

UNIVERSITATEA "ALEXANDRU-IOAN CUZA" DIN IAȘI

**FACULTATEA DE INFORMATICĂ**



LUCRARE DE LICENȚĂ

**Aplicație de utilizare în comun a vehiculelor  
electrice pentru mobilitatea urbană**

propusă de

**Emanuel-Așer Cobaschi**

**Sesiunea: iulie, 2024**

Coordonator științific

**Lector Dr. Frăsinaru Cristian**

UNIVERSITATEA "ALEXANDRU-IOAN CUZA" DIN IAȘI

**FACULTATEA DE INFORMATICĂ**

**Aplicație de utilizare în comun a  
vehiculelor electrice pentru mobilitatea  
urbană**

**Emanuel-Așer Cobaschi**

**Sesiunea: iulie, 2024**

Coordonator științific

**Lector Dr. Frăsinaru Cristian**

Avizat,  
Îndrumător lucrare de licență,  
Lector Dr. Frăsinaru Cristian.

Data: ..... Semnătura: .....

### **Declarație privind originalitatea conținutului lucrării de licență**

Subsemnatul **Cobaschi Emanuel-Așer** domiciliat în **România, jud. Botoșani, mun. Botoșani, aleea Constantin Iordăchescu, nr. 6**, născut la data de **28 iulie 2002**, identificat prin CNP **5020728070080**, absolvent al Facultății de informatică, **Facultatea de informatică** specializarea **informatică**, promoția 2024, declar pe propria răspundere cunoscând consecințele falsului în declarații în sensul art. 326 din Noul Cod Penal și dispozițiile Legii Educației Naționale nr. 1/2011 art. 143 al. 4 și 5 referitoare la plagiat, că lucrarea de licență cu titlul **Aplicație de utilizare în comun a vehiculelor electrice pentru mobilitatea urbană** elaborată sub îndrumarea domnului **Lector Dr. Frăsinaru Cristian**, pe care urmează să o susțin în fața comisiei este originală, îmi aparține și îmi asum conținutul său în întregime.

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

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

Data: .....

Semnătura: .....

### **Declarație de consimțământ**

Prin prezenta declar că sunt de acord ca lucrarea de licență cu titlul **Aplicație de utilizare în comun a vehiculelor electrice pentru mobilitatea urbană**, 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" din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Absolvent **Emanuel-Așer Cobaschi**

Data: .....

Semnătura: .....

# Cuprins

<b>Introducere</b>	<b>2</b>
0.1    Prezentarea problemei . . . . .	2
0.2    Motivație . . . . .	2
0.3    Aplicații similare . . . . .	3
0.4    Diferențe și elemnte de orginalitate . . . . .	3
0.5    Tehonologii utilizate . . . . .	4
<b>1    Specificații funcționale</b>	<b>5</b>
1.1    Descrierea problemei . . . . .	5
1.2    Fluxuri de lucru . . . . .	8
<b>2    Arhitectura Sistemului</b>	<b>12</b>
2.1    Schema bazei de date . . . . .	12
2.2    Explicația tabelelor bazei de date . . . . .	13
2.3    Arhitectura sistemului . . . . .	17
<b>3    Implementare și testare</b>	<b>23</b>
3.1    Baza de date . . . . .	23
3.2    Aplicația server . . . . .	26
3.3    Aplicația ReactJs pentru <i>Frontend</i> . . . . .	36
<b>4    Manual de instalare și utilizare</b>	<b>41</b>
<b>Concluzii</b>	<b>42</b>
<b>Bibliografie</b>	<b>43</b>

# Introducere

## 0.1 Prezentarea problemei

Problema este partajarea autovehiculelor electrice, în scopul asigurării accesului la un mijloc de transport rapid, confortabil și ecologic. Vehiculele sunt repartizate prin oraș, în parcuri special amenajate. De acolo, utilizatorii le pot închiria și se pot deplasa spre alte zone ale orașului. Este necesar ca mașina să fie lăsată într-o altă parcare, din motive ce țin de securitate și de legislație.

Un aspect principal al problemei îl reprezintă rebalansarea parcurilor, în sensul asigurării unui număr optim de mașini în fiecare parcare, proporțional cu importanța parcurii respective. Această importanță este măsurată prin frecvența cu care se închiriază o mașină din parcare respectivă.

Aplicația folosită de utilizatori trebuie să primească datele de la un server, motiv pentru care este folosit un API web. În cadrul acestei aplicații, există un modul dedicat administratorilor, de unde se realizează și operația de rebalansare a parcurilor.

## 0.2 Motivație

Am ales această tematică pentru lucrarea de licență deoarece consider că este o aplicație aplicabilă în viața de zi cu zi.

În primul rând, beneficiile asupra utilizatorului sunt multiple: el nu mai trebuie să dețină o mașină, ceea ce înseamnă resurse financiare salvate (RCA, taxe, impozite, costuri de întreținere), dar poate beneficia în continuare de serviciile uneia. Mai mult, economisește timpul necesar căutării unui loc de parcare, deoarece locul său în parcare de destinație va fi rezervat în momentul închirierii.

În al doilea rând, beneficiile aduse societății în ansamblu sunt însemnate. Aici vorbim despre eliminarea noxelor de dioxid de carbon, aplicația utilizând vehicule

electrice autonome. De asemenea, parcurile publice din oraș sunt gestionate într-un mod mult mai eficient, ceea ce ajută pe toată lumea.

Un alt motiv pentru care am ales această temă pentru lucrarea de licență constă în utilizarea unor tehnologii noi, comparativ cu proiectele anterioare de la facultate. Întrucât am un interes personal pentru dezvoltarea aplicațiilor Web folosind tehnologiile Java și React, consider că elaborarea acestei aplicații mă va ajuta să îmi îmbunătățesc cunoștințele în acest domeniu.

### 0.3 Aplicații similare

Cu toții știm că sistemul de închiriere a mijloacelor de transport, fie că vorbim despre biciclete, trotinete sau mașini este în continuă creștere. La nivelul închirierii de mașini, principalele opțiuni de pe piață se împart în două categorii.

În primul rând există deja bine-cunoscutele aplicații de tip Uber și Bolt, prezente în multe țări, printre care și România. Acestea reprezintă varianta modernă a clasicelor taxiuri. Utilizatorul plasează o comandă, iar transportul este realizat de către un șofer, cu propria lui mașină.

Apoi, există platformele specializate pe închiriere de bunuri sau servicii, de tipul OLX, de unde un utilizator poate închiria o mașină pentru o anumită perioadă.

Mai există o aplicație specializată pe transportul pe distanțe mai lungi, numită BlaBlaCar, acolo unde utilizatorii care merg într-o călătorie cu mașina anunță acest lucru, cu scopul găsirii celor care caută transport pe aceeași rută.

O aplicație mai asemănătoare cu cea realizată de mine la nivelul conceptului este Lime, specializată însă în partajarea trotinetelor.

### 0.4 Diferențe și elemente de originalitate

Aplicația propusă de mine prezintă câteva diferențe față de spectrul de aplicații enunțat anterior.

În primul rând nicio aplicație din cele de mai sus sau asemănătoare cu acestea nu au în evidența lor și un sistem de parcare al mașinilor (sau al resurselor folosite). Se poate pierde mult timp cu căutarea unui loc liber sau există riscul furtului sau vandalizării. Aplicația mea propune un sistem în care locul în parcare de destinație

este rezervat în momentul închirierii unei mașini, în felul acesta evitându-se situații neplăcute enunțate anterior.

De asemenea, un alt element de originalitate constă în implementarea unui mecanism de rebalansare a parcărilor. Acesta presupune determinarea la un moment dat a parcărilor unde este nevoie de mai multe mașini, dar și a celor unde sunt prea multe.

Mecanismul are patru etape:

- determinarea parcărilor implicate cel mai des în curse, pe baza unui istoric
- stabilirea unei ierarhii în sensul acesta
- determinarea numărului maxim de mașini ce pot fi mutate
- relocarea în alte parcuri a mașinilor ce pot fi mutate, proporțional cu importanța parcărilor respective.

În felul acesta se asigură o utilizare cât mai eficientă a resurselor disponibile.

Nu în ultimul rând, este evident faptul că aplicația propusă de mine nu necesită implicarea unei alte persoane (fie ea șofer). Iar faptul acesta, pus alături de utilizarea unor vehicule electrice, conduce la costuri mai reduse pentru utilizator.

## 0.5 Tehnologii utilizate

Pentru realizarea acestei aplicații am folosit mai multe tehnologii. Acestea sunt:

- Java, pentru partea de BackEnd. Am ales acest limbaj deoarece permite utilizarea framework-ului Spring Boot. Am folosit Spring Boot pentru a realiza serviciile web. De asemenea am folosit librăria Hibernate pentru a avea un acces facil asupra bazei de date. Cu aceste tehnologii am putut obține și salva ușor datele, am putut crea endpoint-uri, am putut testa codul.
- React, cu JavaScript pentru partea de FrontEnd.
- API Google Maps pentru a prelua harta orașului și a reprezenta pe hartă pozițiile parcarilor.
- PostgreSQL pentru baza de date.
- Python pentru un script de populare a bazei de date.



# Capitolul 1

## Specificații funcționale

### 1.1 Descrierea problemei

Mașinile vor fi amplasate în locuri special amenajate, numite parcuri, care sunt distribuite în oraș. Fiecare mașină este dotată cu un sistem *hardware* ce permite deblocarea mașinii numai prin scanarea unui cod QR, generat în urma rezervării. Blocarea mașinii se face în mod automat după finalizarea cursei și validarea locului de parcare.

În scopul realizării lucrării de licență, se vor omite părțile *software* și *hardware* ce țin de mașini. Evitând o abordare cu microcontrolere, ce ar fi un lucru greu realizabil, am omis verificările ce țin de generarea și validarea codurilor QR.

De asemenea, deoarece nu există resurse umane care să se ocupe de mutarea efectivă a mașinilor din depou în vreo stație, sau dintr-o stație în alta, am considerat că redistribuirea vehiculelor, în scopul rebalansării parcurilor, se realizează automat de către server.

Problema constă în realizarea a doua aplicații. Prima aplicație este cea de *server* și va fi folosită de a doua. A doua aplicație este una Web și are două module, unul destinat utilizatorilor și unul destinat administratorilor.

*Serverul* trebuie să poată fi accesat de către a doua aplicație, care este Web, deci trebuie să aibă servicii web.

**Cerințele serverului sunt următoarele:**

1. Să se poată conecta la baza de date de unde să poată obține, introduce sau actualiza date.
2. Să prezinte *endpoint-uri* nesecurizate, ce vor fi utilizate pentru înregistrare, autentificare, verificarea adresei de email și resetarea parolei.

3. Să prezinte *endpoint-uri* securizate, accesul la ele fiind validat cu ajutorul unui jeton de autentificare. Acest jeton va permite diferențierea între administratori și utilizatori. În felul acesta, o persoană neautentificată nu va putea accesa *endpoint-uri* securizate iar un utilizator nu va putea accesa *endpoint-uri* ale unui administrator.
4. Să poată trimite e-mailuri.
5. Să cripteze datele de conectare la baza de date și la contul de email.
6. Să folosească o funcție hash la autentificare.
7. Să aibă teste ce atestă corectitudinea codului.
8. Să aibă *endpoint-urile* necesare pentru îndeplinirea cerințelor celeilalte aplicații.
9. Să implementeze un algoritm de ierarhizare a parcărilor în funcție de frecvența cu care au fost selectate pentru curse, și un alt algoritm pentru redistribuirea mașinilor ce pot fi mutate, conform ierarhizării amintite. În felul acesta, se valorifică resursele într-un mod mult mai eficient.

Aplicația Web va facilita utilizarea vehiculelor disponibile, pentru utilizator. Totodată, va permite vizualizarea datelor, a statisticilor și declanșarea operației de rebalansare a parcărilor, pentru administrator.

**Cerințele modului de utilizator sunt următoarele:**

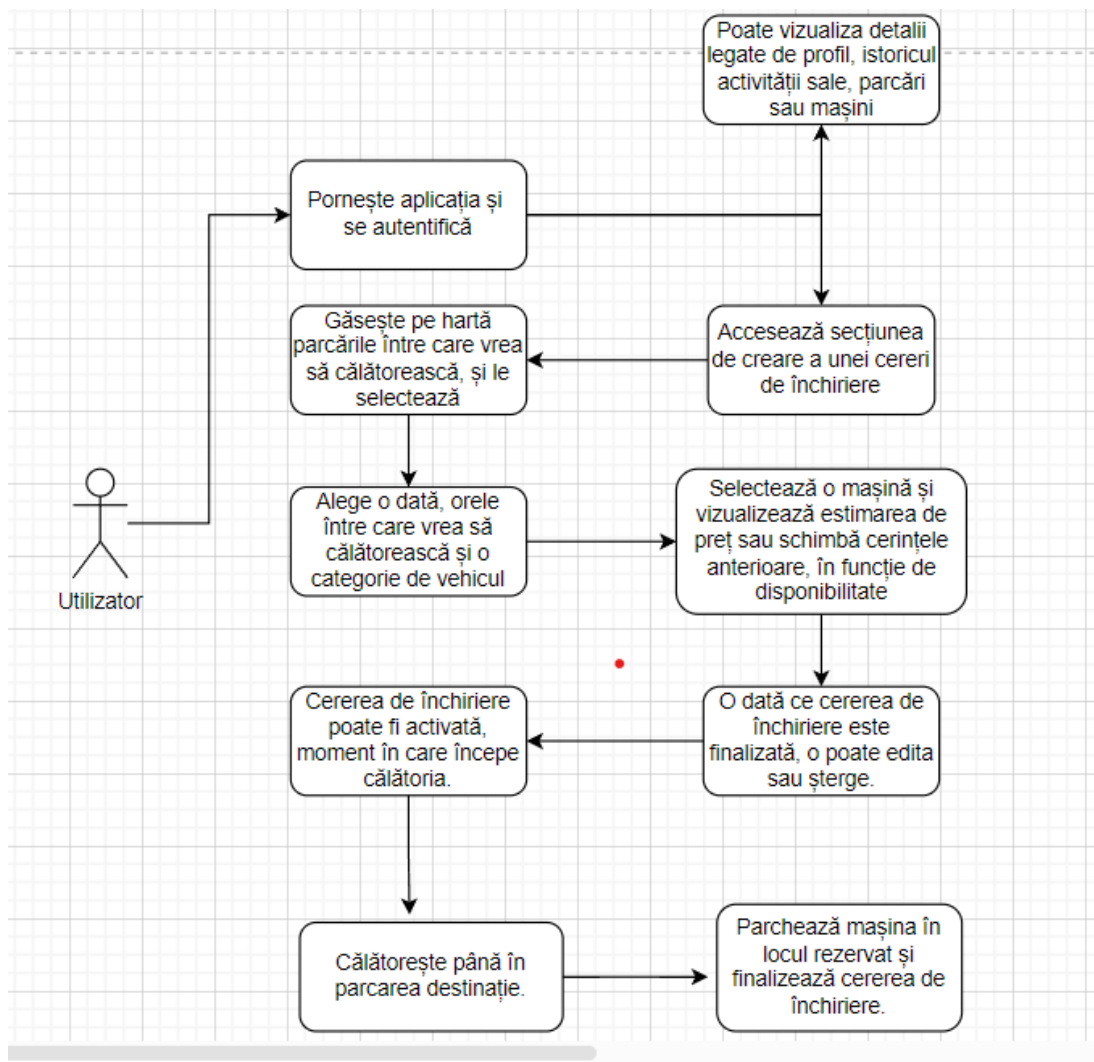
1. Utilizatorul poate să își creeze un cont nou.
2. El trebuie și poate ca înainte de autentificare, să își valideze adresa de e-mail pe baza unui cod de verificare trimis pe adresa introdusă la crearea contului.
3. Se poate autentifica.
4. Își poate vizualiza și edita profilul.
5. Poate vedea și selecta pe hartă parcurile.
6. Poate vizualiza lista parcărilor, iar pentru fiecare parcare poate vedea detaliile acesteia, cât și lista mașinilor aflate în acel moment în parcare respectivă.
7. Poate vizualiza lista mașinilor existente.
8. Poate vedea activitatea sa anterioară, însemnând în închirieri și curse efectuate. Poate vizualiza detalii legate de acestea.

9. Poate închiria o mașină, selectând o dată calendaristică, o parcare din care pleacă, o parcare în care poate ajunge, orele între care își dorește să închirieze mașina și un tip de vehicul pe care și-l dorește. Apoi poate selecta un vehicul dintre cele disponibile sau, dacă nu există vehicule disponibile la acel moment, poate schimba cerințele, până când găsește un vehicul disponibil.

**Cerințele modului de administrator sunt următoarele:**

1. La fel ca și un utilizator, să se poată autentifica și să își vizualizeze sau editeze profilul.
2. Să poată vizualiza, edita sau șterge vehiculele existente. Să poată vizualiza anumite statistici legate de acestea. Să poată adăuga în sistem vehicule noi.
3. Să poată vizualiza, edita sau șterge parcurile existente. Să poată vizualiza anumite statistici referitoare la activitatea fiecăreia. Să poată adăuga în sistem parcuri noi.
4. Să poată adăuga o mașină într-o parcare și să poată elimina una dintr-o parcare.
5. Să poată vizualiza depoul și parcurile din el.
6. Să poată vizualiza, edita sau șterge totii utilizatorii din sistem. Să poată vizualiza anumite statistici referitoare la activitatea lor.
7. Să poată genera rebalansarea parcurilor, pe baza datelor înregistrate într-o anumită perioadă, pe care o poate selecta.

## 1.2 Fluxuri de lucru



În diagrama de mai sus se poate observa fluxul normal de lucru al unui utilizator. Pot exista însă și scenarii secundare.

Când un utilizator pornește aplicația, are mai multe variante.

În cazul în care nu are un cont creat, el se poate înregistra. Etapa următoare necesară este să își confirme adresa de email, prin introducerea unui cod de verificare primit pe această adresă. Apoi, se poate autentifica în aplicație.

În cazul în care are un cont creat dar a uitat parola, o poate reseta, de asemenea prin introducerea unui cod de verificare trimis pe email.

Odată autentificat, utilizatorul este direcționat către o pagină principală, unde are la dispoziție un meniu cu operațiuni pe care le poate realiza, dar și o hartă unde poate vedea marcatoarele parcarilor existente. Prin apăsarea pe un marcator, el poate

vedea detaliile principale despre acea parcare, și un buton către lista vehiculelor din acea parcare.

Utilizatorul poate selecta din meniul disponibil în pagina principală o operațiune pe care vrea să o realizeze.

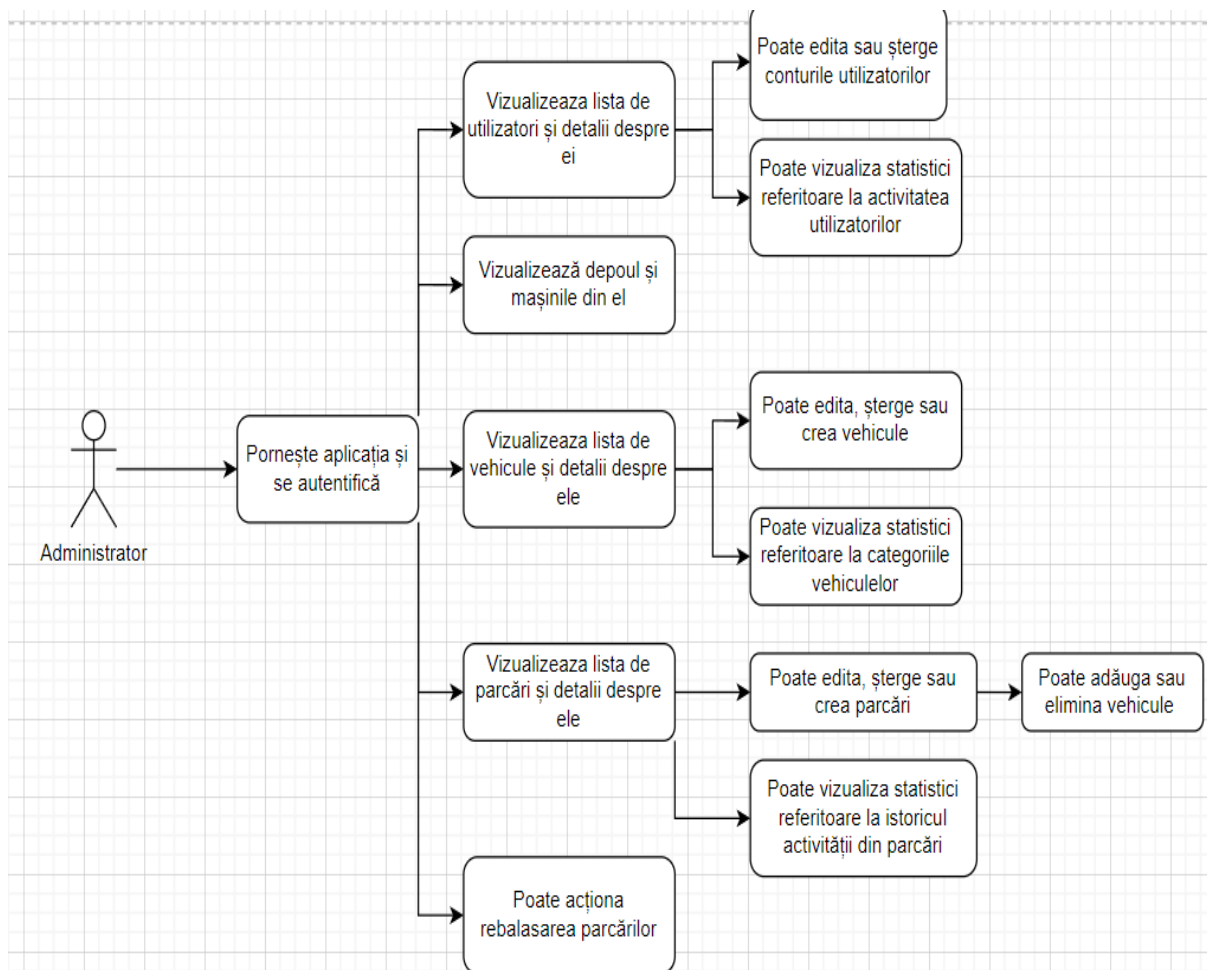
Apăsând pe submeniurile *View Profile* sau *Edit Profile* din meniul *Profile*, el va fi redirecționat către o pagină de unde va putea să își vizualizeze, respectiv editeze profilul.

Apăsând pe meniul *Inspect Parkings* sau *Inspect Vehicles* el va fi redirecționat către o pagină de unde va putea să vizualizeze parcările, respectiv vehiculele existente și detaliile acestora.

Apăsând pe meniul *My Requests* sau *My Rides* el va fi redirecționat către o pagină de unde va putea să vizualizeze cererile de închiriere, respectiv cursele efectuate și detaliile acestora, însemnând în parcările implicate, vehiculul implicat, orele de închiriere și data calendaristică.

Apăsând pe meniul *Create Request* el va fi redirecționat către o pagină de unde va putea crea o nouă cerere de închiriere. Aici, are posibilitatea să se folosească de preferințele sale. Dacă selectează acest lucru, vor fi setate ca implicate parcările și tipul de vehicul implicate cel mai des în cererile lui de închiriere. După selectarea tuturor detaliilor necesare, el poate crea o cerere de închiriere, vizualizând totodată un preț estimat al cursei. Cererea de închiriere poate fi editată ulterior sau ștearsă, până la momentul activării ei.

Din momentul activării cererii de închiriere, începe cursa și utilizatorul călătorește până în parcare de destinație, acolo urmând să închidă cererea de închiriere. În acel moment, este introdusă în istoricul său cursa respectivă.



În diagrama de mai sus este reprezentat fluxul obișnuit de lucru al unui administrator.

Odată autentificat în aplicație, un administrator este direcționat către o pagina principală, unde are un meniu cu acțiuni pe care le poate întreprinde și harta pe care sunt reprezentate marcatoarele parcarilor și a depoului.

Administratorul poate vizualiza sau edita propriul profil, sau alte altor utilizatori, inclusiv activitatea lor. Poate chiar șterge conturile utilizatorilor. De asemenea, poate vizualiza statistici referitoare la activitatea utilizatorilor într-un anumit interval de timp, selectat de el.

Poate vizualiza depoul și mașinile din el.

Poate vizualiza lista de vehicule și detaliile lor. Poate edita, șterge sau adăuga vehicule în sistem. De asemenea, poate vizualiza stastistici referitoare la categoriile vehiculelor.

Poate vizualiza lista de parări și detaliile lor. În parările existene, poate adăuga sau elimina vehicule atunci când acest lucru este posibil. Poate edita, șterge sau adăuga

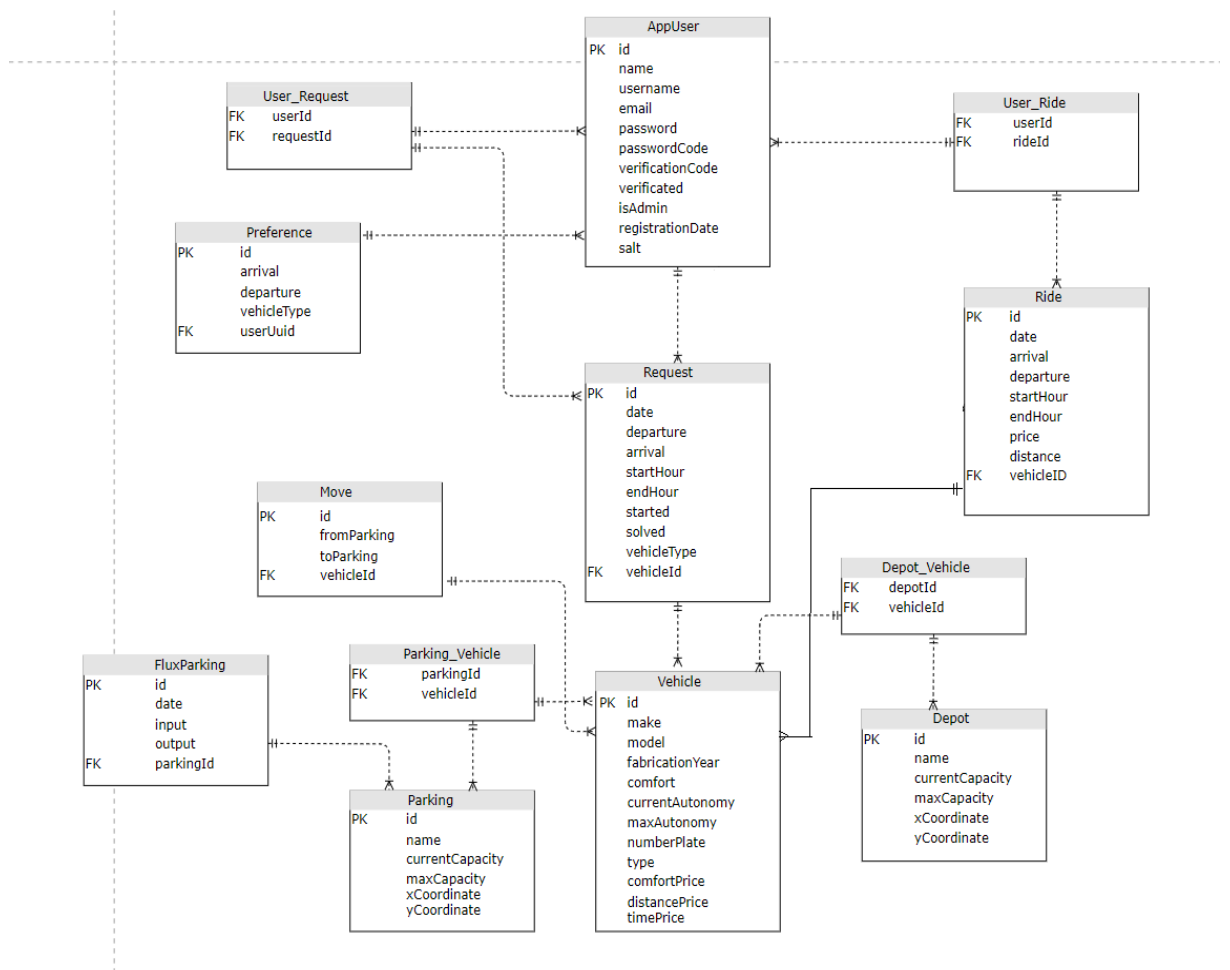
parcări în sistem, De asemenea, poate vizualiza statistici referitoare la istoricul activităților din parcări.

Poate declanșa rebalansarea parcarilor, bazată pe o perioadă de timp selectată de el.

# Capitolul 2

## Arhitectura Sistemului

### 2.1 Schema bazei de date





## 2.2 Explicația tabelelor bazei de date

### 1. AppUser

Tabela "AppUser" va conține lista de conturi ale utilizatorilor, inclusiv ale administratorilor. Cheia primară o constituie coloana "id".

Tabela va mai reține data de înregistrare, numele, username-ul, email-ul și parola utilizatorului. De menționat faptul că fiecare dintre coloanele "email" și "username" trebuie să respecte constângerea de unicitate. Parola este criptată folosind o funcție *hash*. În coloana "salt" este memorat *salt*-ul utilizatorului care se folosește la funcția *hash*.

Coloana "verificationCode" conține un cod de verificare care este trimis utilizatorului pe email în momentul înregistrării în aplicație. Coloana "verified" conține un *boolean* care atestă dacă email-ul utilizatorului a fost verificat sau nu.

Coloana "passwordCode" conține un cod de validare folosit la resetarea parolei care este trimis utilizatorului pe email în momentul cererii de resetare. Scopul lui este de a testa autenticitatea identității utilizatorului. Coloana "isAdmin" memorează 0 dacă contul corespunde unui utilizator obișnuit, respectiv 1 în cazul în care contul corespunde unui administrator.

### 2. Vehicle

Tabela "Vehicle" va conține lista de vehicule existente. Cheia primară o constituie coloana "id", iar coloana "numberPlate" trebuie să respecte constângerea de unicitate.

Sunt memorate în baza de date și anumite detalii tehnice precum marca, modelul și tipul mașinii, anul de fabricare, autonomia maximă și confortul vehiculului. De asemenea, sunt salvate și autonomia curentă a mașinii. Cele trei coloane destinate prețurilor sunt specifice fiecărui vehicul în parte și ajută în calculul prețului unei curse.

### 3. Parking

Tabela "Parking" va conține lista de parări existente. Cheia primară o constituie coloana "id". Coloana "name", consând în numele parării, trebuie să respecte constângerea de unicitate. La fel și perechea formată din coloanele "xCoordinate" și "yCoordinate", constând în coordonatele geografice ale parării.

Sunt memorate în baza de date și capacitatea curentă a parării cât și capacitatea maximă.

#### 4. **Parking\_Vehicle**

Tabela "Parking\_Vehicle" conține repartizarea vehiculelor în parări.

O intrare în tabelă conține două coloane, care sunt chei străine: "parkingId", către tabela "Parking", reprezentând id-ul parării, respectiv "vehicleId", către tabela "Vehicle" reprezentând id-ul vehiculului.

O intrare în tabelă se traduce în felul următor: vehiculul cu id-ul "vehicleId" se găsește în parcare cu id-ul "parkingId".

#### 5. **FluxParking**

Tabela "FluxParking" conține lista istoricelor de activitate specifice parărilor. Cheia primară o constituie coloana "id". Conține și o cheie străină către tabela "Parking", "parkingId", reprezentând id-ul parării a cărei activitate este memorată.

Mai sunt salvate data corespunzătoare activității și numărul de intrări, respectiv de ieșiri din parcare respectivă.

De menționat că un flux se realizează, respectiv modifică doar atunci când o cerere de înregistrare este creată, pentru a evita salvarea fluxurilor pentru date în care nu a existat activitate.

#### 6. **Depot**

Tabela "Depou" va conține lista de depouri existente. Cheia primară o constituie coloana "id". Coloana "name", conștând în numele depoului, trebuie să respecte constângerea de unicitate. La fel și perechea formată din coloanele "xCoordinate" și "yCoordinate", constând în coordonatele geografice ale depoului.

Sunt memorate în baza de date și capacitatea curentă a depoului cât și capacitatea maximă.

De menționat faptul că aplicația va avea un singur depou *hardcodat*.

#### 7. **Depot\_Vehicle**

Tabela "Depot\_Vehicle" conține repartizarea vehiculelor în depouri.

O intrare în tabelă conține două coloane, care sunt chei străine: "depotId", către tabela "Depot" reprezentând id-ul depoului, respectiv "vehicleId", către tabela "Vehicle", reprezentând id-ul vehiculului.

O intrare în tabelă se traduce în felul următor: vehiculul cu id-ul "vehicleId" se găsește în depoul cu id-ul "depotId".

## 8. Request

Tabela "Request" conține lista cu cererile de închiriere existente. Cheia primară o constituie coloana "id". Conține și o cheie străină către tabela "Vehicle", "vehicleId", reprezentând id-ul vehiculului închiriat.

Mai sunt memorate data cererii, numele parcărilor de plecare și destinație, orele între care se efectuează închirierea, cât și tipul mașinii închiriate.

De asemenea, coloanele "started" și "solved" memorează dacă cererea a fost declanșată, respectiv finalizată. Inițial, aceste două coloane vor fi setate la "false". Cererea este declanșată sau finalizată în momentul în care cursa este începută, respectiv, terminată cu bine.

## 9. User\_Request

Tabela "User\_Request" conține repartizarea cererilor de închiriere la utilizatori.

O intrare în tabelă conține două coloane, care sunt chei străine: "userId", către tabela "AppUser" reprezentând id-ul utilizatorului care a făcut cererea de închiriere, respectiv "requestId", către tabela "Request", reprezentând id-ul cererii de închiriere.

O intrare în tabelă se traduce în felul următor: cererea de închiriere cu id-ul "requestId" aparține utilizatorului cu id-ul "userId".

## 10. Ride

Tabela "Ride" conține lista curselor existente. Cheia primară o constituie coloana "id". Conține și o cheie străină către tabela "Vehicle", "vehicleId", reprezentând id-ul vehiculului închiriat.

Mai sunt memorate data cursei, numele parcărilor de plecare și destinație, orele între care se efectuează închirierea, cât și tipul mașinii închiriate.

În plus, coloana "distance" memorează distanța dintre cele două parări iar coloana "price" salvează prețul cursei.

## 11. User\_Ride

Tabela "User\_Ride" conține repartizarea curselor la utilizatori.

O intrare în tabelă conține două coloane, care sunt chei străine: "userId", către tabela "AppUser" reprezentând id-ul utilizatorului care a efectuat cursa, respectiv "rideId", către tabela "Ride", reprezentând id-ul cursei.

O intrare în tabelă se traduce în felul următor: cursa cu id-ul "rideId" aparține utilizatorului cu id-ul "userId".

## 12. Preference

Tabela "Preference" conține lista preferințelor existente. O preferință constă în stabilirea informațiilor care se repetă cel mai mult în cererile de închiriere ale unui utilizator, în scopul prezentării lor ca prestabilite. Astfel, un utilizator are cel mult o preferință, ea fiind actualizată de fiecare dată când conținutul ei se modifică. Cheia primară o constituie coloana "id". Conține și o cheie străină către tabela "AppUser", "userId", reprezentând id-ul utilizatorului.

Mai sunt memorate numele parcărilor de plecare și destinație, respectiv tipul vehiculului folosit de cele mai multe ori de către utilizator în alcătuirea unei cereri de înregistrare.

## 13. Move

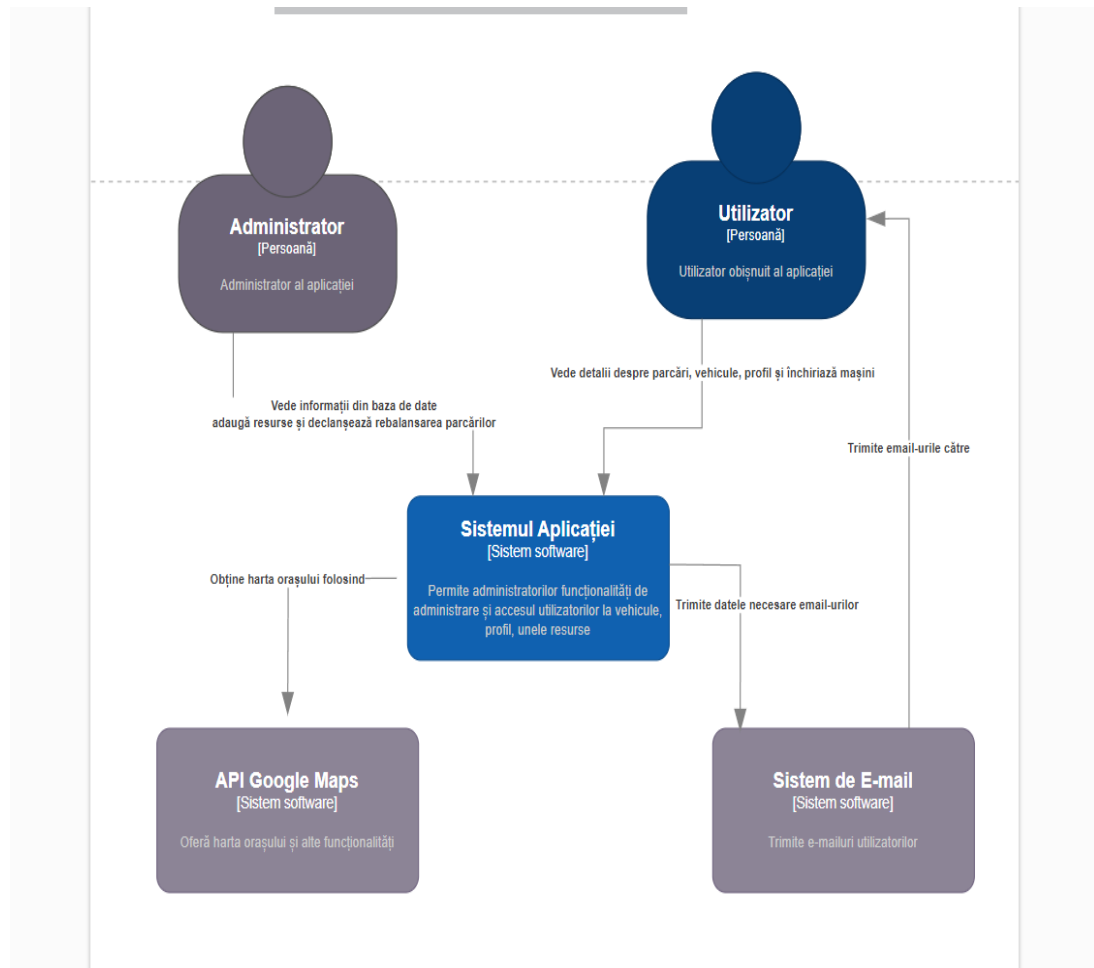
Tabela "Move" conține lista mutărilor de vehicule realizate la ultima rebalansare. Cheia primară o constituie coloana "id". Conține și o cheie străină către tabela "Vehicle", "vehicleId", reprezentând id-ul vehiculului ce a fost mutat.

Mai sunt memorate numele parcării, respectiv depoului de unde vehiculul a fost luat și numele parcării în care vehiculul a fost mutat.

## 2.3 Arhitectura sistemului

### 1. Contextul sistemului

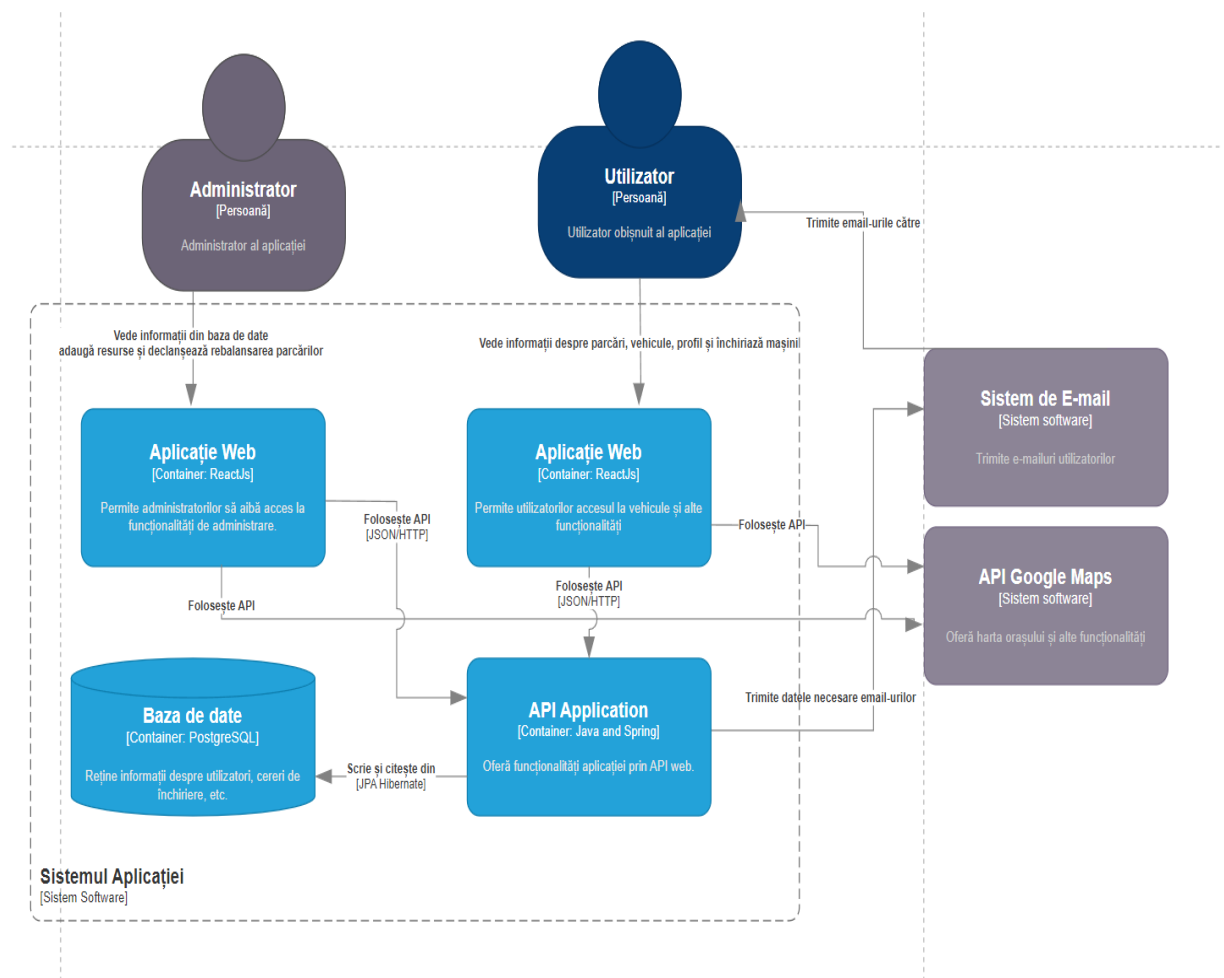
Pentru a detalia arhitectura sistemului este sugestivă o diagramă a contextului, reprezentând primul nivel din modelul C4.



API-ul Google Maps va fi folosit în general pentru a obține harta orașului și pentru a amplasa marcatoarele parcărilor și a depoului.

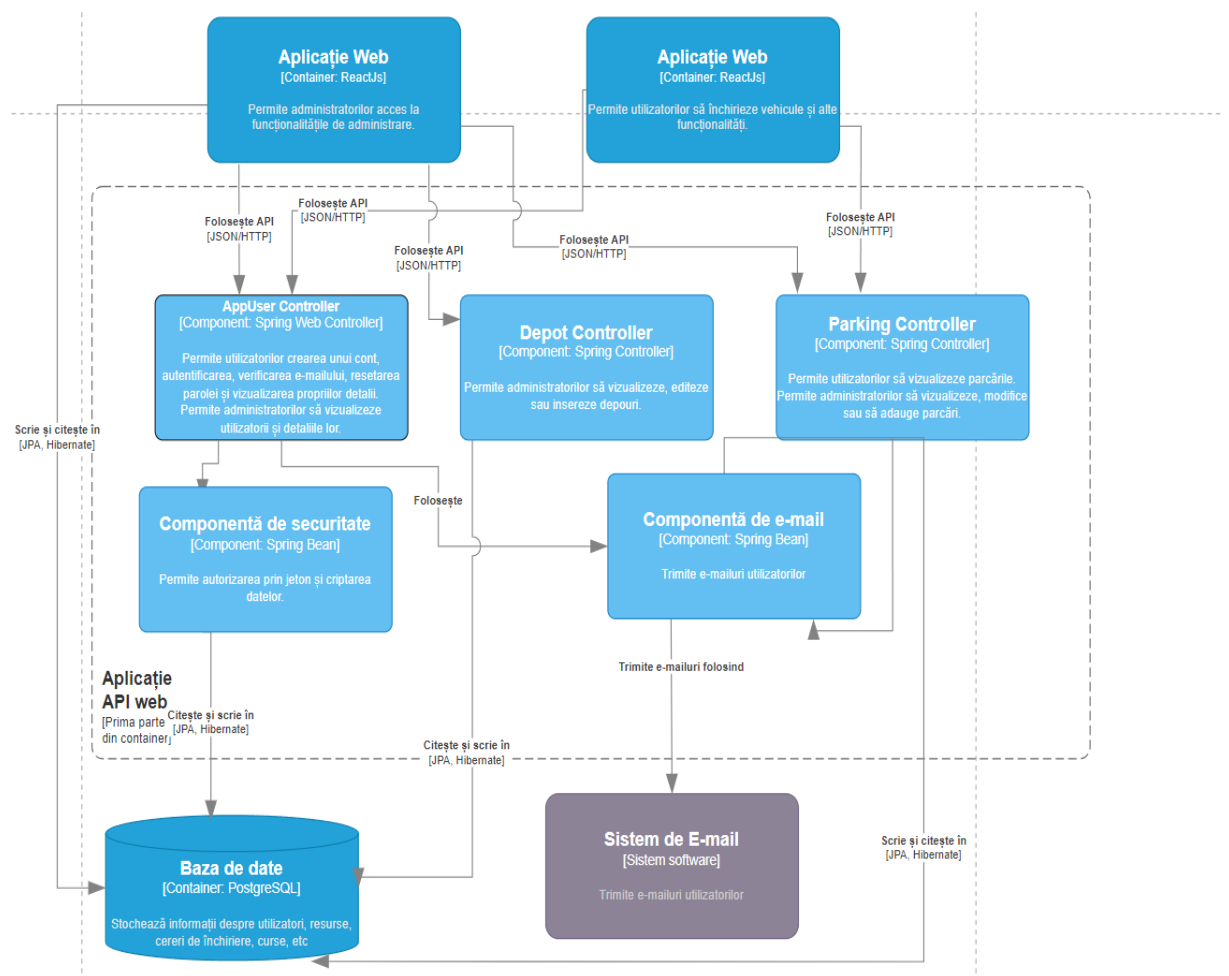
Sistemul de E-mail va fi folosit pentru transmiterea codurilor de verificare a adresei de e-mail și de resetare a parolei.

2. **Diagrama containerului** Pentru a detalia sistemul software este sugestivă o diagramă a containerului, reprezentând al doilea nivel din modelul C4.



Utilizatorul va folosi aplicația Web pentru a închiria vehicule și a vizualiza unele date despre resurse, inclusiv hărțile. Administratorul va utiliza aplicația Web pentru a vedea sau introduce informații în baza de date și a declanșa redistribuirea mașinilor în parcuri în scopul rebalansării acestora.

### 3. Diagramele componentelor



*Controller-ul "AppUser"* va fi folosit de atât de către administrator cât și de către utilizatorul normal. Administratorul se va putea autentifica și va putea interoga lista cu utilizatori existenți, precum și istoricul activității lor. În plus, el poate vizualiza o statistică privitoare la activitatea utilizatorilor. Utilizatorul se poate înregistra, își poate verifica adresa de email, își poate reseta parola și își poate vizualiza profilul și activitatea sa.

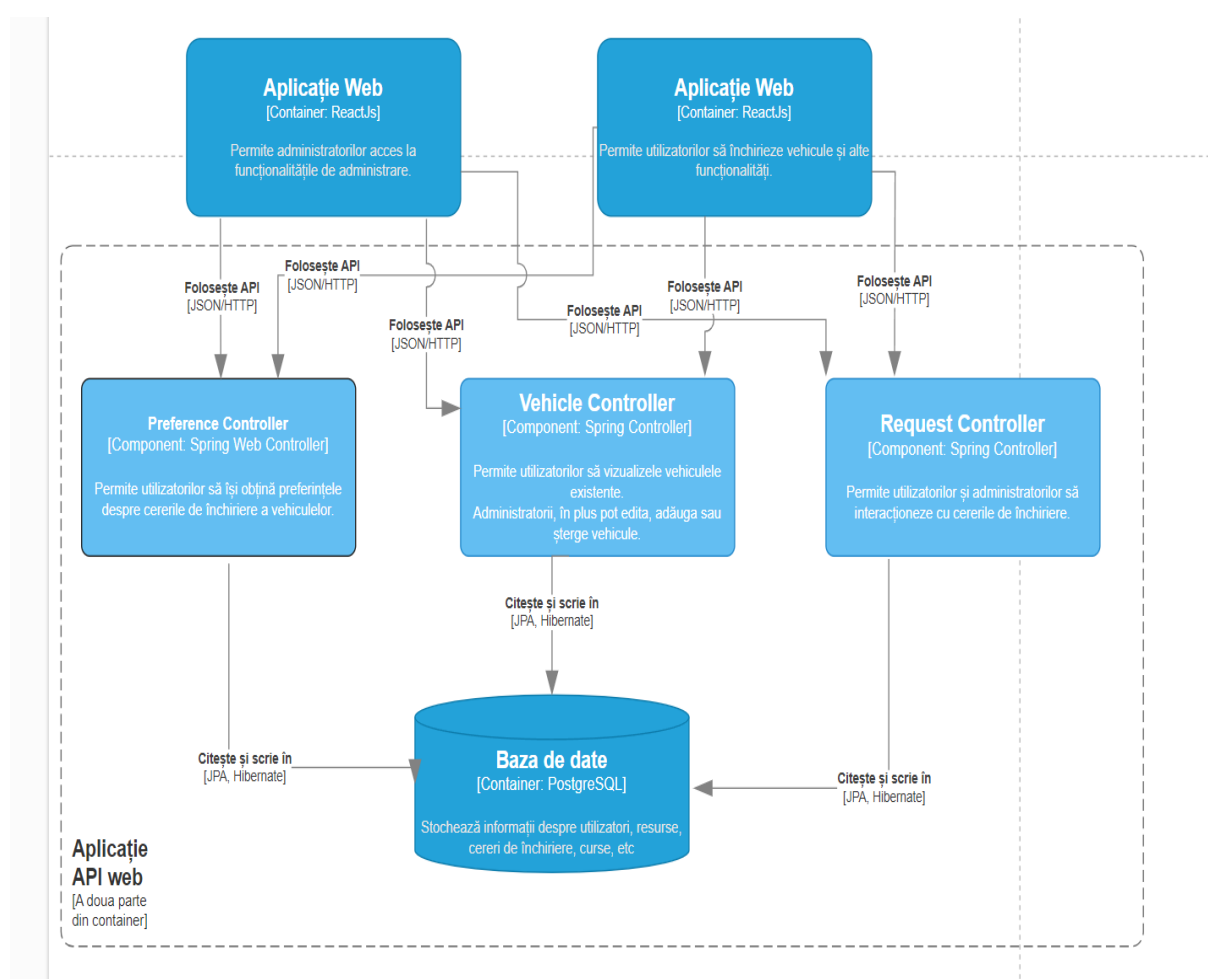
*Controller-ul "Depot"* va fi folosit de atât de către administrator. Administratorul va putea vizualiza lista cu depourile existente și detaliile fiecăruia dintre ele. De asemenea, poate edita depourile existente sau poate crea unul nou.

*Controller-ul "Parking"* va fi folosit de atât de către administrator cât și de către utilizatorul normal. Ambii vor putea vizualiza lista cu parcarile existente și deta-

liile lor. În plus, administratorul poate edita sau adăuga parcări, și poate vizualiza o statistică a activității din fiecare parcare sau poate obține ierarhia parcarilor în funcție de istoricul lor, respectiv ierarhia necesității de a redistribui mașini în parcare respectivă.

Componenta de securitate ajută la generarea jetoanelor de autorizare și apoi validarea autorizării pe baza lor și la criptarea parolelor în baza de date.

Componenta de e-mail va trimite e-mailuri utilizatorilor conținând codurile de verificare a adresei de e-mail și de resetare a parolei.



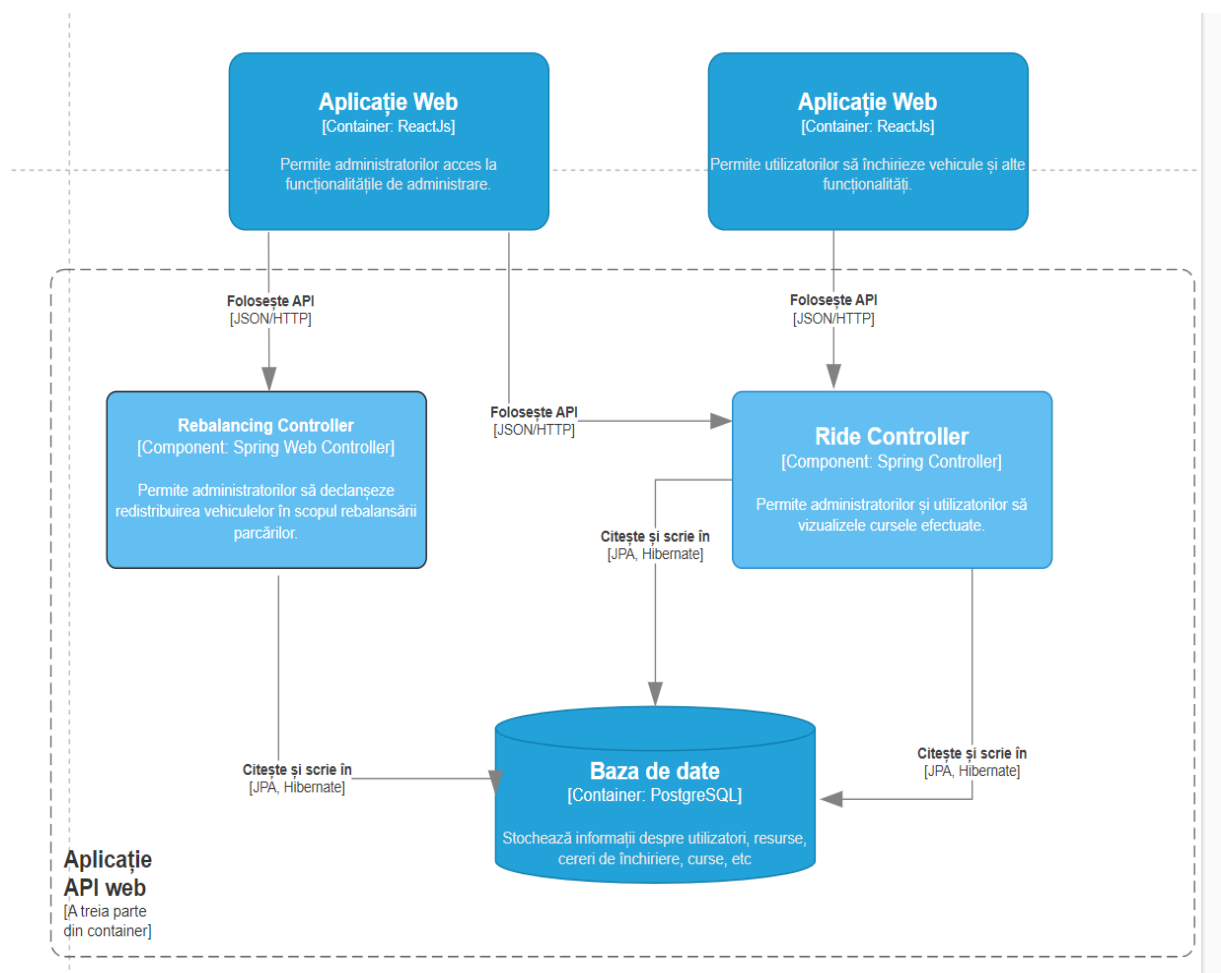
Controller-ul "Preference" va fi folosit de atât de către administrator cât și de către utilizatorul normal. Ambii vor putea obține preferințele utilizatorilor, respectiv propriile preferințe în ceea ce privește închirierea unui vehicul.

Controller-ul "Vehicle" va fi folosit de atât de către administrator cât și de către



utilizatorul normal. Ambii vor putea vizualiza lista cu vehiculele existente și detaliile lor. De asemenea, ambii pot obține listele de vehicule ce pot fi scoase din anumite parcări sau listele de vehicule ce vor fi la un moment dat într-o parcare. În plus, administratorul poate edita sau adăuga vehicule, și poate vizualiza o statistică distribuirii lor în funcție de anumite criterii.

*Controller-ul "Request"* va fi folosit de atât de către administrator cât și de către utilizatorul normal. Utilizatorul poate crea o cerere de închiriere, iar până o activează, o poate edita sau șterge. Apoi, când călătoria este finalizată, o poate închide. De asemenea, își poate vizualiza activitatea trecută. Administratorul are acces la activitatea tuturor utilizatorilor.



*Controller-ul "Ride"* va fi folosit de atât de către administrator cât și de către utilizatorul normal. Utilizatorul își poate vedea istoricul curselor proprii, iar administratorul are acces la istoricul curselor tuturor utilizatorilor.

*Controller-ul "Rebalancing"* va fi folosit doar de către administrator. Acesta poate declanșa operațiunea de redistribuire a vehiculelor ce pot fi mutate în parcuri, pe baza ierarhizării acestora în funcție de necesitate, toate acestea cu scopul rebalansării parcurilor.

# Capitolul 3

## Implementare și testare

### 3.1 Baza de date

#### 1. Implementarea cu Spring Boot și Hibernate

În implementarea bazei de date am utilizat adnotările din Spring Boot și librăria Hibernate. Am setat proprietățile aplicației astfel încât, pe baza adnotărilor din cod, tabelele să fie create automat.

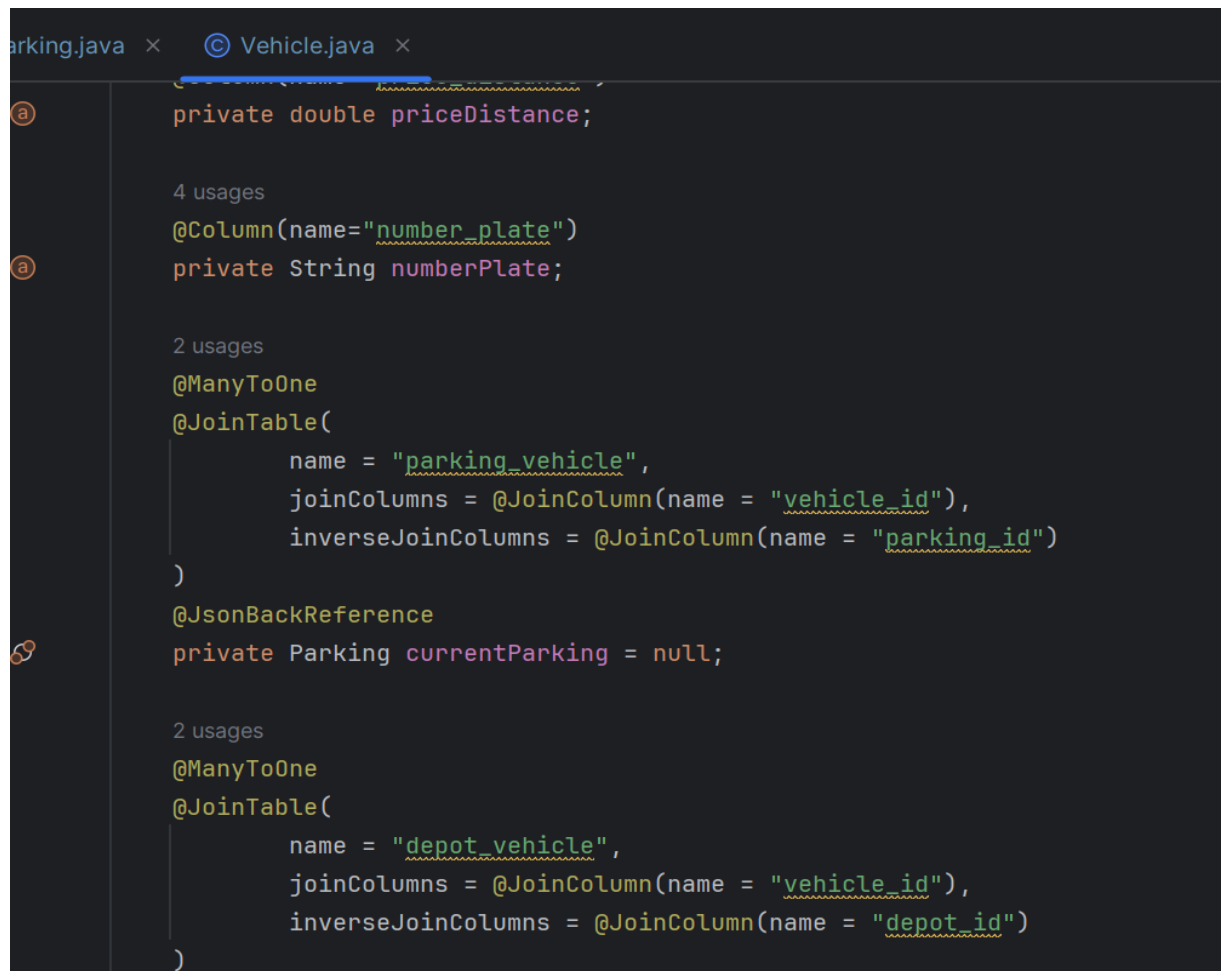
```
## JPA configuration to auto create tables
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.PostgreSQLDialect
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
```

Un tabel este creat în cod în clasa entitate folosind adnotarea `@Table`. Pot fi declarate și constrângerile de unicitate, folosind adnotările `@UniqueConstraint`. Pentru a marca datele membru ale entității ce vor alcătui coloanele tabelului, folosim adnotările `@Column`, iar pentru a marca cheia primară a tabelului, folosim adnotarea `@Id`. În plus, putem crea și secvența pentru incrementarea automată a Id-ului

folosind adnotările de tip @GeneratedValue.

```
© Parking.java x
1 package com.example.webjpademoapplicationsecondtry.entity;
2
3 > import ...|
11
12 CobaschiAser *
12 @Entity
13 @Table(name = "parking",uniqueConstraints = {
14     @UniqueConstraint(columnNames = "name"),
15     @UniqueConstraint(columnNames = {"x_coordinate", "y_coordinate"})
16 })
17 public class Parking {
18
19     @Id
20     @GeneratedValue(strategy = GenerationType.IDENTITY)
21     @Column(name = "id")
22     private Long id;
23
24     5 usages
24     @Column(name = "name")
25     private String name;
26
27     5 usages
27     @Column(name = "x_coordinate")
28     private Double x;
```

Putem marca și cheile străine, respectiv crearea tabelelor ce conțin doar chei străine, folosind adnotările @OneToMany, @ManyToOne și @JoinTable.



```
Vehicle.java
private double priceDistance;

4 usages
@Column(name="number_plate")
private String numberPlate;

2 usages
@ManyToOne
@JoinTable(
    name = "parking_vehicle",
    joinColumns = @JoinColumn(name = "vehicle_id"),
    inverseJoinColumns = @JoinColumn(name = "parking_id")
)
@JsonBackReference
private Parking currentParking = null;

2 usages
@ManyToOne
@JoinTable(
    name = "depot_vehicle",
    joinColumns = @JoinColumn(name = "vehicle_id"),
    inverseJoinColumns = @JoinColumn(name = "depot_id")
)
```

## 2. Script de generare date

Un *script* a fost creat folosind limbajul Python pentru a popula aproape toate tabelele bazei de date. Nepopulat este "Preference", deoarece acest aspect nu era necesar pentru a utiliza aplicația și nici pentru a rula algoritmul de reblanasare a parcărilor. De asemenea, tabelul "Move" nu este populat, deoarece acesta trebuie populat cu rezultatele ultimei rulări a algoritmului de rebalansare, iar scopul este să facem lucrul acesta în demonstrația live.

Tabela "Parking" nu este generată aleator, deoarece altfel reprezentarea pe hartă a marcatoarelor lor ar fi fost lipsită de sens. La fel, tabela "Depot" este populată cu un singur depou, bine localizat.

Tabela "AppUser" a fost generată aleator, folosind un API extern care generează nume. În modul acesta au fost introduși în sistem 150 de utilizatori, având date de înregistrare cuprinse între 1 ianuarie 2024 și 1 martie 2024, pe lângă un admin.

De asemenea, tabela "Vehicle" a fost generată în mod parțial aleator, folosindu-se unele date *hardcode*. În felul acesta au fost introduse în sistem și distribuite în parcări 35 de mașini, niciuna nerămânând în depou, pentru a pune mai bine în evidență algoritmul de rebalansare.

Începând cu o dată aleatoare cuprinsă între 1 martie 2024 și 10 martie 2024, iar apoi tot mai aproape de data curentă, au fost generate aleatoriu dar corect 150 de cereri de închiriere finalizate și 5 cereri de închiriere neîncepute. De asemenea, pentru fiecare cerere finalizată, a fost populat, respectiv modificat tabelul "Ride", respectiv "FluxParking".

## 3.2 Aplicația server

### 1. Tehnologii utilizate

Pentru a realiza aplicația server de servicii web am folosit *framework*-ul Spring Boot. Pentru a accesa cu și a efectua cu ușurință operații asupra bazei de date am folosit pachetele Spring Boot JDBC și Spring Boot Data JPA, împreună cu librăria Hibernate. Pentru a trimite e-mailuri am folosit pachetul Spring Boot Mail și Jakarta Mail.

Partea de securitate conține o clasă creată de mine, JwtUtil, utilizată pentru generarea, respectiv parsarea și validarea jetoanelor de autentificare. Odată autentificați în aplicație, utilizatorii și administratorii vor primi câte un jeton, acesta oferindu-le acces la *endpoint*-urile destinate lor. De asemenea, am folosit librăria Jasypt pentru criptarea datelor din fișierul de proprietăți al aplicației.

Pentru partea de testare automată am folosit *framework*-ul Spring Boot Test și pachetele JUnit, Mockito și AsserJ. Pentru partea de testare manuală am folosit programul Postman.

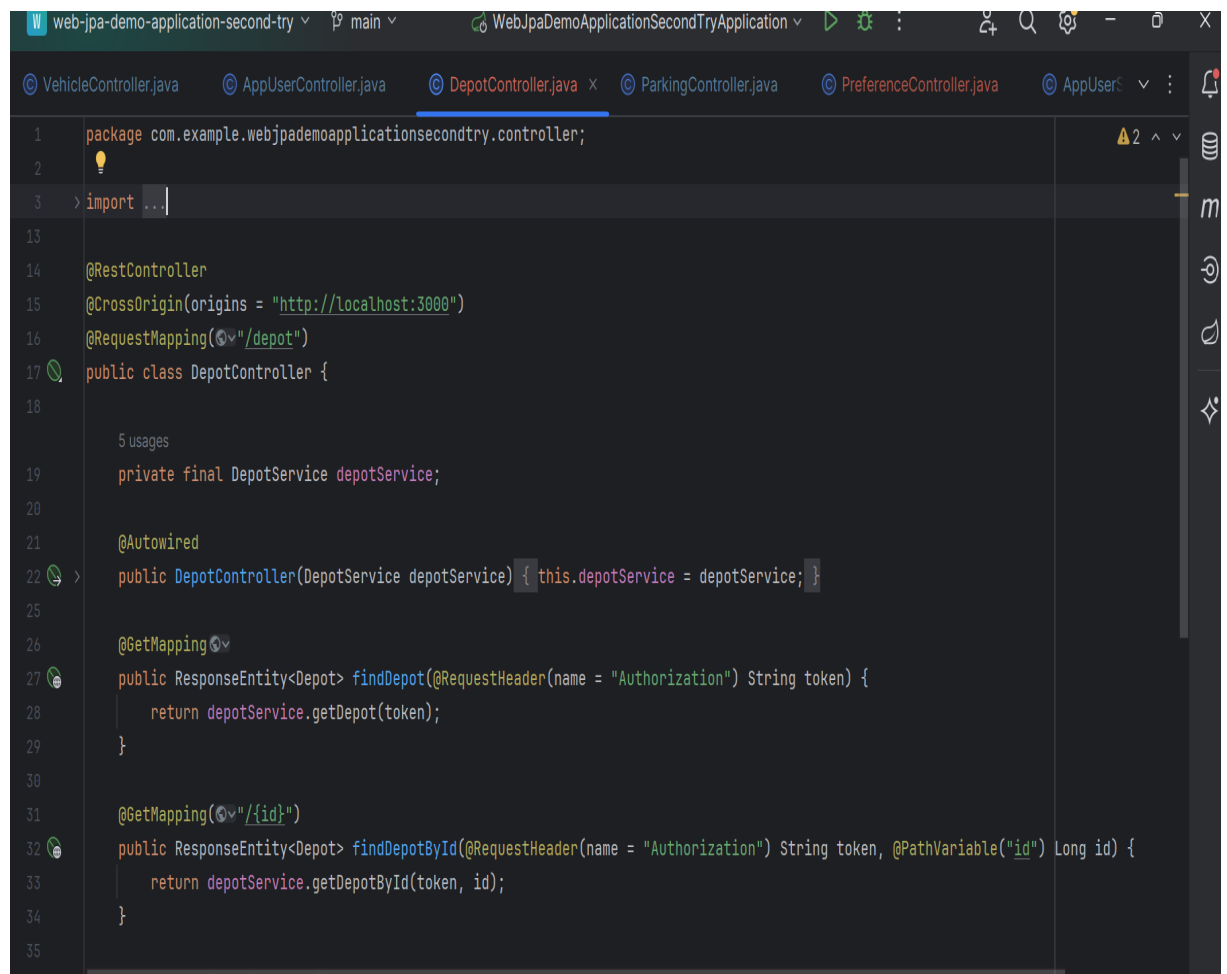
### 2. Straturile aplicației

Aplicația server este structurată după modelul Repository-Service-Controller.

*Controller*-ele conțin *endpoint*-uri ce pot fi accesate atât de către administratori cât și de către utilizatori. Pentru a fi apelat un *endpoint* se folosește adresa URL, tipul de cerere și unele informații în plus, în cazul *endpoint*-urilor care necesită transmiterea către server a unor câmpuri de date sau anumiți parametrii, dar și

un jeton, în cazul *endpoint*-urilor securizate. Informațiile ce pot fi transmise în plus sunt de două tipuri:

- câmpuri de text ce reprezintă valori ale anumitor variabile, folosite la cereri de tip "GET", numite și *request parameters*.
- JSON-uri de date ce sunt folosite adesea împreună cu cereri de tip "POST" și "PUT". Datele acestea vor corespunde unor *DTO-uri* (*Data Transfer Objects*).



```
1 package com.example.webjpademoapplicationsecondtry.controller;
2
3 > import ...
4
13
14 @RestController
15 @CrossOrigin(origins = "http://localhost:3000")
16 @RequestMapping("/depot")
17 public class DepotController {
18
19     5 usages
20     private final DepotService depotService;
21
22     @Autowired
23     public DepotController(DepotService depotService) { this.depotService = depotService; }
24
25
26     @GetMapping
27     public ResponseEntity<Depot> findDepot(@RequestHeader(name = "Authorization") String token) {
28         return depotService.getDepot(token);
29     }
30
31     @GetMapping("/{id}")
32     public ResponseEntity<Depot> findDepotById(@RequestHeader(name = "Authorization") String token, @PathVariable("id") Long id) {
33         return depotService.getDepotById(token, id);
34     }
35
36 }
```

În funcție de *endpoint-ul* apelat, o metodă din *Controller* va fi apelată. Această metodă va apela la rândul ei o metodă dintr-un *Service*.

```
AppUserRepository.java  ParkingController.java  PreferenceController.java  AppUserServiceImpl.java x v
1 package com.example.webjpademoapplicationsecondary.service.implemmentation;
2
3 > import ...
33
34
35 @Service
36 public class AppUserServiceImpl implements AppUserService {
37     private final AppUserRepository appUserRepository;
38
39     private static final Logger logger = LoggerFactory.getLogger(AppUserServiceImpl.class);
40
41
42     @Autowired
43     public AppUserServiceImpl(AppUserRepository appUserRepository) { this.appUserRepository = appUserRepository; }
44
45
46
47
48     @Override
49     public ResponseEntity<List<AppUser>> findAllUser(String token) {
50
51         if (!JwtUtil.isAuthorizedAdmin(token)) {
```

În acel moment, în funcție de tipul de cerere, metoda din *Service* va efectua o operație prin care comunică cu baza de date, printr-un *Repository*. Fie se vor extrage date din baza de date, pe baza *request parameters*, fie se vor insera, edita sau șterge informații din baza de date, folosindu-se în cele mai multe cazuri și JSON-urile de date preluate împreună cu *endpoint*-urile.



```
VehicleController.java  AppUserController.java  DepotController.java  AppUserRepository.java  Pari
1 package com.example.webjpademoapplicationsecondtry.repository;
2
3 > import ...
13
14 @Repository
15 public interface AppUserRepository extends JpaRepository<AppUser, UUID> {
16
17     @Query("SELECT u FROM AppUser u WHERE u.email =:emailAddress")
18
19     public AppUser findUserByEmail(@Param("emailAddress") String email);
20
21     @Query("SELECT u FROM AppUser u WHERE u.username =:username")
22     public AppUser findUserByUsername(@Param("username") String username);
23
24     @Query("SELECT u FROM AppUser u WHERE u.id = :uuid")
25     public AppUser findUserById(@Param("uuid") UUID uuid);
26
27     @Query("SELECT u FROM AppUser u WHERE u.username=:username AND u.password=:password")
28     public AppUser login(@Param("username") String username, @Param("password") String password);
```

Așadar, *Controller* este nivelul din aplicație care preia cererile de la utilizatori sau administratori, *Service* este nivelul din aplicație care implementează logica aplicației, iar *Repository* este nivelul din aplicație care realizează operațiunile asupra bazei de date. Acest lucru este posibil prin utilizarea entităților, însoțite de adnotări, care realizează corespondența dintre fiecare atribut al entităților și coloanele respective din tabele.

### 3. Configurările de securitate

În pachetul "configuration" se găsește clasa "JasyptConfig", a cărei adnotare "@EnableEncryptableProperties" permite criptarea și decriptarea proprietăților aplicației și a adresei de e-mail folosită la transmiterea e-mailurilor către utilizatori. În această clasă, una dintre metode citește parola de criptare dintr-un fișier, iar cealaltă este un *Bean* care construiește un obiect de tipul *StringEncryptor* folosit efectiv la criptare și decriptare.

```

15
16 @Configuration
17 @EnableEncryptableProperties
18 public class JasyptConfig {
19
20     1 usage
21     @ private static String readFile() throws IOException {
22         byte[] encoded = Files.readAllBytes(Paths.get("src/main/resources/jasypt.passwd"));
23         return new String(encoded, StandardCharsets.US_ASCII);
24     }
25
26     @Bean(name = "jasyptStringEncryptor")
27     public StringEncryptor stringEncryptor() throws IOException {
28         PooledPBEStrEncryptor encryptor = new PooledPBEStrEncryptor();
29         SimpleStringPBEConfig config = new SimpleStringPBEConfig();
30         config.setPassword(readFile());
31         config.setAlgorithm("PBESWithMD5AndDES");
32         config.setKeyObtentionIterations("1000");
33         config.setPoolSize("1");
34         config.setProviderName("SunJCE");
35         config.setSaltGeneratorClassName("org.jasypt.salt.RandomSaltGenerator");
36         config.setStringOutputType("base64");
37         encryptor.setConfig(config);
38         return encryptor;
39     }

```

Clasa `JwtUtil`, conține logica privitoare la jetoanele de autentificare. Metoda `generateToken` creează un nou jeton care va fi asignat unui utilizator sau administrator, iar metodele `isAuthorizedAdmin` și `isAuthorizedUser` verifică dacă jetonul trimis împreună cu `endpoint`-ul corespunde unui administrator, respectiv utilizator.

#### 4. *Endpoint*-urile aplicației

Aplicația are opt *Controllere* implementate, fiecare dintre ele având cel puțin câte un *endpoint*.

*Controller*-ul "`AppUserController`" are doisprezece *endpoint*-uri, dintre care cinci sunt nesecurizate. Acestea sunt de tip "`POST`" și se ocupă de înregistrarea unui cont nou, verificarea adresei de e-mail, trimiterea unui cod de resetare al parolei, resetarea propriu-zisă a parolei și autentificarea în aplicație. *Endpoint*-urile securizate, destinate strict administratorilor sunt două, de tip "`GET`". Unul dintre ele obține lista tuturor utilizatorilor din sistem iar celălalt obține statistica privi-

toare la activitatea utilizatorilor. Celelalte *endpoint*-uri securizate, disponibile atât pentru administrator cât și pentru utilizator sunt :

- trei de tip "GET", folosite pentru obținerea profilului, a istoricului de cereri de închiriere, respectiv de curse ale unui utilizator.
- unul de tip "POST", destinat actualizării profilului.
- unul de tip "DELETE", destinat ștergerii contului.

*Controller*-ul "DepotController" are patru *endpoint*-uri securizate, dedicate doar administratorilor. Toate sunt de tip "GET". Două dintre ele permit obținerea informațiilor despre un depou oarecare și lista vehiculelor dintr-un depou oarecare, în timp ce celelalte două *endpoint*-uri oferă detalii despre depoul hardcodat, respectiv lista de vehicule din el.

*Controller*-ul "ParkingController" are doisprezece *endpoint*-uri, toate fiind securizate. Șapte dintre acestea sunt dedicate exclusiv administratorilor. Dintre acestea, unul de tip "GET" permite obținerea ierarhiei parcărilor raportat de numărul de vehicule care trebuie aduse pentru rebalansarea parcărilor. Un altul, tot de tip "GET", permite obținerea statisticii parcărilor în funcție de activitatea desfășurată în ele. Unul de tip "POST" permite salvarea unei noi parări, iar unul de tip "PUT" permite modificarea unei parări. Mai sunt două, de tip "POST" care permit adăugarea, respectiv scoaterea unui vehicul dintr-o parcare. Nu în ultimul rând, mai este unul de tip "DELETE", care permite eliminarea unei parări din aplicație. *Endpoint*-urile la care au acces atât administratorii cât și utilizatorii normali sunt în număr de patru. Toate sunt de tip "GET" și sunt folosite pentru obținerea listei tuturor parcărilor existente, a informațiilor despre o anumită parcare (pe baza numelui ei și a id-ului ei) și pentru obținerea listei de vehicule dintr-o anumită parcare.

*Controller*-ul "VehicleController" are unsprezece *endpoint*-uri, toate fiind securizate. Șase dintre acestea sunt dedicate exclusiv administratorilor. Dintre acestea, unul de tip "GET" permite obținerea statisticii vehiculelor în funcție de anumite criterii. Mai sunt două, de tip "GET" care permit obținerea listelor de vehicule ce pot fi adăugate, respectiv eliminate dintr-o parcare. Unul de tip "POST" permite introducerea în sistem a unui nou vehicul, iar unul de tip "PUT" permite modificarea unui vehicul. Nu în ultimul rând, mai este unul de tip "DELETE",

care permite eliminarea unui vehicul din aplicație. *Endpoint*-urile la care au acces atât administratorii cât și utilizatorii normali sunt în număr de cinci. Toate sunt de tip "GET" și sunt folosite pentru obținerea listei tuturor vehiculelor existente, a informațiilor despre un anumit vehicul, obținerea listei de vehicule ce vor fi într-o anumită parcare după finalizarea cererilor de închiriere până la o anumită oră și listele de vehicule ce pot fi utilizate pentru crearea, respectiv editarea unei cereri de închiriere.

*Controller*-ul "PreferenceController" are un singur *endpoint* securizat, accesibil atât administratorilor cât și utilizatorului. Este de tip "GET" și permite obținerea preferințelor unui utilizator în ceea ce privește realizarea unei cereri de închiriere.

*Controller*-ul "RequestController" are opt *endpoint*-uri securizate. Doar unul dintre ele este specific doar administratorilor, și anume unul de tip "GET" care permite obținerea tuturor cererilor de înregistrare. Celelalte șapte sunt accesibile atât administratorilor cât și utilizatorilor normali. Cinci dintre ele însă sunt utilizate în general doar de către utilizatori, și anume: trei de tip "POST", folosite pentru crearea unei cereri de închiriere, pentru activarea și finalizarea ei, unul de tip "PUT", folosit pentru editarea unei cereri de închiriere, și unul de tip "DELETE" pentru ștergerea ei. Celelalte două sunt de tip "GET", utilizate pentru obținerea unei cereri de închiriere, în funcție de id-ul ei sau de data în care a fost realizată.

*Controller*-ul "RideController" are trei *endpoint*-uri securizate. Doar unul dintre ele este specific doar administratorilor, și anume unul de tip "GET" care permite obținerea tuturor curselor efectuate. Celelalte două sunt accesibile atât administratorilor cât și utilizatorilor normali, însă sunt utilizate în general doar de către utilizatori, și anume: una de tip "GET", folosită pentru obținerea detaliilor unei curse și unul de tip "POST", folosit estimarea prețului unei curse.

*Controller*-ul "RebalancingController" are două *endpoint*-uri securizate. Acestea sunt accesibile doar administratorilor. Unul este de tip "GET" și permite obținerea tuturor mutărilor efectuate la ultima rebalansare a parcărilor. Celălalt este de tip "POST" și permite declanșarea procedurii de rebalansare a parcărilor.

## 5. Clasele de utilitate folosite

Am folosit unele clase utilitare care m-au ajutat să structurez mai bine codul și să ușurez implementarea.

Clasa "PeriodConverter" preia un obiect de tip "String" și în funcție de conținutul lui returnează un obiect de tip "Date". Este folosită la vizualizarea statisticilor și la algoritmul de rebalansare a parcarilor.

Clasa "InputDate" preia în *Controller* un JSON care încapsulează componentele unei date calendaristice și îl trimite mai departe la *Service* sub forma unui obiect de tip "Date".

Clasa "PercentageConverter" ajută la algoritmul de rebalansare a parcarilor, implementând metode pentru calcularea procentajului dintr-o cantitate, sau a unei fracțiuni de cantitate pe procentajului.

Clasa "DistanceConverter" implementează o metodă care preia coordonatele geografice a două puncte sub forma (latitudine, longitudine) și calculează distanța în kilometri între cele două puncte. Este utilă la calcularea distanței dintre două parări.

## 6. Algoritmi principali

Bineînțeles, nivelul algoritmic de interes al aplicației, raportat la problemă, este algoritmul care gestionează redistribuirea vehiculelor în scopul rebalansării parcarilor.

### Necesitatea algoritmului

În cadrul aplicației, utilizatorii au posibilitatea să realizeze cereri de închiriere a vehiculelor, pe care le pot activa, transformându-le în curse între parările din oraș. În felul acesta, distribuția vehiculelor în parări este într-o continuă dinamică. Totuși, există posibilitatea ca, din cauza localizării geografice, a punctelor de interes din zonă, sau alte motive, unele parări să fie mult mai tranzitate decât altele. Sau mai bine zis, este posibil ca pentru unele parări, cererea de închiriere a unui vehicul din acea parcare să fie mult mai mare decât cererea de închiriere a vehiculelor din alte parări. Din acest motiv, ne dorim să utilizăm resursele, respectiv vehiculele într-un mod cât mai eficient, astfel încât, cu vehiculele existente să putem satisface un număr mai mare de cerințe. Dorim să redistribuim vehiculele din depou, sau din parările cu un trafic redus, către parările cu multe cerințe de închiriere. Totuși, este important să avem în vedere faptul că numărul de vehicule mutate dintr-o parcare trebuie să fie unul rezonabil, astfel încât să putem satisface și cerințele acelei parări.

### Descrierea algoritmului

În vederea implementării acestui agloritm, sunt necesare câteva etape.

- În primul rând, este necesar să calculăm o ierarhie a parcărilor în funcție de numărul de cereri de închiriere a mașinilor din fiecare parcare realizate într-o perioadă bine delimitată, care este cunoscută. Așadar, vom considera ca fiind cele mai importante acele parări care au apărut de cele mai multe ori în cererile de închiriere ca loc de plecare, în perioada amintită. Folosim un *HashMap* care conține perechi de tipul (id parcare - număr plecări din parcare respectivă în perioada dată). Calculăm de asemenea, numărul total de plecări înregistrat în perioada stabilită.
- Apoi, este necesar să calculăm numărul maxim de mașini ce ar putea fi mutate, însemnând numărul mașinilor ce se află în depou, la care adăugăm numărul mașinilor neimplicate în vreo cerere de închiriere care încă nu a fost finalizată. În caz de nevoie, aceste mașini pot fi oricând redistribuite.
- În acest moment, putem actualiza structura *HashMap* sus amintit, astfel încât să conțină perechi de tipul (id parcare - număr de vehicule). Numărul de vehicule alocat fiecărei parări este compus în felul următor: calculăm procentajul pe care îl reprezintă plecările din parcare respectivă din totalul de plecări înregistrat în perioada stabilită. Apoi, numărul de vehicule asignat este numărul de vehicule din totalul celor care pot fi mutate, corespundent procentajului calculat. În felul acesta, distribuim mașinile în mod proporțional cu importanța parcărilor.
- Pe baza *HashMap*-ului obținut anterior, implementăm agloritmul de redistribuire a vehiculelor în felul următor: Într-o buclă *for* principală, parcurgem parările în ordinea importanței lor. Pentru fiecare în parte, stabilim numărul de vehicule pe care vom încerca să îl adăugăm efectiv, reprezentând minimul dintre valoarea din *HashMap* și numărul de locuri disponibile rămase după finalizarea cererilor de închiriere aflate în desfășurare, prin care vor ajunge vehicule în parcare respectivă. Odată stabilit acest număr, începem o buclă *while*, în care vom căuta să mutăm mai întâi vehiculele din depou, neasignate niciunei parări, apoi, dacă mai este nevoie, pornind de la cea mai neimportantă parcare, adăugăm vehicule în parcare pe care vrem să o rebalansăm. Astfel, avem pe de o parte parcare pe care vrem să o rebalansăm, în care vrem să mutăm mașini, să zicem P1, și parcare P2, din care

vrem să mutăm mașini.

- Dacă cumva valoarea din *HashMap* corespunzătoare lui P1 este mai mică decât valoarea din *HashMap* a lui P2, atunci trecem la următoarea parcare pe vrem să o rebalansăm.
- Dacă diferența dintre valorile din *HashMap* corespunzătoare lui P1, respectiv P2 este mai mică decât numărul de mașini pe care le putem muta din P2, atunci mutăm doar un număr de mașini egal cu această diferență. Procedăm în felul acesta pentru a menține totuși un nivel echilibrat, în sensul în care să nu luăm mai multe mașini decât ar trebui, pentru ca și parcare P2 să fie în continuare accesibilă.
- Dacă diferența dintre valorile din *HashMap* corespunzătoare lui P1, respectiv P2 este mai mare decât numărul de mașini pe care le putem muta din P2, atunci mutăm toate mașinile din P2 care se pot muta, cu excepția uneia, pentru a fi în continuare posibilă închirierea unei mașini din P2.

Vom realiza operația până când ajungem la o parcare cel puțin la fel de importantă sau până când adăugăm deja numărul de vehicule dorit, moment în care parcare este rebalansată și trecem la următoarea cea mai importantă parcare pentru care repetăm procedeul. Și tot așa, până când rebalansăm toate parcarile care au nevoie de acest lucru.

Sigur că, așa cum este prezentat, algoritmul pune mai bine în valoare rebalansarea parcarilor, în sensul nefolosirii depoului, atunci când depoul este gol. Tocmai din acest motiv, am ales ca pe setul de date de testare, depul să fie gol, iar mutările să se realizeze efectiv între parări.

## 7. Testarea aplicației

Am realizat testare automată pentru unele metode din stratul de servicii, folosind pachetele JUnit, Mockito și AssertJ.

JUnit este folosit pentru adnotări.

Mockito este folosit pentru injectări de servicii și rezultate așteptate și pentru *mock-uri*, reprezentând apeluri la diferite *Repository* sau *Bean-uri*.

AssertJ este folosit pentru afirmații ce urmează a fi verificate în cadrul testelor.

```

201     }
202
203
204     @Test
205     public void whenDeleteParkingIsCalled_WithValidValues_ThenReturnCorrectResponse() {
206         Mockito.when(parkingRepository.findParkingById(parking.getId())).thenReturn(parking);
207
208         try (MockedStatic<JwtUtil> mockedJwtUtil = Mockito.mockStatic(JwtUtil.class)) {
209             // Arrange
210             mockedJwtUtil.when(() -> JwtUtil.isAuthorizedAdmin(token)).thenReturn(true);
211
212             // Act
213             ResponseEntity<String> deleteParking = parkingService.deleteParkingById(token, parking.getId());
214
215             // Assert
216             Assertions.assertThat(deleteParking).isNotNull();
217             Assertions.assertThat(deleteParking.getBody()).isEqualTo("Ok");
218             Assertions.assertThat(deleteParking.getStatusCode()).isEqualTo(HttpStatus.OK);
219         }
220     }
221

```

Testarea manuală a fost implementată în cadrul programului Postman, unde pentru aproape fiecare *endpoint* am creat câte un request pe care l-am testat, inclusiv pe baza jetoanelor.

### 3.3 Aplicația ReactJs pentru *Frontend*

#### 1. Tehnologii utilizate

Aplicația este scrisă în limbajul JavaScript, folosind librăria React. Pentru importul unor containere predefinite, precum Button, Container, etc am folosit librăria Bootstrap.

Harta orașului este preluată și apoi populată cu marcatoarele specifice parcărilor și depourilor folosind API-ul de la Google Maps.

Pentru încărcarea diagramelor am folosit două librării, x-charts.s și charts.js

#### 2. Securitatea

Partea de securitate este dezvoltată în două privințe.

În primul rând, atunci când se preiau datele introduse de utilizator, acestea sunt sanitizate, în scopul evitării atacurilor de tip *SQL Injection*. Această operațiune

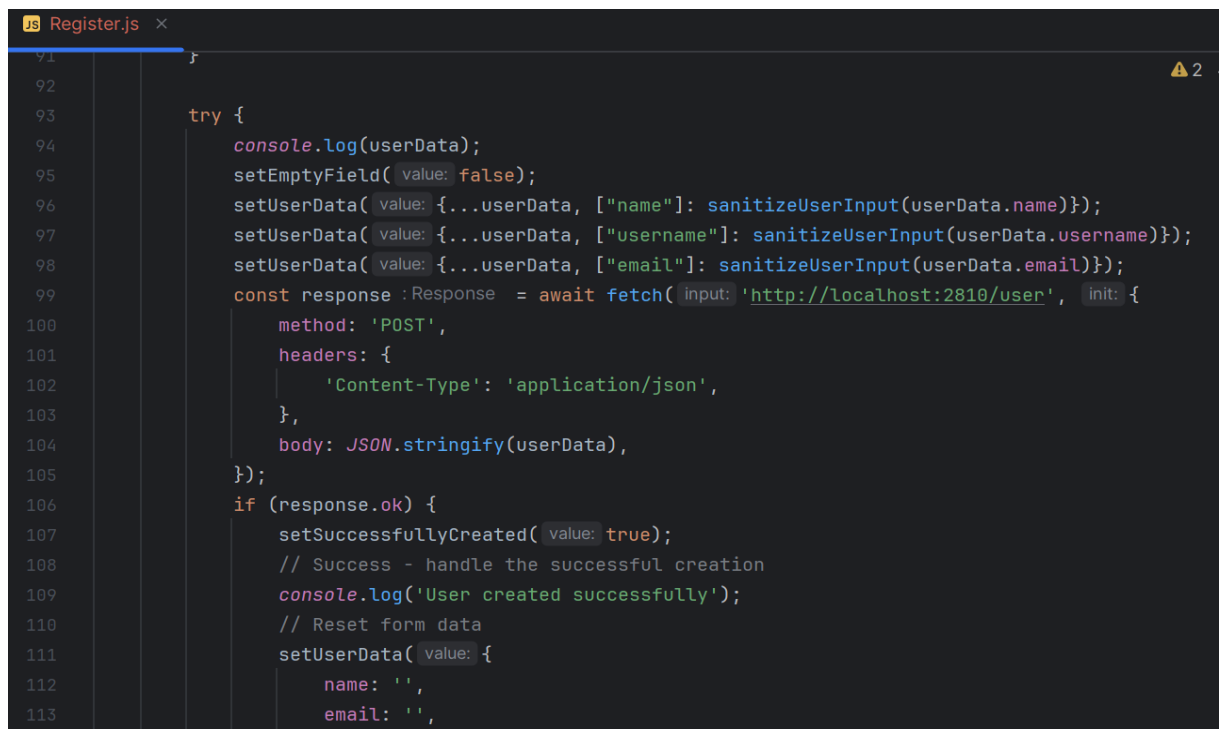


este importantă mai ales la partea de înregistrare și autentificare.

În al doilea rând, în momentul logării în aplicație, este primit de la server jetonul de autentificare. Pe baza lui este permis sau nu accesul în paginile aplicației. Lucrul acesta ajută la prevenirea atacurilor prin care un utilizator, încercând să folosească structura URL-urilor, dorește să obțină acces la pagini sau resurse nepermise lui.

### 3. Comunicarea cu aplicația server

Comunicarea cu server-ul aplicației se realizează prin *fetch*-uri la *endpoint*-urile acestuia, specificând, împreună cu URL-ul ce conține parametrii suplimentari (în cazul metodelor "GET"), *body*-ul (în cazul metodelor "PUT" și "POST"), și *header*-ele apelului, ce conține și jetonul de autentificare.



```
JS Register.js x
91
92
93     try {
94         console.log(userData);
95         setEmptyField( value: false);
96         setUserData( value: {...userData, ["name"]: sanitizeUserInput(userData.name)});
97         setUserData( value: {...userData, ["username"]: sanitizeUserInput(userData.username)});
98         setUserData( value: {...userData, ["email"]: sanitizeUserInput(userData.email)});
99         const response :Response = await fetch( input: 'http://localhost:2810/user', init: {
100             method: 'POST',
101             headers: {
102                 'Content-Type': 'application/json',
103             },
104             body: JSON.stringify(userData),
105         });
106         if (response.ok) {
107             setSuccessfullyCreated( value: true);
108             // Success - handle the successful creation
109             console.log('User created successfully');
110             // Reset form data
111             setUserData( value: {
112                 name: '',
113                 email: '',
```

Pe baza răspunsului primit de la server, se determină comportamentul ulterior al paginii.

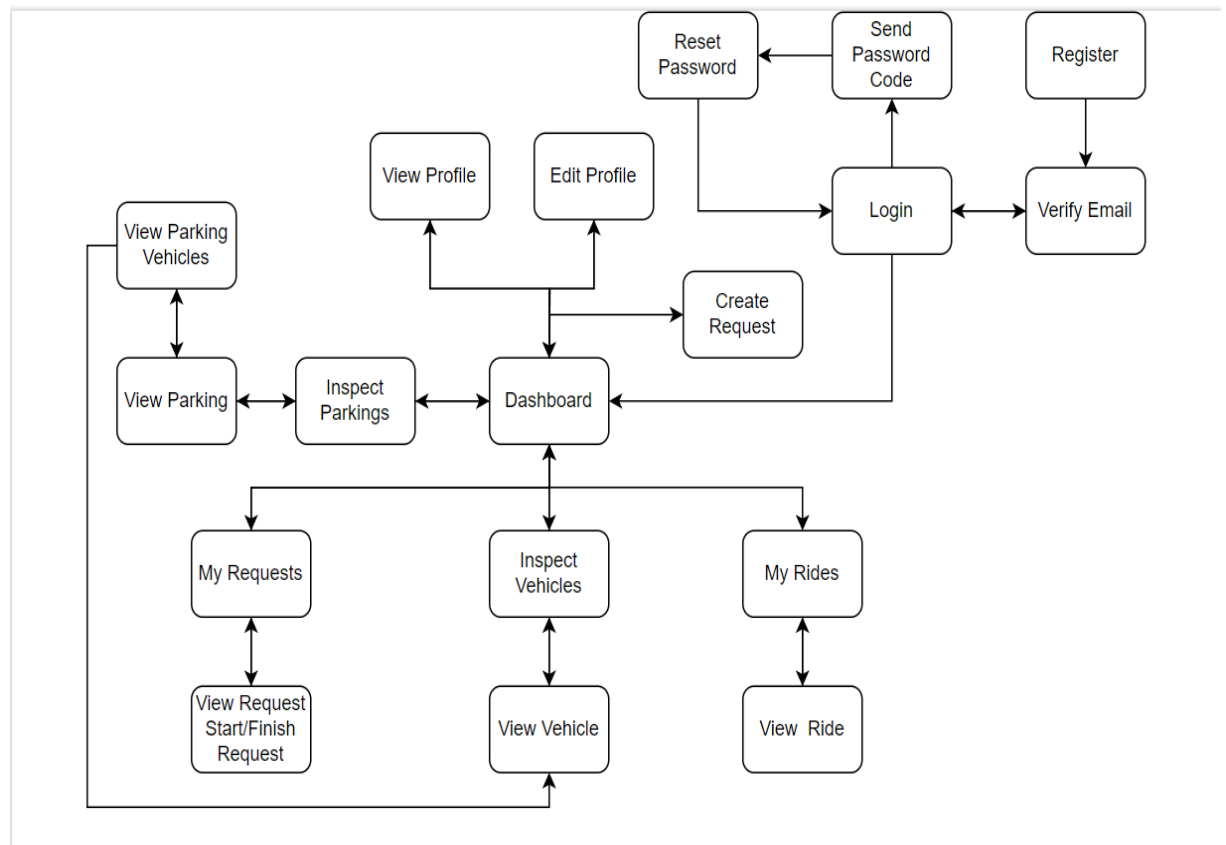
### 4. Resursele aplicației

Fiecărei pagini a aplicației îi corespunde câte un fișier JavaScript în care este descris atât comportamentul paginii în funcție de apelurile către server și răspunsurile primite de la acesta. Paginile JavaScript sunt structurate ca

o funcție ce returnează codul HTML, elementele acestuia având în marea majoritate CSS-ul injectat, prin atributul "style". Aspectul paginilor este unul simplist, dar totuși funcționalitățile sunt ușor de utilizat și intuitive.

Rutarea aplicației are loc în fișierul "App.js", unde pentru fiecare URL al aplicației, se face redirectarea către componenta specifică, ce returnează pagina.

## 5. Modulul de utilizator



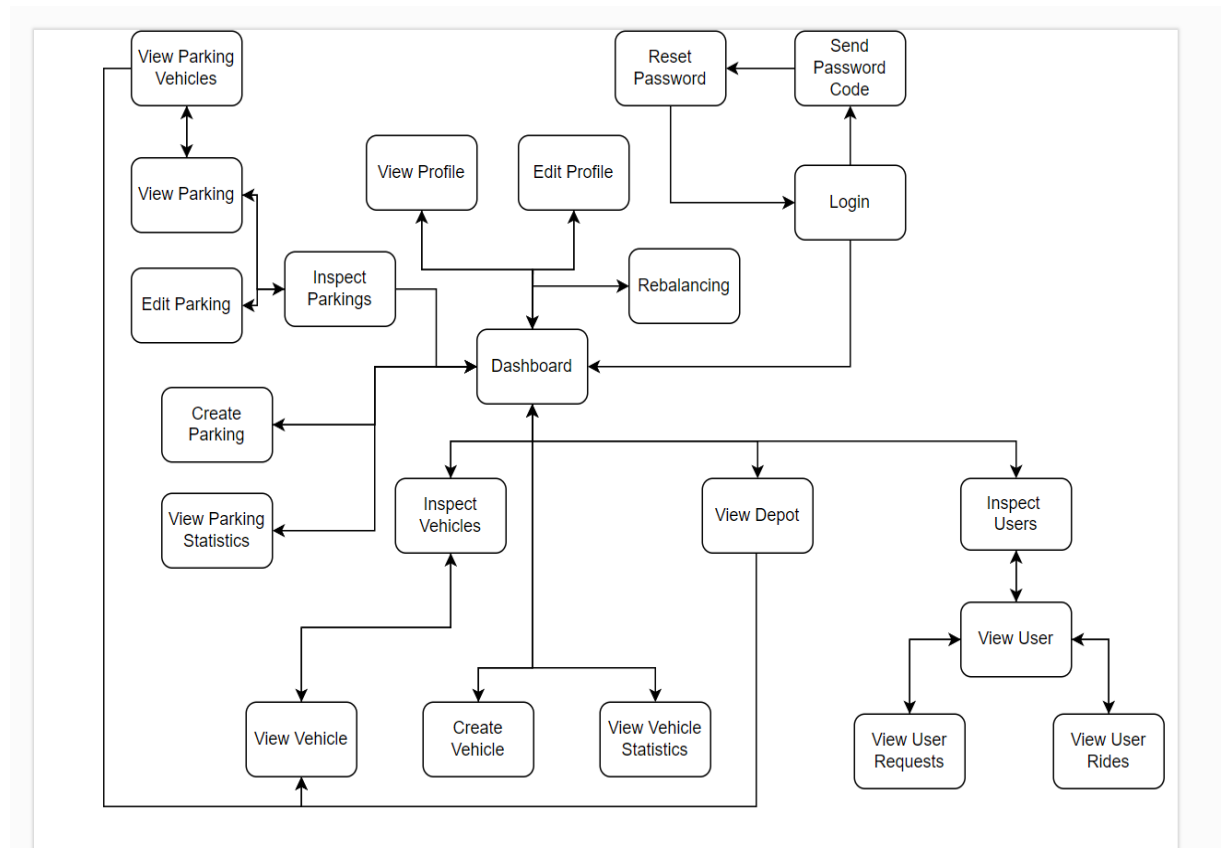
Modulul de utilizator are 18 pagini printre care utilizatorul poate naviga.

În diagrama de mai sus sunt prezentate fluxurile principale între paginile aplicației din modulul de utilizator. Majoritatea acestor pagini conțin butoane și casete de text. Prin apăsarea butoanelor, se preia conținutul casetelor de text, se fac anumite verificări și apoi se face *fetch* la API-ul de la aplicația server. În funcție de răspunsul primit, se afișează un text sau se redirecționează la o altă pagină. Unele dintre pagini, cum ar fi "Dashboard", "Inspect Parkings", "View Parking", "Create Request" conțin harta și marcatoarele uneia sau a mai multor parcuri, preluate

cu ajutorul API-ului de la Google Maps.

Mai multe amănunte despre paginile acestea și modul lor de funcționare sunt descrise în capitolul următor.

## 6. Modulul de administrator



Modulul de administrator are 22 pagini printre care administratorul poate naviga. Fiind prestabilit, nu e nevoie de pagina cu verificarea e-mailului și pagina de înregistrare.

În diagrama de mai sus sunt prezentate fluxurile principale între paginile aplicației din modulul de administrator. Majoritatea acestor pagini conțin butoane și casete de text. Prin apăsarea butoanelor, se preia conținutul casetelor de text, se fac anumite verificări și apoi se face *fetch* la API-ul de la aplicația server. În funcție de răspunsul primit, se afișează un text, diagrame sau se redirecționează la o altă pagină. Unele dintre pagini, cum ar fi "Dashboard", "Inspect Parkings", "View Parking", "Rebalancing", "Parking Statistics" conțin harta și marcatoarele uneia

sau a mai multor parări, preluate cu ajutorul API-ului de la Google Maps.

Mai multe amănunte despre paginile acestea și modul lor de funcționare sunt descrise în capitolul următor.

## **Capitolul 4**

### **Manual de instalare și utilizare**

## Concluzii

# Bibliografie

- Author1, *Book1*, 2018
- Author2, *Boook2*, 2017