

**UNIVERSITATEA „ALEXANDRU IOAN CUZA” IAȘI**  
**FACULTATEA DE INFORMATICĂ**



**LUCRARE DE LICENȚĂ**

# **SocialRoads**

Propusă de

**Popa Eduard**

Sesiunea: **Iulie, 2017**

Coordonator științific

**Asistent, dr. Vasile Alaiba**

**UNIVERSITATEA „ALEXANDRU IOAN CUZA” IAȘI**  
**FACULTATEA DE INFORMATICĂ**

**LUCRARE DE LICENȚĂ**

## **SocialRoads**

**Popa Eduard**

**Sesiunea: Iulie, 2017**

**Coordonator științific**

**Asistent, dr. Vasile Alaiba**

## DECLARAȚIE PRIVIND ORIGINALITATEA ȘI RESPECTAREA DREPTURILOR DE AUTOR

Prin prezenta declar că Lucrarea de licență cu titlul „SocialRoads” este scrisă de mine și nu a mai fost prezentată niciodată la o altă facultate sau instituție de învățământ superior din țară sau străinătate. De asemenea, declar că toate sursele utilizate, inclusiv cele preluate de pe Internet, sunt indicate în lucrare, cu respectarea regulilor de evitare a plagiatului:

- toate fragmentele de text reproduse exact, chiar și în traducere proprie din altă limbă, sunt scrise între ghilimele și dețin referința precisă a sursei;
- reformularea în cuvinte proprii a textelor scrise de către alți autori deține referința precisă;
- codul sursă, imagini etc. preluate din proiecte open source sau alte surse sunt utilizate cu respectarea drepturilor de autor și dețin referințe precise;
- rezumarea ideilor altor autori precizează referința precisă la textul original.

Iași,

Absolvent Popa Eduard

---

(semnătura în original)

## DECLARAȚIE DE CONSIMȚĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licență cu titlul „SocialRoads”, codul sursă al programelor și celelalte 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” 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,

Absolvent Popa Eduard

---

(semnătura în original)

## **Cuprins**

<b>1</b>	<b>Introducere.....</b>	<b>6</b>
1.1	Motivație.....	6
1.2	Context.....	6
1.3	Cerințe funcționale.....	8
1.4	Abordare tehnică.....	9
<b>2</b>	<b>Contribuții .....</b>	<b>13</b>
<b>3</b>	<b>Dezvoltarea aplicației.....</b>	<b>14</b>
3.1	Arhitectura soluției.....	14
3.2	Dezvoltarea aplicației server .....	14
3.2.1	Proiectare .....	14
3.2.1.1	Modelarea datelor.....	15
3.2.1.2	Protocolul de comunicare client-server .....	17
3.2.2	Implementare .....	22
3.2.2.1	Baza de date .....	22
3.2.2.2	Comunicare client-server .....	25
3.3	Dezvoltarea aplicației Client .....	28
3.3.1	Proiectare .....	28
3.3.1.1	Arhitectura aplicației .....	28
3.3.1.2	Structura proiectului .....	31
3.3.1.3	Interfața grafică .....	32
3.3.2	Implementare .....	34
3.3.2.1	Configurare aplicație Client .....	34
3.3.2.2	Configurare API-uri externe.....	35
3.3.2.2.1	Google Maps .....	35
3.3.2.2.2	Facebook .....	39
3.3.2.2.3	Firebase storage.....	40
3.3.2.3	Implementare funcționalități .....	41
3.3.2.3.1	Login .....	41
3.3.2.3.2	Afișarea poziției utilizatorului pe hartă .....	43
3.3.2.3.3	Adaugarea unui marcaj pe harta .....	44
3.3.2.3.4	Afisarea rutelor intre doua puncte .....	47
3.3.2.3.5	Actualizarea automata a hărții .....	48
3.3.2.3.6	Detectarea evenimentelor aflate pe o ruta .....	50
3.3.2.3.7	Trimiterea de mesaje intre utilizatori .....	52
<b>4</b>	<b>Manual de utilizare .....</b>	<b>55</b>
<b>5</b>	<b>Concluzii .....</b>	<b>65</b>
	<b>Bibliografie .....</b>	<b>66</b>

# 1 Introducere

## 1.1 Motivație

Majoritatea dintre noi ne-am confruntat cu diferite probleme în trafic, precum drumuri blocate sau foarte prost construite, ambuteiaje, amenzi de circulație pentru diferite motive sau chiar să ne rătăcim într-o zona necunoscută. Cu toții ne dorim să călătorim cu mașină într-un mod cât mai sigur și liniștit și să ajungem la timp la destinație fără să avem parte de niciun incident neplăcut pe parcursul deplasării.

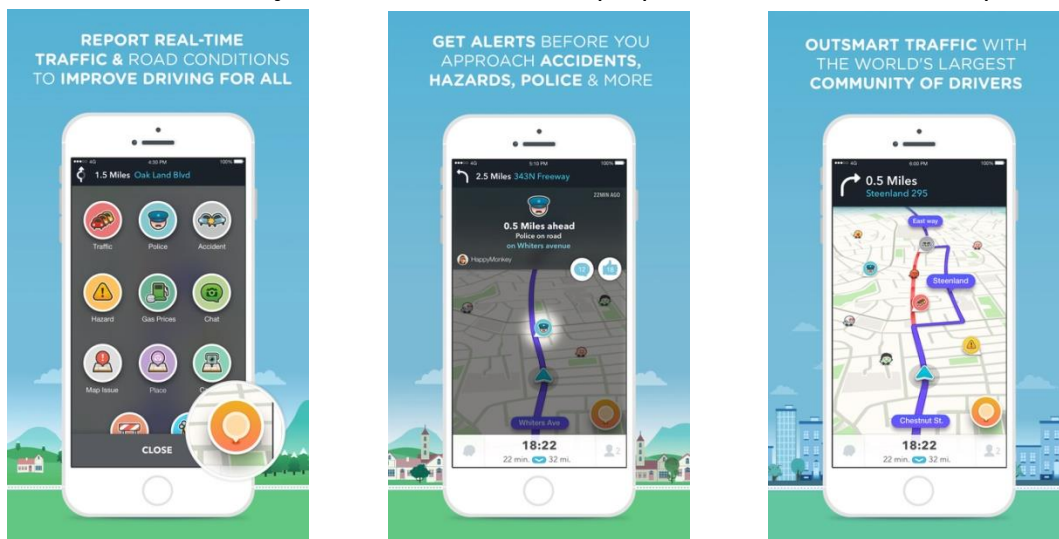
Pentru această lucrare am decis să realizez o aplicație mobile prin intermediul căreia putem scăpa de toate probleme menționate mai sus. Aplicația dorește să vină în sprijinul șoferilor, oferind informații real-time asupra drumului prin intermediul rapoartelor oferite de alți participanți la trafic cu privire la: starea carosabilului, existența unor drumuri blocate, controale ale poliției, raportarea unor accidente sau existența ambuteiajelor. Aplicația oferă utilizatorilor rute alternative pentru a ajunge la destinație evitând evenimentele rutiere nedorite.

Am ales "SocialRoads" ca nume pentru aplicație deoarece aceasta combină atât călătoriile cu autovehiculul cât și socializarea cu ceilalți participanți la trafic, permițând acestora să se conecteze între ei, creând astfel o comunitate care lucrează împreună pentru a îmbunătăți calitatea condusului de zi cu zi.

## 1.2 Context

În prezent sunt dezvoltate tot mai multe aplicații mobile care au ca scop îndrumarea conducătorilor auto spre destinație oferindu-le cât mai multe informații folositoare lor, dar din păcate majoritatea nu se bazează pe datele provenite de la participanții la trafic, ci doar pe informațiile GPS de care dispune aplicația. Există și cazuri în care dezvoltatorii au luat în calcul și acest factor cu privire la comunicarea în trafic. Printre aplicațiile care abordează un subiect asemănător amintim:

Waze<sup>1</sup> – este un program de navigație geografică bazat pe GPS, dezvoltat și popularizat de compania israeliană Waze Mobile. Waze diferă de aplicația tradițională de navigație GPS, fiind bazat pe comunitate și colectează date și informații de trafic de la utilizatorii săi. Oamenii pot raporta accidente, blocaje de trafic, filtre de poliție, radare etc. Waze identifică, de asemenea, cea mai ieftină stație de combustibil în apropierea utilizatorului sau pe traseul acestuia.

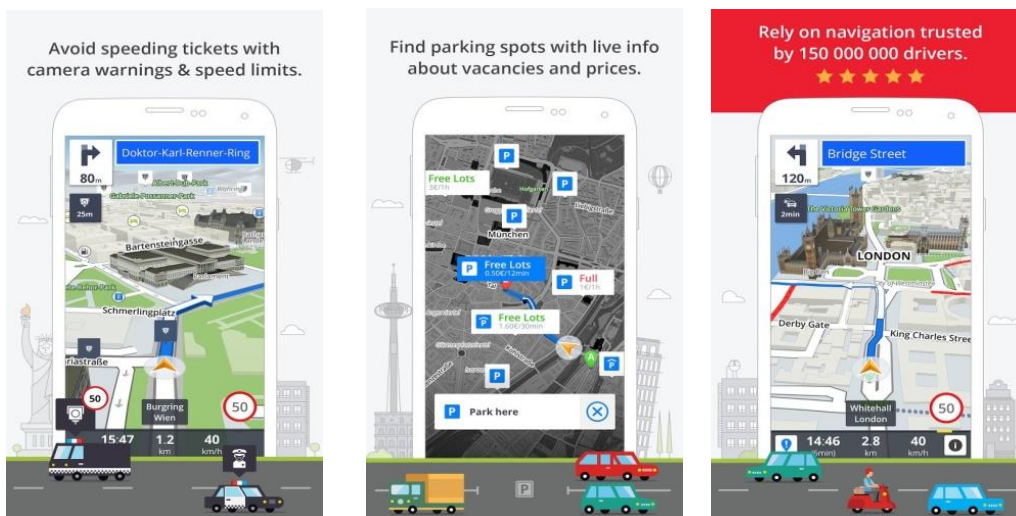


Sygic<sup>2</sup> – este cea mai avansată aplicație de navigare GPS cu hărți 3D offline de la TomTom permițând actualizări gratuite ale hărților. Transformă telefonul mobil într-un dispozitiv de navigație personal. Utilizatorii pot raporta locațiile camerelor de viteză sau a radarelor precum și diferite incidente ale altor utilizatori. Informațiile actualizate sunt livrate tuturor utilizatorilor în timp real. Aplicația mai conține milioane de puncte de interes care vă ajută să găsiți orice: de la restaurante, la aeroporturi și altele.

---

<sup>1</sup> Waze - <https://www.waze.com/>

<sup>2</sup> Sygic - <https://www.sygic.com/gps-navigation>



### 1.3 Cerințe funcționale

- Înregistrarea utilizatorului – Pentru a putea folosi aplicația, fiecare utilizator trebuie să-și facă un cont introducând un nume de utilizator și o parolă.
- Logare – Logarea în aplicație trebuie să se facă cu un nume de utilizator și o parolă valide.
- Accesul la informațiile de profil ale utilizatorului logat – Fiecare utilizator își poate vizualiza și modifica datele asociate profilului și poza de profil.
- Vizualizarea profilului altui utilizator – Profilul fiecărui user poate fi vizualizat de către toți utilizatorii aplicației doar dacă acesta este online.
- Atașarea contului de Facebook la aplicație – Utilizatorul poate să-și anexeze la profil un cont de Facebook prin intermediul căruia va putea face diferite acțiuni.
- Posibilitatea vizualizării contului de Facebook atașat unui utilizator al aplicației – Dacă un utilizator are anexat contul de Facebook atunci acesta este vizibil și poate fi accesat de către ceilalți utilizatori.
- Trimiterea de mesaje între utilizatorii aplicației – Utilizatorii aplicației pot comunica între ei prin intermediul mesajelor. Mesajele primite sau trimise vor fi stocate în pagină de profil a fiecărui utilizator.
- Trimiterea de mesaje către utilizatori apropiați – Fiecare utilizator poate trimite un mesaj către toți utilizatori aflați la o distanță mai mică de 10km de acesta.
- Alertarea utilizatorului atunci când acesta primește un mesaj nou – Atunci când utilizatorul primește un mesaj nou va fi notificat și i se va permite să vizualizeze mesajul. După citire, acesta poate să răspundă celui care a trimis mesajul printr-un alt mesaj.



- Detectarea poziției curente pe hartă – Aplicația are nevoie să cunoască poziția curentă a utilizatorului pentru a-i putea oferi toate informațiile necesare.
- Localizarea pe hartă a unui utilizator - Aplicația va permite să fixeze harta asupra poziției unui utilizator online.
- Afișarea pe hartă a poziției tuturor utilizatorilor online – Poziția fiecărui utilizator va fi marcată pe hartă folosind un marker personalizat.
- Adăugarea pe hartă a rapoartelor de trafic – Fiecare utilizator poate să adauge pe hartă unul dintre cele 5 tipuri de marcaje a rapoartelor rutiere : “Police”, “Accident” , “Road Block” , “Heavy Traffic”, “Bad Road”. De asemenea la fiecare raport rutier utilizatorii trebuie să adauge și o descriere.
- Posibilitatea de a distribui pe Facebook un raport de trafic – Dacă utilizatorul are anexat un cont de Facebook atunci acesta poate selecta un raport rutier de pe hartă și să îl distribuie pe rețeaua de socializare.
- Actualizarea informațiilor referitoare la rapoartele de drum– Harta și toate marcajele asociate acestora vor fi actualizate la fiecare 10 secunde pentru a oferi utilizatorilor informații real-time.
- Căutarea unei locații și poziționarea hărții asupra ei – Aplicația oferă utilizatorului posibilitatea de a naviga în diferite locuri doar introducând numele locației și selectând, dintr-o listă de posibile rezultate, destinația dorită.
- Calcularea și afișarea pe harta a rutei - După selectarea unei destinații, utilizatorul poate alege să i se afișeze o ruta către acea locație. Aplicația afișează cele mai bune 3 rute împreună cu toate informațiile asociate lor: evenimentele rutiere întâlnite, durata și distanța.
- Modificarea setărilor hărții – Utilizatorul poate alege să-și personalizeze harta, acesta având posibilitatea de a modifica tipul hărții și de a ascunde diferite tipuri de evenimente rutiere.

## 1.4 Abordare tehnică

Structura aplicației va fi compusă din client și server. Clientul va fi implementat folosind platforma de dezvoltare Android iar server-ul va fi scris în limbaj de programare JAVA folosind NetBeans ca IDE.

- **GoogleMap API** – Acest API ne permite să adăugăm în aplicația noastră hărți ce au la baza date de la Google. Oferă acces automat la serverele GoogleMaps, afișează harta, răspunde la gesturi ale utilizatorului asupra hărții precum mărirea,

micșorarea, mutarea sau rotirea acesteia. De asemenea acest API ne permite să adăugăm marcaje pe hartă, să trasăm rute între anumite locații facilitând astfel cerințele funcționale ale aplicației. În cadrul aplicației noastre, din pachetul de API oferite de GoogleMaps vom folosi:

- **Google Places API** <sup>3</sup>- Vom folosi acest API pentru a căuta anumite locații și de a le afișa pe hartă. API-ul va returna atât locații geografice cât și puncte de interes: spitale, restaurante, atracții turistice etc.
- **Google Static Maps API** <sup>4</sup>– Ne permite să facem o captură a hartii și să o stocăm ca o imagine . Acest lucru ne va fi de folos atunci când dorim să distribuim un eveniment din cadrul aplicației.
- **Google Map Directions API** <sup>5</sup>– Cu ajutorul acestui API vom putea trasa o rută între două locații. Putem accesa acest API folosind o interfață HTTP cu request-uri construite ca URL-uri care returnează un fișier în format JSON cu toate datele necesare.
- **Facebook API**
  - **Graph API** <sup>6</sup>- este modalitatea principală de a obține date din cadrul platformei Facebook. Este un API bazat pe HTTP pe care îl putem utiliza pentru a crea interogări asupra datelor unui profil, pentru a distribui noutăți sau pentru a încărca fotografii.
- **Adobe Photoshop** <sup>7</sup>– este un editor grafic cu ajutorul căruia vom crea și edita toate elementele grafice din cadrul aplicației cum ar fi: marcajele personalizate pentru fiecare eveniment, logo-ul aplicației, elemente de design din cadrul meniului etc.
- **Firebase Storage** <sup>8</sup> – Este un serviciu de stocare oferit de Google care permite încărcarea și descărcarea de fișiere pentru aplicațiile Firebase. O să utilizăm acest SDK în cadrul aplicației noastre pentru a stoca imaginea de profil a fiecărui utilizator.

---

<sup>3</sup> <https://developers.google.com/places/>

<sup>4</sup> <https://developers.google.com/maps/documentation/static-maps/>

<sup>5</sup> <https://developers.google.com/maps/documentation/directions/>

<sup>6</sup> <https://developers.facebook.com/docs/graph-api>

<sup>7</sup> <http://www.adobe.com/products/photoshop.html>

<sup>8</sup> <https://firebase.google.com/docs/storage/>

- **MySQL<sup>9</sup>** – este un sistem de gestiune a bazelor de date relaționale, produs de compania suedeză MySQL AB. Cu ajutorul lui putem să accesăm, adăugăm sau șterge datele dintr-o bază de date.
- **Android SDK** - Acest SDK ne oferă toate instrumentele necesare dezvoltării unei aplicații Android conținând un debugger, un emulator și o multitudine de librării care conțin documentații, cod exemplu și tutoriale. Dintre librăriile utilizate în aplicație amintim:
  - **android.support.v7.widget.RecyclerView<sup>10</sup>** – ne permite să afișăm un set mare de date într-o fereastră limitată folosind un view flexibil.
  - **android.content.SharedPreferences<sup>11</sup>** – este o interfață care ne permite să accesăm și să modificăm informații și date care persistă chiar și după ce aplicația a fost redeschisă.
  - **android.os.Bundle<sup>12</sup>** – este un set de perechi key/value și îl vom folosi să trimitem date între activități.
  - **android.view.View<sup>13</sup>** – această clasă reprezintă elementul de bază pentru componentele interfeței grafice și constituie clasa de bază pentru widget-uri, care sunt folosite pentru a crea componente interactive cum ar fi: butoane, câmpuri de text, dialog-uri etc.
  - **android.app.AlertDialog<sup>14</sup>** – Clasa AlertDialog ne permite să construim o varietate de modele de dialog. Un dialog este o fereastră care ocupă o mică parte din ecran în care solicită utilizatorul să ia o decizie sau să introducă informații suplimentare înainte de a putea continua.
  - **android.location.Geocoder<sup>15</sup>** – Este o clasă care ne permite să transformăm adresa unei locații într-o coordonată (latitudine, longitudine). De asemenea putem inversa acest proces transformând o coordonată într-o adresă parțială.

---

<sup>9</sup> <https://www.mysql.com/>

<sup>10</sup> <https://developer.android.com/reference/android/support/v7/widget/RecyclerView.html>

<sup>11</sup> <https://developer.android.com/reference/android/content/SharedPreferences.html>

<sup>12</sup> <https://developer.android.com/reference/android/os/Bundle.html>

<sup>13</sup> <https://developer.android.com/reference/android/view/View.html>

<sup>14</sup> <https://developer.android.com/guide/topics/ui/dialogs.html>

<sup>15</sup> <https://developer.android.com/reference/android/location/Geocoder.html>

- **android.animation** <sup>16</sup> - Clasele din această librărie permit animarea proprietăților obiectelor de orice tip . Noi vom folosi aceste clase pentru a anima diferite butoane și texte din cadrul aplicației
- **android.content.Intent** <sup>17</sup> – Clasa Intent ne permite să interacționăm cu alte componente din aplicație sau chiar cu componente ale altor aplicații . De exemplu , o activitate poate porni o nouă activitate externă care să facă o fotografie cu ajutorul camerei foto.
- **JAVA** - Java este un limbaj de programare orientat-obiect, puternic tipizat, conceput de către James Gosling la Sun Microsystems. Limbajul împrumută o mare parte din sintaxă de la C și C++, dar are un model al obiectelor mai simplu și prezintă mai puține facilități de nivel jos. Atât clientul cât și server-ul vor fi implementate folosind limbajul de programare JAVA.
  - Comunicarea dintre client și server se va realiza prin intermediul socket-urilor. Un **socket** este un endpoint pentru comunicarea între două mașini având atribuit o adresă IP și un port. Pachetul **java.net** <sup>18</sup> din cadrul platformei Java oferă clasa ServerSocket care implementează un socket pe care server-ul poate să îl folosească să comunice cu clienții.
  - Conexiunea cu baza de date se face cu ajutorul bibliotecii **java.sql** <sup>19</sup>. Aceasta ne oferă clasele java.sql.Connection, java.sql.PreparedStatement, java.sql.SQLException , java.sql.ResultSet care facilitează lucrul cu baza de date
  - Executarea simultană a mai multor fire de execuție se realizează folosind clasa **java.lang.Thread** <sup>20</sup>. Cu ajutorul Thread-urilor putem să soluționăm în mod concurent toate solicitările clienților .

---

<sup>16</sup> <https://developer.android.com/training/animation/index.html>

<sup>17</sup> <https://developer.android.com/reference/android/content/Intent.html>

<sup>18</sup> <https://docs.oracle.com/javase/7/docs/api/java/net/package-summary.html>

<sup>19</sup> <https://docs.oracle.com/javase/7/docs/api/java/sql/package-summary.html>

<sup>20</sup> <https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html>

## 2 Contribuții

Aplicația SocialRoads are ca scop îmbinarea hărții rudimentare GPS cu elemente de socializare între participanții la trafic. Pentru a atinge acest obiectiv este necesar ca toți clienții să fie conectați între ei fapt pentru care proiectul va fi împărțit în două componente, client și server. De asemenea, structura lucrării va urma aceeași manieră, având două părți importante : dezvoltarea aplicației client și dezvoltarea aplicației server. În fiecare dintre cele două voi prezenta modul în care am proiectat și dezvoltat produsul final oferind toate detaliile de implementare.

În cadrul dezvoltării aplicației server principale contribuții sunt :

- Implementarea unui server TCP concurent care să permită conectarea simultană a mai multor utilizatori, procesarea cererilor realizându-se în mod asincron.
- Stabilirea și implementarea unui protocol de comunicare cu aplicația client prin intermediul căruia are loc schimbul de date.
- Conectarea unei baze de date și crearea tabelelor necesare funcționării aplicației.

În cadrul dezvoltării aplicației client am avut următoarele contribuții:

- Integrarea API-urilor externe în cadrul aplicației pentru a implementa toate cerințele funcționale.
- Crearea unor elemente de design pentru a personaliza diferite componente grafice ale aplicației.
- Dezvoltarea structurii paginilor aplicației și implementarea funcționalităților aferente acestora. Am urmărit, prin modul în care a fost construit design-ul paginilor, să fac interacțiunea utilizatorului cu aplicația cât mai simplă și intuitivă.
- Implementarea unui algoritm de detectare a evenimentelor din trafic în funcție de ruta aleasă de către utilizator.

## 3 Dezvoltarea aplicației

### 3.1 Arhitectura soluției

Pentru dezvoltarea aplicației am folosit modelul client-server. Server-ul va fi conectat la o bază de date din care va prelua informații și le va trimite către clienți. Clientul la rândul lui va fi conectat cu server-ul Google prin intermediul API-ului de la GoogleMaps, cu server-ul Facebook și cu server-ul Firebase Storage, fapt pentru care dispozitivul android va necesita în permanență o conexiune la internet.

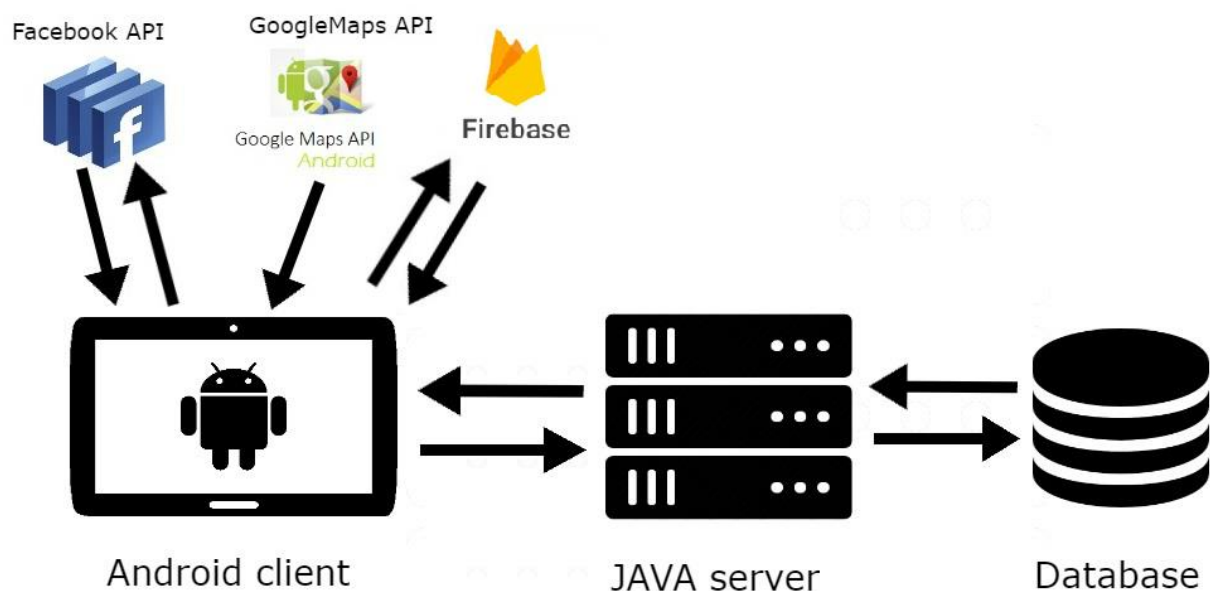


Diagrama 1 Arhitectura solutiei

### 3.2 Dezvoltarea aplicației server

#### 3.2.1 Proiectare

**Server-ul** are rolul de a asculta orice cerere de conectare și de a rezolva orice request din partea clienților. Acesta validează cererea primită de la client și în funcție de tipul ei, server-ul poate să trimită înapoi alte date sau doar să modifice diferite înregistrări din baza de date.

**Comunicarea între server și client** va fi gestionată de ServerSocket din cadrul bibliotecii java.net. Rezolvarea request-urilor primite de către server se va face în mod concurrent. Astfel, după ce se va accepta solicitarea de conexiune a unui client, se va crea un nou fir de execuție care să îndeplinească request-ul primit. În acest mod permitem conectarea mai multor clienți la server în același timp eliminând astfel timpul de așteptare în cazul în care există mai multe solicitări asupra server-ului. Crearea unui nou fir de execuție se va realiza cu ajutorul **Thread**-urilor.

### 3.2.1.1 Modelarea datelor

Server-ul va fi conectat la o bază de date în care vor fi stocate toate datele necesare aplicației. Baza de date va conține 5 tabele: USERS, USERS\_MARKER, UȘER\_PROFILE\_INFO, EVENT\_MARKER, MESSAGES . Mai jos este prezentată structura acestora:

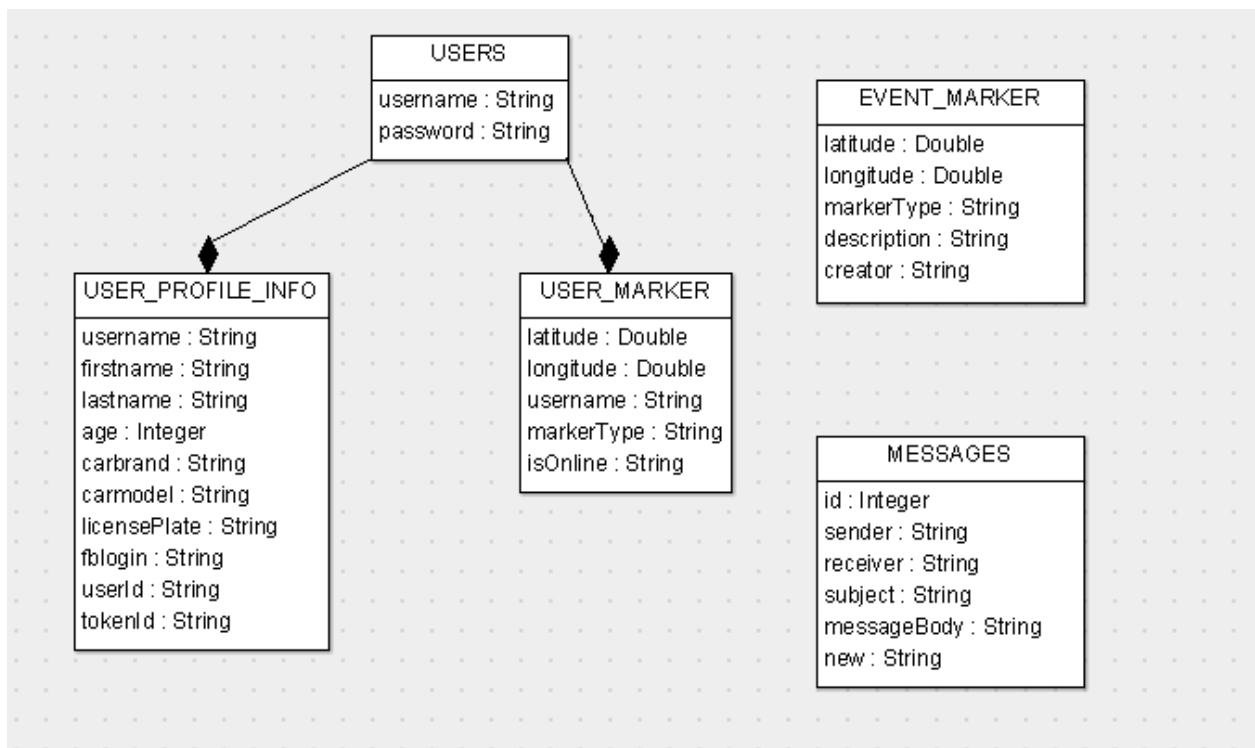


Diagrama 2 Modelarea datelor

- Tabela **USERS** conține informațiile de logare ale utilizatorilor. Câmpul username este de tip primary-key deoarece nu putem avea doi utilizatori cu același nume. Fiecărei înregistrări din cadrul acestei tabele îi va corespunde câte o înregistrare în tabelele **UȘER\_PROFILE\_INFO** și **UȘER\_MARKER**;
- Tabela **UȘER\_PROFILE\_INFO** conține toate informațiile unui utilizator pe care acesta le setează din pagina de profil. De asemenea, utilizatorul are posibilitatea de a se loga cu contul de Facebook, astfel câmpurile **userId** și **tokenId** vor fi inițializate iar **fblogin** va avea valoarea “yes”;
- Tabela **UȘER\_MARKER** conține date despre poziția utilizatorului pe hartă. Câmpul **isOnline** poate să fie “yes” sau “no” în funcție de starea utilizatorului: online sau offline. În cazul în care utilizatorul nu este online, marker-ul de pe hartă asignat acestuia nu va mai fi vizibil;
- Tabela **EVENT\_MARKER** conține informațiile despre toate evenimentele rutiere semnalate de către ceilalți utilizatori. **MarkerType**-ul poate lua 5 valori în funcție de tipul de eveniment rutier pe care îl semnalează: Police, Accident, Heavy traffic, Road block, Bad road;
- În tabela **MESSAGES** sunt stocate toate mesajele care au fost trimise sau primite de către utilizatori. Ele sunt identificate unic printr-un **id** care se autoincrementtează. Tabela conține atât username-ul expeditorului, cât și pe cel al destinatarului, fapt pentru care server-ul știe cui trebuie să trimită mesajul.



### 3.2.1.2 Protocolul de comunicare client-server

Pentru a putea face schimbul de date cu clienții am proiectat un **protocol de comunicare client-server**. Pentru fiecare cerere în parte am construit o diagramă de secvență pentru a indica modul de funcționare a protocolului.

#### 1. “get\_user\_login\_approval”

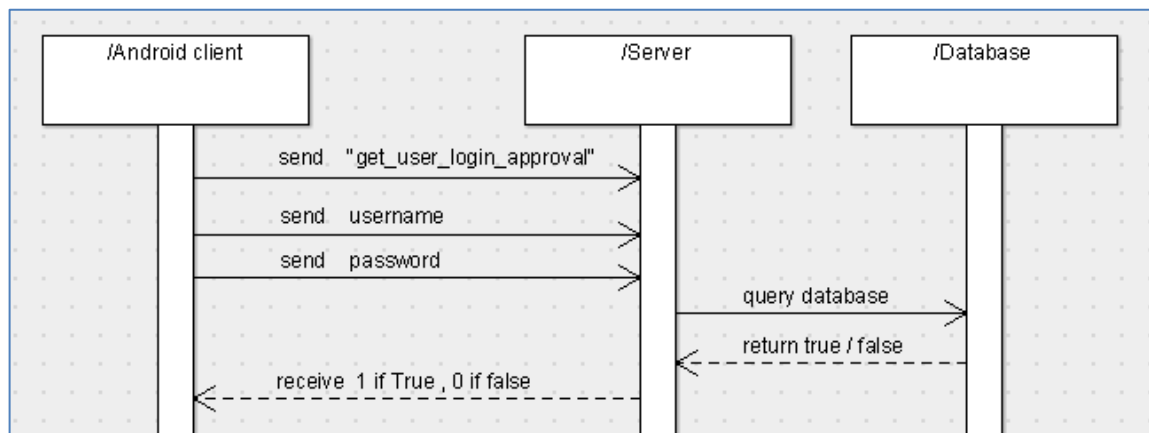


Diagrama 3 : get\_user\_login\_approval

Pentru a primi confirmarea de logare în aplicație clientul va trimite către server “get\_ușer\_login\_approval “ după care username-ul și password-ul și va primi 1 dacă user-ul există în baza de date sau 0 dacă nu există.

## 2. "get\_user\_profile\_information"

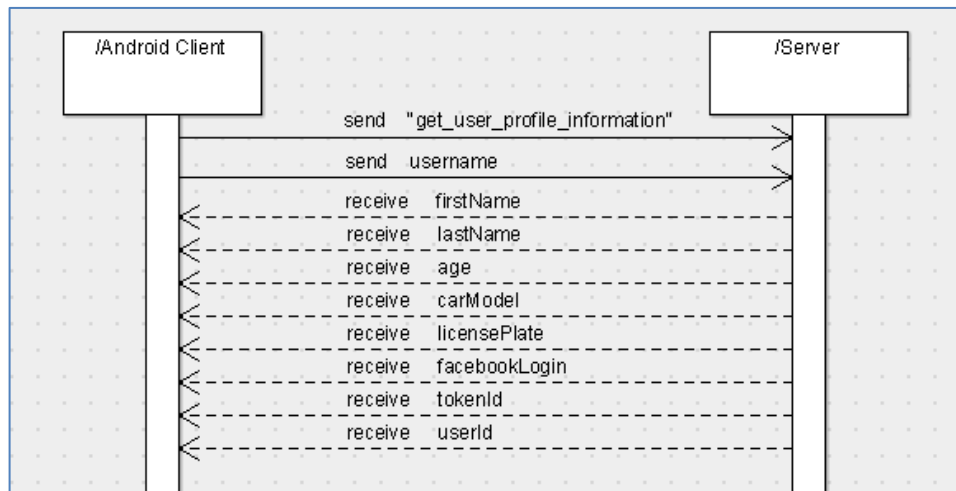


Diagrama 4 : get\_user\_profile\_information

Pentru a obține de la server toate datele despre un utilizator, clientul va trimite cererea "get\_user\_profile\_information" urmat de username-ul utilizatorului și va primi de la server toate informațiile din baza de date anexate acestuia.

## 3. "update\_user\_profile\_information"

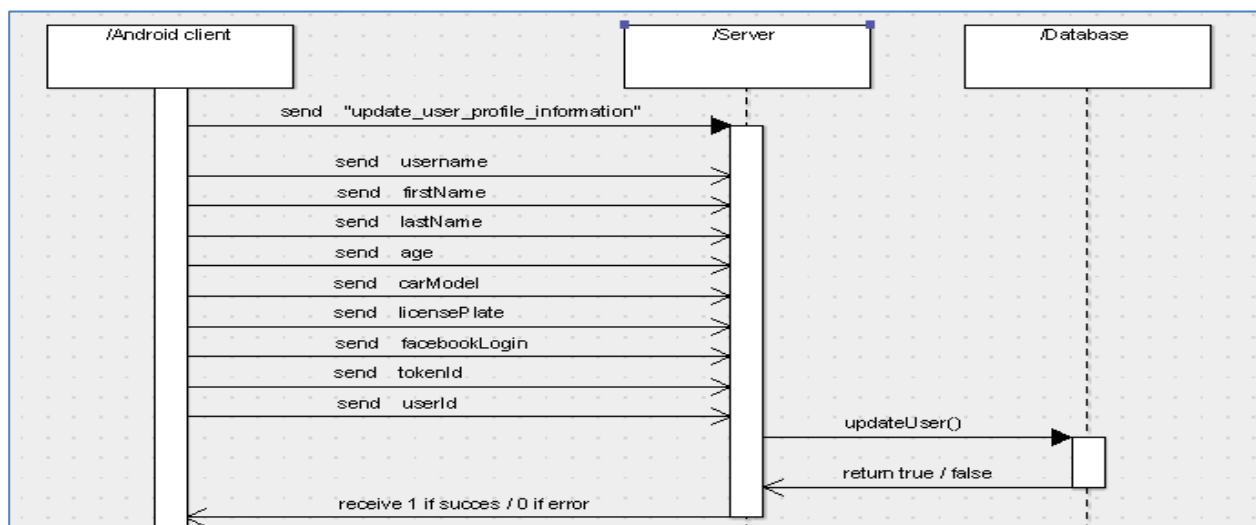


Diagrama 5 : update\_user\_profile\_information

Dacă client-ul dorește să adauge sau să modifice informațiile de profil a unui utilizator, va trimite către server cererea “update\_ușer\_profile\_information” urmată de toate informațiile profilului iar server-ul le va prelua și le va introduce introduce în baza de date.

#### 4. “register\_user”

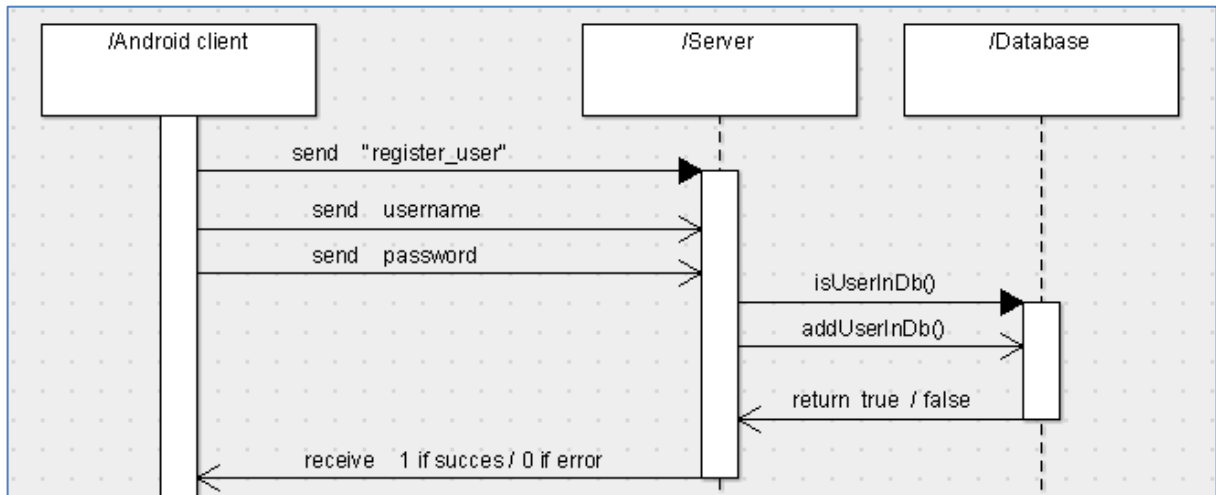


Diagrama 6 : register\_user

Pentru a înregistra un utilizator client-ul va trimite cererea “register\_ușer” urmată de username-ul și password-ul pe care dorește să le introducă în sistem. Server-ul va verifica dacă username-ul există deja în sistem și dacă nu există, îl va adăuga în baza de date.

#### 5. “add \_marker”

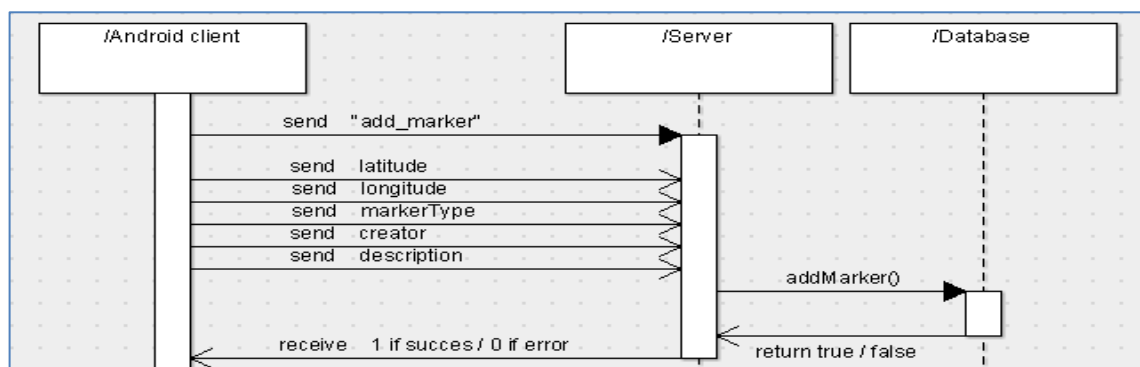


Diagrama 7 : add \_marker

Pentru a adăuga un marker pe hartă, client-ul trimite mai întâi cererea “add\_marker” după care trimite informațiile marker-ului. După ce server-ul primește toate informațiile necesare adăugă marker-ul în baza de date și trimite către client 1 sau 0 dacă operațiunea s-a încheiat cu succes sau a intervenit o eroare.

#### 6. “get \_markers”

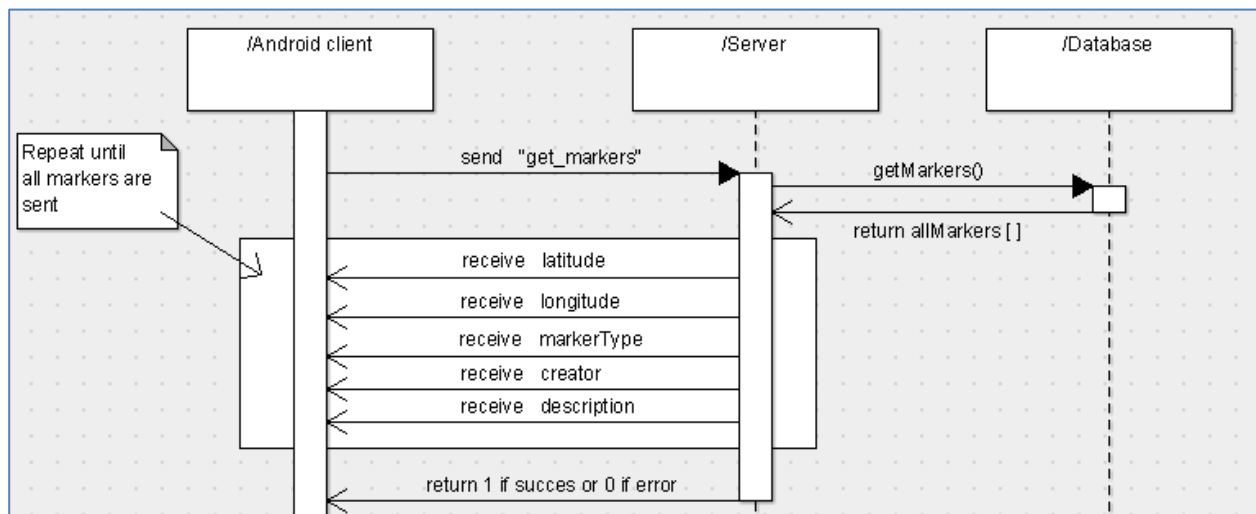


Diagrama 8 : get \_markers

Pentru a obține date despre marcajele ce trebuie afișate pe hartă, client-ul va trimite cererea “get \_markers” către server iar acesta va returna toate informațiile existente în baza de date.

## 7. "add\_message"

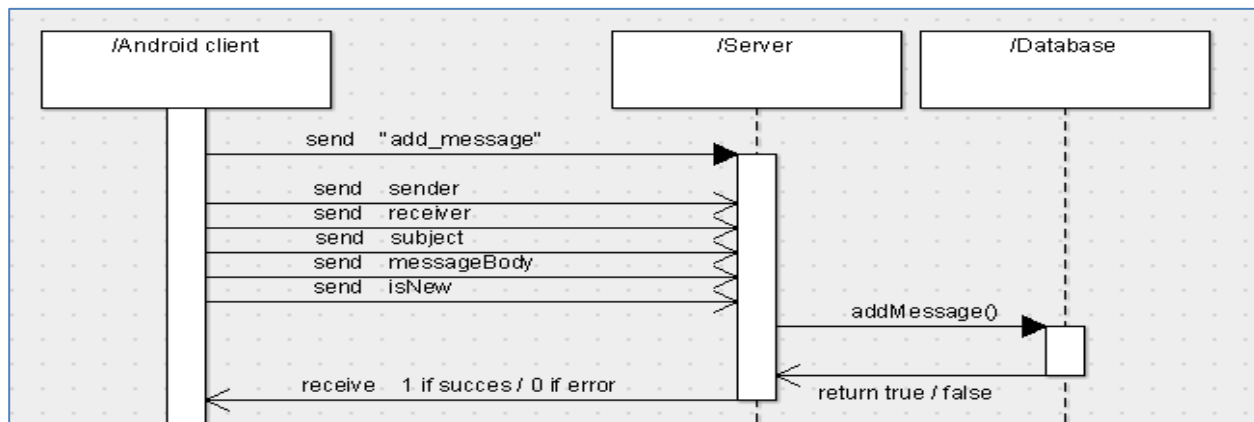


Diagrama 9 : add\_message

## 8. "get\_messages"

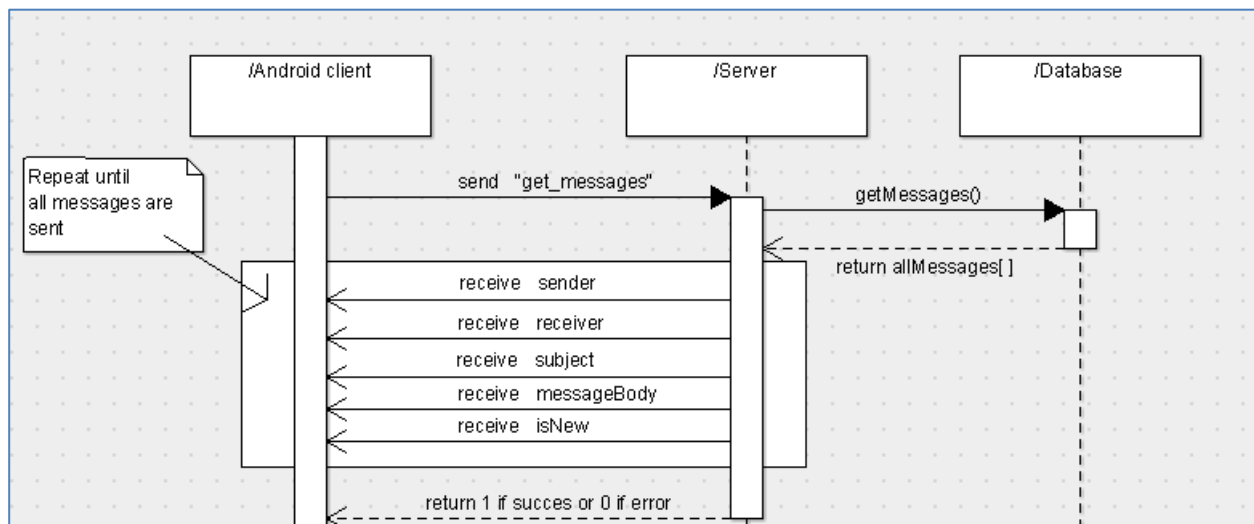


Diagrama 10 : get\_messages

## 3.2.2 Implementare

### 3.2.2.1 Baza de date

Conexiunea Server-ului la baza de date se face prin intermediul API-ului JDBC (Java Database Connectivity) care face posibilă efectuarea a trei lucruri :

#### 1. Stabilește o conexiune cu o gama largă de baze de date

Pentru realizarea conexiunii am folosit clasa DriverManager care prin intermediul metodei .getConnection ne returnează un obiect Connection prin intermediul căruia putem accesa baza de date specificată ca parametru.

```
Class.forName("org.sqlite.JDBC");  
c = DriverManager.getConnection("jdbc:sqlite:SocialRoads.db");
```

Secțiune cod 1 : Conexiune baza de date

#### 2. Trimite instrucțiuni SQL

Formularea instrucțiunilor SQL se face cu ajutorul claselor :

- **Statement** – Este utilizată pentru a executa interogări normale SQL și nu permite folosirea de parametri în cadrul interogării. Crearea unui Statement se realizează prin intermediul metodei .createStatement() din clasa Connection.

```
statement = connection.createStatement();  
String sql = "CREATE TABLE IF NOT EXISTS USERS (" +  
            " USERNAME          STRING      NOT NULL, " +  
            " PASSWORD          STRING      NOT NULL); ";  
statement.executeUpdate(sql);  
statement.close();
```

Secțiune cod 2 : Exemplu Statement

- **PreparedStatement** – Este utilizată pentru a executa interogări dinamice sau parametrizate. Crearea unui obiect PreparedStatement se realizează cu ajutorul metodei `.prepareStatement(sql)` din cadrul clasei Connection, unde **sql** reprezintă interogarea dinamică pe care dorim să o asociem.

```
String sql= "SELECT * FROM USERS WHERE USERNAME= ? AND PASSWORD= ?";
PreparedStatement statement=connection.prepareStatement(sql);
statement.setString(1, username);
statement.setString(2, password);
```

Sectiune cod 3 : Exemplu PreparedStatement

### 3. Proceseaza rezultatele obținute în urma instrucțiunilor SQL

Comenzile de interogare returnează un set de rezultate. Acesta este procesat folosind clasa ResultSet care ne permite să parcurgem setul de rezultate obținut. Coloanele dintr-un rând sunt preluate fie după nume, fie după numărul coloanei. Parcurgerea ResultSet-ului se realizează folosind metoda `.next()`.

```
Statement statement = c.createStatement();
ResultSet rs = statement.executeQuery( "SELECT * FROM MARKERS;" );
while(rs.next())
{
    Marker marker=new Marker();
    marker.latitude=rs.getDouble("LATITUDE");
    marker.longitude=rs.getDouble("LONGITUDE");
    marker.markerType=rs.getString("MARKERTYPE");
    marker.username=rs.getString("CREATOR");
    marker.description=rs.getString("DESCRIPTION");
    markerList.add(marker);
}
```

Sectiune cod 4 : Exemplu ResultSet

## Crearea tabelelor :

```
statement=c.createStatement();
String sql_2 = "CREATE TABLE IF NOT EXISTS MARKERS (" +
    " LATITUDE          DOUBLE    NOT NULL, " +
    " LONGITUDE          DOUBLE    NOT NULL, " +
    " MARKERTYPE          STRING    NOT NULL, " +
    " CREATOR             STRING    NOT NULL, " +
    " DESCRIPTION          STRING    NOT NULL);";
statement.executeUpdate(sql_2);
statement.close();

statement=c.createStatement();
String sql_3 = "CREATE TABLE IF NOT EXISTS USER_MARKER(" +
    " LATITUDE          DOUBLE    NOT NULL, " +
    " LONGITUDE          DOUBLE    NOT NULL, " +
    " MARKERTYPE          STRING    NOT NULL, " +
    " USERNAME           STRING    PRIMARY KEY, " +
    " DESCRIPTION          STRING    NOT NULL);";
statement.executeUpdate(sql_3);

statement=c.createStatement();
String sql_4 = "CREATE TABLE IF NOT EXISTS USER_PROFILE_INFO(" +
    " USERNAME           STRING    PRIMARY KEY, " +
    " FIRSTNAME           STRING    NULL, " +
    " LASTNAME            STRING    NULL, " +
    " AGE                 STRING    NULL, " +
    " CARBRAND            STRING    NULL, " +
    " CARMODEL            STRING    NULL, " +
    " LICENSEPLATE        STRING    NULL, " +
    " FBLOGIN             STRING    NULL, " +
    " USERID              STRING    NULL, " +
    " TOKENID             STRING    NULL); ";
statement.executeUpdate(sql_4);

statement=c.createStatement();
String sql_5 = "CREATE TABLE IF NOT EXISTS MESSAGES(" +
    " ID                 INTEGER    PRIMARY KEY    AUTOINCREMENT, " +
    " SENDER              STRING    NOT NULL, " +
    " RECEIVER            STRING    NOT NULL, " +
    " SUBJECT             STRING    NOT NULL, " +
    " MESSAGEBODY         STRING    NOT NULL, " +
    " NEW                 STRING    NOT NULL);";
statement.executeUpdate(sql_5);
```

### Sețiune cod 5 : Creare table



### 3.2.2.2 Comunicare client-server

Pentru comunicarea server-ului cu clientul am folosit din librăria java.net clasa Socket și clasa SocketServer. Un socket este un punct final al unei legături de comunicare bidirecțională între două programe care rulează pe rețea, în cazul nostru client-ul și server-ul. Un socket are atribuit o adresă IP și un număr de port, astfel încât putem identifica aplicația la care datele sunt trimise.

Pentru implementarea comunicării la nivelul server-ului am folosit clasa ServerSocket care implementează un socket care ascultă și acceptă conexiunile cu clienții.

```
public class SimpleServer {
    // Definim portul la care se ruleaza serverul
    public static final int PORT = 8080;

    public SimpleServer() throws IOException, ClassNotFoundException, SQLException
    {
        ServerSocket serverSocket = null ;
        try
        {
            serverSocket= new ServerSocket(PORT);

            while ( true )
            {
                System.out.println (" Asteptam un client ...");
                Socket socket = serverSocket.accept();
                // Executam solicitarea clientului intr -un fir de executie
                new ClientThread(socket).start();
            }
        }
        catch ( IOException e)
        {
            System.err. println (" Eroare IO \n" + e);
        }
        finally
        {
            serverSocket.close();
        }
    }
}
```

Sectiune cod 6 : Instantiere ServerSocket

Client-ul trimite cererea de conexiune către server, iar dacă totul este în regulă, aceasta este acceptată și se creează un nou socket prin intermediul căruia server-ul poate comunica cu clientul.

După ce se realizează conexiunea, se pornește un nou fir de execuție care să rezolve solicitarea venită de la client iar server-ul va continua să asculte alte cereri de conectare de la alți clienți. Firul de execuție se creează cu ajutorul clasei Thread din cadrul librăriei java.lang.

```
public class ClientThread extends Thread{
    private Socket socket = null ;
    public ClientThread ( Socket socket ) throws ClassNotFoundException, SQLException
    {
        this.socket = socket ;
    }
    public void run ()
    {
        try
        {
            DataOutputStream output = new DataOutputStream(socket.getOutputStream());
            DataInputStream input = new DataInputStream(socket.getInputStream());

            String request = input.readUTF(); //Primim cererea de la client

            if(request.compareTo("get_user_login_approval")==0){
                /* rezolvare cerere */ }
            if(request.compareTo("get_user_profile_information")==0){
                /* rezolvare cerere */ }
            if(request.compareTo("update_user_profile_information")==0){
                /* rezolvare cerere */ }
            if(request.compareTo("register_user")==0){
                /* rezolvare cerere */ }
            if(request.compareTo("add_marker")==0){
                /* rezolvare cerere */ }
            if(request.compareTo("get_markers")==0){
                /* rezolvare cerere */ }
            if(request.compareTo("get_user_marker")==0){
                /* rezolvare cerere */ }
            if(request.compareTo("add_message")==0){
                /* rezolvare cerere */ }
            if(request.compareTo("get_messages")==0){
                /* rezolvare cerere */ }

            output.close();
            input.close();
        }
    }
}
```

#### Secțiune cod 7 ClientThread

Pentru citirea și scrierea în rețea prin intermediul socket-ului vom folosi clasele DataInputStream respectiv DataOutputStream. După ce este citită cererea, server-ul o identifică și folosind protocolul de comunicare explicat în Capitolul 3.2.1.2 primește datele de la client și returnează răspunsul dorit.

În Secțiune cod 8 și Secțiune cod 9 am prezentat un exemplu de implementare a rezolvării unei cereri atât din postura Server-ului cât și a Client-ului.

```
if(request.compareTo("add_user")==0){
    String username=input.readUTF();//Primim username
    String password=input.readUTF();//Primim password

    boolean raspuns=false;
    if(!userDatabase.isUserInDatabase(username))
    {
        raspuns=userDatabase.addUser(username, password);
    }

    if(raspuns == true){
        output.writeUTF ( "1" );// Trimitem raspuns la client
    }
    else{
        output.writeUTF ( "0" ); // Trimitem raspuns la client
    }
}
```

Secțiune cod 8 : Exemplu rezolvare cerere Server

```
Socket socket = new Socket(adresaServer, PORT);
DataOutputStream output=new DataOutputStream(socket.getOutputStream());
DataInputStream input = new DataInputStream(socket.getInputStream());

output.writeUTF("add_user");//Trimitem cererea la server

output.writeUTF(s.username);//Trimitem username
output.writeUTF(s.password);//Trimitem password

String raspunsServer = input.readUTF(); //Primesc raspuns server
if (raspunsServer.compareTo("1") == 0)
|   s.raspuns = true;
else
|   s.raspuns = false;

socket.close();
```

Secțiune cod 9 : Exemplu trimitere cerere Client

## 3.3 Dezvoltarea aplicatiei Client

Pentru dezvoltarea aplicației Client am folosit API-ul Android. Acest API este scris în limbaj JAVA și este oferit de compania Google în regim deschis și gratuit. Există mai multe versiuni ale acestui API, fiecare corespunzând versiunii sistemului de operare Android. Pentru a avea acces la cele mai noi funcționalități, dar și pentru a acoperi un număr cât mai mare de dispozitive am ales că nivelul minim al API-ului Android pe care aplicația să îl suporte să fie 21, corespunzător versiunii de Android 5.0 Lollipop.

```
defaultConfig {  
    applicationId "com.example.edward.socialroads"  
    minSdkVersion 21  
    targetSdkVersion 25  
    versionCode 1  
    versionName "1.0"  
    testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"  
    multiDexEnabled true  
}
```

Sectiune cod 10 Configurari Client

### 3.3.1 Proiectare

#### 3.3.1.1 Arhitectura aplicatiei

Instrumentele oferite de Android, cum ar fi Layout-uri, Activități, Fragmente și structuri de date, par să ne orienteze în direcția modelul arhitectural Model-View-Controller (MVC). Acest șablon care își propune să izoleze responsabilitățile, are rolul de a modulariza aplicația, delimitând în mod clar părțile componente pentru a fi ușor de modificat iar după modificare, acestea să fie compatibile cu celelalte module ce formează aplicația.

Schema de funcționare a aplicației după arhitectura MVC decurge în felul următor:

- Utilizatorul interacționează cu interfața reprezentată de către **View**;
- **View**-ul notifică **Controller**-ul cu privire la acțiunea utilizatorului;
- **Controller**-ul interacționează și modifică **Model**-ul în funcție de acțiunea utilizatorului;

- **Controller**-ul actualizează **View**-ul pe baza **Model**-ului;
- Interfața așteaptă acțiuni suplimentare din partea utilizatorului, ciclul reluându-se;

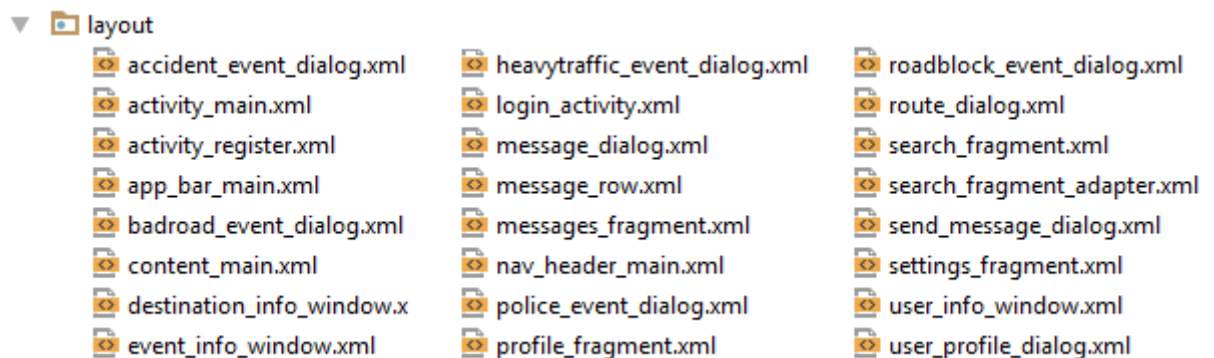


Diagrama 11 MVC

Cele trei componente ale MVC sunt Model, View și Controller.

**Model**-ul se ocupă de comportarea aplicației și datele acesteia. Nu este legat de View sau de Controller, fapt pentru care este reutilizabil în multe contexte.

**View**-ul este reprezentarea modelului având responsabilitatea de a crea interfață cu utilizatorul (User Interface) și de a transmite informații către Controller atunci când utilizatorul interacționează cu aplicația. View-ul este reprezentat prin intermediul layout-urilor. Un layout definește o structură vizuală pentru interfață cu utilizatorul, fiind declarat cu ajutorul fișierelor XML.



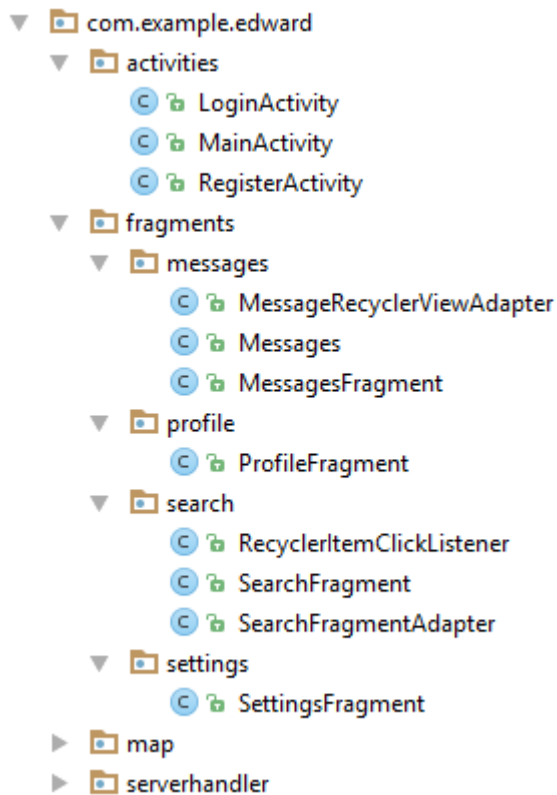
```

graph TD
    A([Activity launched]) --> B[onCreate()]
    B --> C[onStart()]
    C --> D[onResume()]
    D --> E([Activity running])
    E -- "Another activity comes into the foreground" --> F[onPause()]
    F -- "User returns to the activity" --> G[onRestart()]
    G --> C
    F -- "The activity is no longer visible" --> H[onStop()]
    H -- "User navigates to the activity" --> G
    H -- "Apps with higher priority need memory" --> I([App process killed])
    I -- "User navigates to the activity" --> B
    H -- "The activity is finishing or being destroyed by the system" --> J[onDestroy()]
    J --> K([Activity shut down])
  
```

### Diagrama 12 Ciclul de viata al activitatilor

O activitatea nouă poate fi creată de către o altă activitate. După ce activitatea este creată, cealaltă o să fie adăugată într-o stivă de activități, numită *back-stack* (o să fie readusă în *foreground* după ce activitatea nouă este distrusă). Prima metodă ce se apelează când aceasta este creată este metoda *onCreate()*. Aceasta este urmată de metodele *onStart()* și *onResume()*. Când activitatea este trimisă în *background*, se apelează pe rând metodele *onPause()* și *onStop()*, iar dacă urmează să fie distrusă se apelează *onDestroy()*. Dacă o activitate este în *background*, când va reveni în *foreground* se va apela metoda *onResume()*. Toate aceste metode din ciclul de viață al unei activități sunt prezentate mai sus în **Diagrama 12**.

### 3.3.1.2 Structura proiectului



Structura proiectului este format din 4 pachete principale și anume:

- **activities** – Acest pachet contine cele 3 activitati ale aplicației:
  - `LoginActivity` – activitatea care permite utilizatorului să acceseze aplicația prin intermediul unui nume de utilizator și a unei parole.
  - `RegisterActivity` – activitatea care permite unui utilizator să se înregistreze în baza de date a aplicației.
  - `MainActivity` – reprezintă activitatea principală care îi oferă utilizatorului acces la toate funcționalitățile aplicației, conținând un panou prin intermediul căruia se poate naviga între fragmente.
- **fragments** - Acest pachet este format din 4 sub-pachete care reprezintă fragmentele din cadrul activității principale:
  - `messages` – reprezintă fragmentul care afișează mesajele unui utilizator, permițând diferite acțiuni asupra lor;
  - `profile` – reprezintă fragmentul care afișează toate datele utilizatorului

- search – este fragmentul care identifică poziția pe hartă a unei locații introdu-se de utilizator;
- settings – este fragmentul care permite utilizatorului să modifice setările hărții;
- **map** – Acest pachet conține toate clasele care au ca scop instanțierea, setarea și modificarea hărții.
- **serverhandler** – Acest pachet are rolul de a manipula toate cererile trimise la Server și răspunsurile primite de la acesta.

### 3.3.1.3 Interfața grafică

Utilizatorul are la dispoziție o interfață grafică simplă și intuitivă, care utilizează pe cât posibil denumiri și pictograme sugestive pentru a face navigarea în cadrul aplicației cât mai simplă și cursivă.

Aplicația va fi împărțită în trei ecrane principale, fiecare ecran fiind controlat de o Activitate (Activity).



FIGURA 1

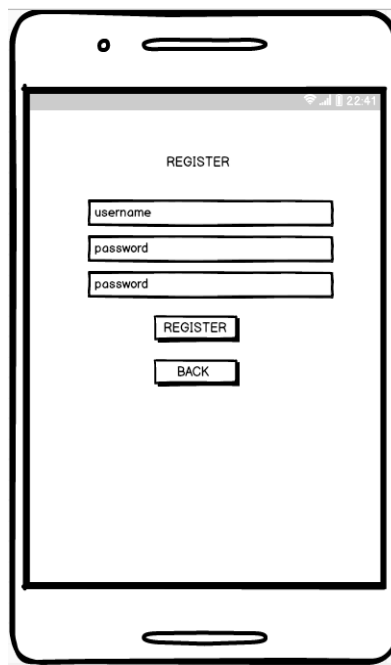


FIGURA 2

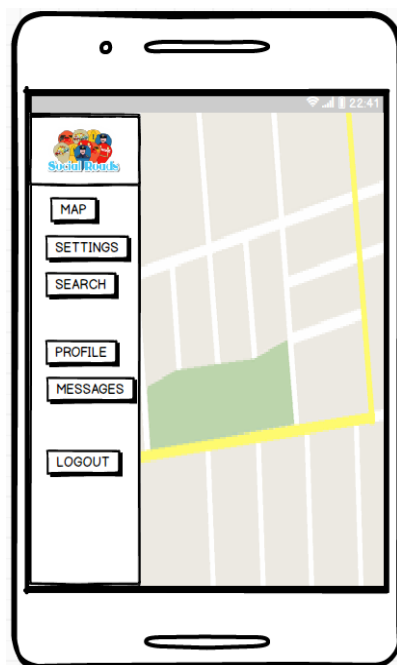


FIGURA 3



- **Pagina de login – Figura 1**

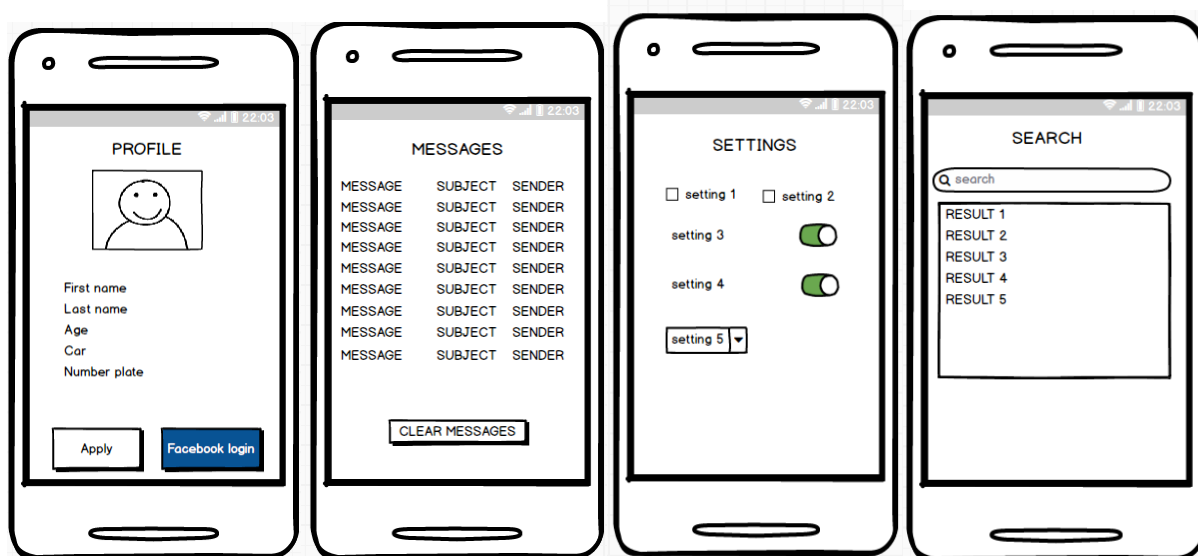
Va afișa două câmpuri de text în care utilizatorul va introduce numele de utilizator și parola și va fi înaintat către pagina principală prin apăsarea butonul **Login**. Prin apăsarea butonului **Register**, utilizatorul va fi redirecționat către pagina de înregistrare.

- **Pagina de înregistrare – Figura 2**

Este accesibilă doar din cadrul paginii de login și conține trei câmpuri de text în care utilizatorul trebuie să introducă datele cu care dorește să se înregistreze pe viitor în aplicație. Prin apăsarea butonului **Register** datele vor fi preluate și trimise la server, iar prin apăsarea butonului **Back**, utilizatorul va fi întors la pagina de login.






- **Pagina principala – Figura 3**

Pagina principală este accesibilă doar după ce utilizatorul s-a autentificat în cadrul paginii de login cu un nume de utilizator și o parolă corecte. În cadrul acestei pagini, utilizatorul va putea selecta unul dintre cele 4 fragmente (**Settings**, **Search**, **Profile**, **Messages**) prin intermediul meniului situat în partea stânga care va fi ascuns în cea mai mare parte a timpului. Prin apăsarea butonului **Logout** utilizatorul va fi redirecționat către pagina de login și va fi nevoit să se autentifice din nou pentru a folosi aplicația. Meniul va fi dezvăluit de fiecare dată când utilizatorul va glisă degetul în partea stângă a ecranului.



Pentru a păstra consumul de resurse al aplicației Client cât mai scăzut, în activitatea principală (Figura 3) am recurs la utilizarea obiectelor de tip Fragment, care fiind reutilizabile în cadrul aplicației permit crearea unei interfețe grafice dinamice.

Pentru afișarea evenimentelor rutiere pe hartă am creat 5 marcaje personalizate cât mai sugestive:

- Roadblock ( Drum blocat ) - 
- Police ( Poliție ) - 
- Heavy traffic ( Trafic aglomerat ) - 
- Accident ( Accident ) - 
- Bad road ( Drum rău ) - 

## 3.3.2 Implementare

### 3.3.2.1 Configurare aplicație Client

Aplicația Client dezvoltată în Android are nevoie de permisiuni pentru accesarea unor componente hardware și software. Permisiunile trebuie menționate în fișierul AndroidManifest.xml. Acest fișier conține informații esențiale, pe care sistemul trebuie să le aibă înainte de a putea executa codul aplicației.

```

package="com.example.edward.socialroads">

<uses-permission android:name="android.permission.INTERNET" />
<uses-permission
    android:name="android.permission.READ_EXTERNAL_STORAGE"
    android:maxSdkVersion="21" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>

```

Secțiune cod 11 : Permisuni Client

- android.permission.INTERNET – permite aplicației să aibă acces la internet, acest lucru fiind necesar pentru a comunica cu API-urile externe (Google, Facebook, Firebase). De asemenea conexiunea la internet este necesară și pentru comunicarea cu Server-ul care se realizează prin intermediul Socket-ului. Fără acces la internet, aplicația nu mai poate oferi toate funcționalitățile.
- android.permission.READ\_EXTERNAL\_STORAGE și .WRITE\_EXTERNAL\_STORAGE – permite utilizatorului să acceseze prin intermediul aplicației date din memoria dispozitivului mobil. Acest lucru este necesar pentru a putea încărca imaginea de profil a unui utilizator.
- android.permission.ACCESS\_COARSE\_LOCATION și .ACCESS\_FINE\_LOCATION – permit aplicației să acceseze locația dispozitivului mobil, acest lucru fiind foarte important întrucât majoritatea funcționalităților aplicației necesită localizarea geografică pe hartă a utilizatorului.

### 3.3.2.2 Configurare API-uri externe

#### 3.3.2.2.1 Google Maps

Pentru a integra API-ul oferit de Google este necesar să adăugăm pachetul Google Play services care cuprinde și API-urile GoogleMaps. Acest lucru se realizează prin adăugare în fișierul build.gradle a următoarei dependente:

```
compile 'com.google.android.gms:play-services:10.2.1'
compile 'com.google.android.gms:play-services-maps:10.2.1'
```

#### Sectiune cod 12 : Google Maps

Pentru a putea folosi serviciile oferite de API-ul Google este necesară o cheie de acces care se obține înregistrând proiectul în Google API Console. Cheia va fi adăugată în cadrul fișierului AndroidManifest.xml.

```
<meta-data
    android:name="com.google.android.geo.API_KEY"
    android:value="AIza[redacted]GRXmtvgeII" />
```

#### Sectiune cod 13 : Google Maps Api Key

După adăugarea API key-ului vom putea folosi în cadrul aplicației noastre serviciile oferite de GoogleMaps cum ar fi:

- **Google Maps API** - este folosit pentru a putea integra harta oferită de Google Maps în interiorul aplicației. API-ul ne oferă harta sub formă de Fragment prin intermediul clasei **SupportMapFragment**;

```
SupportMapFragment sMapFragment;

sMapFragment = SupportMapFragment.newInstance();

//instantiere fragment
sMapFragment.getMapAsync(this);
```

#### Sectiune cod 14 Map Fragment

SupportMapFragment-ul va fi declarat în MainActivity ( activitatea principală ) care implementează interfața **OnMapReadyCallback**.

```
public interface OnMapReadyCallback {
    void onMapReady(GoogleMap var1);
}
```

Metoda ***.getMapAsync(this)*** sincronizează harta și când aceasta este pregătită se va apela metoda ***onMapReady(GoogleMap map)*** care ne va pune la dispoziție un obiect GoogleMap prin intermediul căruia vom putea să modificăm harta.

```
@Override
public void onMapReady(GoogleMap googleMap) {

    this.map=googleMap;
    googleMap.setMyLocationEnabled(true);
    mapIsReady=true;
}
```

Sectiune cod 15 : Initializare GoogleMap

Prin metoda ***.setMyLocationEnable(true)*** din cadrul clasei GoogleMap, permitem afișarea pe hartă a poziției dispozitivului mobil.

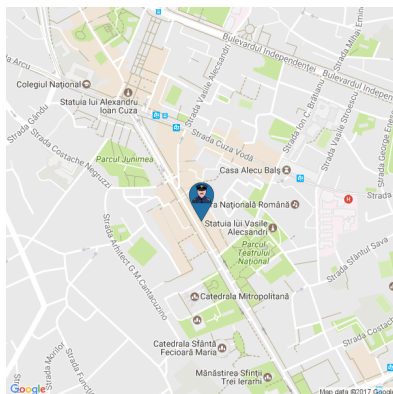
- **Google Maps Static API** – este folosit pentru a captura harta sub formă de imagine pentru a o putea distribui prin intermediul rețelei de socializare Facebook. Serviciul Google Maps Static creează harta pe baza parametrilor unei adrese URL trimisă printr-o solicitare HTTP standard returnând captura hărții ca o imagine.

Exemplu de URL :

```
URL_photo = "https://maps.googleapis.com/maps/api/staticmap?"+ "markers=icon:http://i64.tinypic.com/2qn1lxk.png|"
+String.valueOf(marker.getPosition().latitude)+" , "+String.valueOf(marker.getPosition().longitude)
+"&size=600x600&zoom=16&maptype=roadmap&key=AIza[redacted]eII";
```

Sectiune cod 16 : URL Static Map

În interiorul URL-ului au fost dați ca parametri poziția marcajului rutier, iconița personalizată pentru marcaj, mărimea hărții, nivelul de apropiere a hărții, tipul acesteia și cheia de acces GoogleMaps. Exemplul de URL prezentat va returna următoarea imagine:



Imagine 1: Exemplu Static Map

- **Google Places API** – cu ajutorul acestui API vom putea căuta anumite locații pe hartă.

Pentru folosirea API-ului o să avem nevoie să creăm un obiect **GoogleApiClient**. Prin intermediul acestuia vom putea face cereri de căutare a unei locații introduse de utilizator.

```
mGoogleApiClient = new GoogleApiClient.Builder(getActivity())
    .addConnectionCallbacks(this)
    .addOnConnectionFailedListener(this)
    .addApi(LocationServices.API)
    .addApi(Places.GEO_DATA_API)
    .build();
```

Sectiune cod 17 Google Places

După crearea obiectului verificăm conexiunea, iar dacă totul este în regulă atunci putem să interogăm server-ul Google pentru a obține locațiile dorite. Locația căutată în exemplul de mai jos este reținută în variabilă **constraint**.

```
if (mGoogleApiClient.isConnected()) {
    PendingResult<AutocompletePredictionBuffer> results =
        Places.GeoDataApi.getAutocompletePredictions(mGoogleApiClient, constraint.toString(), mBounds, null);
    return results;
}
else {
    Log.e("", "ERROR GOOGLE API");
}
```

### 3.3.2.2 Facebook

Vom folosi API-ul oferit de Facebook pentru a face posibilă adăugarea contului de Facebook în aplicație și pentru a putea distribui pe pagina de profil a utilizatorului diferite evenimente rutiere din cadrul aplicației.

Pentru a folosi API-ul vă trebuie să adăugăm Facebook SDK în proiect, acest lucru realizându-se prin adăugarea următoarei dependente în fișierul build.gradle:

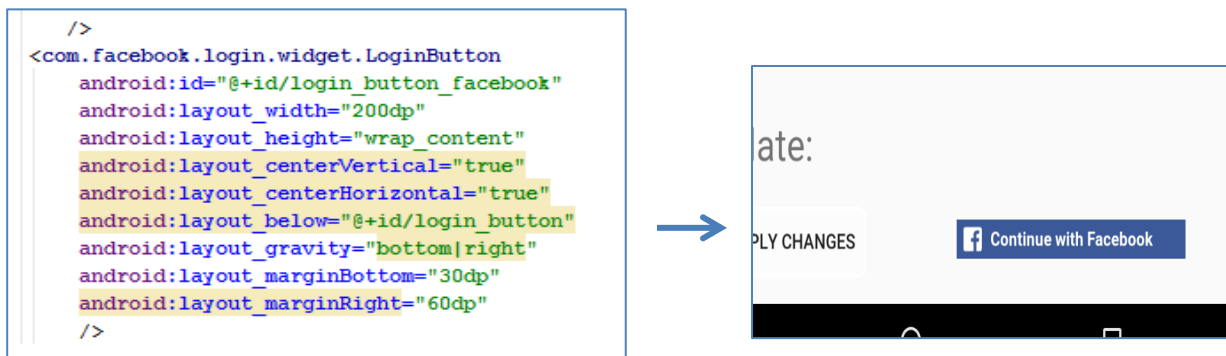
```
compile 'com.facebook.android:facebook-android-sdk:4.20.0'
```

La fel ca și la API-ul Google, pentru folosirea lui trebuie să introducem o cheie de acces oferită de Facebook, în fișierul AndroidManifest.xml. Cheia se obține înregistrând aplicația în cadrul **Facebook Developer Console**.

```
<meta-data
    android:name="com.facebook.sdk.ApplicationId"
    android:value="739[ ]345" />
```

Secțiune cod 18 : Facebook API key

Pentru a face posibilă logarea utilizatorului cu profilul de Facebook, am atașat în interfață grafică a aplicației butonul de login oferit de Facebook SDK (**LoginButton**).



Secțiune cod 19 : Facebook LoginButton

```
LoginButton facebookLogin;
facebookLogin=(LoginButton) rootView.findViewById(R.id.login_button_facebook);
facebookLogin.setFragment(this);
```

Sectiune cod 20 Facebook LoginButton

După ce utilizatorul se va înregistra prin contul de Facebook, se va returna un obiect de tipul LoginResult oferit de Facebook SDK, care conține un **token** de acces cu ajutorul căruia se vor putea realiza operații asupra profilului de Facebook a utilizatorului.

```
facebookLogin.registerCallback(callbackManager, new FacebookCallback<LoginResult>() {
    @Override
    public void onSuccess(LoginResult loginResult) {

        loginSession.setFacebookTokenId(loginResult.getAccessToken().getToken());
        loginSession.setFacebookUserId(loginResult.getAccessToken().getUserId());
    }
});
```

Sectiune cod 21 Inregistrarea Facebook

### 3.3.2.2.3 Firebase storage

Pentru a instala SDK-ul oferit de Firebase, va trebui să adăugăm în fișierul **build.gradle** următoarea dependentă:

```
compile 'com.google.firebase:firebase-storage:10.2.1'
```

Instantiere Firebase:

```
//initializam Firebase-ul
StorageReference mStorageRef;
mStorageRef = FirebaseStorage.getInstance().getReference();
```

Sectiune cod 22 : Instantiere Firebase



API-ul de la **Firestore storage** va fi folosit pentru a încărca imaginile de profil ale tuturor utilizatorilor pe server-ul Firebase, astfel încât să fie accesibile pentru oricine folosește aplicația. Cu ajutorul metodei **.getFile(File)** se va descărca fișierul **File** de pe server-ul Firebase, iar prin intermediul metodei **.putFile(File)** se va încărca fișierul **File** pe server-ul Firebase.

```
StorageReference childRef = mStorageRef.child("Profile pictures").child(username + ".jpg");
final File finalLocalFile = localFile;
childRef.getFile(localFile)
    .addOnSuccessListener((OnSuccessListener) (taskSnapshot) → {
        profileImageURI=Uri.fromFile(finalLocalFile);
        profileImage.setImageURI(profileImageURI);
        progressDialogDownload.dismiss();
    })
```

Sectiune cod 23 : Descarcare fisier Firebase

```
StorageReference childRef=mStorageRef.child("Profile pictures").child(username+".jpg");
childRef.putFile(selectedImage).addOnSuccessListener((OnSuccessListener) (taskSnapshot) → {
    displayToast("UPLOAD DONE");
    progressDialogUpload.dismiss();
});
```

Sectiune cod 24 : Incarcare fisier Firebase

### 3.3.2.3 Implementare funcționalități

#### 3.3.2.3.1 Login

Pentru păstrarea informațiilor de login a utilizatorului am folosit clasa **SharedPreferences** oferită de Android SDK. În acest fel datele utilizatorului se vor salva chiar dacă aplicația va fi închisă și utilizatorul va putea accesa pagina principală a aplicației fără a se loga din nou următoarea dată când intră în aplicație. Pentru administrarea datelor am creat clasa **LoginSession** :

```

public class LoginSession {

    SharedPreferences prefs;
    SharedPreferences.Editor editor;
    Context ctx;

    public LoginSession(Context ctx){
        this.ctx=ctx;
        prefs=ctx.getSharedPreferences("loginPrefereces",Context.MODE_PRIVATE);
        editor=prefs.edit();
    }
}

```

#### Secțiune cod 25 Sesiune de autentificare

După acționarea butonului de login se va apela metoda **login()** care va verifica dacă credențialele introduse sunt corecte trimițând un *request* la Server. Dacă totul este în regulă, se va face tranziția de la **LoginActivity** la **MainActivity** (pagina principală a aplicației) .

```

LoginSession session;
private void login() throws InterruptedException {
    String etUsername=username.getText().toString();
    String etPassword=password.getText().toString();
    if(server.getUserLoginApproval(etUsername,etPassword))
    {
        session.setLoggedIn(true);
        session.setUsername(etUsername);
        Intent intent=new Intent(LoginActivity.this,MainActivity.class);
        startActivity(intent);
        finish();
    }
    else
        Toast.makeText(getApplicationContext(),"Wrong username or password",Toast.LENGTH_SHORT).show();
}

```

#### Secțiune cod 26 Login Activity

### 3.3.2.3.2 Afișarea poziției utilizatorului pe hartă

Pentru a monitoriza poziția dispozitivului mobil am creat clasa **MyLocation** care implementează interfața **LocationListener**.

```
public class MyLocation implements LocationListener {  
  
    LocationManager locationManager;  
    public Location currentLocation;  
  
}
```

Sectiune cod 27 MyLocation

În variabilă **currentLocation** vom memora locația dispozitivului, aceasta fiind actualizată în cadrul metodei **onLocationChanged** oferită de interfața **LocationListener**, care se va apela automat atunci când dispozitivul își va schimba poziția pe hartă.

```
@Override  
public void onLocationChanged(Location location) {  
    currentLocation=location;  
}
```

De asemenea pentru a putea accesa locația dispozitivului trebuie să verificăm mai întâi dacă toate permisiunile necesare sunt acordate ( **ACCES\_FINE\_LOCATION** , **ACCES\_COARSE\_LOCATION**). După verificarea permisiunilor, monitorizarea dispozitivului va începe prin apelarea metodei **.requestLocationUpdates** din cadrul clase **LocationManager**.

```
private void registerGPS() {  
  
    locationManager=(LocationManager)ctx.getSystemService(ctx.LOCATION_SERVICE);  
    Criteria criteria = new Criteria();  
    String best = locationManager.getBestProvider(criteria, true);  
  
    if (ActivityCompat.checkSelfPermission  
        (ctx, android.Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED &&  
        ActivityCompat.checkSelfPermission  
        (ctx, android.Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED) {  
        //...  
        return;  
    }  
    locationManager.requestLocationUpdates(best, 50, 0, this);  
}
```

Sectiune cod 28 Localizare GPS

Poziția dispozitivului va fi trimisă la **Server** în mod automat și va fi marcată pe hartă cu o imagine aleasă de utilizator ( Vezi **3.3.2.3.5 Actualizarea automată a hărții** ). Pentru fiecare **Client** online se va adăuga un marcaj personalizat astfel încât să fie vizibil pentru ceilalți utilizatori și care va dispărea atunci când acesta va ieși din aplicație. Toate datele despre pozițiile marcajelor vor fi stocate în baza de date de pe **Server**.

```
public void addUserMarkersFromDatabase(){
    clearUserMarkers();
    List<MyMarker> myMarkerList;
    myMarkerList = server.getUserMarkersFromDatabase(); // Trimitem cererea la Client
    while (!server.isFinish)//asteptam pana server-ul trimite toate datele
    {
    }
    server.isFinish = false;

    Log.d(TAG, "marker list= " + myMarkerList.toString());
    userMarkers.clear();
    for (MyMarker myMarker : myMarkerList)
        addMarker(myMarker.latitude, myMarker.longitude, myMarker.markerType, myMarker.username, myMarker.description);
}
```

Secțiune cod 29 : Adăugare marcaje pe harta

**Client**-ul va parcurge lista cu locațiile utilizatorilor primită de la **Server** și le va adăuga pe hartă apelând metoda **addMarker** ( Vezi **3.3.2.3.3 Adăugarea unui marcaj pe hartă** )

### 3.3.2.3.3 Adăugarea unui marcaj pe harta

Pentru adăugarea unui marcaj pe harta, utilizatorului îi va fi pus la dispoziție o casetă de dialog în care va introduce toate datele despre marcajul respectiv. Dialog va fi creat cu ajutorul clasei **AlertDialog** . Pentru fiecare tip de marcaj am creat câte un dialog personalizat folosind *layout-uri*.

```
final AlertDialog.Builder mBuilder=new AlertDialog.Builder(MainActivity.this);
View mView=getLayoutInflater().inflate(R.layout.police_event_dialog,null);

mBuilder.setView(mView);
final AlertDialog dialog=mBuilder.create();
dialog.show();
```

Secțiune cod 30 : Dialog eveniment rutier

După ce se vor introduce informațiile marcajului , acesta va fi adăugat pe harta după care va fi trimis la **Server** pentru a putea fi vizibil și pentru ceilalți utilizatori.

```
double latitude=myLocation.currentLocation.getLatitude();
double longitude=myLocation.currentLocation.getLongitude();
mapHandler.addMarker(latitude,longitude,"POLICE",username,policeEvent_addDescription.getText().toString());
server.addMarkerInDatabase(latitude,longitude,"POLICE",username,policeEvent_addDescription.getText().toString());
dialog.dismiss();
```

Secțiune cod 31 : Adaugare marcaj

Adăugarea unui marcaj pe hartă se realizează prin intermediul metodei **addMarker** din clasa MapHandler ( clasă care are ca scop modificarea și actualizarea hărții ).

```
public void addMarker(final double latitude, final double longitude,
String markerType, final String username, String description) {
```

Secțiune cod 32 Funcție adaugare marcaj

În cadrul funcției **addMarker** se va verifica tipul marcajului care urmează să fie adăugat iar informațiile anexate acestuia vor fi adăugate folosind clasa **MarkerOptions** din pachetul GoogleMaps API. Unui marcaj putem să îi setăm patru informații : **Snippet** (Descriere), **Title** (Titlu), **Icon** (Imaginea iconiței), **Position** (Poziția pe hartă).

```

if (markerType.compareTo("POLICE") == 0) {
    MarkerOptions options = new MarkerOptions();
    options.snippet("POLICE EVENT BY " + username + " \nDescription: "+description)
        .title("POLICE")
        .icon(BitmapDescriptorFactory.fromResource(R.drawable.rsz_police_pin))
        .position(new LatLng(latitude, longitude));

    map.addMarker(options);
}

```

Secțiune cod 33 Marcaj politie

```

if (markerType.compareTo("ACCIDENT") == 0) {
    MarkerOptions options = new MarkerOptions();
    options.snippet("ACCIDENT EVENT BY " + username + " \nDescription: "+description)
        .title("ACCIDENT")
        .icon(BitmapDescriptorFactory.fromResource(R.drawable.rsz_accident_pin))
        .position(new LatLng(latitude, longitude));

    map.addMarker(options);
}

```

Secțiune cod 34 Marcaj accident

În cazul adăugării unui marcaj de tip utilizator va fi nevoie să preluăm imaginea de profil a acestuia înaintea setării marcajului. Acest lucru se realizează cu ajutorul API-ului **Firestore** ( Vezi 3.3.2.2.3 **Firestore storage** )

```

if (markerType.compareTo("USER") == 0) {

    mStorageRef = FirebaseStorage.getInstance().getReference();
    StorageReference childRef = mStorageRef.child("Profile pictures").child(username + ".jpg");
    final File finalLocalFile = localFile;
    childRef.getFile(localFile)
        .addOnSuccessListener((OnSuccessListener) (taskSnapshot) -> {
            try {
                s.bitmap = MediaStore.Images.Media.
                    getBitmap(ctx.getContentResolver(), Uri.fromFile(finalLocalFile));
                MarkerOptions options = new MarkerOptions();
                options .snippet("USER")
                    .title(username)
                    .icon(BitmapDescriptorFactory.fromBitmap(
                        getCroppedBitmap(Bitmap.createScaledBitmap(s.bitmap, 95, 95, false))))
                    .position(new LatLng(latitude, longitude));
                addUserMarker(options);
            }
        });
}

```

### 3.3.2.3.4 Afisarea rutelor intre doua puncte

Pentru trasarea și afișarea pe hartă a unei rute o să folosim **GoogleMaps Direction API**. Deoarece acest **API** nu ne oferă o librărie pentru aplicațiile Android (apelurile făcându-se prin operații HTTP care returnează datele în format JSON sau XML ), o să folosim o bibliotecă externă implementată de **akexorcist** numită **GoogleDirectionLibrary**.

```
compile 'com.akexorcist:googledirectionlibrary:1.0.5'
```

Pentru a putea obține ruta între două puncte trebuie să specificăm cheia de acces **Google**, coordonatele locației de start și coordonatele locației finale. Funcția de return va conține două metode: **onDirectionSuccess** și **onDirectionFailure**. Dacă calcularea rutelor între cele două puncte este realizată cu succes se va returna un obiect de tip **Direction** care va conține toate informațiile rutelor.

```
GoogleDirection.withServerKey("AIzaSyCY-o7mGScrYCyEP1txD8u9ZOEkeCSXAIw")
    .from(startPoint)
    .to(endPoint)
    .execute(new DirectionCallback() {
        @Override
        public void onDirectionSuccess(Direction direction, String rawBody) {...}
        @Override
        public void onDirectionFailure(Throwable t) {...} });
```

Secțiune cod 35 : Calculare ruta între două puncte

Obiectul **Direction** va conține o listă cu rutele dintre cele două puncte fixate de către utilizator.

```
myRoutesCount = direction.getRouteList().size();
for (int i = 0; i < myRoutesCount; i++)
{
    Route route = direction.getRouteList().get(i);
```

Secțiune cod 36 : Parcurgere lista rute

Rutele sunt obiecte de tip **Route** care conțin la rândul lor o listă cu segmentele de drum din cadrul rutei. Fiecare segment este alcătuit dintr-o mulțime de puncte cu ajutorul cărora se trasează linia poligonală pe hartă ( **PolylineOptions**).

```
Route route = direction.getRouteList().get(i);
Leg leg = route.getLegList().get(0);
ArrayList<LatLng> directionPositionList = leg.getDirectionPoint();
PolylineOptions polylineOptions =
    DirectionConverter.createPolyline(ctx, directionPositionList, size[i], colors[i]);
```

Sectiune cod 37 : Creare linie poligonala

De asemenea această bibliotecă ne mai permite să aflăm distanța și durata rutei, acest lucru fiind realizabil prin apelarea metodelor **.getDistance()** și **.getDuration()**.

```
myRoute.distanceInfo = route.getDistance();
myRoute.durationInfo = route.getDuration();
```

Sectiune cod 38 : Distanța și durata rutei

Adăugarea unei linii poligonale pe hartă se realizează prin intermediul metodei **.addPolyline** din cadrul clasei **GoogleMap**.

```
map.addPolyline(polylineOptions) ;
```

Sectiune cod 39 : Adaugare linie poligonala

### 3.3.2.3.5 Actualizarea automata a hărții

Pentru a actualiza harta în mod automat o să folosim clasa **Handler**. Aceasta ne permite prin intermediul metodei **.postDelayed** să apelăm automat un obiect de tip **Runnable** la un interval de timp într-un mod automat. În variabilă **mInterval** se va reține numărul de secunde între actualizările hărții.

```
private int mInterval; // timetask interval
private Handler mHandler;
```



Am creat un obiect de tip **Runnable** în care am apelat funcțiile care au ca scop actualizarea diferitelor elemente ale hărții :

- ***addUserMarkerInDatabase*** din cadrul clasei **Server** - această funcție va trimite către server poziția curentă a utilizatorului pentru a putea fi actualizată în baza de date.
- ***addUserMarkersFromDatabase()*** din cadrul clasei **MapHandler** – această funcție va actualiza pozițiile marcajelor de tip utilizator și le va afișa pe hartă.
- ***addMarkersFromDatabase()*** din cadrul clasei **MapHandler** – această funcție va actualiza pozițiile evenimentelor rutiere și le va afișa pe hartă.
- ***showNewMessageNotification()*** – această funcție verifică dacă utilizatorul a primit mesaje noi și va afișa o casuță de dialog prin care îl va anunța pe utilizator.

```
Runnable updateMarkersOnMap = new Runnable() {
    @Override
    public void run() {
        try {
            mapHandler.addUserMarkersFromDatabase();
            server.addUserMarkerInDatabase(myLocation.currentLocation.getLatitude(),
                myLocation.currentLocation.getLongitude(), "USER", username, "VISIBLE");
            mapHandler.addMarkersFromDatabase();
            showNewMessageNotification();
        } finally {
            mHandler.postDelayed(updateMarkersOnMap, mInterval);
        }
    }
}
```

Sectiune cod 40 : Apelarea automata a metodelor

Funcțiile ***startRepeatingTask()*** și ***stopRepeatingTask()*** vor fi apelate la pornirea respectiv închiderea activității principale.

```
void startRepeatingTask() {
    updateMarkersOnMap.run();
}

void stopRepeatingTask() {
    mHandler.removeCallbacks(updateMarkersOnMap);
}
```

Sectiune cod 41 : Pornirea / Oprirea actualizarii automate

### 3.3.2.3.6 Detectarea evenimentelor aflate pe o ruta

Aplicația identifica pe baza unei rute alese de utilizator, ce evenimente rutiere va întâlni acesta pe drum. Fiecare rută este formată din mai multe segmente, fiecare reprezentând o porțiune dreaptă de drum. Pentru fiecare segment, biblioteca **GoogleDirectionLibrary** ne pune la dispoziție atât coordonatele punctului de început cât și coordonatele punctului de final ale respectivei porțiuni de drum.

```
List<Step> pointsList = leg.getStepList();
for(Step segment :pointsList) {
    startPoint =segment.getStartLocation().getCoordination();
    endPoint = segment.getEndLocation().getCoordination();
```

Sectiune cod 42 : Extremitati portiuni de drum

Pentru verificarea existenței unui eveniment pe o rută, vom împărți traseul în segmente de drum și vom verifica cu ajutorul funcțiilor geometrice dacă coordonatele unui eveniment rutier se află pe acel segment.

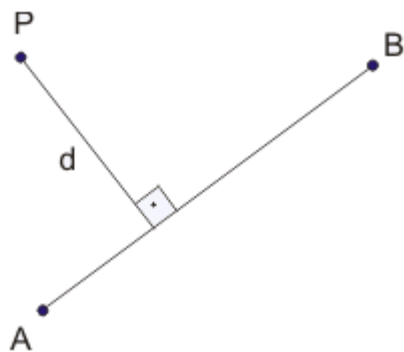
```
for(Marker marker : eventMarkers) {
    for(Step segment :segmentList) {

        startPoint =segment.getStartLocation().getCoordination();
        endPoint = segment.getEndLocation().getCoordination();

        if (isMarkerOn(startPoint, endPoint, marker))
            myRoute.MarkersOnRoute.add(marker);
    }
}
```

Sectiune cod 43 : Verificare existenta marcaj pe ruta

Funcția **isMarkerOn** returnează *true* dacă coordonatele marcajului rutier se află pe segmentul format de punctele de început și sfârșit a segmentul sau *false* în caz contrar.



$$d = \frac{\mathbf{v} \cdot \mathbf{AP}}{|\mathbf{v}|} = \frac{|(x_p - x_1)(y_2 - y_1) - (y_p - y_1)(x_2 - x_1)|}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}}$$

Formula 1 : Distanța dintre un punct și un segment

Pentru aflarea distanței dintre P ( evenimentul rutier ) și segmentul format de A și B vom avea două cazuri :

- Punctul P se află între A și B : pentru calcularea distanței vom folosi formula descrisă mai sus
- Punctul P nu se află între A și B : distanța dintre punct și segment va fi egală cu  $\min(\text{dist}[A,P], \text{dist}[B,P])$

Distanța dintre două puncte o calculăm folosind formula :

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Formula 2 : Distanța dintre două puncte

```
private double distanceTwoPoints(LatLng startPoint, LatLng endPoint)
{
    double distance = Math.sqrt(
        (startPoint.latitude - endPoint.latitude) * (startPoint.latitude - endPoint.latitude)
        + (startPoint.longitude - endPoint.longitude) * (startPoint.longitude - endPoint.longitude));
    return distance;
}
```

Sectiune cod 44 : Distanța între două puncte

Pentru că un marcaj să fie considerat pe ruta, distanța dintre coordonatele acestuia și segmentul de drum trebuie să fie mai mică de 15m.

```
double xx=endPoint.latitude-startPoint.latitude;
double yy=endPoint.longitude-startPoint.longitude;
double length=((xx*(marker.getPosition().latitude-startPoint.latitude))
               +(yy*(marker.getPosition().longitude-startPoint.longitude)))
               /((xx*xx)+(yy*yy));
double X_final = startPoint.latitude+ xx * length;
double Y_final = startPoint.longitude + yy * length;
double length1=Math.abs(distanceTwoPoints(endPoint,marker.getPosition()));
double length2=Math.abs(distanceTwoPoints(startPoint,marker.getPosition()));
double minLength=Math.min(length1,length2);
if( X_final < endPoint.latitude && X_final > startPoint.latitude
    || Y_final < endPoint.longitude && Y_final > startPoint.longitude)
{
    double final_len=Math.min(
        Math.abs(distanceTwoPoints(new LatLng(X_final,Y_final),marker.getPosition()))
        ,minLength);
    if(final_len<0.015) { // 15 meters
        return true;}
}
else
    if(minLength<0.015) {
        return true;}
return false;
```

Secțiune cod 45 : Detectare marcaj pe ruta

### 3.3.2.3.7 Trimiterea de mesaje între utilizatori

Trimiterea mesajelor între utilizatori se realizează cu ajutorul Server-ului, acestea fiind stocate în baza de date. Mesajele unui utilizator vor fi listate în pagina de mesaje a profilului cu ajutorul clasei **RecyclerView** care ne permite să afișăm în mod dinamic lista de mesaje.

```
<android.support.v7.widget.RecyclerView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/message_recycler_view"
    android:scrollbars="vertical"
    >
</android.support.v7.widget.RecyclerView>
```

#### Messages

Item 0  
Item 1  
Item 2  
Item 3  
Item 4  
Item 5  
Item 6  
Item 7  
Item 8  
Item 9

```

recyclerView.setOnItemClickListener(
    new RecyclerViewItemClickListener(
        getActivity(),
        new RecyclerViewItemClickListener.OnItemClickListener()
        {
            @Override
            public void onItemClick(View view, final int position) {
                displayToast("Ai selectat "+position);
                serverHandler.setMessageAsRead(serverResponse.myMessageList.get(position).id);

                final AlertDialog.Builder mBuilder
                    =new AlertDialog.Builder(getActivity());
                final View mView=getActivity().getLayoutInflater()
                    .inflate(R.layout.message_dialog,null);
                final AlertDialog dialog;
                mBuilder.setView(mView);
                dialog=mBuilder.create();
            }
        }
    )
);

```

Sectiune cod 46 : Selectare mesaj din lista

La selectarea unui mesaj din lista se va afișa un dialog în care vor fi afișate toate informațiile despre mesajul respectiv. De asemenea, utilizatorul va putea vizualiza poziția celui care a trimis mesajul său va putea să îi trimită un răspuns la mesajul primit.

Folosind clasa **CameraUpdate** și metodele **.moveCamera()** ,**animateCamera()** din cadrul clasei **GoogleMap** vom putea muta centrul hărții pe poziția utilizatorului care a trimis mesajul.

```

viewUserOnMap.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        LatLng positionOfUser=mapHandler.getPositionOfUser
            (serverResponse.myMessageList.get(position).sender);
        if(positionOfUser!=null) {
            CameraUpdate center = CameraUpdateFactory.newLatLng(positionOfUser);
            CameraUpdate zoom = CameraUpdateFactory.zoomTo(18);
            mapHandler.map.moveCamera(center);
            mapHandler.map.animateCamera(zoom);
            dialog.dismiss();
            //Facem switch la mapFragment
            ((MainActivity) getActivity()).onNavigationItemSelected(new MenuItem() {...});
        }
        else
            displayToast("USER IS NOT ONLINE");
    }
});

```

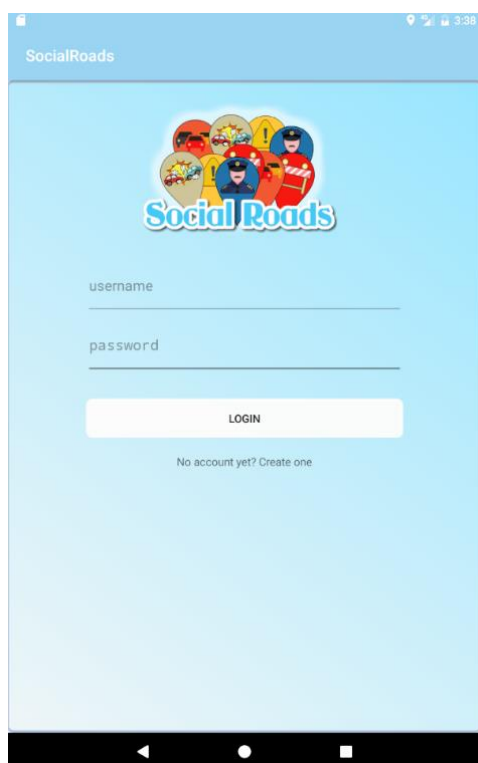
Utilizatorul mai poate opta pentru trimiterea unui mesaj la toți utilizatorii care se afla în jurul lui la o distanță mai mică de **10 km**. Pentru aceasta, vom parcurge locațiile tuturor utilizatorilor și vom trimite mesajul doar la cei care respectă limita de distanță impusă.

```
public void sendMessageToNearbyUsers(double currentLatitude, double currentLongitude,
                                     String messageBody, String subject, String username)
{
    for(Marker m: userMarkers)
    {
        double distance=distanceBetween(currentLatitude,currentLongitude,
                                         m.getPosition().latitude,m.getPosition().longitude);
        if(distance<10)
        {
            try {
                server.addMessageInDatabase(username,m.getTitle(),subject,messageBody,"yes");
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

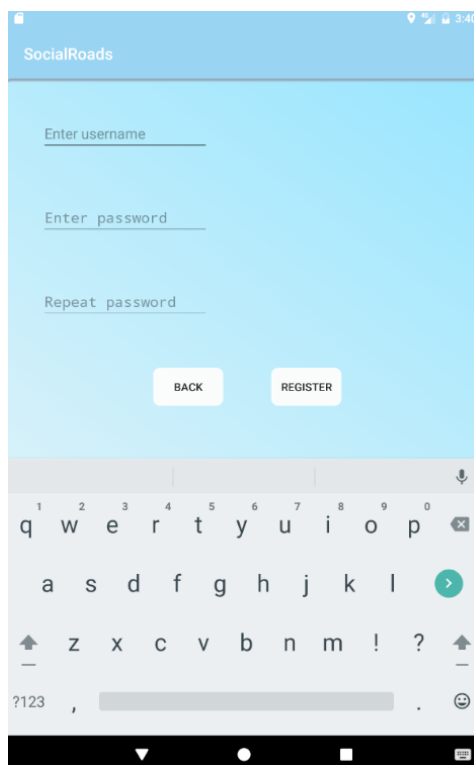
Secțiune cod 47 : Trimitere mesaj la utilizatorii apropiați

## 4 Manual de utilizare

### — Login / Register



Captura ecran 1 : Pagina inregistrare



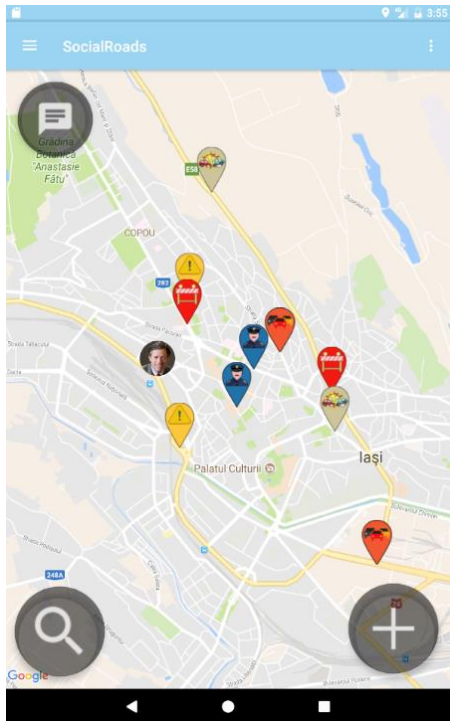
Captura ecran 2 : Pagina login

Primul ecran cu care utilizatorul este întâmpinat atunci când intră în aplicație este cel de autentificare. Accesul în aplicație se face numai după ce utilizatorul a introdus un nume de utilizator și o parolă corecte.

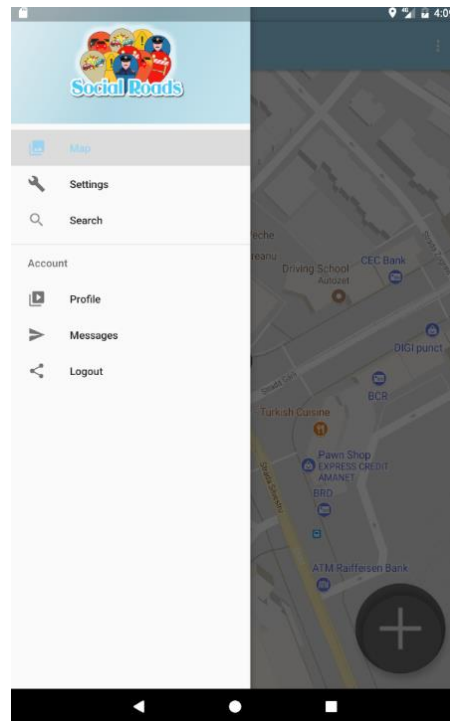
În cazul în care utilizatorul nu are niciun cont de autentificare, acesta se poate înregistra selectând ***“No account yet? Create one”*** de sub butonul de login. Utilizatorul va trebui să introducă un nume de cont valid și să introducă parola dorită de două ori. În cazul în care numele contului este deja folosit sau conține caractere speciale se va afișa un mesaj de eroare și înregistrarea nu se va realiza.

După ce utilizatorul va introduce numele de cont și parola, va fi redirecționat către pagina principală a aplicației.

## — Pagina principală



Captura ecran 3 : Meniu pagina principala



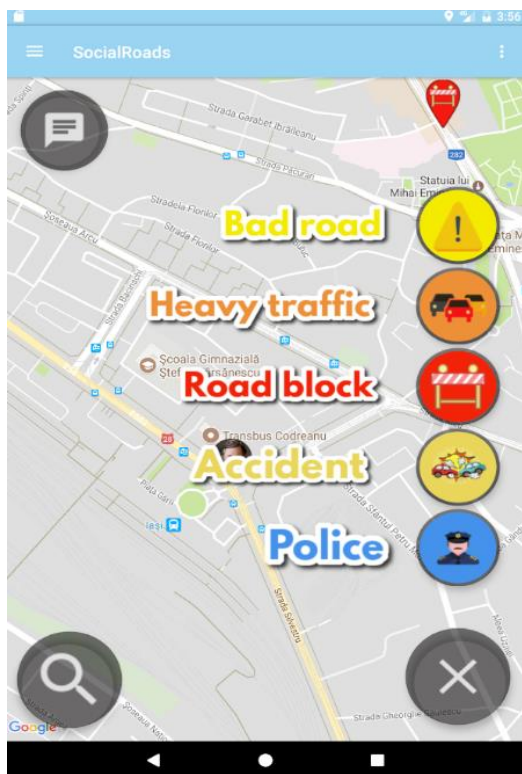
Captura ecran 4 : Pagina principala

În pagina principală va fi afișată harta împreună cu toate elementele acesteia. Indiferent de poziția hărții, utilizatorul va avea acces la trei butoane :

- Butonul de adăugare a marcajelor rutiere (dreapta jos)
- Butonul de căutare (stânga jos)
- Butonul pentru trimiterea unui mesaj utilizatorilor apropiați (stânga sus)

De asemenea, utilizatorul va putea accesa oricând meniul aplicației aflat în colțul stângă sus prin apăsarea pictogramei sau prin glisarea de la stânga spre dreapta a degetului pe ecran. Meniul aplicației permite navigarea între cele 5 ecrane ale aplicației și anume :





Captura ecran 5 : Meniu marcaje rutiere

- Map ( Hartă )
- Settings ( Setări )
- Search ( Căutare )
- Profile ( Profil )
- Messages ( Mesaje )

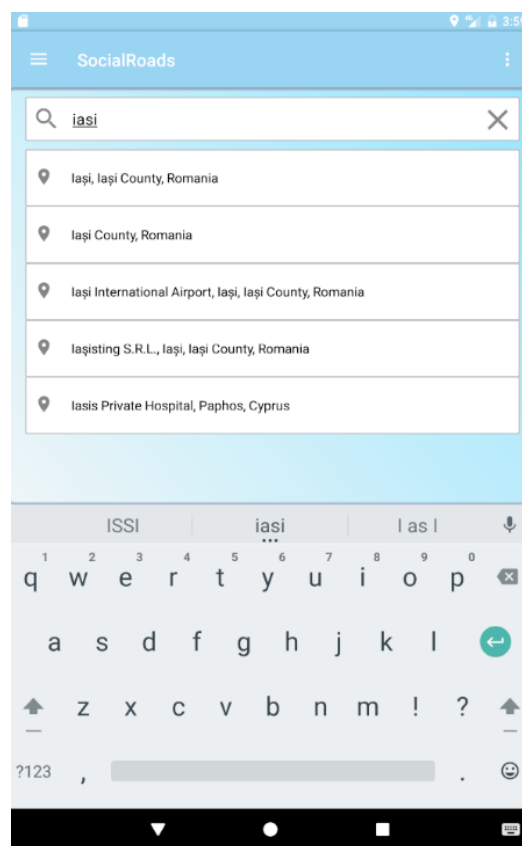
#### ○ Butonul de adăugare a marcajelor rutiere

• Prin apăsarea acestui buton se va afișa o listă cu toate tipurile de marcaje rutiere pe care utilizatorul poate să le introducă pe harta și anume :

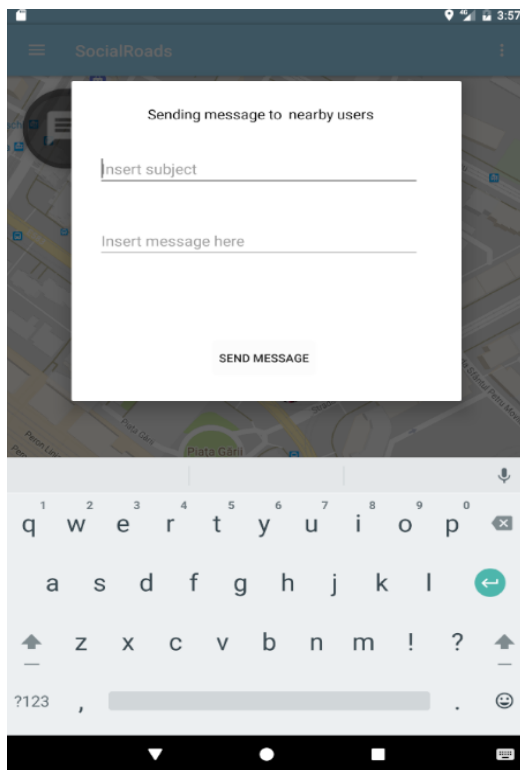
- Bad road ( Condiții de drum rele )
- Heavy traffic ( Circulație foarte greoaie )
- Road block ( Drum blocat )
- Accident ( Accident )
- Police ( Poliție )

#### ○ Butonul de cautare

Prin apăsarea acestui buton utilizatorul va fi redirecționat pe pagina de căutare unde va putea introduce numele unei adrese și prin selectarea unuia dintre rezultatele afișate va putea adăuga un marcaj pe poziția locației respective.



Captura ecran 6 : Pagina de cautare



Captura ecran 7 : Dialog trimitere mesaj utilizatorilor apropiați

○ Butonul pentru trimiterea unui mesaj utilizatorilor apropiați

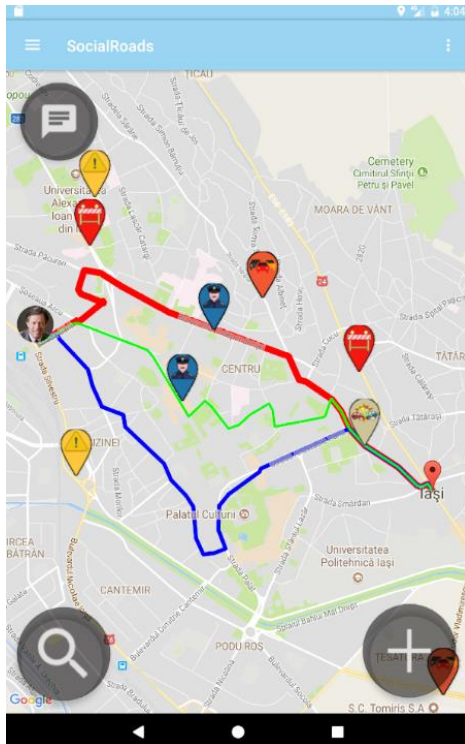
Prin apăsarea acestui buton se va afișa un dialog în care utilizatorul va putea introduce **subiectul** și **mesajul propriu-zis**, urmând ca acesta să fie trimisă la toți utilizatorii on-line pe o rază de 10 km.

## – Afișarea rutelor

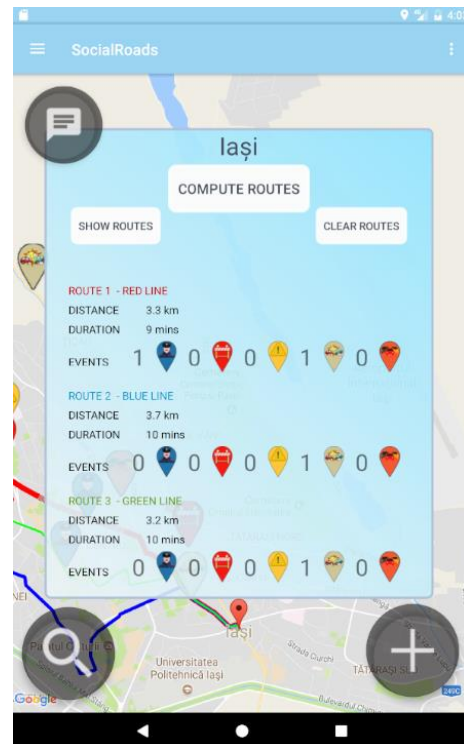
După selectarea unei locații din cadrul paginii de căutare, se va afișa un marcaj pe harta care va conține butonul **COMPUTE ROUTES**, acesta având ca scop calcularea tuturor rutelor dintre poziția utilizatorului și marcajul respectiv.



După apăsarea butonului **COMPUTE ROUTES** vor mai apărea încă două butoane și anume **SHOW ROUTES** și **CLEAR ROUTES** care au ca scop afișarea respectiv ștergerea rutelor de pe hartă.

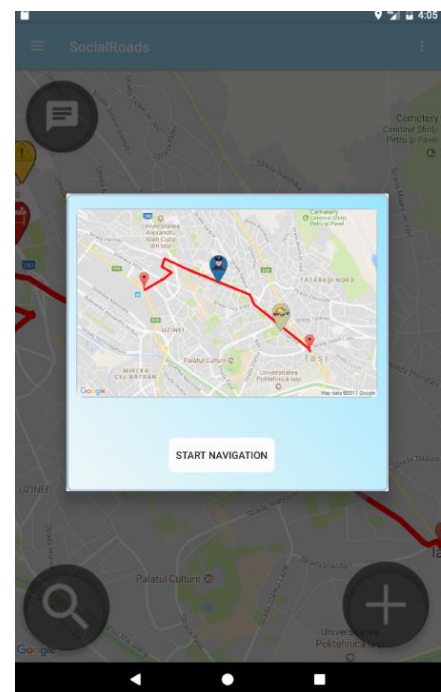


Captura ecran 8 : Rute între doua puncte



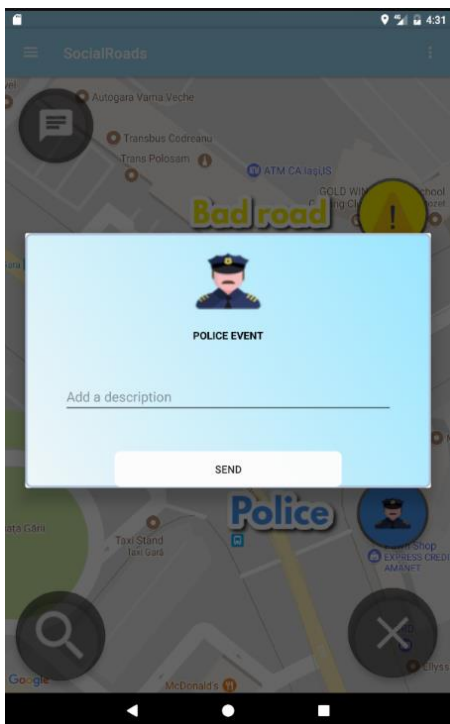
Se vor prezenta trei rute către punctul de destinație, fiecare fiind reprezentată cu o culoare diferită. Pentru fiecare rută se vor afișa distanța, durata de timp și toate evenimentele rutiere care se găsesc pe acel drum.

După selectarea rutei dorite se va afișa un dialog în care este reprezentată ruta împreună cu evenimentele rutiere. De asemenea utilizatorul poate să înceapă călătoria către destinație apăsând butonul **START NAVIGATION**.



Captura ecran 9 : Imagine traseu

## – Adăugare / Vizualizare marcaje rutiere

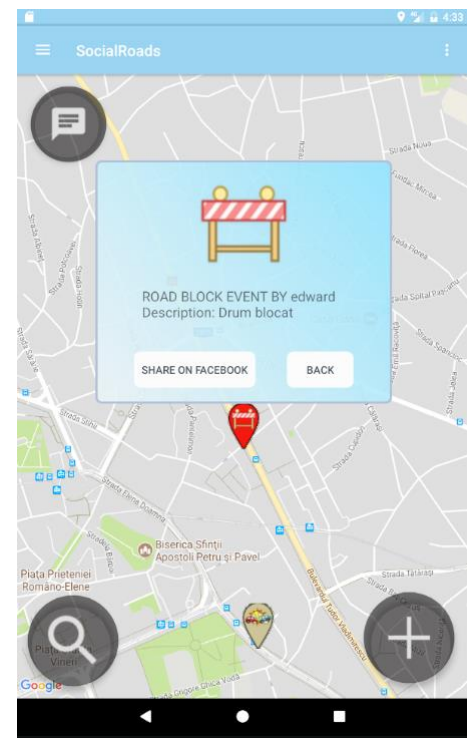


Captura ecran 10 : Adaugare marcaj rutier

Adăugarea unui marcaj rutier se realizează prin selectarea unuia dintre cele cinci tipuri de marcaje din cadrul paginii principale. Fiecare marcaj rutier trebuie să fie însoțit de o descriere, în funcție de evenimentul raportat.

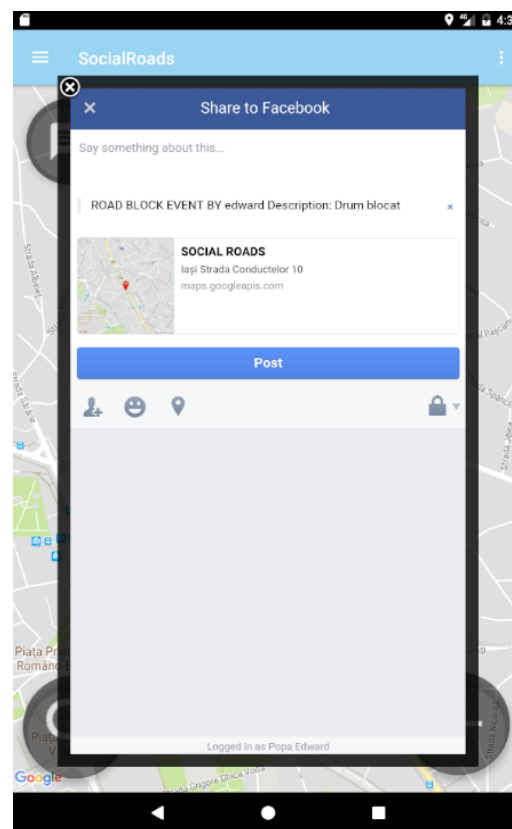
Prin apăsarea butonului **SEND**, evenimentul rutier va fi trimis la **Server** și va putea fi văzut de către toți utilizatorii aplicației.

Vizualizarea detaliilor unui eveniment rutier se realizează prin simpla selectare a marcajului rutier. În acest fel se va afișa o căsuță de dialog în care vor fi afișate tipul marcajului rutier, numele utilizatorului care l-a adăugat și descrierea evenimentului.



Captura ecran 11 : Dialog marcaj rutier

De asemenea, utilizatorul poate opta pentru distribuirea evenimentului rutier pe rețeaua de socializare Facebook prin apăsarea butonului “**SHARE ON FACEBOOK**” .



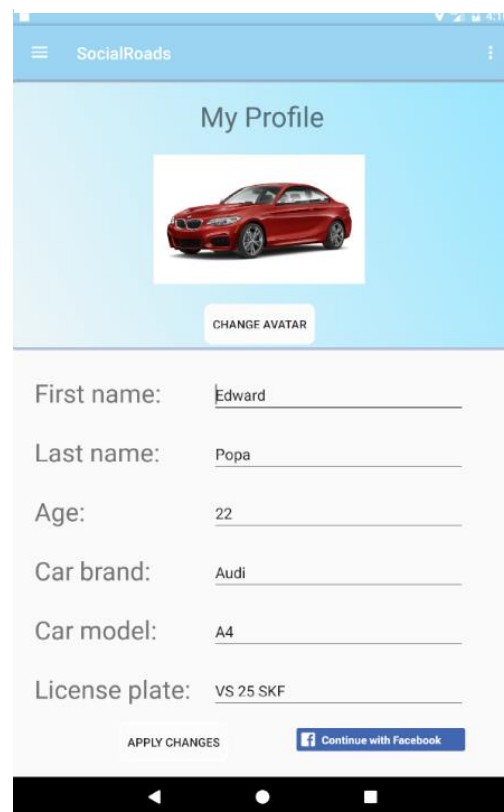
Captura ecran 12 : Distribuie pe Facebook a unui eveniment rutier

## — Profilul utilizatorului

Pagina de profil a utilizatorului poate fi accesată prin selectarea paginii **Profile** din cadrul meniului.

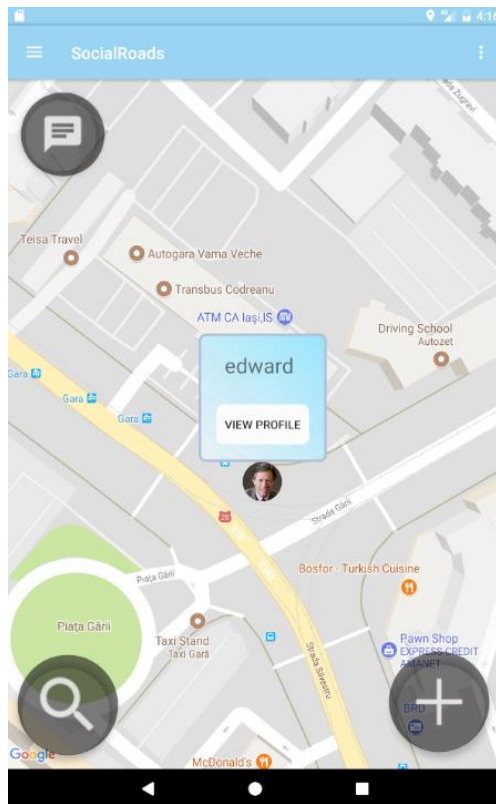
În această pagină vor fi afișate toate informațiile de profil care vor putea fi vizualizate de ceilalți utilizatori ai aplicației.

Prin intermediul butonului **CHANGE AVATAR**, utilizatorul va putea selecta o imagine din galeria foto internă a dispozitivului pentru a o folosi pe post de imagine de profil.

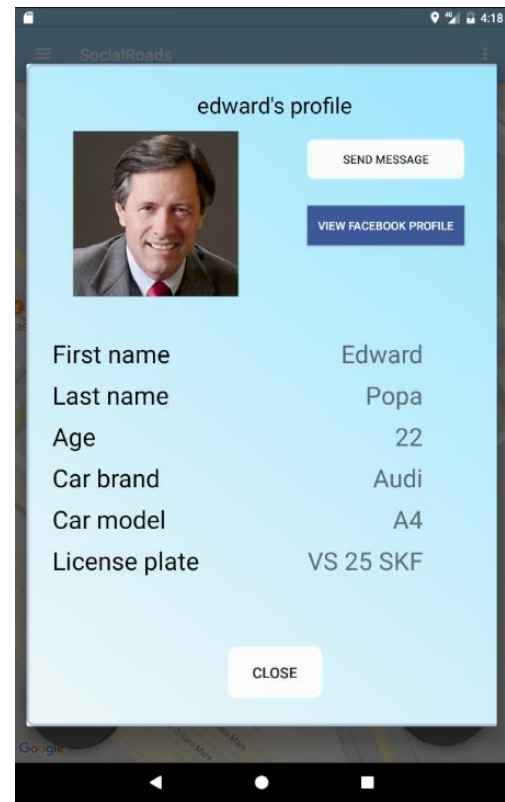


Captura ecran 13 : Pagina profil

Utilizatorul poate să adauge contul său de Facebook pentru a distribui diferite evenimente rutiere și pentru a putea fi vizualizat de către ceilalți utilizatori ai aplicației.



Captura ecran 14 : Reprezentare pozitie utilizator



Captura ecran 15 : Afisare informatii utilizator

Prin apăsarea butonului **VIEW PROFILE** se va afișa o căsuță de dialog în care vor fi afișate toate datele utilizatorului respectiv. De asemenea prin intermediul butonului **VIEW FACEBOOK PROFILE**, vom fi redirectionați către pagina de Facebook a respectivei persoane.

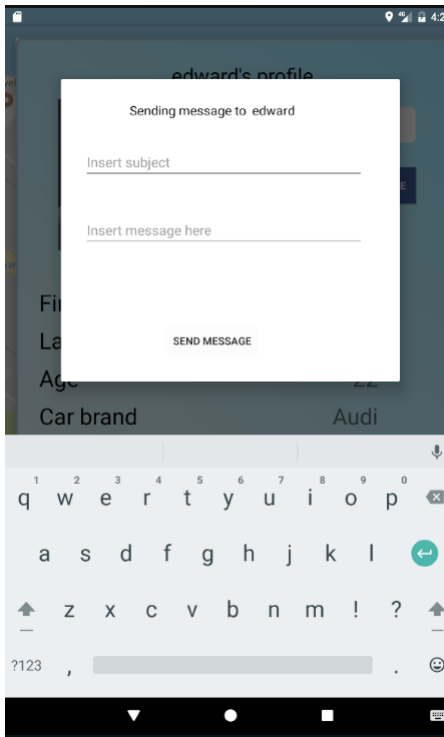
În cazul în care utilizatorul nu are anexat profilul de Facebook , butonul **VIEW FACEBOOK PROFILE** nu va fi vizibil.



## – Trimiterea / Vizualizarea mesajelor

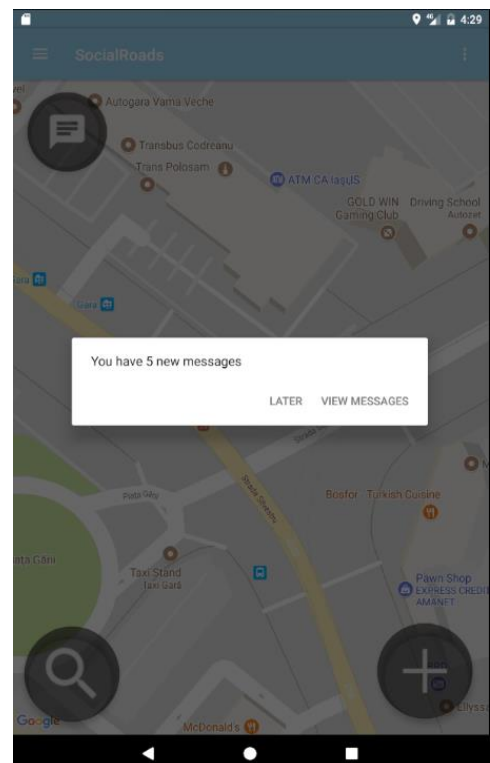
2

Pentru trimiterea unui mesaj către un utilizator, se va selecta butonul **SEND MESSAGE**, care va afișa o casuță de dialog în care se vor introduce subiectul și conținutul mesajului.



Captura ecran 16 : Dialog trimitere mesaj

Utilizatorul va fi anunțat la primirea unui mesaj nou prin intermediul unei notificări. Pentru a vizualiza mesajele, utilizatorul poate selecta opțiunea **VIEW MESSAGES** din cadrul dialogului sau opțiunea **MESSAGES** din cadrul meniului aplicației.



Captura ecran 17 : Notificare mesaje noi



Captura ecran 19 : Pagina de mesaje



Captura ecran 18 : Afisare mesaj

În pagina de mesaje vor fi stocate toate mesajele primite de utilizator, cele necitite fiind marcate cu un contur negru. Utilizatorul poate răspunde mesajului primit prin intermediul butonului **REPLY** sau poate să vizualizeze poziția curentă a celui care a expediat mesajul.



## 5 Concluzii

În concluzie, putem afirma că am reușit să atingem toate punctele pe care ni le-am propus la începutul acestei lucrări. Am dezvoltat o aplicație care reușește să transforme harta rudimentară într-un spațiu de socializare creând astfel o comunitate rutieră care facilitează călătoriile pe drumurile publice.

În prezenta lucrare am încercat să descriu principalele etape ale dezvoltării aplicației, furnizând atât detalii de proiectare cât și fragmente de cod care surprind principalele funcționalități ale aplicației. Dintre funcționalitățile care se remarcă în această aplicație amintim: trimiterea de mesaje între utilizatori, detectarea evenimentelor rutiere pe un anumit traseu, afișarea rutelor între două puncte, distribuirea evenimentelor rutiere pe rețeaua de socializare Facebook.

Aplicația ar putea fi îmbunătățită prin adăugarea unor noi funcționalități cum ar fi:

- Ghidarea utilizatorului pe ruta aleasă prin intermediul indicațiilor vizuale și sonore;
- Adăugarea unei liste de prieteni;
- Alertarea utilizatorului prin intermediul notificărilor atunci când se apropie de un eveniment rutier ;
- Adăugarea unor noi rețele de socializare cum ar fi Twitter sau Instagram ;

# Bibliografie

- [1] **Frăsinaru, Cristian** - *Curs practic de Java*.
- [2] **Orton, John** - *Android Programming for Beginners*. 2015.
- [3] **Google Maps API**. [Online] <https://developers.google.com/maps/android/>
- [4] **Google Places API**. [Online] <https://developers.google.com/places/android-api/>
- [5] **Google Static Maps API**. [Online] <https://developers.google.com/maps/documentation/static-maps/>
- [6] **Firebase Cloud Storage**. [Online] <https://firebase.google.com/docs/storage/>
- [7] **Facebook Graph API**. [Online] <https://developers.facebook.com/docs/android/graph>
- [8] **Documentație oficială JAVA** . [Online] <https://docs.oracle.com/javase/tutorial/>
- [9] **Documentație oficială Android**. [Online] <https://developer.android.com/guide/index.html>
- [10] **Adobe Photoshop**. [Online] <https://helpx.adobe.com/photoshop/user-guide.html>
- [11] **Waze**. [Online] <https://www.waze.com/about>
- [12] **Akexorcist**. Google Direction Library API. [Online] <http://www.akexorcist.com/2015/12/google-direction-library-for-android-en.html>