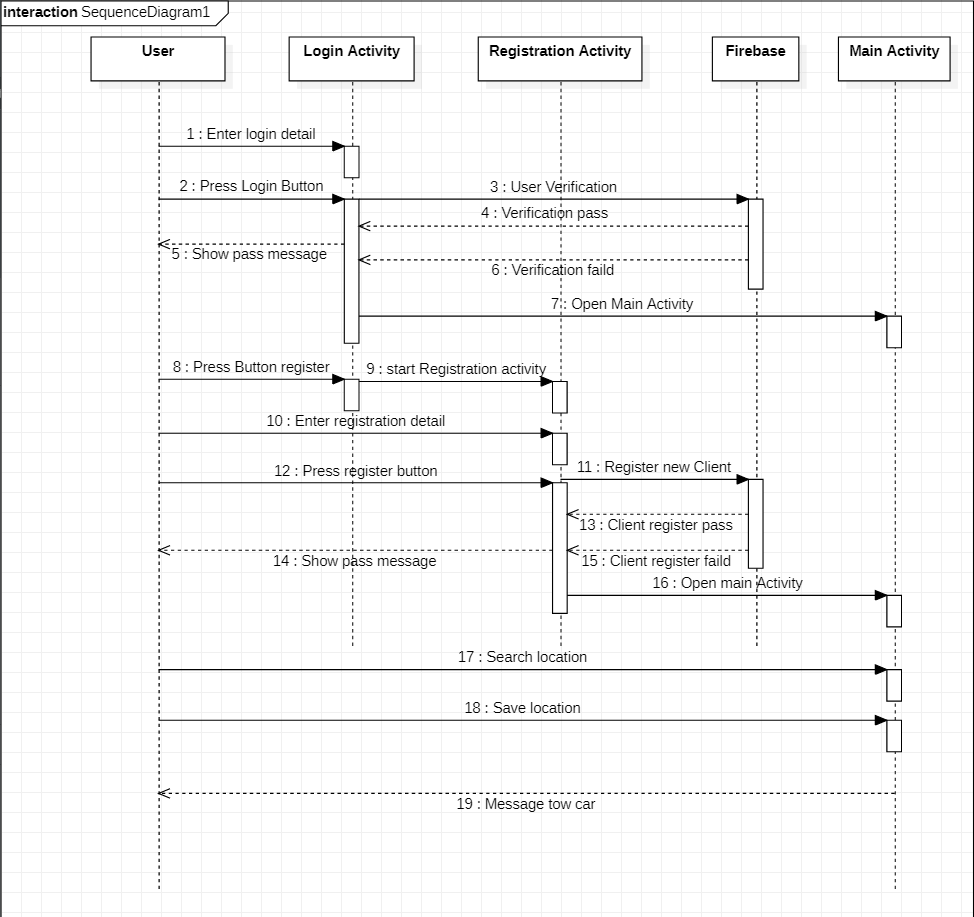
După autentificare apare ecranul principal unde utilizatorul poate alege mai multe opțiuni: Get Location, Save location car, find car, Send alert, Account



## Diagrama de secvență

Diagramele de secvenţă prezintă interacţiunile care au loc si diverse obiecte ale unui sistem, ordonate cronologic.  Ele determină obiectele şi clasele implicate intr-un scenariu şi secvenţele de mesaje transmise si obiecte necesare indeplinirii funcţionalităţii scenariului.  Diagramele de secvenţă sunt asociate unui caz de utilizare.

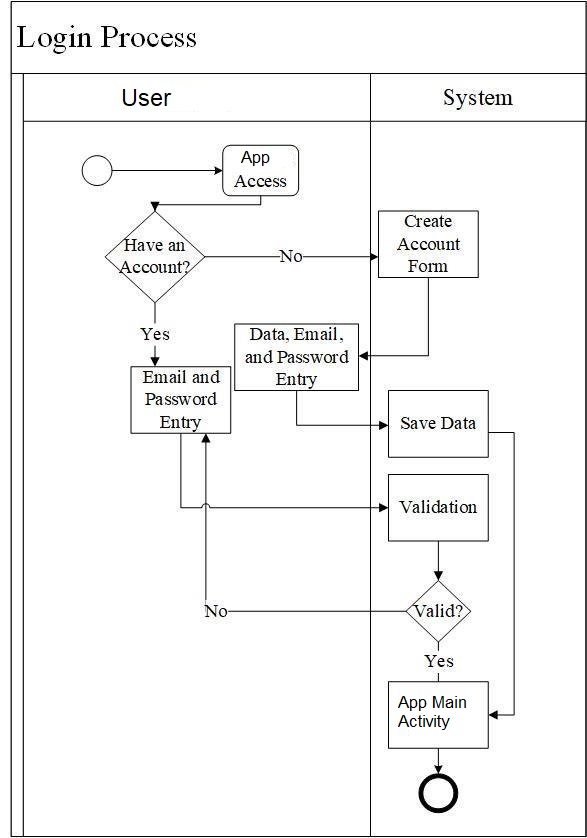
Diagrama de secvența realizată este asociata la doua cazuri de utilizare Login si Registration.



## Diagrama de activitate

Se folosesc pentru modelarea aspectelor dinamice ale unui sistem, la diferite nivele: incepand de la nivelul „business process”, pana la nivel de operatie a unei clase. Din acest motiv, in diagramele de activitate se folosesc un numar mare de simboluri.

Diagrama de activitate realizată prezinta aspectul dinamic al procesului de logare si înregistrare



## Algoritmi

In acest proiect nu s-au folosit algoritmi speciali, doar alogoritmi cu funcționalitate simpla pentru implementarea aplicației.

# Implementare

## Implementarea funcționalitații

Pentru realizarea acestei aplicații s-a folosit un singur pachet com.rdk.ridicariv4 care cuprinde toate clasele necesare funcționalitații aplicației.

Pachetul com.rdk.ridicariv4 cuprinde următoarele clase și interfețe:

* Account
* Home
* HomeActivity
* IsaveLocationCallback
* MainActivity
* MyFirebaseInstanceIdService
* MyFirebaseMessagingService
* MySingleton
* RegisterActivity

**MainActivity** extinde clasa AppCompatActivity(este clasa de bază pentru activități care utilizează libraria suport pentru Action Bar), MainActivity este clasa care serveste drept Launcher in aplicație, in fișierul Manifest.xml este descris ca aceasta activitate sa porneasca prima prin anumite tag-uri specifice.

Clasa MainActivity suprascrie metoda onCreate(), metoda care se apeleaza automat in momentul in care activitatea este creată, unde se seteaza continutul vizual prin setContentView(R.layout.activity\_main), unde activity\_main este un fisier .xml prin care sunt specificate componentele grafice ale acestei activitati. In aceasta clasa avem o instanță de Firebase si anume FirebaseAuth clasa prin care putem comunica cu serverul Firebase pentru a putea stoca date. Aceasta clasa ofera metoda signInWithEmailAndPassword pentru a putea face sing in in aplicatie daca utilizatorul are deja un cont in baza de date Firebase prin datele preluate fin interfata grafica.

**RegisterActivity** extinde AppCompatActivity, este clasa care ofera functionalitate de inregistrare a unui nou utilizator. Aceasta clasa are doua instante catre Firebase, una FirebaseAuth(pentru introducerea unui nou utilizator in baza de date prin email si parola) si FirebaseFirestore pentru accesarea unei baze de date configurata de dezvoltator, pentru a putea retine alte detalii despre utilizator precum ar fi: numele, numarul masinii.

**HomeActivity** este clasa care ofera functionalitatea aplicatiei. Aceasta extinde clasa AppCompatActivity si implementeaza OnMapReadyCallback. Clasa HomeActivity are atributele FirebaseFirestore, FirebaseAuth, GoogleMap(clasa care ofera metodele necesare pentru implementarea unei harti de la google si pentru obtinerea anumitor informatii). In aceasta clasa sunt implementate cererile de permisiune asupra locatiei, permisiune necesara pentru functionalitatea aplicatiei, este implementat un bottom bar care ofera un mic meniu pentru utilizator prin care poate alege urmatoarele optiuni: Find car, Account, Send alert. Functionalitatiile implementate asupra acestor butoane sunt: pentru Find car functionalitatea este de a verifica daca utilizatorul are salvata in baza de date locatia masini, in caz afirmativ se preiau aceste date si se creeaza un marker care este afisat pe harta, pentru Account functionalitatea este de deschide o noua activitate pentru afisarea unor optiuni de modificare a contului, pentru Send alert functionalitatea este de a trimite o cerere, catre serverul Firebase Messaging, de a trimite tuturor utilizatorilor un mesaj prin care sunt avertizati ca este posibil ca masina lor sa fie ridicata, aceasta functionalitate este posibila cu ajutorul obiectelor Json care ofera metode de setare a mesjului si a obiectului JsonObjectRequest care contine ca atribute url catre server si obiectele JsonObjsects care contin mesajul; aceste obiecte JsonObjectRequest sunt trimise clasei MySingleton.

**Account** extinde AppCompatActivity, este clasa care ofera functionalitate butonului, din clasa HomeActivity, Account. Aceasta clasa are ca atribute principale FirebaseFirestore prin care putem modfica baza de date a utilizatorului si FirebaseAuth prin care putem accesa baza de date a utilizatorului pentru a putea modifica. In aceasta clasa sunt trei butoane: unul de a salva modificarile introduse, unul de a oferi posibilitatea utilizatorului de a se deloga si ultimul de a oferi posibilitatea utilizatorului de a-si schimba parola prin deschiderea altei activitati.

**MyFirebaseMessagingService** extinde FirebaseMessagingService, clasa de baza pentru primirea mesajelor de la Firebase Cloud Messaging, Extinderea acestei clase este necesară pentru a putea gestiona mesajele primite de la server. De asemenea, oferă funcționalitate pentru afișarea automată a notificărilor și are metode care sunt invocate pentru a da starea mesajelor upstream. Toate metodele sunt invocate pe un thread de fundal și pot fi apelate atunci când aplicația este în fundal sau nu este deschisă.

Prin extinderea acestei clase putem afisa pentru utilizator un push notification prin care este avertizat ca este posibil ca masini sa fie ridicata. Pentru acest lucru s-a adaugat functionalitate suplimentare fata de cea oferita de FirebaseMessagingService, prin verificarea in baza de date daca utilizatorul are parcata masina pe strada primita ca si mesaj de la server.

**MySingleton** este clasa care restricționează instantierea unei clase la un singur obiect. Această clasă oferă o modalitate de a accesa singurul său obiect care poate fi accesat direct fără a fi necesar să instantieze obiectul clasei. Aceasta clasa utilizeaza biblioteca Volley. Volley este o bibliotecă HTTP care face rețeaua pentru aplicațiile Android mai ușoară și, cel mai important, mai rapid.

## Design pattern

In ingineria software, un design pattern este o solutie generala pentru o problema aparuta in software design. Un design pattern nu este un design fix care poate fi direct transformat in cod. Este o descriere sau un template care poate fi folosit in mai multe situatii diferite pentru a rezolva problema.

Modelele de proiectare pot accelera procesul de dezvoltare prin furnizarea de paradigme de dezvoltare testate și dovedite.

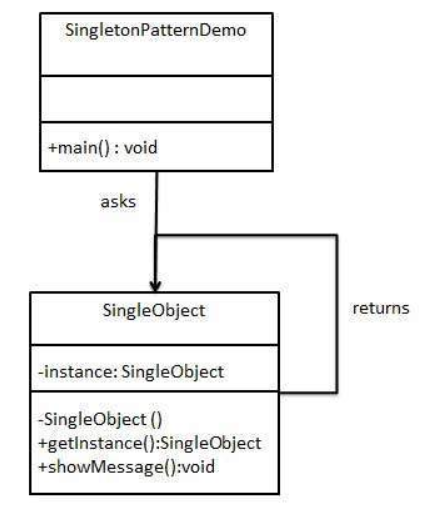
Reutilizarea modelelor de proiectare ajută la prevenirea problemelor subtile care pot cauza probleme majore și îmbunătățește lizibilitatea codurilor pentru programatorii familiarizați cu aceste modele de proiectare.

Design Pattern folosite in aplicatie sunt:

* Singleton Pattern
* Facade Pattern

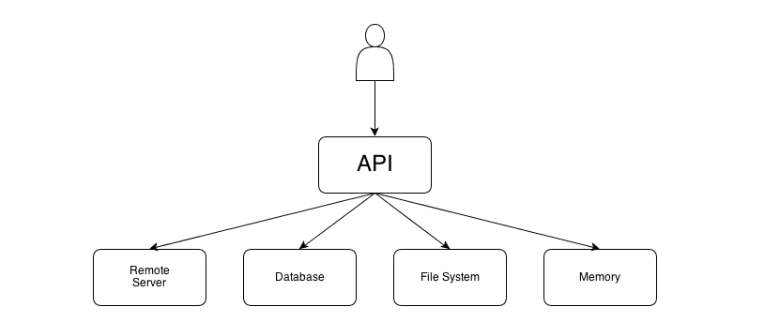
**Singleton Pattern**

Modelul singleton este un model de design care restricționează instantiarea unei clase la un singur obiect. Această clasă oferă o modalitate de a accesa singurul său obiect care poate fi accesat direct fără a fi necesar să instantieze obiectul clasei.



**Facade Pattern**

Facade pattern oferă o interfață de nivel superior care face un set de alte interfețe mai ușor de utilizat. Următoarea diagramă ilustrează mai detaliat această idee:



Analogic cu o fațadă din arhitectură, o fațadă este un obiect care servește ca o interfață care maschează mai mult codul complex.

In aplicatia dezvoltata: aceste design pattern este folosit pentru firebase, pentru implementare harta (google maps).

# Testare

Testarea s-a realizat pe parcursul dezvoltarii aplicatiei, urmarind comportamentul si rezultatele( comparate cu cele asteptate de noi) astfel s-au corectat unele erori si s-a optimizat programul.

# Concluzii si Dezvoltari Ulterioare

Realizarea acestui proiect m-a ajutat sa aprofundez cunostintele legate de android si de a invata noi implementari in android. Am reusti sa comuni cu serverul Firebase cloud messaging si utilizare unui design pattern in implementarea aplicatiei.

Referitor la aplicatie, functionalitatiile principale au fost implementate, incercand sa ofer utilizatorului o experienta de utlizare cat mai buna.

Dezvoltarile ulterioare:

* Crearea unei interfete grafice mai interactive si organizate
* Optimizarea codului
* Adaugarea de functionalitati noi precum ar fi: trimitere poze cand o masina este ridicata, recenzie asupra utilizatorului pentru a putea restrictiona utilizatorii care dau alarme false
* Oprirea notificarilor din aplicatie
* Setarea timpului de avertizare prin notificare
* Posibilitatea de selectie a locului unde se ridica masini
* Multiple posibilitati de inregistrare: facebook, google etc.
* Contul utilizatorului mai complex cu imagini, alarme trimise etc.

# Bibliografie

<https://developer.android.com/training/volley/requestqueue>

<https://firebase.google.com/docs/reference/android/com/google/firebase/messaging/FirebaseMessagingService>

<https://developer.android.com/guide/components/fundamentals>

<https://firebase.google.com/docs/auth/android/start>

<https://firebase.google.com/docs/firestore>

<https://firebase.google.com/docs/cloud-messaging>

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