

# Cuprins

<b>Introducere</b>	<b>6</b>
<b>Contribuții</b>	<b>9</b>
<b>1. Specificații funcționale</b>	<b>10</b>
1.1 Descrierea problemei	10
1.2 Fluxuri de lucru	13
1.3 Elemente de originalitate	16
<b>2. Arhitectura sistemului</b>	<b>17</b>
2.1 Schema bazei de date	17
2.2 Explicația tabelelor bazei de date	18
2.3 Arhitectura sistemului	23
<b>3. Implementare și testare</b>	<b>28</b>
3.1 Baza de date	28
3.2 Aplicația <i>server</i>	32
3.3 Aplicația Android	39
3.4 Aplicația de administrare	43
<b>4. Manual de utilizare</b>	<b>45</b>
4.1 Modul de instalare	45
4.2 Modul de utilizare a aplicației Android	46
4.3 Modul de utilizare a aplicației de administrare	51
<b>Concluzie</b>	<b>53</b>
<b>Bibliografie</b>	<b>54</b>

# Introducere

## 1. Prezentarea problemei

Problema este partajarea utilizării bicicletelor. Bicicletele trebuie să fie repartizate prin oraș, de unde utilizatorii să le poată folosi pentru a ajunge în altă zonă a orașului sau chiar întoarce la aceeași folosind o aplicație pentru telefon.

Aplicația folosită de utilizatori trebuie să primească date de la server, astfel fiind necesară folosirea unui API *web*. De asemenea, trebuie să existe și o aplicație separată, pentru administratori.

Asigurarea unui număr optim de biciclete dintr-o stație la orice oră din zi reprezintă aspectul principal al problemei.

## 2. Motivație

Am ales această temă pentru proiectul de licență deoarece are o aplicabilitate în viața de zi cu zi și realizarea ei constă în folosirea a noi tehnologii față de proiectele pe care le-am făcut pe perioada primilor doi ani de facultate.

De asemenea, am un interes personal pentru dezvoltarea aplicațiilor Android și de API *web*. Realizând acest proiect de licență, voi căpăta cunoștințe și experiență dezvoltând astfel de aplicații.

## 3. Aplicații similare

Închirierea unor mijloace de transport pentru un timp scurt este un concept ce a existat de mult timp pretutindeni. De exemplu, mașini pot fi închiriate pentru o durată de câteva zile persoanelor care au călătorit în altă țară fără mașina lor personală, schiuri și sănii persoanelor care merg la munte etc.

Recent, s-a încercat automatizarea acestui concept la o scară mai mică. Un exemplu este serviciul modern Lime de închiriat trotinete electrice pentru deplasare rapidă prin oraș pe distanțe scurte, disponibil în 70 de țări, inclusiv și în România. Pentru a putea închiria aceste trotinete este

necesară utilizarea unei aplicații mobile, plata făcându-se cu cardul tot prin intermediul ei. Folosind aplicația, utilizatorul găsește o trotinetă disponibilă în apropiere, scanează codul QR, plătește și deblochează trotineta, după care o poate folosi pentru un timp limitat. La final, trotineta se parchează într-un loc sigur.

#### **4. Diferențe si elementul de originalitate**

Modul de operare a serviciului Lime oferă conveniență ridicată utilizatorilor prin faptul că trotinetele pot fi găsite și parcate oriunde înafara străzilor și a proprietăților private. Acest lucru vine cu mai multe dezavantaje. În primul rând, trotinetele pot fi furate cu ușurință. Din cauza utilizării microcontrolerelor, nu vor putea fi folosite fără a fi plătite, dar întreg circuitul electric poate fi schimbat pentru un cost mai mic decât cel al trotinetei. În al doilea rând, poate crea inconveniențe oamenilor care merg pe trotuar, fiind nevoiți să ocolească un număr potențial ridicat de trotinete din calea lor.

Folosind o abordare care prioritizează siguranța vehiculului și confortul publicului, aplicația se va folosi de spații speciale pentru biciclete. Aceste spații ar putea fi monitorizate prin intermediul unor camere de luat vederi pentru a preveni furtul și vor fi situate în locuri ce nu vor crea inconveniențe populației orașului.

Elementul de originalitate constă în existența unui serviciu care va monitoriza utilizarea bicicletelor pentru a determina timpul unde anumite spații vor avea nevoie de mai multe sau mai puține biciclete. Va determina prețuri mai mici pentru anumite rute și perioade de timp pentru a se ajunge la un număr optim de biciclete în stațiile respective. În cazul în care diferența dintre numărul optim și cel actual este prea mare, se vor folosi autovehicule pentru a transporta bicicletele. Acest serviciu va fi apelat în mod automat. De asemenea, vor exista setări ce pot fi modificate de administratori, cum ar fi procentele de reducere a prețurilor sau dacă sunt folosite transporturile.

#### **5. Tehnologii utilizate**

Java este limbajul de programare folosit pentru toate cele trei proiecte. Am ales Java deoarece permite utilizarea *framework*-ului Spring Boot. De asemenea, acest limbaj de

programare poate fi folosit pentru a realiza o aplicație Android precum și un program desktop folosind JavaFX.

Spring Boot a fost folosit pentru a realiza serviciile *web*. Acesta este un framework ce facilitează obținerea și salvarea datelor din baza de date, crearea de *endpoint*-uri, testarea codului etc.

Android pentru aplicația utilizatorilor. Câteva avantaje față de un website sunt performanța crescută, capacitatea de utilizare a notificărilor și consumul scăzut de date mobile.

API Google Maps pentru a prelua harta orașului și pentru a reprezenta pozițiile stațiilor precum și ale utilizatorului.

JavaFX pentru aplicația de administrare.

PostgreSQL pentru baza de date.

Python pentru a genera date aleatorii în baza de date.

## **6. Structura lucrării**

- I.    Specificații funcționale: descrierea detaliată a problemei ce va fi rezolvată, fluxuri de lucru și elemente de originalitate.
- II.   Arhitectura sistemului: schema bazei de date și diagrame.
- III.   Implementare și testare: detalii tehnice, tehnologii folosite și algoritmi.
- IV.   Manual de utilizare: modul de instalare și de utilizare.

# Contribuții

1. Crearea bazei de date folosind PostgreSQL precum și a tabelor, indecșilor și a *trigger*-elor.
2. Crearea unui *script* folosind Python pentru a popula baza de date cu date aleatorii.
3. Crearea unei aplicații folosind *framework*-ul Spring Boot pentru servicii web.
  - 1) Cu ajutorul *framework*-ului Spring Security, am implementat *endpoint*-uri separate pentru utilizatori și administratori.
  - 2) Am implementat un algoritm ce prezice numărul de biciclete care ar trebui să fie într-o stație la o anumită ora și un alt algoritm care încearcă să aducă numărul de biciclete la cel prevăzut.
4. Crearea unei aplicații Android pentru utilizatori folosind Android Studio.
5. Crearea unei aplicații *desktop* pentru administratori folosind JavaFX.

# I. Specificații funcționale

## 1. Descrierea problemei

Bicicletele vor fi situate în spații special amenajate numite stații. Aceste stații ar dispune de lacăte ce se vor deschide automat după ce un utilizator plătește suma cerută și s-ar închide când un utilizator își depune bicicleta.

Dacă o bicicletă acumulează un număr de rapoarte minore sau majore, va deveni indisponibilă utilizatorilor. La anumite intervale de timp, aceste biciclete vor fi inspectate și înlocuite dacă este necesar pentru a fi reparate. Dacă un utilizator raportează greșit de prea multe ori, i se va interzice accesul la aplicație.

În scopul lucrării de licență, se vor omite părțile software și hardware ce țin de biciclete. Acestea ar avea nevoie de microcontrolere și ar necesita mult timp pentru a fi realizate. Așadar, nu va exista nici o verificare legată de lacăte sau de locația bicicletelor.

Astfel încât nu există angajați care să poată realiza transporturi de biciclete, acest aspect va fi simulat de aplicația *server*. De asemenea, nu vor fi folosite coduri QR, fiind o altă componentă a unei biciclete.

Problema constă în crearea a trei aplicații. Prima aplicație este cea de *server*, care va fi folosită de celelalte două. A doua este cea destinată utilizatorilor, și anume o aplicație Android. Ultima aplicație este destinată administratorilor, fiind o aplicație desktop.

Aplicația *server* trebuie să fie accesibilă de cea a utilizatorului, și fiind o aplicație de tip Android, *server*-ul trebuie să aibă servicii web. Cerințele sale sunt următoarele:

- 1) Să se poată conecta la baza de date de unde să poată obține, introduce și actualiza date.
- 2) Să aibă *endpoint*-uri nesecurizate, care vor fi folosite pentru autentificare, înregistrare, resetare parolă și pentru a prelua lista cu orașe.
- 3) Să aibă *endpoint*-uri securizate, accesibile pe baza unui jeton de autentificare. Acest jeton va putea fi de două tipuri, și anume de utilizator sau de administrator. Astfel, o persoană

neautentificată nu va putea apela niciun *endpoint* securizat și un utilizator nu va putea apela *endpoint*-uri menite administratorilor.

- 4) Să poată trimite e-mailuri folosind o metodă asincronă.
- 5) Să poată avea metode ce rulează automat la anumite intervale din zi.
- 6) Să creeze datele de autentificare pentru baza de date și contul de e-mail.
- 7) Să folosească o funcție hash pentru autentificarea utilizatorilor și a administratorilor.
- 8) Să aibă teste ce verifică corectitudinea codului.
- 9) Să aibă *endpoint*-urile necesare pentru cerințele celorlalte aplicații.
- 10) Să implementeze un algoritm care prezice numărul de biciclete necesar la o anumită oră și stație, și un alt algoritm care să aducă numărul actual la cel prevăzut prin reduceri sau transporturi.

Aplicația Android pentru utilizatori va facilita utilizarea bicicletelor disponibile. Cerințele sale sunt următoarele:

- 1) Utilizatorul să își poată crea un cont.
- 2) Să se poată autentifica.
- 3) Să poată să își reseteze parola folosind un cod primit pe e-mail.
- 4) Să poată vedea mesajele primite de la administratori.
- 5) Să poată insera/modifica datele necesare pentru a plăti cu un card de credit.
- 6) Să poată schimba e-mailul sau parola.
- 7) Să poată vedea și selecta pe hartă stațiile.
- 8) Să poată vedea o listă cu bicicletele din o anumită stație și aproximații de timp până când vor ajunge mai multe biciclete.
- 9) Să poată vedea numărul de rapoarte împreună cu severitatea lor pentru o anumită bicicletă.
- 10) Odată ajuns la o stație, să poată raporta o bicicletă dacă nu este în condiție bună.
- 11) Să poată vedea tranzacțiile și rapoartele făcute anterior.
- 12) Odată ajuns la o stație, să poată alege o stație destinație, timpul până la care consideră că va ajunge acolo, și bicicleta disponibilă pe care dorește să o folosească. Înainte ca bicicleta să fie deblocată, va trebui să plătească suma de bani specificată.
- 13) Dacă există reduceri disponibile de la stația respectivă, să poată fi selectate.
- 14) Să poată vedea detalii despre tranzacția curentă.

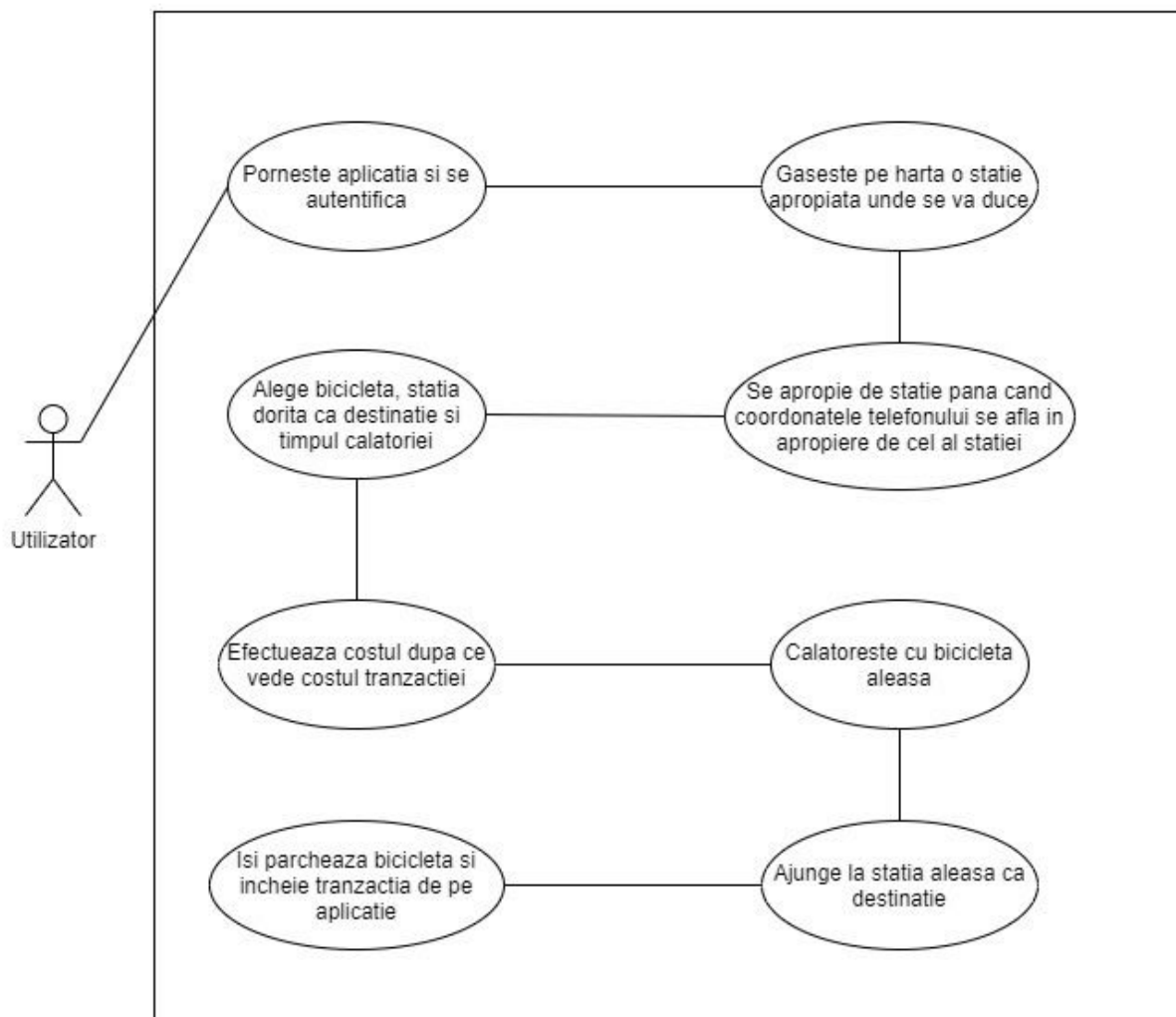
- 15) Să primească notificări pe telefon atunci când timpul sosirii se apropie și atunci când este depășit.
- 16) Să poată depună bicicleta la o stație. Dacă este altă stație față de cea specificată sau dacă a fost depășit timpul va plăti o sumă în plus. De asemenea, dacă a folosit o reducere, va trebui să plătească suma de bani economisită de aceasta inițial.
- 17) Să poată fi penalizat dacă depășește termenul de depunere a bicicletei cu foarte mult timp, astfel încât se va considera furt.
- 18) Să poată primi avertismente dacă raportează în mod greșit biciclete sau dacă întârzie la sosire. Odată ce se acumulează un număr de avertismente, să i se interzică accesul la aplicație.

Aplicația *desktop* este destinată administratorilor și va facilita realizarea de statistici, schimbarea unor setări și vizualizarea datelor. Cerințele sale sunt următoarele:

- 1) Să poată schimba setări ce țin de algoritmul de realocare a bicicletelor, fie prin reduceri automate, fie prin utilizarea de autovehicule.
- 2) Să poată vedea bicicletele și detaliile lor pentru fiecare stație și o statistică pentru starea bicicletelor.
- 3) Să poată vedea detalii despre activitatea din orice stație pentru intervale de timp din trecut.



## 2. Fluxuri de lucru



În diagrama de mai sus se poate vedea fluxul obișnuit de lucru pentru un utilizator. Însă, pot exista mai multe scenarii secundare.

Când utilizatorul pornește aplicația, are mai multe opțiuni. Poate să se autentifice, dacă are deja un cont creat și își știe numele de utilizator și parola. Dacă nu are un cont creat, se poate înregistra, și dacă și-a uitat parola poate să o reseteze. Pentru a-și reseta parola, utilizatorul va primi un cod prin e-mail, pe care îl va folosi la resetare.

Odată autentificat, utilizatorul va vedea harta cu marcatoare pentru fiecare stație și pentru poziția curentă. Aici poate selecta o stație pentru a vedea detalii despre aceasta, sau poate naviga la altă pagină folosind meniul.

În pagina de mesaje utilizatorul își poate vedea mesajele primite de la administratori. Acestea includ avertismente primite în mod automat atunci când utilizatorul raportează în mod fals o bicicletă sau ajunge prea târziu cu bicicleta la stație.

În pagina de tranzacții utilizatorul își poate vedea toate tranzacțiile făcute în trecut, împreună cu detalii despre acestea. Detaliile includ timpul de plecare și de sosire, stația de plecare și de sosire, costul inițial și de penalizare precum și reducerea, dacă a fost folosită.

În pagina metodei de plata utilizatorul poate introduce sau înlocui informațiile cardului de credit cu care va plăti tranzacțiile. Dacă are deja informații salvate, va fi notificat cu un mesaj în josul paginii.

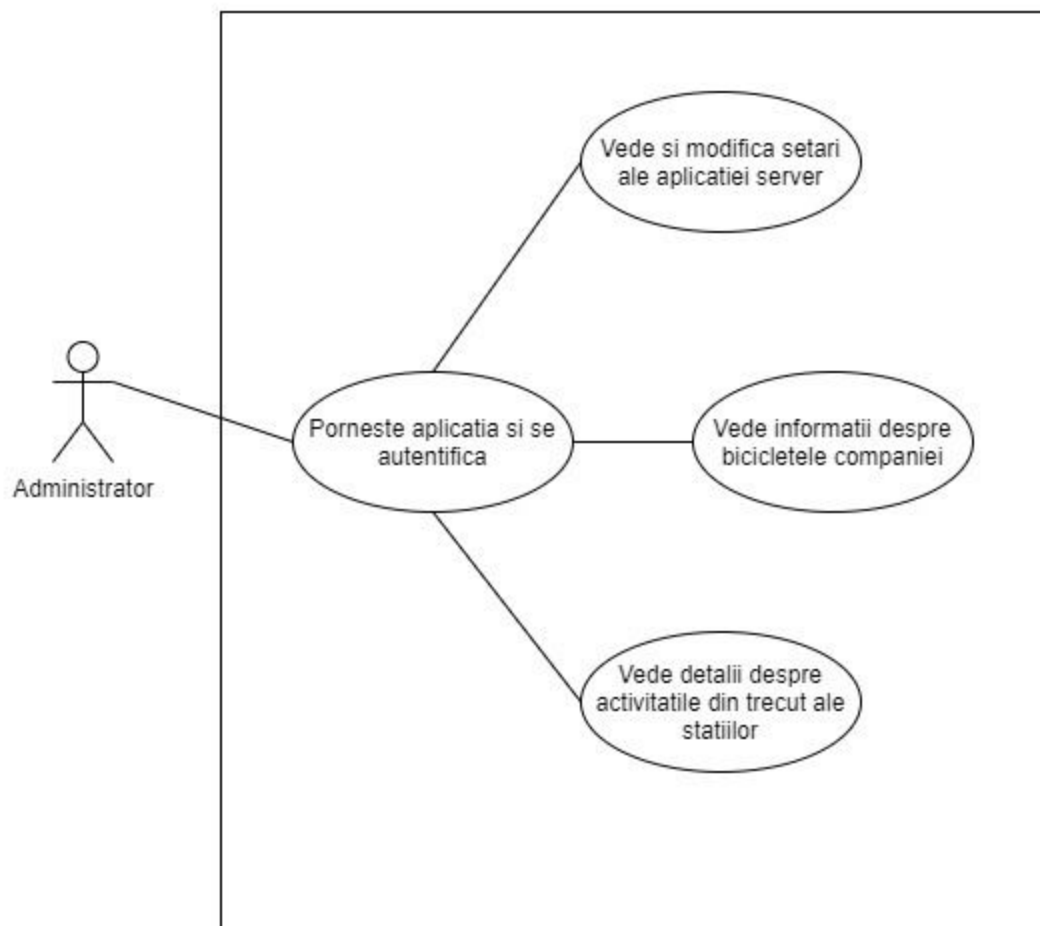
În pagina de cont utilizatorul își poate schimba adresa de e-mail sau parola. De asemenea, poate vedea detalii despre tranzacția curentă, dacă există, și numărul de avertismente pe care le are.

Când utilizatorul apasă pe marcatorul unei stații, acesta va vedea lista de biciclete din acea stație, împreună cu mai multe detalii despre fiecare. Poate apasă pe butonul de rapoarte pentru a vedea rapoartele create în trecut și crea un nou raport, dacă utilizatorul se află lângă stație.

În timpul creării tranzacției, utilizatorul poate să aleagă una din reducerile disponibile. Făcând asta, stația de sosire va fi selectată automat, schimbarea ei anulând reducerea.

După ce este creată tranzacția, utilizatorul va primi notificări cu câteva minute înainte de timpul de sosire selectat, precum și după ce a trecut. Odată ce trece timpul de sosire, utilizatorul își va pierde locul din stația de sosire, însemnând că nu mai are nicio garanție că va exista un loc liber. De asemenea, va fi penalizat pentru timpul întârziat și dacă o reducere a fost selectată, suma de bani economisită va fi adăugată la acea penalizare.

Dacă utilizatorul întârzie prea mult timp, bicicletă va fi considerată ca fiind furată. Astfel, utilizatorul va fi penalizat cu o sumă mare de bani, și își va pierde accesul la aplicație.



În diagrama de mai sus se poate vedea fluxul obișnuit de lucru pentru un administrator. Aplicația de administrare poate fi considerată o demonstrație, având numai o parte din funcționalitățile pe care ar trebui să le aibă. Implementarea tuturor funcționalităților nu ar fi ceva complicat, dar ar necesita mult timp. Astfel încât nu este ceva original sau complex, am ales să nu dedic prea mult timp acestuia.

### 3. Elemente de originalitate

Pentru anumite acțiuni realizate de către utilizatori, în baza de date se vor incrementa numere ce reprezintă activitatea unei stații. Aceste acțiuni includ selectarea unei stații fără biciclete cât timp utilizatorul nu are o tranzacție activă, selectarea unei stații fără locuri libere cât timp utilizatorul caută o stație de sosire, plecarea dintr-o stație, sosirea într-o stație și utilizarea reducerilor.

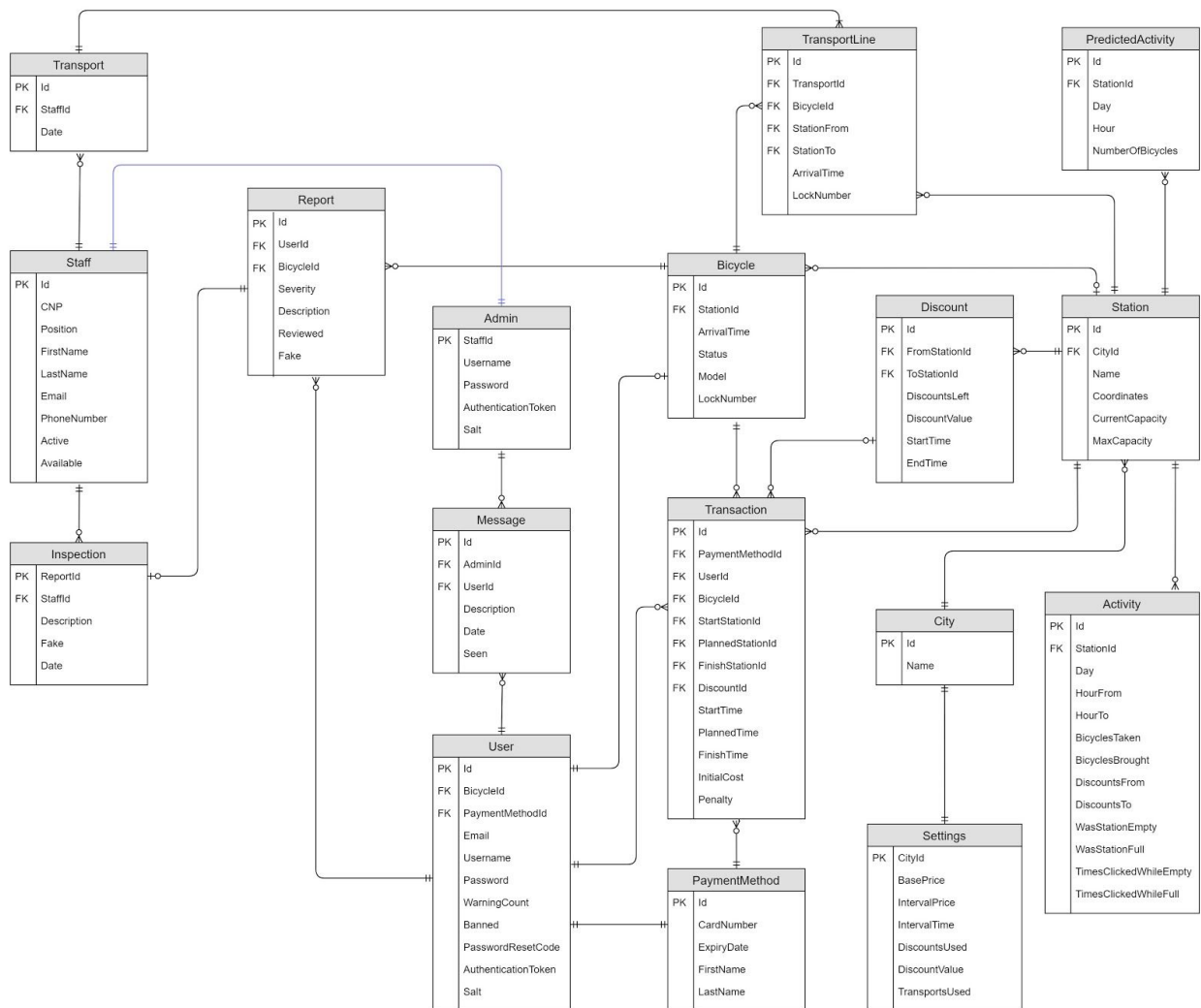
Folosind variabilele independente menționate anterior, un algoritm de predicție va rula în fiecare dimineață. Acesta va genera numărul de biciclete ce ar trebui să fie într-o stație la începutul fiecărei ore. Algoritmul va lua în considerare numai variabilele independente pentru aceeași stație și aceeași oră. Cu cât variabilele independente sunt mai recente, cu atât vor avea o pondere mai mare în generarea variabilelor dependente.

Folosind variabilele dependente, va exista un alt algoritm care va încerca să aducă numărul total de biciclete dintr-o stație la numărul prevăzut. Va determina care stații au mai multe biciclete și care au mai puține, apoi în funcție de numărul de biciclete se vor crea reduceri sau transporturi.

De la stațiile cu puține biciclete în plus se vor crea reduceri către stațiile cu puține biciclete în minus și de la cele cu multe biciclete în plus se vor crea transporturi către stațiile cu multe biciclete în minus. După aceea se vor crea reduceri între stațiile rămase, oricare ar fi numărul de biciclete în plus sau în minus.

## II. Arhitectura sistemului

### 1. Schema bazei de date



## 2. Explicația tabelelor bazei de date

### 1) User

Tabela “User” va avea lista de conturi ale utilizatorilor obișnuiți. Va avea o cheie străină pentru tabelele “PaymentMethod” și “Bicycle”. Cheia pentru tabela de biciclete va fi *null* dacă utilizatorul nu are o bicicletă la momentul respectiv.

Tabela va mai reține e-mailul, numele de utilizator și parola utilizatorilor. Parola va fi criptată folosind o funcție *hash*. Coloana “Salt” va reține în text simplu *salt*-ul utilizatorului pentru funcția *hash*.

Coloana “WarningCount” va reține numărul de avertismente ale utilizatorilor. Odată ce ajunge la un anumit număr, utilizatorul își va pierde dreptul de acces la aplicație, atributul “Banned” devenind true.

Coloana “PasswordResetCode” va reține codul pentru a reseta parola. Ca valoare implicită va fi *null*, memorând un text generat în mod aleator la cererea utilizatorului. Coloana “AuthenticationToken” va reține jetonul generat în mod aleator la momentul autentificării care va fi cerut de *endpoint*-urile securizate.

### 2) Bicycle

Tabela “Bicycle” va avea lista de biciclete accesibile prin intermediul aplicației. Va avea o cheie străină pentru tabela “Station”, ce reprezintă stația în care se află bicicleta. Dacă bicicleta este în uz, va reprezenta stația la care va ajunge.

Atributul “Status” va avea o enumerație pentru stări referitoare la bicicletă: *Station*, *Warehouse*, *Transport*, *User*, *Damaged* și *Stolen*. Dacă acest atribut nu are valoarea *Station*, nu va fi valabil pentru utilizatori.

Dacă nu este în uz, atributul “ArrivalTime” va fi *null*. Altfel, atributul va avea o aproximare pentru ora la care va ajunge în stație.

Coloana “LockNumber” va avea numărul lacătului din stația în care este pusă bicicleta. Dacă este în folosință sau în depozit, atributul va fi *null*.

### **3) Station**

Tabela “Station” va avea lista de stații pentru biciclete. Va conține orașul și coordonatele unde este situată precum și numărul curent și maxim de biciclete.

### **4) Transaction**

Tabela de tranzacții “Transaction” va avea lista de tranzacții efectuate. Va avea chei străine pentru tabelele “User”, “Bicycle”, “PaymentMethod”, “Station” și “Discount”.

Va conține chei străine către stația de unde utilizatorul a luat o bicicletă, unde a spus că va duce bicicleta și unde a dus-o în realitate. De asemenea, va conține timpul de plecare, timpul când utilizatorul a spus că va ajunge și timpul când a ajuns.

Coloana “InitialCost” va conține costul pentru estimarea inițială și “Penalty” va conține costul adăugat. “Penalty” poate fi adăugat din mai multe motive, cum ar fi întârzierea de a ajunge la stație, schimbarea stației sau furtul bicicletei. Acesta va fi dedus de pe cardul de credit al utilizatorului.

“Discount” reprezintă valoarea ce a fost dedusă din “InitialCost” deoarece utilizatorul a selectat o reducere. Dacă stația unde este dusă bicicletă diferă sau dacă utilizatorul întârzie cu prea mult timp, valoarea va fi adăugată în întregime la atributul “Penalty”.

### **5) PaymentMethod**

Această tabelă va avea lista cu detalii de plată ale fiecărui utilizator.

### **6) Discount**

Tabela “Discount” va avea lista cu reducerile între stații. Va avea o cheie străină către stația de plecare și una către stația de sosire. De asemenea, va avea numărul de reduceri rămase, valoarea reducerii și timpul de start și de final al reducerii. Aceste reduceri sunt generate în mod automat de un algoritm.

## 7) Report

Tabela “Report” va avea lista cu rapoartele făcute de utilizatori. Va avea chei străine pentru tabelele “User” și “Bicycle”. De asemenea, va avea un grad de severitate și o descriere.

Atributul *boolean* “Reviewed”, inițial fals, va deveni adevărat odată ce un angajat verifică bicicleta. Atributul *boolean* “Fake” va fi implicit *null*. Odată ce a fost verificată bicicleta, va deveni fals sau adevărat, dacă raportul e fals respectiv adevărat. Dacă raportul este fals și severitatea raportului mare, utilizatorul va primi un avertisment.

## 8) Inspection

Tabela “Inspection” va avea lista cu inspecțiile făcute de angajați. Va avea chei străine pentru tabelele “Staff” și “Report”, cea pentru “Report” fiind și cheia primară. De asemenea, va avea o descriere, data în care a fost efectuată inspecția și dacă angajatul considera ca raportul este adevărat.

## 9) Message

Tabela “Message” va avea mesajele utilizatorilor făcute de către administratori. Va avea chei străine pentru tabelele “Admin” și “User”. De asemenea, va conține textul mesajului, data la care a fost trimis și un atribut *boolean* dacă a fost văzut de utilizator.

## 10) Staff

Tabela “Staff” va avea lista cu angajați. Va conține detalii personale și tipul de muncă efectuat. Angajații cu tipul de muncă *Driver* pot fi selectați de algoritmul de relocare al bicicletelor.

## 11) Admin

Tabela “Admin” va avea lista de administratori. Va conține numele și parola criptată cu o funcție *hash*. Restul datelor vor putea fi preluate printr-un *join* cu tabela “Staff”, cheia primară fiind și una străină totodată.



## 12) Activity

Această tabelă va avea activitatea zilnică pentru fiecare stație și interval de timp. O constrângere de unicitate va fi formată din trei atribute: *Id*-ul stației, ziua și ora de început, astfel încât acest tuplu trebuie să fie unic. Ora de final va fi ora de început plus unu.

Coloanele “BicyclesTaken” și “BicyclesBrought” vor memora numărul de biciclete luate din stație respectiv aduse. Coloanele “DiscountsFrom” și “DiscountsTo” vor memora numărul de reduceri pentru plecarea respectiv ajungerea în stație.

Atributele *booleene* “WasStationEmpty” și “WasStationFull” vor memora dacă stația a fost goală respectiv plină în intervalul de timp dat. Atributele “TimesClickedWhileEmpty” și “TimesClickedWhileFull” vor memora de câte ori a fost selectată stația de un utilizator cât timp stația era goală respectiv plină de biciclete.

Informațiile din acest tabel vor fi folosite de un algoritm pentru a genera predicții ale numărului de biciclete ce ar trebui să fie într-o stație specifică la o anumită oră.

## 13) PredictedActivity

Această tabelă va avea predicțiile generate de un algoritm pentru numărul de biciclete ce ar trebui să fie într-o stație la o anumită oră.

O constrângere de unicitate va fi formată din trei atribute: *Id*-ul stației, ziua și ora, astfel încât acest tuplu trebuie să fie unic. Atributul “NumberOfBicycles” reprezintă numărul de biciclete prevăzut de către algoritm.

## 14) Settings

Tabela “Settings” va avea setările folosite pentru aplicație. Cheia primară va fi *id*-ul orașului pentru care sunt menite setările.

Coloana “BasePrice” va memora prețul de start al unui drum cu o bicicletă și coloana “IntervalPrice” costul care va fi adăugat pentru fiecare “IntervalTime” de minute.

Atributele *booleene* “DiscountsUsed” și “TransportsUsed” vor influența algoritmul de partajare al bicicletelor și “DiscountValue” va reprezenta valoarea de bază a unei reduceri.

### **15) Transport**

Tabela “Transport” va avea lista cu transporturile de biciclete cu un autovehicul. Va avea o cheie străină pentru tabela “Staff” și data efectuării transportului.

### **16) TransportLine**

Tabela “TransportLine” va avea lista de biciclete transportate. Va avea chei străine pentru tabela “Transport”, “Bicycle” și “Station”. Atributul “LockNumber” va memora numărul lacătului unde a fost pusă bicicleta. Atributul “ArrivalTime” va avea o aproximare pentru când va ajunge la destinație.

*Id*-ul de transport indică transportul din care face parte această linie de transport. *Id*-ul de bicicletă reprezintă bicicleta transportată. *Id*-urile de stație indică stația de unde este luată bicicleta și unde este dusă.

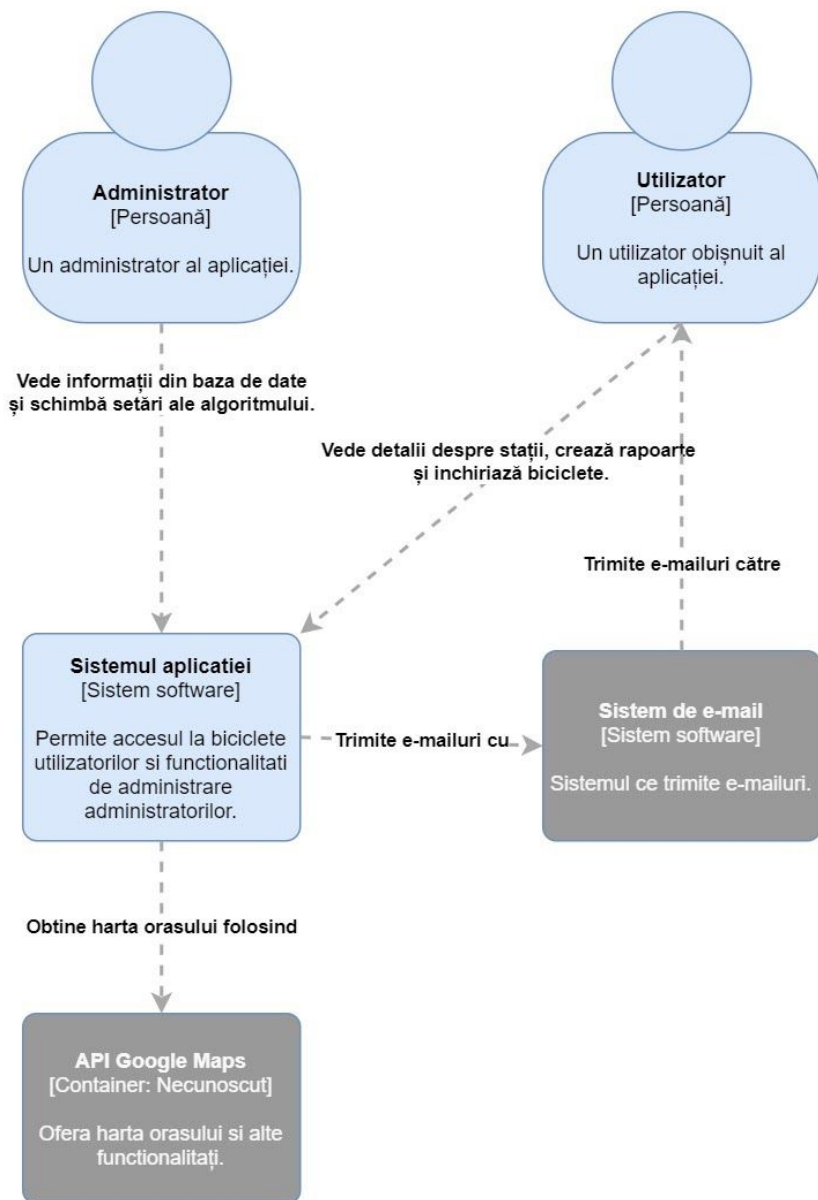
### **17) City**

Tabela “City” va avea lista cu orașe. Va avea două atribute: *Id*-ul și numele orașului.

### 3. Arhitectura sistemului

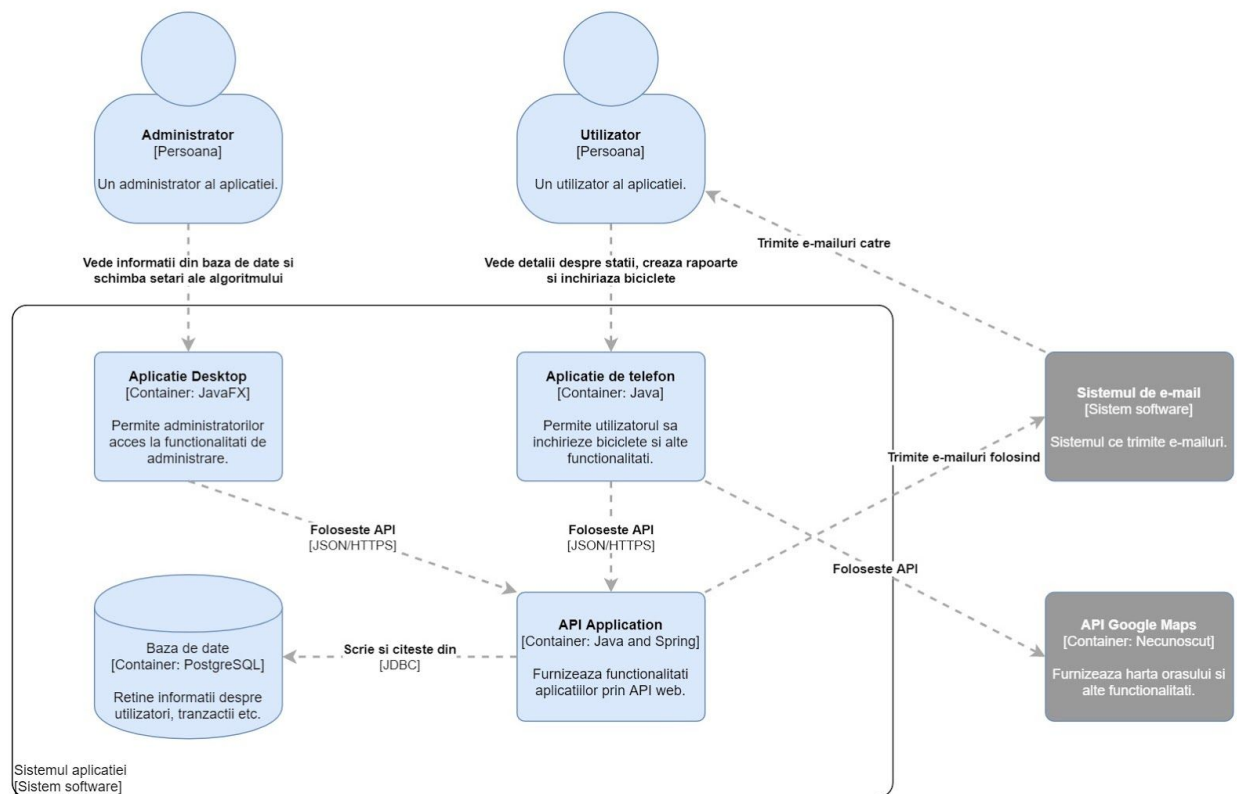
#### 1) Contextul sistemului

Diagrama contextuală a sistemului reprezintă primul nivel din modelul C4. Prezintă cele două tipuri de utilizator, sistemul *software* al aplicației precum și sistemele externe ce vor fi folosite.



API-ul Google Maps va fi folosit în principal pentru a prelua harta orașului iar serviciile e-mail pentru trimiterea codului de resetare a parolei.

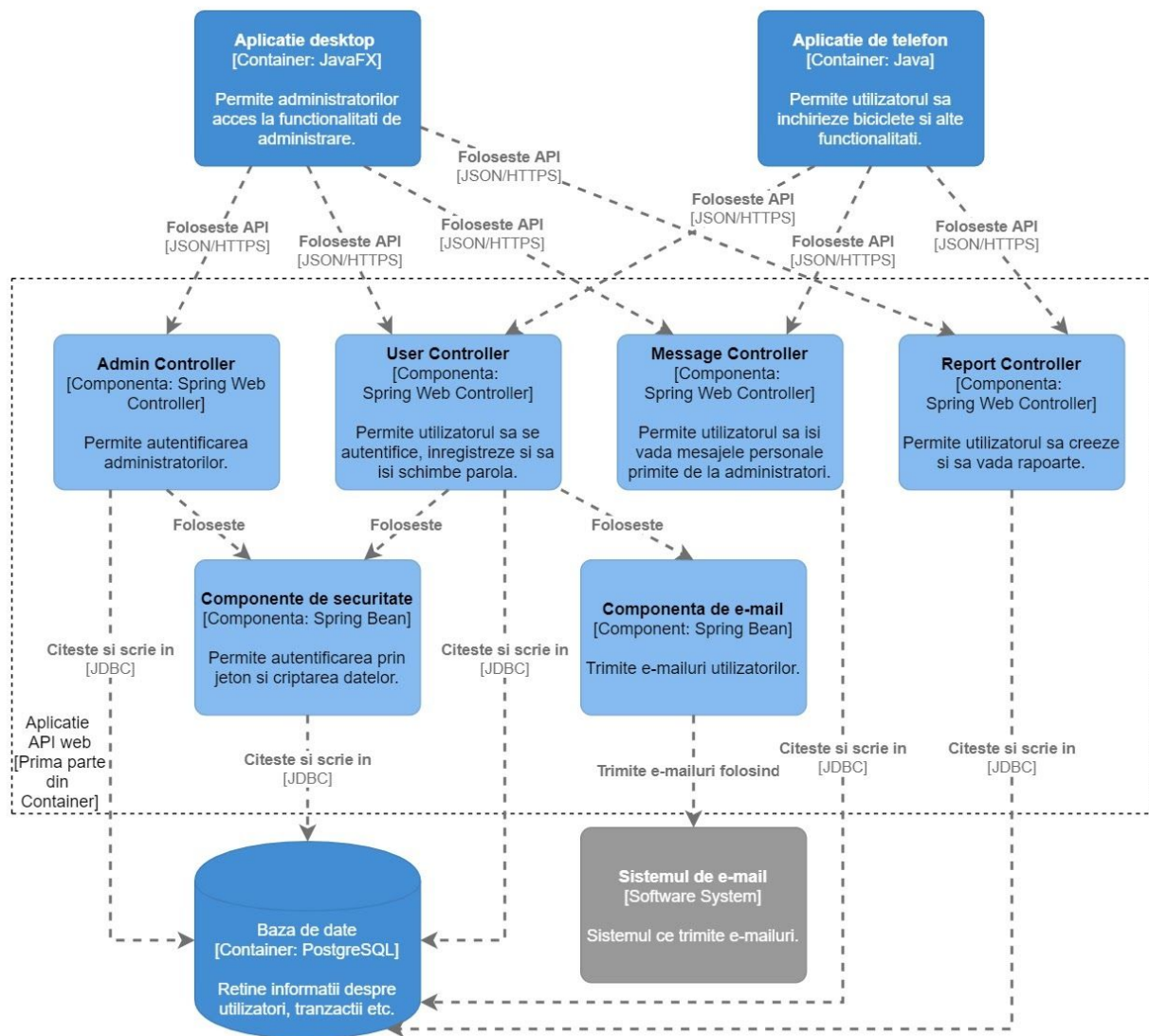
## 2) Diagrama containerului



Utilizatorul va folosi o aplicație de Android, prin intermediul căreia va putea să închirieze biciclete. Administratorul va folosi o aplicație *desktop*, prin intermediul căreia va putea vedea informații din baza de date și schimba opțiuni ale aplicației.

Cele două aplicații vor prelua și modifica informații prin intermediul unei aplicații de API web, care este conectată la baza de date.

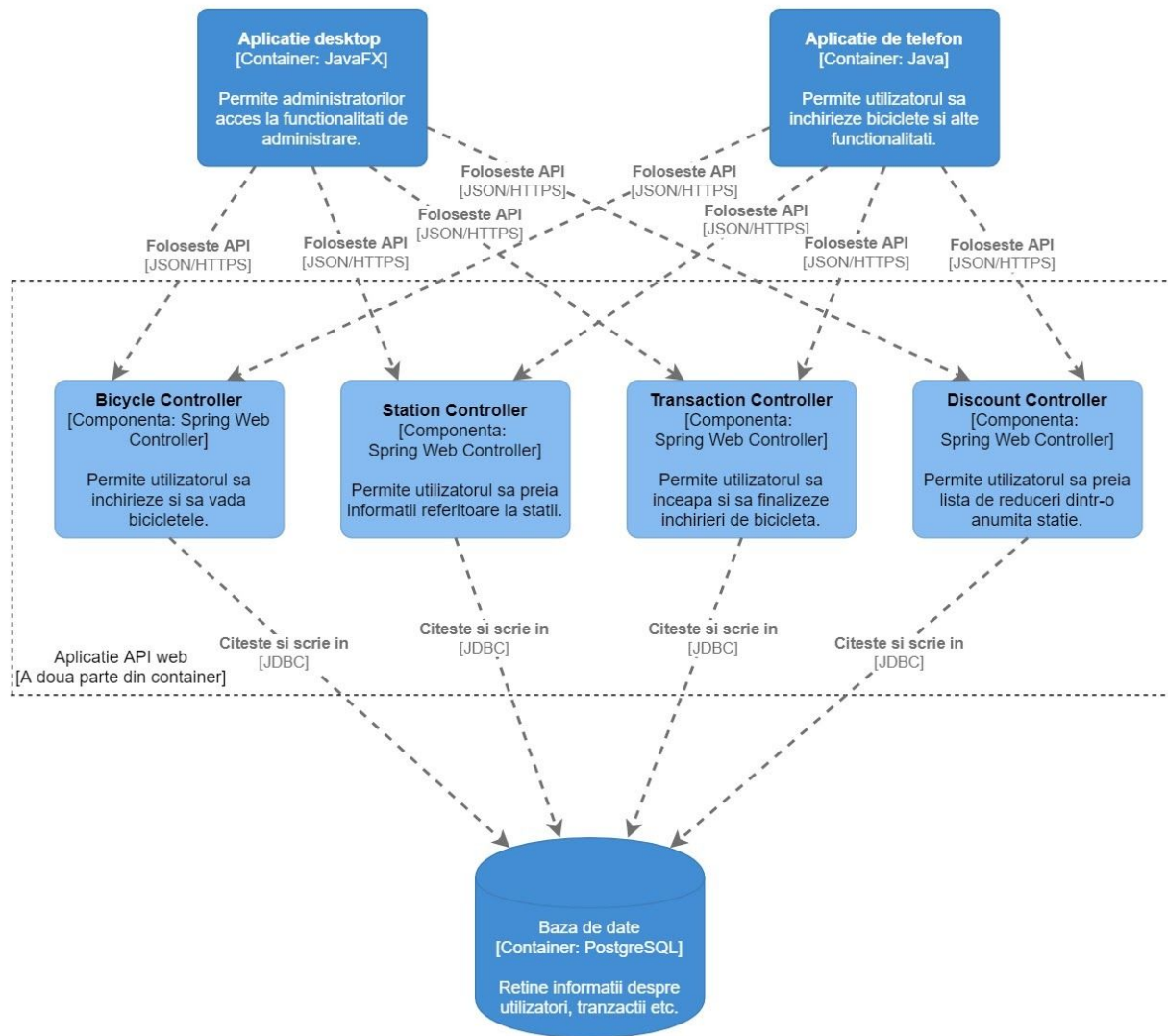
### 3) Diagramele componentelor



*Controller*-ul “Admin” va fi folosit exclusiv de aplicația desktop menită pentru administratori. Va permite logarea administratorilor. *Controller*-ul “User” va fi folosit de utilizatori pentru logare, înregistrare și resetarea parolei. *Controller*-ul “Message” va fi folosit de utilizatori pentru a prelua lista lor de mesaje.

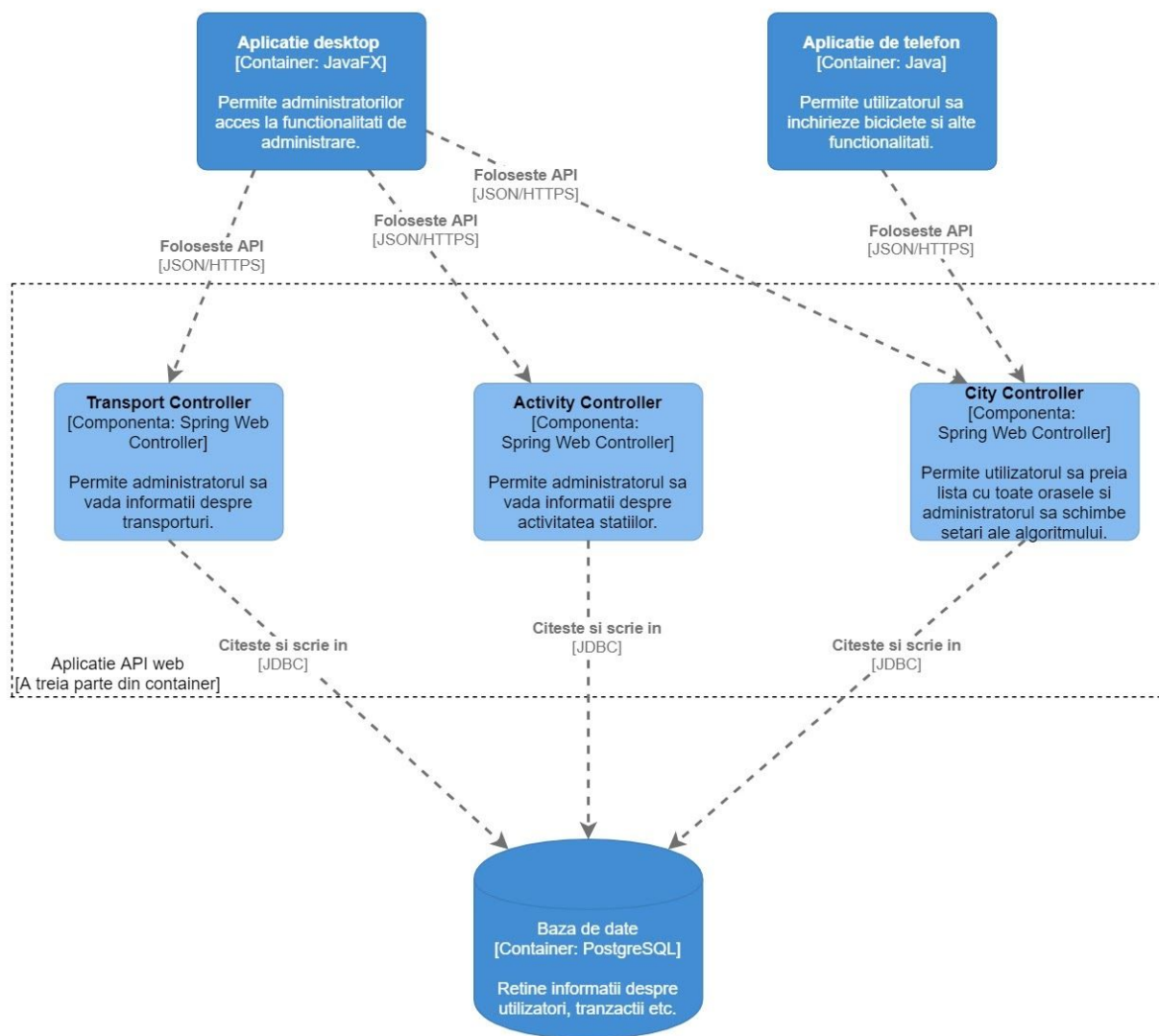
*Controller*-ul “Report” va fi folosit de utilizatori pentru a raporta biciclete. De asemenea, va fi folosit pentru a prelua lista de rapoarte a unei biciclete și de utilizatori pentru a-și vedea rapoartele facute în trecut.

Componenta de securitate se va ocupa de securizarea datelor de logare, prin intermediul funcțiilor *hash*, si de autentificarea prin jeton. Componenta de e-mail va trimite e-mailuri utilizatorilor pentru resetarea parolei.



*Controller*-ul “Bicycle” va fi folosit de utilizatori și administratori pentru a prelua lista de biciclete dintr-o stație precum și detaliile lor. *Controller*-ul “Station” va fi folosit de utilizatori și administratori pentru a prelua lista de stații dintr-un oraș și detaliile lor.

*Controller*-ul “Transaction” va fi folosit de utilizatori pentru a prelua lista lor de tranzacții și de a realiza o tranzacție, adică a închiria o bicicletă. *Controller*-ul “Discount” va fi folosit de utilizatori pentru a vedea reducerile între stații.



*Controller*-ul “Transport” va fi folosit de administratori pentru a vedea lista de transporturi.

*Controller*-ul “Activity” va fi folosit de administratori pentru a prelua informații despre activitatea zilnică a aplicației.

*Controller*-ul “City” va fi folosit de utilizatori pentru a prelua informații despre orașe iar de administratori pentru a și schimba setări ale algoritmului la nivel de oraș.

# III. Implementare și testare

## 1. Baza de date

### a) Declanșatoare

Pentru baza de date am creat declanșatoare care se ocupă cu modificarea datelor din rândurile modificate sau ale unor rânduri din tabele diferite. Acestea nu sunt complexe și astfel încât se declanșează în mod frecvent, scad timpul de rulare ale metodelor din server. Există nouă declanșatoare, fiecare având câte o funcție pe care o apelează.

- 1) Pentru fiecare rând inserat în tabela “TransportLine”, tabelele “Station” și “Bicycle” vor fi actualizate. În tabela “Station” stațiile de plecare și de sosire vor avea capacitatea curentă schimbată (decrementata pentru stația de plecare, incrementata pentru stația de sosire), iar în tabela “Bicycle” va fi schimbată starea (devine *Transport*) și timpul de sosire (devine timpul estimat de sosire).
- 2) Pentru fiecare rând inserat în tabela “Inspection”, tabela “Report” va fi actualizată. Coloanele *review* și *fake* vor fi schimbate (sunt preluate din Inspection).
- 3) După fiecare actualizare din tabela “Report”, în funcție de nivelul de severitate și dacă nu este fals raportul, starea bicicletei va deveni *Damaged* și nu va mai putea fi folosită, urmând să fie luată din stație. Dacă raportul este fals, utilizatorul va primi un avertisment.
- 4) După fiecare actualizare pe coloana de avertismente în tabela “User”, va fi verificat numărul lor. Dacă este mai mare sau egal decât trei, utilizatorul își va avea accesul la aplicație restricționat.
- 5) Pentru fiecare rând inserat în tabela “Discount”, tabela “Activity” va fi actualizată. Pentru stația de plecare coloana *discounts\_from* și pentru stația de sosire coloana *discounts\_to* vor fi actualizate, fiind incrementate cu numărul total de reduceri.



- 6) După fiecare actualizare din tabela “Station”, dacă stația rămâne fără biciclete sau devine plină atunci coloanele *was\_station\_empty* respectiv *was\_station\_full* vor fi actualizate în tabela “Activity”, devenind adevărate.
- 7) Pentru fiecare rând inserat în tabela “Transaction”, mai multe tabele vor fi actualizate. Tabela “Bicycle” își va avea stația și timpul de sosire schimbat în funcție de alegerea utilizatorului. Tabela “Station” va avea *current\_capacity* actualizat atât pentru stația de plecare cât și de sosire. Tabela “User” va avea *id*-ul bicicletei schimbat, astfel încât utilizatorul să nu poată avea mai multe biciclete simultan. Dacă o reducere a fost folosită, tabela “Discount” va fi modificată astfel încât numărul de reduceri rămase să fie decrementat. Tabela “Activity” va fi actualizată prin incrementarea numărului de biciclete luate din stația de plecare.
- 8) După fiecare actualizare din tabela “Transaction”, vor fi calculate penalizările utilizatorilor dacă au întârziat sau au dus bicicleta în stația greșită. În tabela “User” utilizatorul va avea *id*-ul bicicletei actualizat la valoarea *null*. În tabela “Activity” numărul de biciclete aduse va fi incrementat pentru stația de sosire și dacă stația de sosire este alta față de cea aleasă inițial de către utilizator, acesta va primi un avertisment și coloanele de *current\_capacity* vor fi schimbate pentru stația aleasă inițial și cea de sosire.
- 9) Pentru fiecare actualizare pe coloana *warning\_count* din tabela “User” un mesaj va fi creat pentru utilizator, notificându-l de avertisment.

## b) Indecși

În baza de date există paisprezece indecși diferiți, toți fiind de tip B-tree. În PostgreSQL, indecșii de tip Bitmap nu există și cei de tip Hash prezintă anumite probleme de utilizare, în cazul unui *crash*.

- 1) Report (bicycle\_id, reviewed): Index pentru rapoartele fiecărei biciclete.
- 2) Admin (username): Index pentru logarea administratorilor.
- 3) Message (user\_id): Index pentru preluarea mesajelor utilizatorilor.
- 4) User (username): Index pentru logarea utilizatorilor.
- 5) Bicycle (station\_id): Index pentru preluarea bicicletelor din stații.
- 6) Transaction (start\_time): Index pentru statistici legate de tranzacții de-a lungul timpului.
- 7) Transaction (start\_station\_id, start\_time): Index pentru statistici legate de tranzacții de-a lungul timpului pentru plecarea din stații specifice.
- 8) Transaction (finish\_station\_id, start\_time): Index pentru statistici legate de tranzacții de-a lungul timpului pentru sosirea în stații specifice.
- 9) Station (city\_id): Index pentru preluarea stațiilor dintr-un oraș.
- 10) PredictedActivity (station\_id, day, hour): Index pentru preluarea activității prevăzute a unei stații pentru o zi și ora specifică.
- 11) Discount (from\_station\_id, start\_time): Index pentru preluarea reducerilor de la o stație la o oră specifică.
- 12) Activity (station\_id, day, hour\_from): Index pentru preluarea activității unei stații pentru o zi și oră specifică.
- 13) Transport (date): Index pentru statistici legate de transporturi.
- 14) TransportLine (transport\_id): Index pentru transportul din care face parte linia respectivă.

### c) *Script de generare date*

Un *script* a fost creat folosind limbajul de programare Python pentru a popula majoritatea tabelelor din baza de date. Tabelele “Inspection”, “Report”, “Message”, “Discount” și “PredictedActivity” au rămas nepopulate astfel încât popularea lor cu date realiste ar fi fost destul de complicată și nu este necesară pentru utilizarea aplicației sau pentru algoritmul de prezicere a activității stațiilor.

Tabelele “City”, “Settings” și “Station” nu sunt generate aleator, astfel încât să aibă cât mai mult sens. Tabelele “User”, “PaymentMethod”, “Admin”, “Bicycle” și “Staff” au fost generate în mod aleator, folosind un API extern pentru a popula o listă de nume și de prenume din care au fost alese.

Începând cu 1 ianuarie până la 30 iulie, au fost populate în mod aleator tabelele “Transaction”, “Activity”, “Transport”, “TransportLine”. Luând oră cu oră, de la 7 la 21, au fost generate tranzacții pentru fiecare stație în parte. Dacă o stație era aproape de a fi goală sau plină, atunci erau create transporturi pentru a remedia situația.

## 2. Aplicația server

### a) Tehnologii utilizate

*Framework*-ul Spring Boot a fost folosit pentru a realiza aplicația *server* de servicii web. Pachetele Spring Boot JDBC și Spring Boot Data JPA au fost folosite pentru a accesa baza de date și a face operații pe această cu ușurință. Pachetul Spring Boot Mail a fost folosit pentru a putea trimite e-mailuri utilizatorilor.

Pentru partea de securitate am folosit pachetul Spring Boot Security și librăria Jasypt. Folosind pachetul Spring Boot Security am realizat autentificarea prin jeton. Utilizatorii și administratorii vor primi un jeton odată ce se autentifică pe aplicația lor, jetonul oferindu-le acces la *endpoint*-urile server-ului. Un jeton de utilizator nu poate fi folosit pentru *endpoint*-uri de administrare. Librăria Jasypt a fost folosită la *hash*-ul parolelor de utilizatori și administratori, precum și pentru criptarea datelor din fișierul de proprietăți ale aplicației.

Pentru partea de testare automată am folosit framework-ul Spring Boot Test și librăriile Mockito, AssertJ și JUnit. Pentru a vedea și testa manual *endpoint*-urile am folosit framework-ul Swagger și programul Postman.

### b) Straturile aplicației

Aplicația *server* urmează *design pattern*-ul de Repository-Service. Controller-ele formează *endpoint*-uri ce pot fi folosite de către utilizatori și administratori folosind adresa, tipul de cerere și în funcție de *endpoint* informații în plus. În funcție de adresa și tipul de cerere primit, o metodă din Controller va fi apelată.

Informațiile ce pot fi primite în plus sunt de două tipuri, și anume în *request parameters* ce se regăsesc în adresa în *plain text* sau într-un JSON ce va fi criptat de client și decriptat de server. JSON-ul va reprezenta datele unui DTO (*dată transfer object*).

O metodă din *Controller* va apela o altă metodă dintr-un *Service*. În funcție de tipul de cerere, va lua date din baza de date folosind un *Repository* și efectua modificări asupra lor sau datelor primite de la *Controller*. De asemenea, se pot insera, actualiza sau șterge date din baza de date, din nou folosind un *Repository*.

*Repository*-ul este stratul aplicației responsabil cu realizarea operațiunilor pe baza de date. Va exista câte un *Repository* pentru fiecare tabelă din baza de date. Pentru a face acest lucru, se vor folosi entități. Entitățile se folosesc de adnotări pentru a conecta fiecare atribut al clasei cu cel din tabelă.

### c) Configurările și securitatea

În pachetul “config” se regăsesc fișierele de configurare ale aplicației. Clasele “AsyncConfig” și “ScheduledConfig” permit utilizarea metodelor asincrone respectiv programate. Clasele nu au decât o adnotare care activează funcționalitatea în aplicație.

În clasa “SwaggerConfig” se regăsește o singură metodă care se ocupă de crearea documentației Swagger și o adnotare ce activează funcționalitatea.

În clasa “JasyptConfig” adnotarea @EnableEncryptableProperties permite criptarea și decriptarea proprietăților aplicației. Metodă *readFile* permite citirea unui fișier a cărui path e primit ca String, și *return*-ul conținutului fișierului ca String. Această metodă este folosită pentru a obține parola la *encryptor*. Cealaltă metodă a clasei este un *Bean* ce construiește un obiect de tipul *StringEncryptor* ce va fi folosit pentru criptare și decriptare.

În clasa “SecurityConfig” se regăsește configurația pentru securitatea aplicației. Această configurație se folosește de alte trei clase, și anume “AuthenticationProviderUser”, “AuthenticationProviderAdmin” și “AuthenticationFilter”. În configurație sunt setate adresele care sunt securizate și care nu sunt. De asemenea, pentru adresele securizate se stabilește autoritatea jetonului ce le pot folosi.

Clasa “AuthenticationFilter” extrage jetonul din cererea făcută la *endpoint*. Clasele “AuthenticationProviderUser” și “AuthenticationProviderAdmin” se ocupă cu găsirea utilizatorului respectiv a administratorului folosind jetonul primit în cererea pentru *endpoint*. Dacă jetonul nu este găsit și *endpoint*-ul este securizat, atunci utilizatorul sau administratorul va primi o eroare. Autoritatea jetonului este stabilită în metoda de autentificare apelată de fiecare dintre cele două clase.

Pentru securitate am încercat să folosesc protocolul HTTPS. L-am făcut funcțional pentru partea de server, dar nu mai puteam să apelez API-ul din aplicațiile client deoarece certificatul folosit era *self signed*. După mai multe încercări eșuate, am renunțat la acest protocol.

#### **d) *Endpoint*-urile aplicației**

Aplicația are zece *controller*-e implementate, fiecare având una sau mai multe *endpoint*-uri.

*Controller*-ul “ActivityController” are două *endpoint*-uri. Amândouă fac un GET-ALL pe tabela “Activity” respectiv “PredictedActivity”. Sunt securizate și folosite pentru aplicația de administrare.

*Controller*-ul “AdminController” are un singur *endpoint*. Acest *endpoint* nu este securizat și se ocupă cu autentificarea administratorilor, fiind de tipul POST.

*Controller*-ul “TransactionController” are șapte *endpoint*-uri. Două dintre ele sunt securizate pentru administratori. Acestea fac un GET pe tabela “Transaction” pentru tranzacțiile cu o stație de plecare respectiv stație de sosire specifică. Celelalte cinci *endpoint*-uri sunt securizate pentru utilizatori. Două dintre ele fac un GET pe aceeași tabela, obținând toate tranzacțiile respectiv tranzacția activă a unui utilizator. Ultimele trei *endpoint*-uri sunt de tip POST, primind de la utilizator un DTO. Ele se ocupă cu crearea, finalizarea și previzualizarea unei tranzacții.

*Controller*-ul “UserController” are șase *endpoint*-uri, dintre care patru nesecurizate. *Endpoint*-urile nesecurizate sunt de tip POST și se ocupă de autentificarea, înregistrarea, trimiterea codului de resetare a parolei și resetarea parolei. Trimiterea codului de resetare este realizată printr-o metodă asincronă, folosind Spring Boot Email pentru a trimite e-mailul. Cele două *endpoint*-uri rămase sunt de tip PUT, fiind folosite pentru a actualiza informațiile de plată sau de cont ale utilizatorului.

*Controller*-ul “BicycleController” are patru *endpoint*-uri, toate securizate. Două dintre ele sunt securizate pentru utilizatori. Ele sunt de tip GET și obțin toate bicicletele dintr-o stație specifică respectiv o singură bicicletă pentru un *id* specific. Celelalte două sunt securizate pentru administratori și sunt de tip GET. Prima va obține un DTO cu fiecare stare de bicicletă și numărul

de biciclete pentru starea respectivă, informație ce va fi folosită pentru un grafic de proporții. A doua va obține o listă cu biciclete dintr-o stație folosind o limită și un decalaj, astfel încât va fi folosit cu paginare.

*Controller*-ul “CityController” are trei *endpoint*-uri. Primul *endpoint* este nesecurizat, fiind folosit pentru a obține lista de orașe. Ultimele două sunt securizate pentru administratori, fiind folosite pentru a obține sau schimbă setările unui oraș.

*Controller*-ul “DiscountController” are un singur *endpoint*. Acesta este securizat pentru utilizatori și este de tip GET. Este folosit pentru a obține toate reducerile de la o stație specifică.

*Controller*-ul “MessageController” are un singur *endpoint*. Acesta este securizat pentru utilizatori și este de tip GET. Este folosit pentru a obține toate mesajele unui utilizator.

*Controller*-ul “ReportController” are patru *endpoint*-uri securizate pentru utilizatori. Trei *endpoint*-uri sunt de tip GET, fiind folosite pentru a obține raportul unui *id* specific, toate rapoartele neverificate ale unei biciclete sau toate rapoartele făcute de un utilizator. Ultimul *endpoint* este de tip POST și este folosit pentru a salva un nou raport.

*Controller*-ul “StationController” are cinci *endpoint*-uri securizate. Primele patru sunt securizate pentru utilizatori. Două dintre ele sunt de tip GET, fiind folosite pentru a obține o stație pentru un *id* sau toate stațiile pentru un oraș. Celelalte două sunt de tip PUT, incrementând numărul de apăsări pentru o stație goală respectivă plină. Ultimul *endpoint* este securizat pentru administratori și este folosit pentru a obține numai anumite informații pentru fiecare stație dintr-un oraș specific.

#### **e) Serviciile apelate în mod automat**

Există mai multe servicii ce sunt apelate în mod automat. În “ActivityService” există o metodă apelată zilnic pentru a crea rândurile din tabela “Activity” pentru ziua respectivă. În “TransactionService” există o metodă apelată zilnic foarte devreme dimineața pentru a finaliza toate tranzacțiile care încă sunt active. Bicicletele vor fi considerate ca fiind furate, iar utilizatorul își va avea accesul la aplicație restricționat.

În “BicycleService” există o metodă apelată de mai multe ori pe zi. Pentru fiecare bicicletă cu timpul de sosire depășit, stația de sosire va fi setată la *null*. Asta înseamnă că stația va avea un loc liber în plus, iar utilizatorul nu va mai avea un loc rezervat.

În “DiscountService” există o metodă apelată la fiecare oră de activitate. Aceasta va crea reduceri și transporturi de la stațiile cu biciclete în plus către stațiile cu biciclete în minus. Pentru a determina stațiile cu biciclete în plus sau în minus, se va folosi numărul prevăzut de algoritmul de predicție.

În “PredictedActivityService” există o metodă apelată odată pe zi. Această va prezice numărul de biciclete ce ar trebui să fie pentru fiecare stație și oră. Pentru a face asta se va folosi de datele din trecut.

În “TransportService” există o metodă apelată de mai multe ori pe zi. Aceasta va identifica bicicletele deteriorate și le va transporta către depozit.

#### **f) Algoritmi principali**

În aplicația server există doi algoritmi principali. Primul creează predicții pentru numărul de biciclete care ar trebui să fie pentru fiecare stație și oră de activitate. Al doilea se folosește de aceste numere pentru a crea reduceri și transporturi între stații.

Primul algoritm va rula pentru fiecare stație și pentru fiecare oră de activitate. Va prelua activitățile din trecut pentru stația și ora curentă din buclă. În aceste activități se regăsesc mai multe informații, cum ar fi numărul de biciclete luate și aduse, numărul de reduceri folosite pentru a veni și pentru a pleca, numărul de apăsări pe stație cât timp utilizatorul caută o stație de plecare și stația este goală precum și numărul de apăsări pe stație cât timp utilizatorul caută o stație de sosire și stația este plină.

Pentru a prezice numărul de biciclete, attributele menționate anterior vor fi înmulțite cu o proporție și adunate. Numărul obținut va fi la rândul sau înmulțit cu o proporție și adunat la o sumă totală. Această sumă este împărțită la suma proporțiilor și apoi aproximat.

Acest algoritm este unul simplu de predicție. Am încercat să folosesc algoritmul de regresie liniară, dar nu am reușit astfel încât nu am găsit o variabilă independentă pentru ecuație.



În ecuația  $y = a + b \cdot x$  folosită pentru regresie,  $y$  reprezintă variabilă dependentă ce este generată de algoritmul de regresie și  $x$  reprezintă variabila independentă, cea cunoscută. Litera **a** reprezintă interceptul, adică locul pe ordonată unde dreapta de regresie se intersectează cu OY, adică valoarea lui Y pentru X egal cu 0, și **b** reprezintă panta de regresie care arată cu cât se modifică Y atunci când X se modifică cu o unitate.

Am încercat să folosesc numărul obținut după aplicarea proporțiilor pe atributele din activitate ca și variabila independentă. Într-un final mi-am dat seama că acest lucru este imposibil, astfel încât nu se știe acest număr atunci când se dorește a fi creată predicția. Nu poate fi folosită nici activitatea cu o ora înainte, astfel încât nu ar fi cunoscută la timpul potrivit. Reducerile și transporturi trebuie create cu aproximativ douăzeci de minute înainte, ceea ce ar însemna că se cunoaște activitatea doar pentru primele patruzeci de minute.

De asemenea, am încercat să folosesc ziua săptămânii și ora ca variabila independentă, dar asta ar fi însemnat că variabila dependentă s-ar fi repetat pentru fiecare zi și oră. Pentru regresie trebuie ca valoarea variabilei independente să fie unică pentru a determina o nouă variabilă dependentă.

Am ajuns la concluzia că trebuia să folosesc o rețea neuronală sau să nu mai folosesc stații pentru aplicație, putând astfel utilizatorii să se oprească oriunde cu bicicleta. În cazul acesta aș fi putut să folosesc un algoritm de clusterizare. Din cauza lipsei de timp, am implementat un algoritm simplu de predicție.

**Al doilea algoritm** va rula automat cu douăzeci de minute înaintea fiecărei ore de activitate. Se vor crea două *HashMap*-uri, primul conținând perechi de tipul (stație - număr de biciclete în plus) și al doilea de tipul (stație - număr de biciclete în minus). Pentru fiecare stație și oră se va lua din baza de date numărul prevăzut de primul algoritm. În funcție de diferența dintre numărul prevăzut și cel actual, se va introduce stația și diferența în *HashMap*-ul potrivit.

După ce sunt formate și populate aceste *HashMap*-uri, o metodă va fi apelată de două ori. Prima oară metoda încearcă să cupleze stațiile care au multe biciclete în plus cu stațiile care au multe biciclete în minus pentru a realiza transporturi respectiv stațiile care au puține biciclete în plus cu stațiile care au puține biciclete în minus pentru a realiza reduceri. În a doua iterație,

stațiile rămase vor avea reduceri realizate între ele, indiferent de numărul de biciclete în plus sau în minus.

Acest algoritm ar putea fi îmbunătățit. Transporturile ar putea fi optimizate astfel încât angajatul să mute cât mai multe biciclete folosind ruta cea mai eficientă.

#### **g) Testarea aplicației**

Folosind pachetele AssertJ pentru afirmații, JUnit pentru adnotări de testare și Mockito pentru *mock*-uri și injectari am realizat teste automate pentru majoritatea metodelor din stratul de servicii. Acestea simulează scenariile posibile pentru o metodă și verifică dacă rezultatele sunt corecte.

Folosind Swagger la început am testat în mod manual *endpoint*-urile API-ului. După ce am implementat autentificarea prin jeton, m-am folosit de programul Postman pentru a face acest lucru.

### **3. Aplicația Android**

#### **a) Tehnologii folosite**

Pentru a realiza această aplicație am folosit Android Studio.

Pentru a face conversia din JSON în obiect și din obiect în JSON am folosit librăria Jackson. Clasa `ObjectWriter` se ocupă de conversia din JSON în obiect iar clasa `ObjectMapper` de conversia din obiect în JSON.

Pentru a valida adrese de e-mail am folosit clasa `EmailValidator` din librăria `Commons Validator`.

Pentru a programa trimiterea notificărilor după începerea unei tranzacții am folosit clasa de bază `Worker` din API-ul `WorkManager`.

#### **b) Permisunile aplicației**

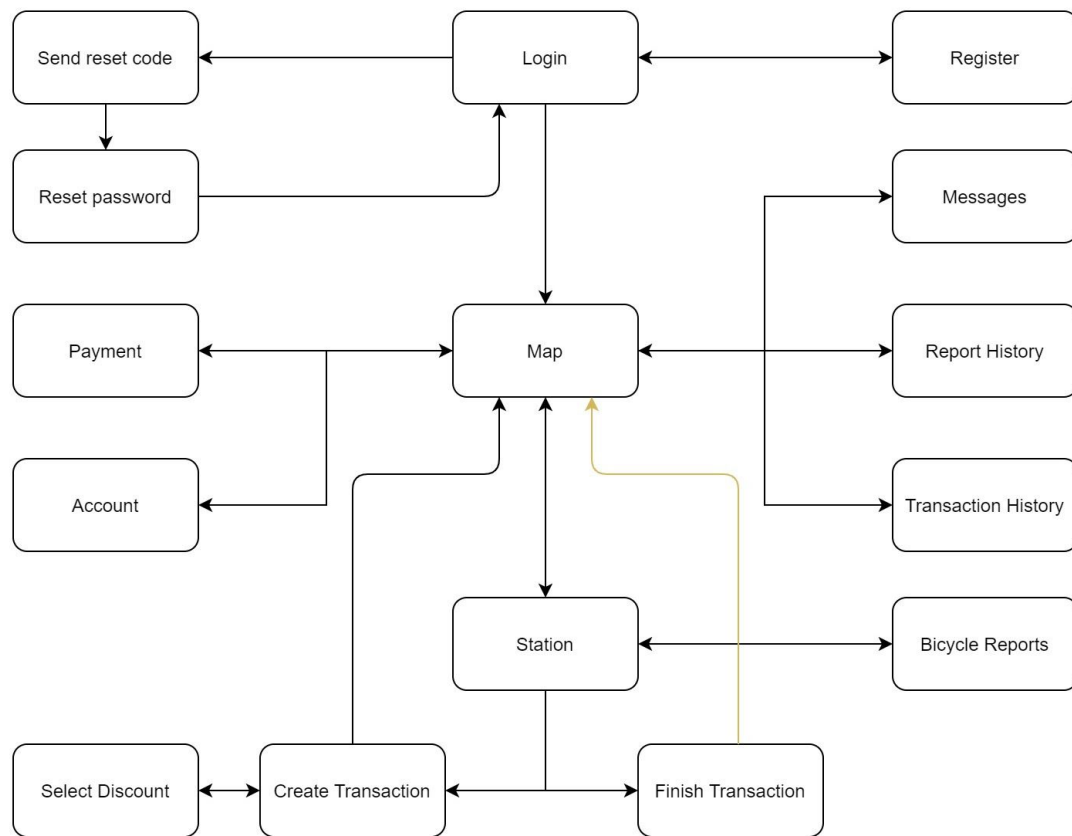
Aplicația are nevoie de două permisiuni de la utilizator. Prima este cea de “`ACCESS_FINE_LOCATION`”, ce va fi folosită pentru a determina locația curentă a utilizatorului. A doua permisiune cerută este de “`INTERNET`”, ce va fi folosită pentru a realiza cereri către aplicația server.

#### **c) Paginile aplicației**

Aplicația are cincisprezece pagini. Pentru fiecare pagină a aplicației există o clasa ce extinde clasa “`AppCompatActivity`” și un fișier XML în *folder*-ul de resurse.

În majoritatea paginilor există butoane și casete de text. Aceste butoane în general preiau informația din casete, le verifică și apoi apelează API-ul de la aplicația server. Folosind informația primită de la server pot fi actualizate liste, casete de text sau poate fi schimbată pagină aplicației.

Pentru pagina cu harta se folosește un API Google Maps. La pornirea paginii, harta este preluată și se face *zoom in* pe locația utilizatorului. Pentru această se desenează un marcator, precum și pentru fiecare stație din orașul selectat.



În diagrama de mai sus se poate vedea fluxul dintre pagini. Mai multe detalii despre pagini și funcționalitățile lor sunt valabile în următorul capitol, cel pentru ghidul de instalare și modul de utilizare.

#### d) Clasele de utilitate folosite

Prima clasă de utilitate folosită este “ActivityStarter”. Această clasă are metode statice de deschidere a paginilor aplicației. A fost folosită pentru a evita codul duplicat.

A doua clasă de utilitate folosită este “ApiCaller”. Această clasă se ocupă de crearea și trimiterea cererilor către server. Are ca și clasă de bază pe “AsyncTask” astfel încât procesul nu poate rula pe firul principal de execuție. Primește ca parametru o serie de String-uri. Primul String reprezintă tipul de cerere, al doilea adresa cererii, al treilea jetonul de autentificare care poate fi *null* iar al patrulea reprezintă informația ce va fi trimisă. Această informație suplimentară poate fi *null*, un JSON pentru cererile de tipul POST și PUT iar pentru cererile de tipul GET parametrii ce vor fi concatenați la adresa.

Pentru a realiza această cerere, folosesc clasele URL și HttpURLConnection. Pentru conexiunea de tipul HttpURLConnection sunt setate proprietățile corespunzătoare cererii, apoi după trimiterea ei sunt primite un JSON și alte informații cum ar fi codul de răspuns. Folosind aceste informații se va crea un obiect de tipul “ApiResponse”, care este a treia clasă de utilitate, și va fi returnată.

Următoarea clasă de utilitate este “DistanceCalculator”. Primind un String ce reprezintă coordonatele unei locații, calculează dacă utilizatorul poate fi considerat în apropierea sa sau nu.

O altă clasă de utilitate este “JsonConverter”, care are o metodă ce transformă un obiect de orice clasă într-un String JSON. Pentru a face acest lucru m-am folosit de clasa ObjectWriter din librăria Jackson.

Ultima clasă de utilitate este “NotificationWorker”. Aceasta are ca și clasă de bază pe “Worker”. Astfel, orice cod ce rulează în această clasă o va face pe un alt fir de execuție, și va continua să ruleze chiar și dacă aplicația este oprită. Un obiect al acestei clase va fi creat de fiecare dată când un utilizator creează o nouă tranzacție. În funcție de timpul de sosire, vor fi create patru notificări. Două din acestea vor fi înainte ca timpul de sosire să treacă, iar două după.

#### **e) Resursele aplicației**

Există câte un fișier de tip XML pentru fiecare pagină a aplicației. Acestea sunt relativ simple, având în principal text, butoane și liste. Excepția este pagina hărții, care se folosește de un *fragment*.

Pentru liste am creat adaptoare de listă, având ca și clasă de bază pe ArrayAdapter. Pentru fiecare astfel de listă există câte un fișier XML, la care am folosit LinearLayout-uri ca informațiile unui element din listă să fie pe câte o linie.

Meniurile utilizate au și ele câte un fișier XML, fiecare element din lista meniului fiind reprezentat de un *item*.

Internaționalizarea aplicației constă în utilizarea fișierelor de String-uri din *folder*-ul de resurse. Folosind metode de obținere a resurselor utilizarea acestui fișier este simplă. Localizarea

poate fi implementată cu ușurință, prin crearea a noi fișiere similare cu “strings.xml” și utilizarea fișierului corespunzător locației utilizatorului.

CSS-ul aplicației este unul simplu. Culorile folosite se află în fișierul “colors.xml” și tema folosită pentru fiecare pagină a aplicației se află în fișierul “styles.xml”.

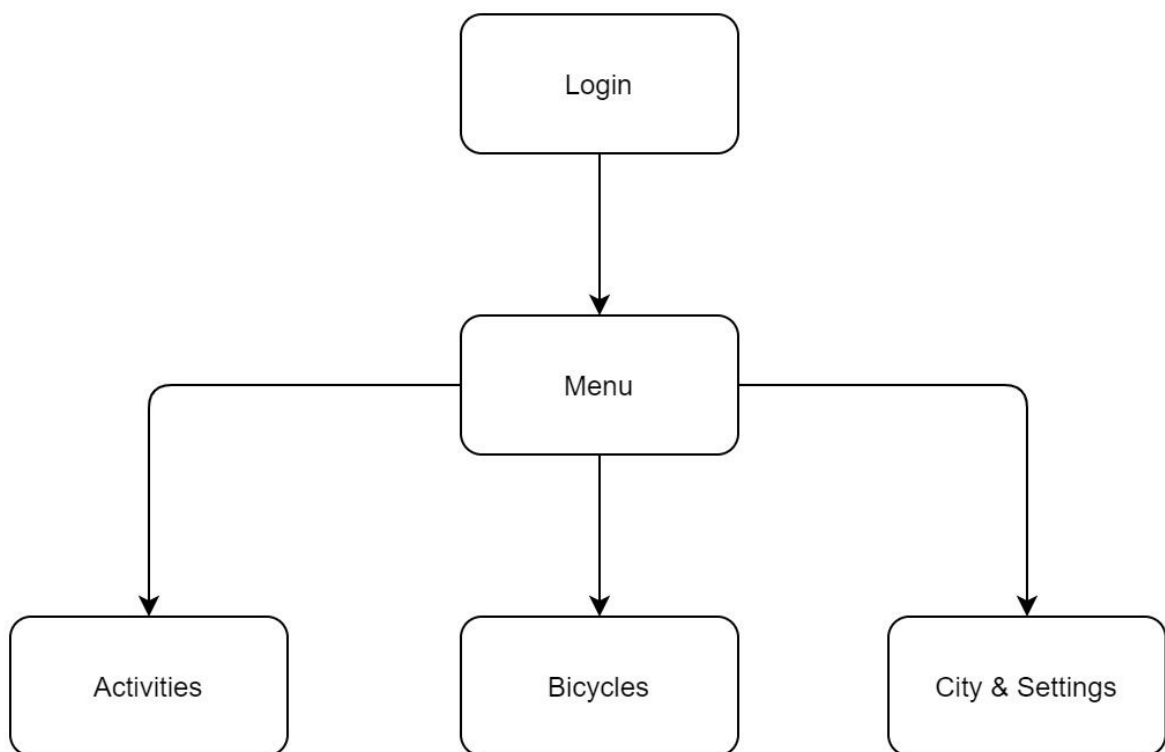
## 4. Aplicația de administrare

### a) Tehnologii utilizate

Pentru interfața grafică de utilizator am folosit JavaFX. Pentru a converti obiectele în JSON și invers am folosit librăria Jackson.

### b) Paginile aplicației

Aplicația are patru pagini. Prima pagină se ocupă de autentificarea administratorului. După ce se autentifică, poate accesa oricare dintre celelalte trei pagini. Ele sunt simple, având mai multe casete de text și butoane care apelează API-ul *server*-ului.



În diagrama de mai sus se poate vedea fluxul dintre pagini. Mai multe detalii despre pagini și funcționalitățile lor sunt valabile în următorul capitol, cel pentru ghidul de instalare și modul de utilizare.

### c) Detalii tehnice

Am folosit *design pattern*-ul de MVC. Modelul este reprezentat de aplicația server cu care comunică această aplicație prin API, *View*-ul de fișierele cu extensia *fxml* și *Controller*-ul de fișierele cu sufixul “Controller”.

Partea de apelare a API-ului este aproape identică cu cea din aplicația de Android. Singura diferență este faptul că nu mai este nici o clasă de baza pentru clasa “ApiCaller”.

Pentru fiecare pagină există o clasă Controller precum și un fișier cu extensia *fxml*. De asemenea, mai există un Controller și fișier cu extensia *fxml* pentru meniul principal. În acest meniu principal se ajunge după autentificare. Va avea un *BorderPane* unde vor fi încărcate paginile selectate din bara de meniu.



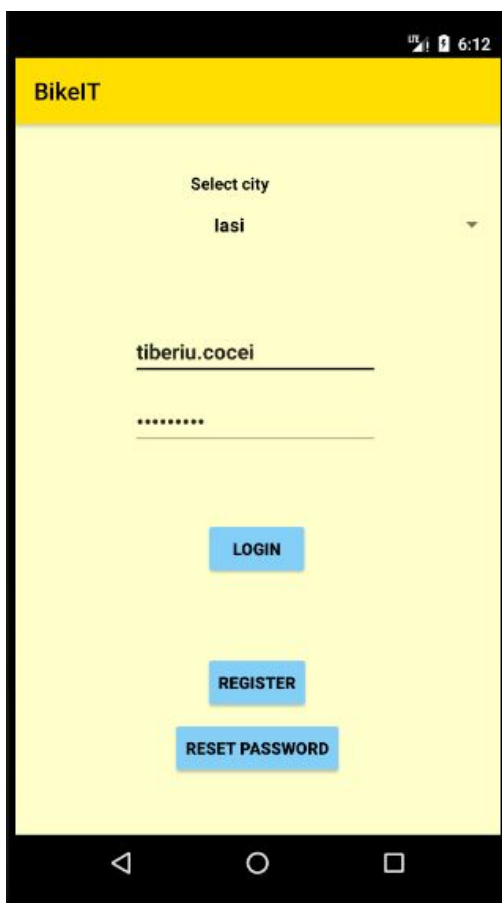
# IV. Manual de utilizare

## 1. Modul de instalare

Pașii ce trebuie urmați pentru a instala și rula toate cele trei aplicații sunt:

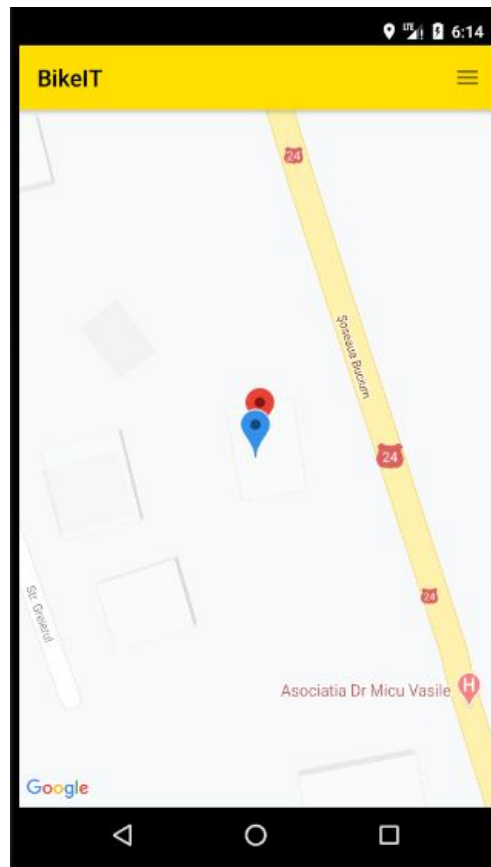
- 1) Se instalează PostgreSQL, pgAdmin4, Android Studio și un IDE ce poate folosi Spring Boot
- 2) Se instalează și configurează versiunea 1.8 de Java și 3.6 de Python
- 3) Se creează baza de date folosind pgAdmin
- 4) Din *folder*-ul “Database” se rulează codul din fișierul “Tables” urmat de “Indexes”, “Rules”, “Functions”, “Triggers”, “Casts” și “Operators”
- 5) Numele bazei de date, de utilizator și parola se scriu în fișierul “database.ini”
- 6) Se rulează *script*-ul de populare a bazei de date “data\_script.py”
- 7) Se lansează aplicația server din *folder*-ul “WebAPI” folosind IDE-ul instalat
- 8) Folosind librăria Jasypt, se criptează numele bazei de date, de utilizator și parola apoi se trec în fișierul “application.properties”
- 9) În același fișier se va face același lucru pentru numele de utilizator și parola contului de Gmail folosit pentru a trimite e-mailuri utilizatorilor
- 10) Se rulează aplicația server
- 11) Se lansează aplicația de Android din *folder*-ul “Android” folosind Android Studio
- 12) Se instalează un emulator din Android Studio
- 13) În fișierul “strings.xml” se modifică porturile pentru API dacă diferă
- 14) Se rulează aplicația de Android
- 15) Se lansează aplicația de administrare din *folder*-ul “Desktop” folosind IDE-ul instalat
- 16) Se modifică porturile folosite în *Controller*-e dacă diferă
- 17) Se rulează aplicația de administrare

## 2. Modul de utilizare a aplicației Android



Prima pagină a aplicației este cea de autentificare, care se poate vedea în imaginea de mai sus. Utilizatorul trebuie să aleagă orașul și să își introducă datele de autentificare. Dacă autentificarea este realizată cu succes, va fi trimis la pagina cu harta orașului.

De asemenea, utilizatorul poate să navigheze la pagina de înregistrare sau la cele de resetare a parolei. Înregistrarea este una standard, iar pentru resetarea parolei există două pagini. În prima pagină utilizatorul își introduce adresa de e-mail, unde va primi codul de resetare și va fi trimis la a doua pagină. În această utilizatorul trebuie să introducă codul precum și parola nouă.



În imaginea de mai sus se poate vedea pagina cu harta. În aceasta vor fi desenate marcatoare roșii pentru fiecare stație și un marcator albastru pentru locația curentă a utilizatorului. Utilizatorul poate să apese pe oricare din marcatoarele roșii, astfel navigând la pagina acelei stații.

Apăsând pe iconiță din partea de dreapta sus a ecranului, lista meniului va apărea. Există cinci pagini ce pot fi accesate din aceasta.

Prima pagină este cea a contului său, de unde își poate modifica parola sau adresa de e-mail. De asemenea, își poate vedea detaliile tranzacției curente, dacă există. Altă pagină ce poate fi accesată folosind meniul este cea de mesaje, unde utilizatorul își va vedea toate mesajele primite de la administratori.

În a treia pagină unde utilizatorul își poate introduce informațiile cardului de credit cu care va plăti. Ultimele două pagini din acest meniu sunt cele de vizualizare a tranzacțiilor și rapoartelor făcute în trecut.

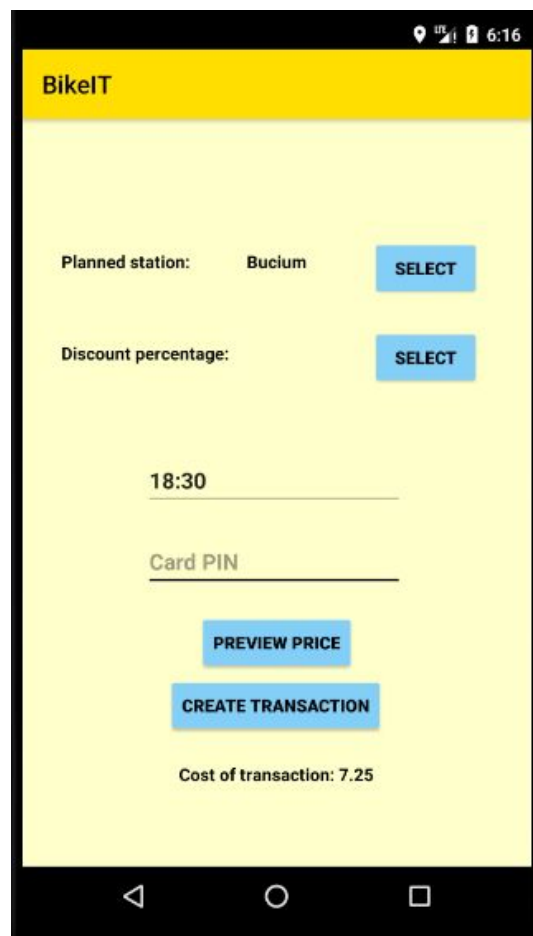


În imaginea de mai sus se poate vedea pagina stației. Lista conține toate bicicletele care sunt deja sau urmează să ajungă în acea stație. Pentru biciclete care nu sunt încă în stație, se va putea vedea o aproximare pentru timpul de sosire. Starea fiecărei biciclete va fi indicată în dreptul textului “Status:”. Pentru fiecare bicicletă din lista vor exista două butoane, și anume de selectare și de rapoarte.

Butonul de selectare a unei biciclete va fi valabil doar dacă locația curentă a utilizatorului este lângă stație, dacă utilizatorul nu are o tranzacție activă și dacă bicicleta are starea de stație. Altfel, butonul nu va putea fi apăsat. Dacă condițiile enunțate anterior sunt întâlnite, apăsând pe buton utilizatorul va fi trimis la pagina creării unei noi tranzacții.

Butonul de rapoarte a unei biciclete poate fi apăsat oricând. Pagina de rapoarte va fi deschisă, unde utilizatorul poate să vadă toate rapoartele existente ale bicicletei. Dacă locația utilizatorului este lângă stație, va putea crea un nou raport.

Butonul de finalizare tranzacție poate fi apăsat dacă utilizatorul are o tranzacție activă, adică are o bicicletă închiriată, și dacă locația sa curentă este lângă stație. Apăsând butonul, utilizatorul va fi trimis la o altă pagină a aplicației, unde va vedea informații despre tranzacția activă și un nou buton ce va confirma decizia.

The image shows a mobile application interface for 'BikeIT'. At the top, there's a yellow header with the text 'BikeIT'. Below this, the screen has a light yellow background. There are two rows of input fields: 'Planned station:' with the value 'Bucium' and a blue 'SELECT' button; and 'Discount percentage:' with a blue 'SELECT' button. Below these, there's a time input field showing '18:30' and a 'Card PIN' input field. At the bottom, there are two blue buttons: 'PREVIEW PRICE' and 'CREATE TRANSACTION'. Below the 'CREATE TRANSACTION' button, the text 'Cost of transaction: 7.25' is displayed. The top status bar shows 'LTE', signal strength, and the time '6:16'. The bottom navigation bar shows standard Android icons: back, home, and recent apps.

În imaginea de mai sus se poate vedea pagina de creare a unei tranzacții. În această pagină utilizatorul își va selecta stația și timpul de sosire și introduce codul PIN pentru cardul de credit salvat. După aceea poate confirma crearea tranzacției și va fi redirecționat la pagina cu harta.

Apăsând pe butonul de previzualizare, utilizatorul poate vedea costul tranzacției dacă este introdusă stația și ora de sosire.

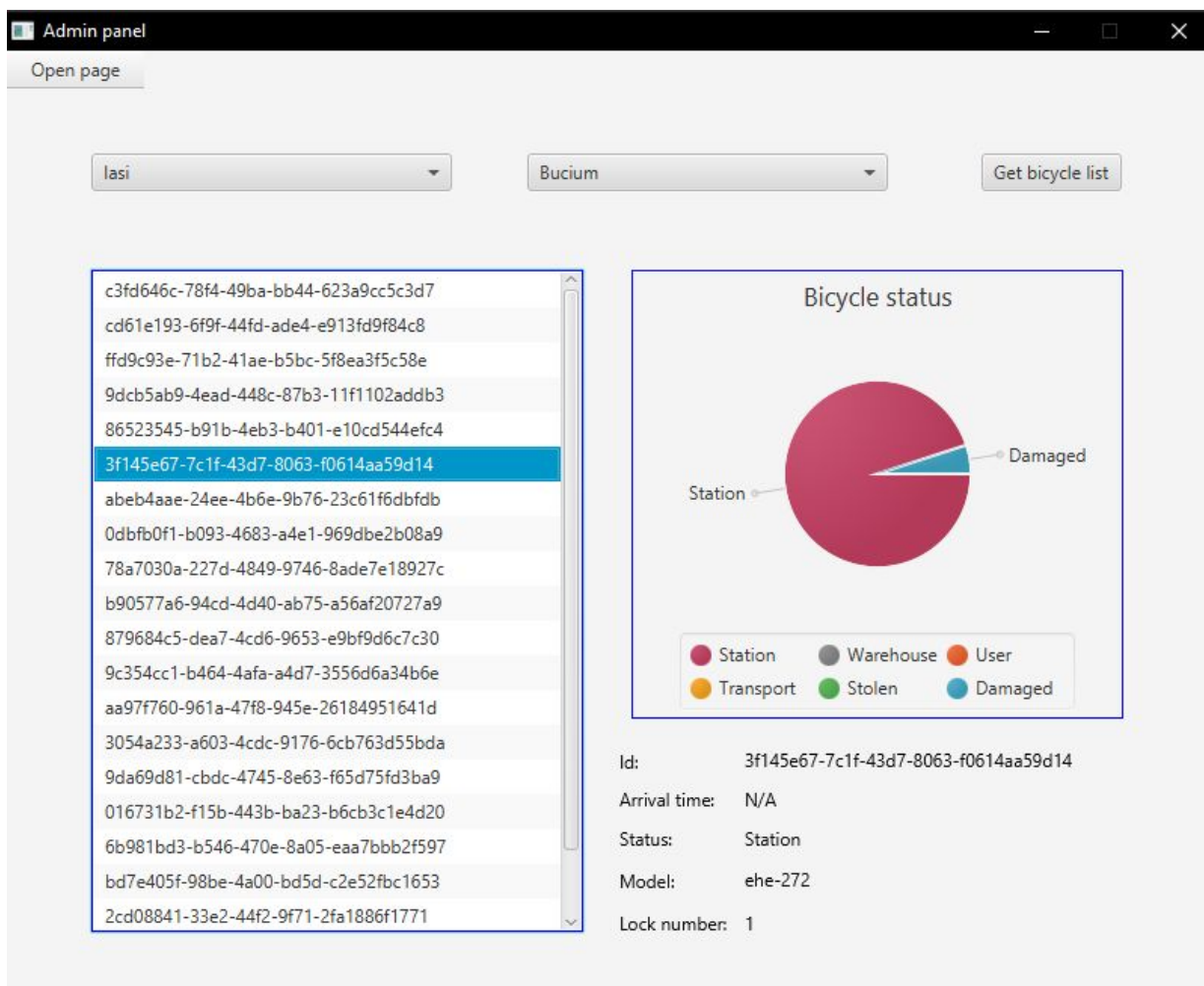
Când utilizatorul apasă pe butonul de alegere stație, va fi redirecționat la pagina cu harta. Apăsarea pe marcatorul unei stații o va selecta ca și stație de sosire, întorcându-se apoi la pagina cu tranzacția.

Apăsând pe butonul de reduceri, utilizatorul va vedea lista cu reduceri pentru acea stație. Selectând o reducere va înlocui stația de sosire cu cea specificată de reducere. Dacă utilizatorul alege o altă stație, reducerea va dispărea.

După crearea tranzacției utilizatorul va primi notificări cu câteva minute înainte să îi expire timpul de sosire specificat și cu câteva minute după ce va expira.

### 3. Modul de utilizare a aplicației de administrare

Prima pagină este cea de autentificare. Aceasta este una simplă, având numai două casete de text și un buton. După o autentificare de succes, administratorul va fi redirecționat la pagina meniului. De aici va putea deschide una din cele trei pagini existente.



În imaginea de mai sus se poate vedea pagina ce conține detalii legate de biciclete. Graficul proporțiilor va reprezenta procentajul fiecărei stări de bicicletă pentru acea stație. În lista vor apărea toate bicicletele stației. Apăsând pe un element din listă îi va afișa detaliile în partea dreaptă a listei.

A doua pagină este cea de activități. Selectând un oraș și o stație din acel oraș, se vor încărca activitățile din trecut și cele prevăzute pentru acea stație. Aici se folosește paginare,

administratorul apăsând pe butoane pentru a afișa pagina următoare sau anterioară. Din nou, apăsând pe un element din listă îi va afișa detaliile.

Ultima pagină este cea de setări. Selectând un oraș, setările sale vor fi afișate. Administratorul poate să le modifice, apăsând pe un buton după editarea câmpurilor.



# Concluzie

În concluzie, aplicația utilizatorului este completă și funcțională iar cea a administratorului este numai parțial implementată, putând fi considerată o demonstrație a întregului. De asemenea, aplicația *server* are toate metodele necesare celorlalte aplicații din *Controller*-e implementate. Chiar și așa, lucrarea de licență poate fi îmbunătățită prin mai multe moduri.

În primul rând, securitatea aplicației *server* poate fi îmbunătățită folosind protocolul HTTPS. Pentru a face acest lucru ar trebui obținut un certificat autentic, care să nu fie *self signed*, sau configurarea aplicațiilor de Android și de administrare pentru a le accepta. Însă, acest lucru poate fi periculos dacă această aplicație ar fi folosită de utilizatori reali.

În al doilea rând, algoritmul de predicție ar putea fi refăcut. Pentru asta ar putea fi folosită o rețea neuronală pe iterația curentă ce se folosește de stații. Dacă ar fi schimbată aplicația pentru a se folosi de biciclete poziționate oriunde, renunțând în întregime la stații, ar putea fi folosit un algoritm de clusterizare. Astfel, s-ar obține rezultate mai bune.

În al treilea rând, algoritmul de partajare al bicicletelor folosind reduceri și transporturi ar putea fi îmbunătățit. Pentru transporturi se folosește câte un șofer pentru fiecare cuplu de stație cu biciclete în minus și biciclete în plus. Calculând drumul cel mai eficient dintre punctul de start și de final și maximizând numărul de biciclete ce ar putea fi luat din mai mult de o stație, transporturile ar fi realizate într-un mod mult mai eficient.

De asemenea, aplicația de administrare ar putea fi implementată în întregime și împreună cu aplicația de Android ar putea avea CSS-ul îmbunătățit.

# Bibliografie

- Documentație Swagger pentru aplicația *server*,  
<https://www.baeldung.com/swagger-2-documentation-for-spring-rest-api>
- Criptare si decriptare folosind librăria Jasypt, <https://www.baeldung.com/jasypt> și  
<https://www.baeldung.com/spring-boot-jasypt>
- Trimiterea E-mailurilor din aplicația server, <https://www.baeldung.com/spring-email>
- Validarea DTO-urilor, <https://www.baeldung.com/javax-validation>
- *String hashing*, <https://www.baeldung.com/java-password-hashing>
- Autentificare prin jeton,  
<https://www.javadevjournal.com/spring/securing-a-restful-web-service-with-spring-security> și  
<https://docs.spring.io/spring-security/site/docs/current/reference/html5/>
- Metode asincrone, <https://www.baeldung.com/spring-async>
- Metode ce rulează automat, <https://www.baeldung.com/spring-scheduled-tasks>
- Protocolul HTTPS, <https://www.thomasvitale.com/https-spring-boot-ssl-certificate/>
- Regresie liniară, <http://statisticasociala.tripod.com/regresie.htm> și  
<https://www.codesansar.com/numerical-methods/linear-regression-method-algorithm.htm>
- Librăria Jackson, <https://github.com/FasterXML/jackson>
- Android Worker, <https://developer.android.com/reference/androidx/work/Worker> și  
<https://developer.android.com/topic/libraries/architecture/workmanager>