

**UNIVERSITATEA BABEȘ-BOLYAI CLUJ-NAPOCA**

**FACULTATEA DE MATEMATICĂ ȘI  
INFORMATICĂ**

**SPECIALIZAREA MATEMATICĂ-INFORMATICĂ**

**LUCRARE DE LICENȚĂ**

**Gestionarea locurilor de parcare**

**Conducător științific**

**Lector Univ. Dr. Vasile Cioban**

**Absolvent**

**Katuna Norbert**

**2019**

## Cuprins

Cap.0.Introducere .....	3
Cap.1. Tema, istoric .....	5
1.1    Tema .....	5
1.2    Istoric .....	5
Cap.2. Analiza aplicației .....	8
2.1 Specificarea exactă a problemei .....	8
2.2 Analiza problemei (deducerea subproblemei și stabilirea arhitecturii, FE/BE) .....	8
2.2.1 Deducerea subproblemei .....	8
2.2.2 Stabilirea arhitecturii .....	9
2.3 Limbaje folosite .....	13
2.3.1 Back-End .....	13
2.3.2 Front-End .....	21
Cap 3. Proiectare .....	27
3.1 Diagrama de clase .....	29
3.2 Diagrama cazurilor de utilizare .....	31
3.3 Diagrama de interacțiune .....	32
3.4 Diagrama bazei de date .....	34
3.5 Diagrama de pachete .....	35
Cap 4. Utilizarea aplicației .....	36
4.1 Utilizare aplicației de către client .....	38
4.2 Utilizare aplicației de către administrator .....	44
Cap 5. Concluzii .....	47
Bibliografie .....	49

## Cap.0.Introducere

Odată ce m-am mutat în Cluj-Napoca ca și student, un oraș mare și aglomerat, fiind al patrulea oraș din România ca și număr de locuitori, având o populație de peste 320.000 de locuitori, prima problemă pe care am sesizat-o la acest oraș este problema locurilor de parcare. Încă din primele zile principala greutate la deplasarea prin Cluj-Napoca este găsirea unui loc de parcare liber, care să fie aproape de locul în care doresc să mă deplasez, de aceea am decis să rezolv această problemă prin crearea unui soft pentru gestionarea locurilor de parcare cu scopul de a permite rezervarea din timp al unui loc de parcare, atât în Cluj-Napoca cât și în celelalte orașe mari din țara noastră care întâmpină această problemă.

Primul capitol conține informații despre istoria locurilor de parcare, când au început să apară locurile de parcare, cum s-au dezvoltat și cum au ajuns să devină o problemă în zilele de azi, în principal pentru marile orașe din România. De asemenea sunt descrise problemele legate de locurile de parcare împreună cu soluțiile pe care aplicația aceasta le-ar aduce în rezolvarea acestora. Capitolul conține și exemple de aplicații care rezolvă parțial această problema, și au reprezentat o sursă de inspirație pentru mine.

Al doilea capitol conține o descriere a problemei generale pe care aplicația dorește să o rezolve și descompunerea acesteia în subprobleme împreună cu soluțiile care sunt aduse acestora. Este prezentată arhitectura generală pe care se va construi aplicația dar și alte arhitecturi care vor fi folosite pe partea de backend sau frontend. În cele din urmă, capitolul va conține o parte de descriere a limbajelor de programare folosite, istoricul lor, motivul pentru care am ales să folosesc limbajul respectiv dar și mecanisme deosebite precum și alte programe folosite.

Capitolul trei conține date legate de proiectarea aplicației, care a fost realizată cu ajutorul diagramelor UML. O scurtă descriere generală a tipurilor de diagrame folosite și includerea fiecărei diagrame cu ajutorul unei imagini și al unei descrieri concrete pe aplicație.

În capitolul patru am prezentat modul în care se utilizează aplicația. Sunt prezentate funcționalitățile, prin parcurgerea aplicației de către utilizator de la crearea de cont până la delogarea acestuia. Modul de utilizare este împărțită pe două tipuri de utilizatori care sunt clientul și administratorul. Pentru o mai bună înțelegere a aplicației prezentarea conține atât text cât și imagini.

Capitolul cinci, ultimul, conține o descriere a problemelor care nu au fost abordate sau care nu au fost complet acoperite de către o soluție potrivită, și posibile îmbunătățiri care se pot realiza în viitor pentru a face aplicația mai captivantă, mai ușor de utilizat și mai folositoare prin adăugarea de funcționalități noi.

Noutatea acestei lucrări constă în aplicație.

Lucrarea aceasta este rezultatul propriei mele activități. Nu am primit asistență neautorizată pentru această lucrare.

# Cap.1. Tema, istoric

## 1.1 Tema

Tema acestei lucrări este dezvoltarea unui soft care rezolvă probleme legate de găsirea și folosirea unui loc de parcare în marile orașe din România.

## 1.2 Istoric

Problema locurilor de parcare în țara noastră este destul de recentă. Aceasta a început odată cu modernizarea orașelor cu aproximativ 20 de ani în urmă prin creșterea locurilor de muncă, dezvoltare nivelului de studiu, astfel oamenii aspirând la o viață mai bună în mediul urban. Această dorință a dus la creșterea nivelului de populație în marile orașe ale României. În același timp și Industria Automobilitică a avut o creștere mare, iar din ce în ce mai mulți oameni au început să folosească mașinile ca principal mijloc de deplasare. Acești factori au ca și consecință un număr ridicat de locuitori în marile orașe, fiecare dintre aceștia deținând unul sau mai multe autovehicule, iar numărul locurilor de parcare a rămas aproximativ același.

Un alt factor care a dus la scăderea numărului locurilor de parcare și intensificare problemelor este distrugerea spațiilor de parcare pentru a se construi spații pietonale.

În acest context numărul locurilor de parcare disponibile este cu atât mai mic, și mai greu de găsit cu cât acestea sunt mai aproape de principalele locuri de interes.

Aplicația va avea ca scop principal rezolvarea problemelor legate de parcarile pe o durată scurtă. În majoritatea situațiilor oamenii doresc să folosească un loc de parcare pentru o perioadă scurtă de timp. De multe ori se întâmplă să ajungem într-un asemenea loc de interes în care se găsesc mulți oameni care utilizează locul de parcare pentru o perioadă mai scurtă de timp și vedem că toate locurile sunt ocupate, astfel trebuie să mergem mai departe pentru a găsi un loc de parcare sau să așteptăm să se elibereze un loc, lucru care în multe cazuri nu ne este permis. Însă s-ar putea ca la câteva minute după ce plecăm noi să se

elibereze un loc de parcare. Dacă toate acestea s-ar realiza cu ajutorul unei aplicații care să permită ca fiecare persoană să știe în ce moment este disponibil un loc de parcare și să-l poară rezerva, am putea evita această așteptare și să câștigăm timp.

Un alt avantaj ar fi accesul ușor la datele locului de parcare pe care dorim să-l folosim precum prețul și intervalul de ore în care se poate parca fără plată. De multe ori trebuie să parcăm mașina și să căutăm locul în care sunt plasate informațiile despre locul de parcare. Astfel ajungem să realizăm faptul că parcare nu este potrivită pentru necesitatea noastră. Prin urmare trebuie să repetăm același lucru într-un alt loc, sau aceste informații s-ar putea să nu existe și va trebui să întrebăm persoanele care se află în trecere sperând că ei vor ști aceste informații.

O altă problemă pe care aplicația o tratează este problema automatelor. Majoritatea acestora accepta doar monede, iar oamenii sunt nevoiți să caute un loc în care să poată schimba bancnotele în monede, mai ales când este nevoie de un bilet pe o durată mai lungă de timp. Cu ajutorul acestei aplicații oamenii plătesc prin intermediul cardurilor bancare pentru o plată cât mai rapidă și sigură.

În acest moment la noi în țară nu există un soft care să rezolve această problemă în totalitate, însă există aplicații care rezolvă o parte din problemă și anume cumpărarea de bilete, oferind posibilitatea de a achiziționa biletele online prin intermediul unor aplicații. Exemple de astfel de aplicații sunt: **SMS Parking**, **TPARK**. Pe de altă parte în alte țări există aplicații care conțin funcționalitățile descrise mai sus și astfel încep să devină din ce în ce mai populare. Aplicațiile care m-au inspirat au fost **RingGo** și **JustPark**.

**RingGo** este o aplicație care se adresează în principal telefoanelor mobile neglijând partea de web, oferind o interfață utilizator mult mai intuitivă pe mobile. Aplicația este folosită în Marea Britanie și oferă posibilitatea găsirii unui loc de parcare în funcție de locația curentă sau de o locație dorită oferind informații despre locul de parcare precum:

- Prezentarea locației prin adresă și hartă cu ajutorul coordonatelor.
- Cât de tare este solicitată parcare.
- Prețul pe oră și prețul total după ce utilizatorul alege intervalul de timp.

Pentru a rezerva locul de parcare dorit, utilizatorul trebuie să fie înregistrat, iar apoi să aleagă un loc de parcare și să specifice informațiile autovehiculului pe care urmează să-l parcheze. Pentru a plăti trebuie să se utilizeze un card bancar deoarece plata se realizează online.

**JustPark** este folosită tot în Marea Britanie cu un număr de 2.5 milioane de utilizatori și 45,000 de locuri de parcare. Aplicația are o interfață plăcută atât pentru web cât și pentru mobile. Oferă în principal aceleași funcționalități ca și RingGo, însă spre deosebire de aceasta JustPark oferă și informații suplimentare despre locul de parcare precum:

- Parcare este acoperită sau nu.
- Cât de sigur este locul de parcare.
- Dacă locul de parcare conține camere video.

## **Cap.2. Analiza aplicației**

### **2.1 Specificarea exactă a problemei**

Problema pe care aplicația dorește să o rezolve este găsirea și folosirea unui loc de parcare pentru o anumită perioadă de timp în orașele mari din România. În principal pentru locurile de interes crescut din orașele respective, într-un mod cât mai eficient astfel încât oamenii să câștige timp, să plătească mult mai ușor pentru locurile de parcare, iar traficul din orașe să fie cât mai scăzut.

### **2.2 Analiza problemei (deducerea subproblemei și stabilirea arhitecturii, FE/BE)**

#### **2.2.1 Deducerea subproblemei.**

Problema locurilor de parcare este foarte amplă și conține foarte multe aspecte, prin urmare am încercat să aduc soluții la problemele mai semnificative, urmând să aduc soluții și pentru celelalte aspecte în viitoarele actualizări ale aplicației.

Subproblemele pe care aplicația le abordează și pentru care aceasta aduce soluții sunt următoarele:

1. Găsirea unui loc de parcare. Aplicația rezolvă problema găsirii unui loc de parcare prin oferirea unei liste cu spațiile care folosesc această aplicație. Utilizatorii pot să caute locurile de parcare dintr-un anumit oraș și să afle informații utile care să-i ajute la găsirea locului respectiv precum adresa și vizualizarea pe o hartă a locației în care se află locul de parcare. De asemenea utilizatorii pot să vadă într-un anumit interval de timp, dacă o anumită parcare are sau nu are locuri libere prin prezentarea numărului de locuri libere din acel interval de timp.

2. Prezentarea informațiilor legate de prețul parării. Cu ajutorul aplicației utilizatorii vor putea să găsească prețul pe ora și prețul total pe care trebuie să-l plătească pentru intervalul de timp dorit.



3. Cumpărarea de tichete. Utilizatorii vor primi un cod QR în urma realizării rezervării unui loc de parcare, la care vor avea acces în orice moment. Astfel, aceștia nu vor mai avea nevoie să caute o mașină de tichete sau să riște să-și piardă tichetul ci vor putea să intre în locul de parcare prin intermediul codului QR primit. Acest cod se poate accesa atât din aplicație cât și prin intermediul unui email primit la momentul creării unei rezervări.

4. Realizarea plății. Aplicația permite realizarea plății online cu un card de PayPal, astfel utilizatorii nu sunt nevoiți să caute un anumit loc în care să plătească sau să fie nevoiți să dețină valoarea exactă de bancnote sau monezi pentru realizarea plății.

5. Reducerea traficului și diminuarea poluării. Prin oferirea posibilității de a vizualiza disponibilitatea locurilor de parcare și prin rezervarea din timp al acestora persoanele nu vor trebui să conducă de la o parcare la alta până în momentul în care reușesc să găsească un loc de parcare mărind astfel traficul și poluarea. Aceștia își vor rezerva un loc de parcare într-un moment în care acesta este disponibil iar în cazul în care nu există locuri libere aceștia nu vor mai deplasa la spațiul parcare.

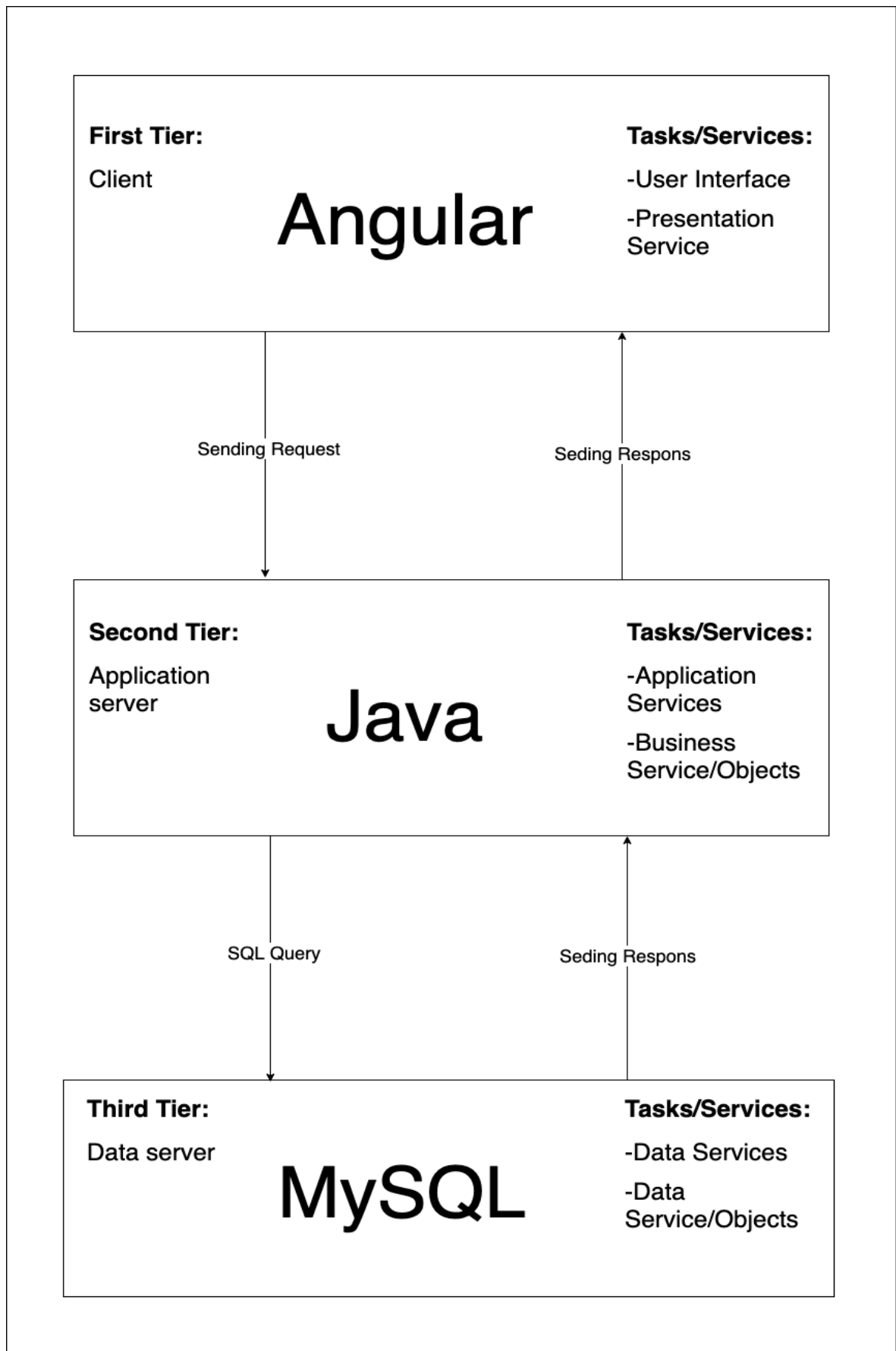
### **2.2.2 Stabilirea arhitecturii.**

În cadrul realizării acestei aplicații a fost folosită arhitectura **Client-Server**. Această arhitectură are rolul de a partaja procesarea dintre cei care oferă servicii, numiți servere și elementele care solicită servicii care se numesc clienți. Clienții și serverele comunică printr-o rețea de calculatoare, de obicei prin Internet, având suporturi hardware diferite, dar pot rula și pe același sistem fizic.

Un server rulează unul sau mai multe programe server, care partajează resursele existente cu clienții. Clientul nu partajează niciuna dintre resursele proprii, ci apelează la resursele serverului prin funcțiile server.

Clienții inițiază comunicația cu serverele și așteaptă mesajele acestora. Pentru menținerea legăturii dintre cei doi, indiferent de pauzele care intervin, se folosește conceptul de sesiune, care de obicei este limitată în timp. [2]

În cadrul acestei aplicații arhitectura **Client-Server** este implementată astfel: clientul care este reprezentat de parte de Frontend, care comunică cu serverul care este reprezentat de partea Backend prin intermediul protocolului HTTP, aceasta trimite un request pe baza unui url și primește un răspuns. Backendul de asemenea comunică cu un alt server care este baza de date prin intermediul query-urilor. Această comunicare este întreținută printr-o sesiune implementată pe partea de Backend care este activă în timpul în care utilizatorul este logat.



Pentru partea de Frontend am folosit **Angular**, un framework care urmează arhitectura **MVC**. Această arhitectură are scopul de izola partea de logică de partea de interfață cu utilizatorul rezultând astfel o aplicație în care aspectul vizual și partea de logică sunt mai ușor de modificat, fără a afecta și alte nivele. Arhitectura împarte aplicația în trei părți interconectate:

- **Model.**

Această parte se ocupă cu manipularea operațiunilor logice și de utilizare a informației pentru a ajunge la o formă ușor de înțeles.

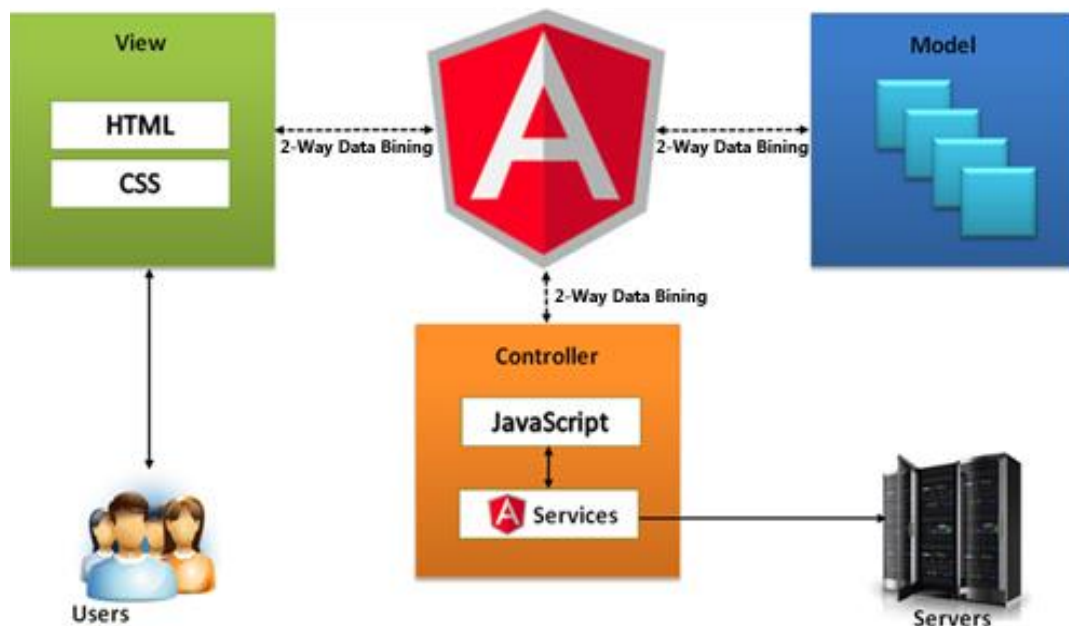
- **View.**

Acestuia îi corespunde partea de reprezentare grafică, de prezentare a informației în forma sa finală utilizatorului. Rolul este acela de a evidenția informația obținută de la controller.

- **Controller.**

Această componentă face legătura dintre celelalte două părți, partea de Model și View și controlează accesul la aplicație.

În **Angular** arhitectura **MVC** este realizată astfel : **View**-ul este reprezentat de pagina **HTML** prin intermediul căruia utilizatorul interacționează cu aplicația, **Controllerul** este reprezentant de partea de **TypeScript** a componentelor care face legătura dintre obiecte, interfață și server decizând când ce obiecte să folosească și ce informații să ceară de la server, iar partea de **Model** este reprezentat de obiectele din interiorul aplicației.



[9]

## 2.3 Limbaje folosite

### 2.3.1 Back-End

Pe partea de **Back-End** am folosit următoarele tehnologii:

- Server.
- Baza de date.
- Java.
- Hibernate.
- Jersey.
- Maven.
- JavaMail.

#### • Server

Am folosit un server local numit **Apache Tomcat**, un web server open source și container servlet dezvoltat de Apache Software Foundation. Versiunea folosită este 9.0. Am ales să folosesc acest server deoarece este gratuit, este un server stabil, simplu și rapid.[21]

- **Baza de date**

Pentru stocarea datelor am folosit **MySQL**. Aceasta este un sistem de gestionare a bazelor de date relaționale distribuit sub Licență Publică Generală. Este cel mai popular sistem de gestionare a bazelor de date open-source la ora actuală. Există multe scheme API disponibile pentru **MySQL** ce permit scrierea aplicațiilor în numeroase limbaje de programare pentru accesarea bazelor de date precum: C, C++, C#, Java, Perl, PHP, Python, FreeBasic, etc., fiecare dintre acestea folosind un tip specific API.

Pentru a administra baza de date MySQL am folosit interfața grafică **MySQL Workbench** cu scopul de a avea o menținere mai ușoară a acesteia.[18]

- **Java**

**Java** este un limbaj de programare orientat-obiect, puternic tipizat, conceput de către James Gosling la Sun Microsystems la începutul anilor 90, fiind lansat în 1995. Cele mai multe aplicații distribuite sunt scrise în **Java**, iar noile evoluții tehnologice permit utilizarea sa și pe dispozitive mobile gen telefon, agenda electronică, palmtop etc. În felul acesta se creează o platformă unică, la nivelul programatorului, deasupra unui mediu eterogen extrem de diversificat. Acesta este utilizat în prezent cu succes și pentru programarea aplicațiilor destinate intranet-urilor.

Limbajul împrumută o mare parte din sintaxă de la **C** și **C++**, dar are un model al obiectelor mai simplu și prezintă mai puține facilități de nivel jos. Un program Java compilat, corect scris, poate fi rulat fără modificări pe orice platformă care e instalată o mașină virtuală Java (engleză Java Virtual Machine, prescurtat JVM). Acest nivel de portabilitate (inexistent pentru limbaje mai vechi cum ar fi C) este posibil deoarece sursele Java sunt compilate într-un format standard numit cod de octeți (engleză byte-code) care este intermediar între codul mașină (dependent de tipul calculatorului) și codul sursă.

Mașina virtuală Java este mediul în care se execută programele Java. În prezent, există mai mulți furnizori de JVM, printre care:

- Oracle.
- IBM.
- Bea.
- FSF.

În 2006, Sun a anunțat că face disponibilă varianta sa de JVM ca open-source.

Am ales să folosesc Java deoarece este un limbaj de programare orientat obiect ceea ce face programare mult mai ușoară iar codul mult mai organizat. De asemenea acest limbaj conține multe framework-uri din care poți să alegi după propria preferință și care acoperă multe nevoi.[14]

Pentru o structură logică și organizată a codului Java, am folosit design pattern-ul **MVC**. **MVC** este o abreviere pentru Model-View-Controller. Acest pattern este folosit pentru a separa rolurile din interiorul aplicației.

**Model**-ul în Java reprezintă un obiect care conține informație. De asemenea poate să conțină logica necesară pentru a face actualizări în controller în cazul în care datele ei se schimbă.

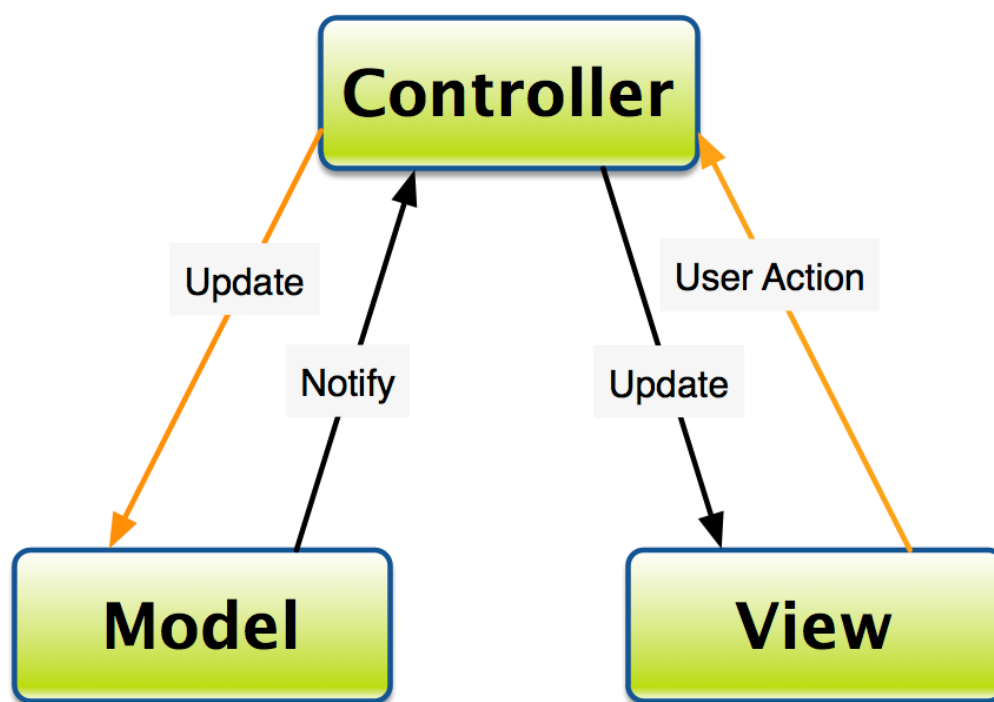
**View**-ul reprezintă partea vizuală, modul în care sunt prezentate datele pe care parte Model le conține utilizatorului.

**Controller**-ul acționează atât asupra Model-ului cât și asupra View-ului. Controlează cursul de date spre interiorul model-ului și actualizează View-ul în cazul în care datele suferă o schimbare. Acesta ține celelalte două componente separate una de cealaltă.

În implementare aplicației, partea de **Model** este reprezentată de clasele care conțin și grupează date precum:

- Utilizator
- Loc de parcare
- Clasele care conțin metode cu query-uri care au rolul de a comunica cu baza de date și de a trimite datele obținute Controller-ului.

Partea de **Controller** conține clasele care conțin url-urile la care se fac cereri din parte de View. Aceste clase din Controller sunt împărțite în cereri care au loc pentru anumite obiecte care se regăsesc în parte de Model, spre exemplu clasa **UserController** va conține toate cererile legate de utilizatori. Partea de View nu este implementată în Java deoarece de această parte se va ocupa framework-ul Angular care va trimite cereri spre Controller-ul din Java și se va ocupa de afișare răspunsului primit utilizatorului.



[11]

Ca și mediu de dezvoltare am folosit **Eclipse**, un mediu de dezvoltare open-source scris preponderent în Java.

Motivele pentru care am ales acest mediu de dezvoltare sunt:

- ușurința găsirii erorilor, în momentul în care apare o eroare în cod, cu ajutorul unui click pe eroare poți să ajungi la linia în care se află eroarea respectivă.
- ușurința descărcării unor pachete sau extensii și autocompletarea codului. Acesta este de mare ajutor în momentul în care dorești să folosești o metoda a unei clase deoarece prin inserarea primului caracter al acesteia vei obține o listă cu toate metodele clasei respective care încep cu acea literă.



- **Hibernet**

Hibernate ORM(Object-Relational Mapping) este un instrument pentru limbajul de programare Java. Acesta oferă un framework care permite aplicației să acceseze baza de date prin asocierea claselor din Java cu tabelele din baza de date. Hibernate se ocupă de problemele legate de nepotrivirea impedanței relaționale obiect prin înlocuirea unui acces direct și persistent la baza de date cu funcții de nivel înalt care se ocupă de gestionarea obiectelor.

Crearea conexiunii se realizează prin crearea unui fișier cu extensia XML și prin adăugare de adnotații claselor cărora doresc să le fie asociate tabelele din baza de date. În fișierul XML se specifică datele de conexiune cu baza de date care conțin: url-ul, numele utilizatorului și parola și locația claselor care reprezintă tabelele din baza de date

Am ales să folosesc acest framework deoarece este gratuit și oferă posibilitatea de a crea query-uri și de a actualiza datele unei baze de date relaționale.[7]

- **Jersey**

Pentru limbajul Java am folosit ca și framework open-source **Jersey** care este folosit pentru programare serviciilor web de tip REST.

Am ales acest framework pentru ușurința creării serviciilor de tip REST cu ajutorul adăugării adnotațiilor:

- @GET
- @PUT
- @DELETE
- @POST

De asemenea este ușoară setarea tipului de dată produs care pot fii de tip text **HTML** sau **JSON** și tipul de dată pe care-l consumă tot cu ajutorul adnotațiilor. În concluzie acest framework simplifică crearea serviciilor **REST** pe care ulterior pot fii accesate prin intermediul request-urilor din partea de frontend făcând mai ușoară comunicarea Client-Server.[15]

- **Maven**

O altă tehnologie folosită este **Maven**. Maven este un sistem de build și administrare a proiectelor, scris în **Java**. Face parte din proiectele găzduite de Apache Software Foundation. Funcționalitățile sale principale sunt descrierea procesului de build al software-ului și descrierea dependențelor acestuia. Proiectele sunt descrise printr-unul sau mai multe fișier **XML**, denumite **POM-uri** (Project Object Model), dar au o structură implicită, ceea ce încurajează structurarea similară a proiectelor. POM-ul principal conține informații despre module, precum și despre dependențele proiectului (alte proiecte). Ordinea operațiunilor de build este definită prin declararea unor pluginuri folosite, din cadrul cărora unele scopuri sunt plasate și configurate în diferitele faze predefinite din ciclul de viață al unui build. Maven descarcă dinamic bibliotecile Java și pluginurile, din unul sau mai multe repository-uri.

Conceptul central în **Maven** este acela de **POM** (Project Object Model), un document **XML** care descrie un proiect sau un modul al acestuia. Proiectele multimodulare au un POM pentru fiecare modul, precum și un POM general care le agregă. În mod minimal, POM-ul unui proiect îi definește o denumire unică (formată din `groupId` care reprezintă id-ul grupului de proiect și `artifactId` care este id-ul proiectului) și o versiune. Componentele denumirilor trebuie să asigure unicitate proiectului, astfel că s-a răspândit pe scară largă convenția ca `groupId` să fie inversul unui nume complet calificat de domeniu, la care se adaugă elemente comune unui set de proiecte ale persoanei sau firmei care le dezvoltă, similar cu numele package-ului în care sunt grupate clasele **Java**. `groupId` poate fi sau nu identic cu un package părinte al claselor din proiect.

Un proiect care definește aceste denumiri și versiunea este un proiect minimal pe care **Maven** îl poate construi fără alte elemente de bază, inferând toate celelalte configurații necesare din convenții, implementate într-un așa-numit „Super POM”.

Proiectele din viața reală mai au însă și alte elemente incluse în **POM**, care completează sau suprascriu unele configurații ale Super POM-ului.

Pentru un proiect **Java**, convențiile sunt acelea de a compila toate sursele din subdirectorul `src/main/java`, de a alătura fișierelor *bytecode* rezultate toate fișierele din `src/main/resources` și apoi de a le archiva într-un fișier cu extensia *jar*. Se compilează apoi testele automate definite în `src/test/java` și se rulează având în classpath și resursele statice (fișierele) din `src/test/resources`. Dacă toți pașii au reușit, la final, în subdirectorul *target* al directorului proiectului va apărea fișierul `artifactId-version.jar`.

Una din facilitățile pentru care a fost dezvoltat **Maven** de la bun început, ca software diferit de celălalt proiect al Fundației **Apache** dedicat buildurilor, Ant, l-a constituit gestiunea dependențelor. Proiectele care depind de alte biblioteci puneau probleme mari de gestiune a acestor dependențe: întreținerea classpath-urilor diferite la compilare, rulare, compilare de teste și rulare de teste, la care se adăuga problema dependențelor tranzitive: biblioteci care depind de alte biblioteci de care depind alte biblioteci, într-un arbore cu adâncime care poate fi mare și a cărui integritate (versiunile corecte, eliminarea duplicatelor și a versiunilor diferite ale aceleiași biblioteci din arborele de dependențe) putea deveni un coșmar. Astfel, **Maven** a fost construit în jurul acestui concept, că fiecare modul produce după build un așa-numit *artifact*. Artifactul este un fișier care împachetează codul livrabil al modulului, care poate fi reutilizat mai departe de alte module.

Când se cere un artifact dependent de modulul buildat, **Maven** este capabil să identifice dependențele tranzitive și să includă toate bibliotecile care depind de cea cerută. Dacă există dependențe care se folosesc doar la anumite operațiuni (de exemplu, cele pentru teste sau necesare doar pentru compilare), ele se pot declara împreună cu un `scope` care oferă lui **Maven** această informație; artifactele astfel declarate se folosesc

doar în momentele la care este nevoie. Scope-urile principale sunt:

- **compile** (cel implicit, care înseamnă că artifactul este prezent în toate etapele);
- **runtime** (artefacte care trebuie să fie prezente la rulare și testare, dar nu și la compilare — de exemplu, implementări ale API-urilor folosite);
- **provided** (similar cu runtime, artefacte despre care există așteptarea că vor exista la rulare, dar care urmează să fie furnizate de către containerul care rulează aplicația sau de către JDK);
- **test** (artefacte necesare testării codului, care însă nu trebuie distribuite și nu sunt necesare la rulare). Alte două scope-uri (**system** și **import**) sunt disponibile pentru unele cazuri speciale, mai rare.[16][17]

Am decis să folosesc în principal pentru a ușura descărcarea și instalarea dependențelor.

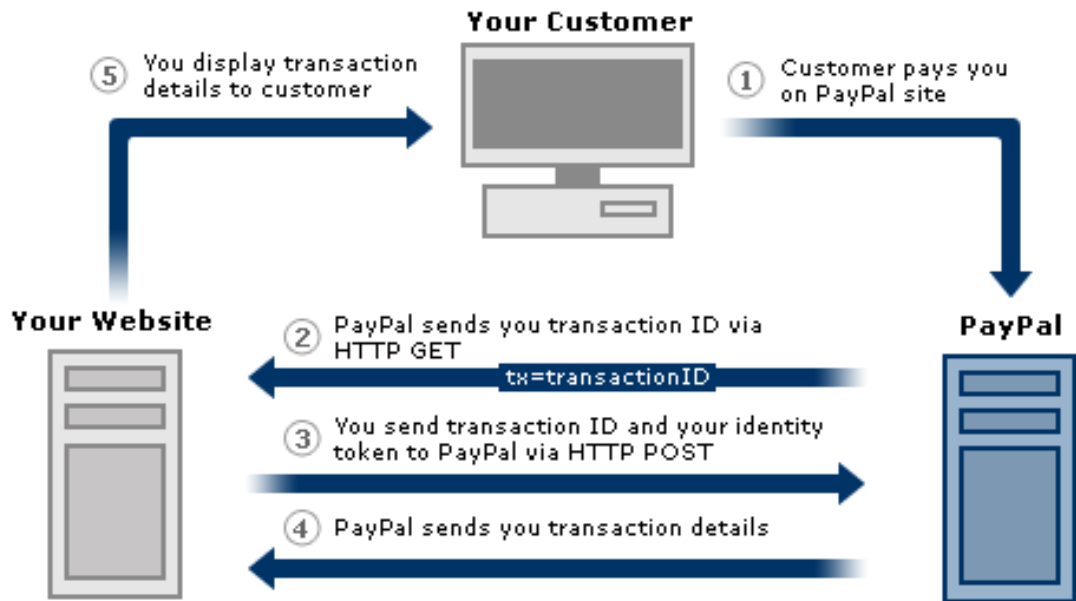
- **PayPal**

**Payment REST API** este folosit pentru o ușoară și sigură acceptare al unei plăți online de pe o pagină web sau un telefon mobil. Se poate oferi utilizatorilor posibilitatea de:

- a efectua plăți **PayPal** cu doar câteva click-uri.
- a primi informații despre plăți efectuate.
- a primi informații despre restituiri.

Cu ajutorul acestui API poți să efectuezi restituiri complete sau parțiale.

Adițional se poate folosi și **Payouts REST API** pentru a efectua și administra payout-uri, care sunt plăți către mai multe conturi PayPal. **Payout**-urile sunt metode rapide și convenabile pentru a trimite comision, reduceri, recompense și plăți generale. [20]



[12]

- **JavaMail**

JavaMail este un API Java folosit pentru a trimite și pentru a primi email-uri. Este construit în platforma Java EE, însă oferă și un pachet opțional pentru a putea fi folosit în Java SE. Versiunea folosită este 1.4 și este open source.

### 2.3.2 Front-End

Pe partea de **Front-End** am folosit tehnologiile:

- HTML.
- CSS.
- TypeScript.
- Angular.
- OpenStreetMap.
- Bootstrap.

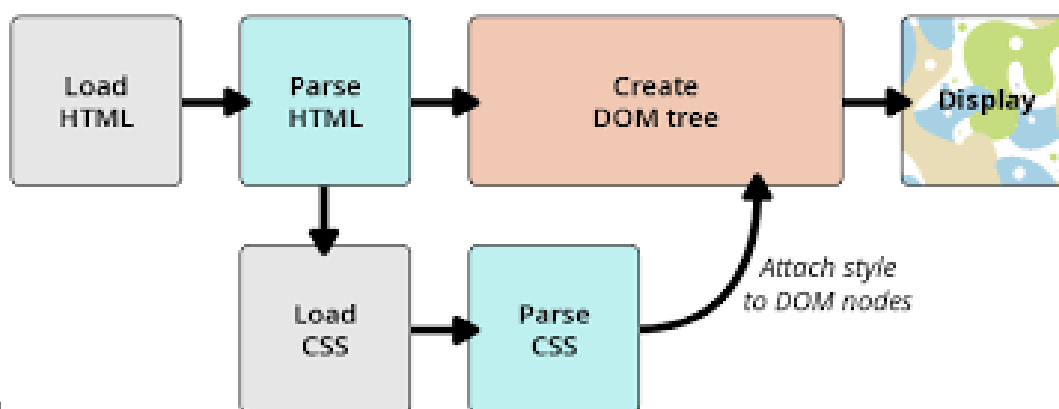
- **HTML**

Este un limbaj de marcare utilizat pentru crearea paginilor web ce pot fi afișate într-un browser. Scopul HTML este mai degrabă prezentarea informațiilor – paragrafe, fonturi, tabele ș.a.m.d. – decât descrierea semanticii documentului.

**HTML** este o formă de marcare orientată către prezentarea documentelor text pe o singura pagină, utilizând un software de redare specializat, numit agent utilizator HTML, cel mai bun exemplu de astfel de software fiind browserul web. HTML furnizează mijloacele prin care conținutul unui document poate fi adnotat cu diverse tipuri de metadata și indicații de redare. Indicațiile de redare pot varia de la decorațiuni minore ale textului, cum ar fi specificarea faptului că un anumit cuvânt trebuie subliniat sau că o imagine trebuie introdusă, până la scripturi sofisticate, hărți de imagini și formulare. Metadatale pot include informații despre titlul și autorul documentului, informații structurale despre cum este împărțit documentul în diferite segmente, paragrafe, liste, titluri etc. și informații cruciale care permit ca documentul să poată fi legat de alte documente pentru a forma astfel hyperlink-uri (sau web-ul).[8]

- **CSS(Cascading Style Sheets)**

Este un standard pentru formatarea elementelor unui document **HTML**. **CSS** este o tehnologie de baza a Web-ului alături de **HTML** și **JavaScript**. Aceasta este concepută pentru a face posibilă separarea între prezentare și conținut, incluzând layout-uri, culori și fonturi. Această separare poate îmbunătăți accesibilitatea conținutului, oferă mai multă flexibilitate și control în specificare caracteristicilor prezentației, face posibilă utilizare aceluiași format prin specificarea CSS-ului necesar într-un fișier separat având extensia **.css**, și reduce complexitatea și repetarea în conținutul structural.



[10]

**CSS** folosește selectori pentru a determina partea din pagină pentru care se aplică un anumit stil prin potrivirea tag-urilor și atributelor. Selectorii pot fi aplicați:

- tuturor elementelor unui tip specific, spre exemplu paragrafe, tabele, header-uri.
- elementelor specificate prin attribute precum id-ul care este un identificator unic în cadrul documentului sau clasa care este un identificator ce se poate referi la mai multe elemente din document.
- elementelor depinzând de modul în care sunt plasate relativ la alte elemente din document.[4]

- **TypeScript**

Este un limbaj de programare open source dezvoltat și menținut de Microsoft. Este un superset sintactic al limbajului **JavaScript** și asigură un sistem de tipuri opțional.

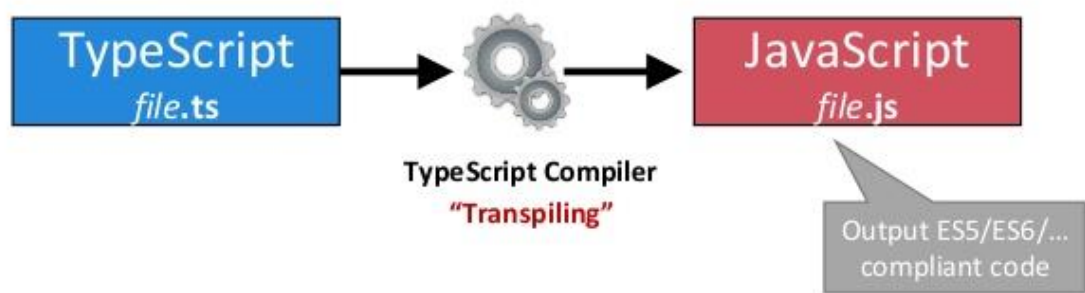
Acesta a fost făcut public pentru prima dată în Octombrie 2012, după doi ani de dezvoltare internă la Microsoft. La scurt timp după anunțare, limbajul a primit răspunsuri pozitive, însă a fost criticat pentru faptul că nu existau alte medii de dezvoltare care suportau acest limbaj în afară de **Microsoft Visual Studio** care nu era disponibil pentru Linux și OS X în acea perioadă. În ziua de azi limbajul este suportat și de alte medii de dezvoltare precum Eclipse. Varioase editare de text precum:

- Vim.
- Sublime.
- Webstorm.
- Atom.
- Visual Studio Code deținut de Microsoft

suportă TypeScript.

**TypeScript** este proiectat pentru dezvoltarea de aplicații de mari dimensiuni și se compilează în JavaScript. Deoarece TypeScript este un superset peste JavaScript programele JavaScript existente sunt, de asemenea, programe valide TypeScript. TypeScript poate fi utilizat atât pentru a dezvolta aplicații JavaScript pentru parte de client, cât și pentru parte de server.[22]

# How Does TypeScript Work?



[13]

- **Angular**

Este o platformă și un framework folosit pentru crearea aplicațiilor de tip client combinând tehnologiile **HTML**, **CSS** și **TypeScript**. Angular este scris în TypeScript. Acesta implementează funcționalități nucleu și de tip opțional ca și un set de biblioteci TypeScript pe care-l importă în aplicația ta.

Principalele blocuri de construcție al unei aplicații Angular sunt **Modulele**, care furnizează un context de compilare pentru componente. Aceste module colectează cod asociat în seturi funcționale. O aplicație Angular este definită de către un set de astfel de module. Angular conține de asemenea componente care definesc view-ul, un set de ecrane din care aplicația poate să aleagă și să modifice în funcție de logica programului și date. **Componentele** obțin funcționalități care nu sunt asociate într-un mod direct cu view-ul prin folosirea serviciilor. **Serviciile** pot fi injectate în componente ca și dependențe, făcând codul modular și eficient. Atât componentele cât și serviciile sunt simple clase, cu decoratori care



marchează tipul lor și furnizează metadate care oferă informații programului despre cum trebuie să le folosească.

**Modulele** Angular diferă și completează modulele **JavaScript**. Un modul Angular declară un context de compilare pentru un set de componente care este dedicat unui domeniu al aplicației, unui flux de lucru, sau unui set apropiat de capabilități. Un modul poate asocia componentele cu care se află în legătură cu serviciile. Fiecare aplicație Angular are un root modul, denumit convențional **AppModule**, care furnizează mecanismul **bootstrap** care pornește aplicația. O aplicație tipică conține mai multe module funcționale. Asemenea modulelor din JavaScript, modulele din Angular pot importa funcționalități din alte module, și pot permite exportarea propriilor funcționalități de către alte module. Spre exemplu dacă vrei să folosești serviciul root în aplicația ta, trebuie să importi modulul Router. Organizând codul în diverse module funcționale ajută la întreținerea dezvoltării unei aplicații complexe, și, de asemenea, timpului necesar încărcării aplicației.

**Componentele.** Fiecare aplicație **Angular** are cel puțin o componentă, componenta root care conectează o componentă ierarhic cu pagina document object model(DOM). Fiecare componentă definește o clasă care conține date ale aplicației și logica, și este asociată cu modelul **HTML** care definește un view care trebuie afișat în mediul vizat. O componentă se creează prin adăugarea decoratorului „@Component()” unei clase.

**Serviciile** sunt clase care se creează în momentul în care o dată sau o logică nu este asociată unui view, și se dorește împărțire acestora de-a lungul componentelor. Definirea unei clase de tip service se face prin adăugarea decoratorului „@Injectable()”.[1]

Am ales să folosesc framework-ul **Angular** deoarece acesta unește toate fișierele **HTML**, **CSS** și **TypeScript** într-o singură aplicație complexă și le împarte într-un mod logic pe componente astfel încât fiecare componentă să reprezintă o pagină sau o secțiune a unei pagini conținând atât elementele din interfață cât și elementele de logică, astfel codul devenind mult mai ordonat și ușor de citit și modificat. Un alt motiv este reutilizarea codului prin moșteniri, care pot avea loc între componente sau prin crearea unei componente comune care poate fi inserată în interiorul mai multor pagini. De asemenea **Angular** conține multe librării de componente utile care pot fi importate.

- **OpenStreetMap**

Este un proiect colectiv, in regim open source, ce are ca scop construirea unei baze de date geografice globale cum ar fi atlasele rutiere, folosind atât date introduse manual având ca fundal imagini spațiale, cât și date colectate de pe dispozitivele **GPS**

**OpenStreetMap** acoperă toată rețeaua de drumuri europene care traversează România și o mare parte a drumurilor naționale ajungând la o acoperire de aprox. 90% din aceste drumuri. [19]

Am folosit această hartă deoarece este open source și conține majoritatea drumurilor și clădirilor din România.

- **Bootstrap**

Este un framework open source gratuit, folosit pentru crearea de interfețe responsive pentru aplicații web, pe principiul mobile-first. Acesta conține șabloane bazate pe **CSS** și opțional **JavaScript** pentru:

- Tipografie
- Formulare
- Butoane
- Navigare

și alte componente pentru interfață. **Bootstrap** folosește clase predefinite pentru a aplica stilizări predefinite în **CSS** și **JavaScript** elementelor din **HTML**. [3]

Am ales să folosesc **Bootstrap**, pentru a crea o interfață care să se poată adapta în funcție de dispozitivul pe care rulează aplicația, în principal pentru telefoanele mobile și desktop, dar și pentru a adăuga unele șabloane predefinite elementelor din **HTML**.

## Cap 3. Proiectare

### Diagrame UML.

O diagramă UML este o diagramă bazată pe UML(Unified Modeling Language) cu scopul de a reprezenta vizual un sistem împreună cu principalii săi actori, artefacte, roluri, acțiuni, sau clase, cu scopul de a se înțelege, modifica sau menține informații despre sistem.

**UML** a fost creat ca rezultat al unei mod de programare și documentare dezordonat. În jurul anului 1990, au fost mai multe posibilități de reprezentare și documentare al unei aplicații. A apărut nevoia pentru o modalitate comună de a reprezenta vizual acele sisteme. În perioada 1994-1996, UML-ul a fost creat de către trei ingineri lucrând la Rational Software.[5]

#### Tipuri de diagrame **UML**:

- Diagrama de clase.

Diagrama de clase face parte din modelarea obiectuală a sistemului, descrie structura acestuia în termeni de obiecte, attribute, asocieri și operații.

Sunt reprezentate în UML folosind dreptunghiuri cu 3 compartimente, primul conține numele, al doilea attributele și ultimul operațiile. Ultimul poate fi omis.

- Diagrama cazurilor de utilizare

Acest tip de diagramă face parte din modelarea funcțională a sistemului, și descrie funcționalitatea sistemului din perspectiva utilizatorului. Cazurile de utilizare sunt folosite în cadrul activităților de colectare și analiză a cerințelor.

Diagrama cazurilor de utilizare este compusă din actori, cazuri de utilizare, dreptunghi și etichete cu denumirile aferente.

- Diagrama bazei de date

Diagrama bazei de date este o diagramă de modelare conceptuală a informației de nivel înalt. Se bazează pe notarea entităților din lumea reală împreună cu relațiile care se găsesc între ele. Aceasta te ajută să analizezi nevoile de date sistematic pentru a produce baze de date bine structurate.

- Diagrama de interacțiune

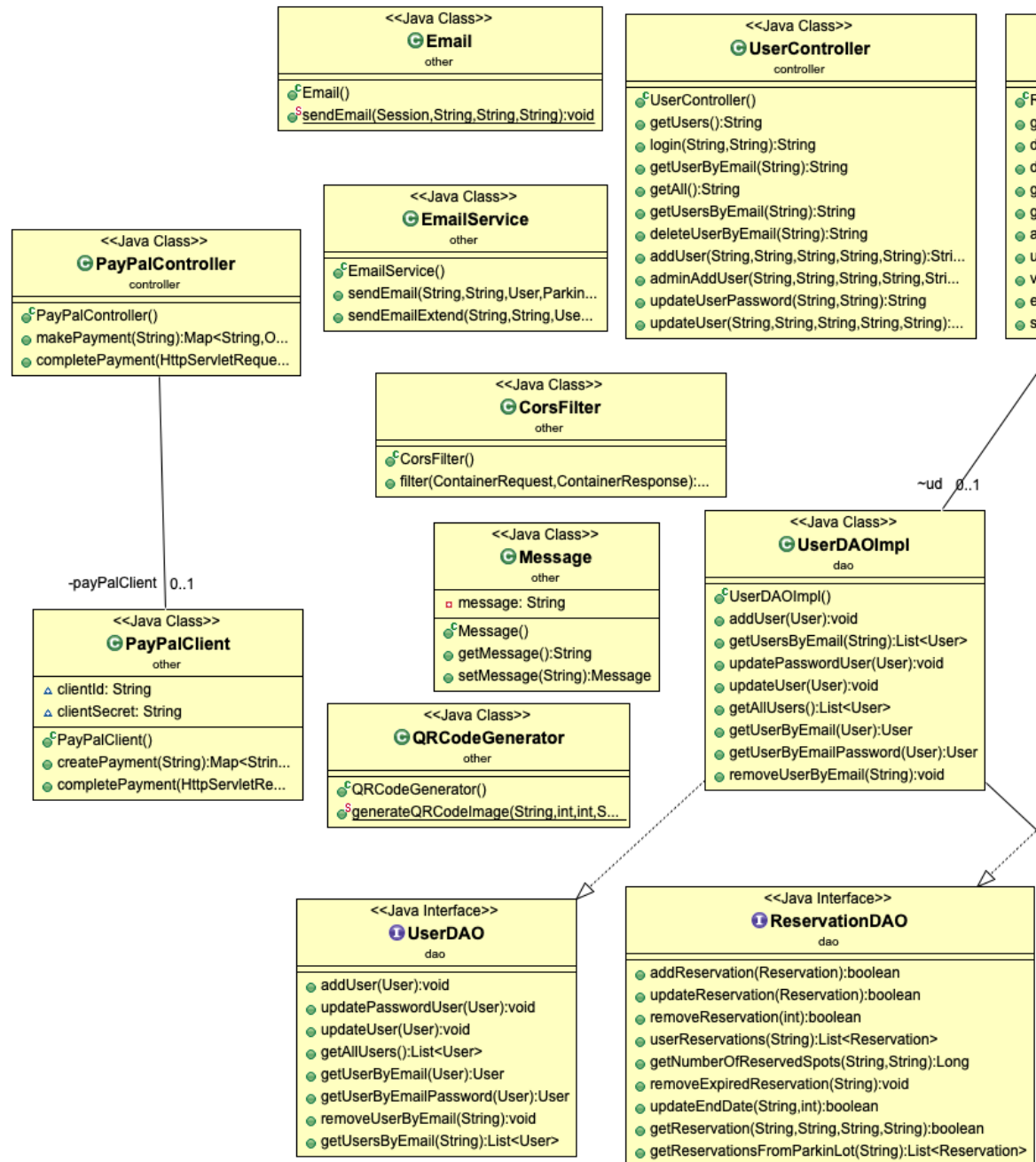
Face parte din modelarea dinamică a sistemului și descrie șabloanele de comunicare într-o mulțime de obiecte care interacționează. Obiectele interacționează între ele prin schimb de mesaje. Recepționarea unui mesaj de către un obiect declanșează execuția unei metode a obiectului în cauză, fapt ce determină, de obicei, trimiterea unor mesaje către alte obiecte.

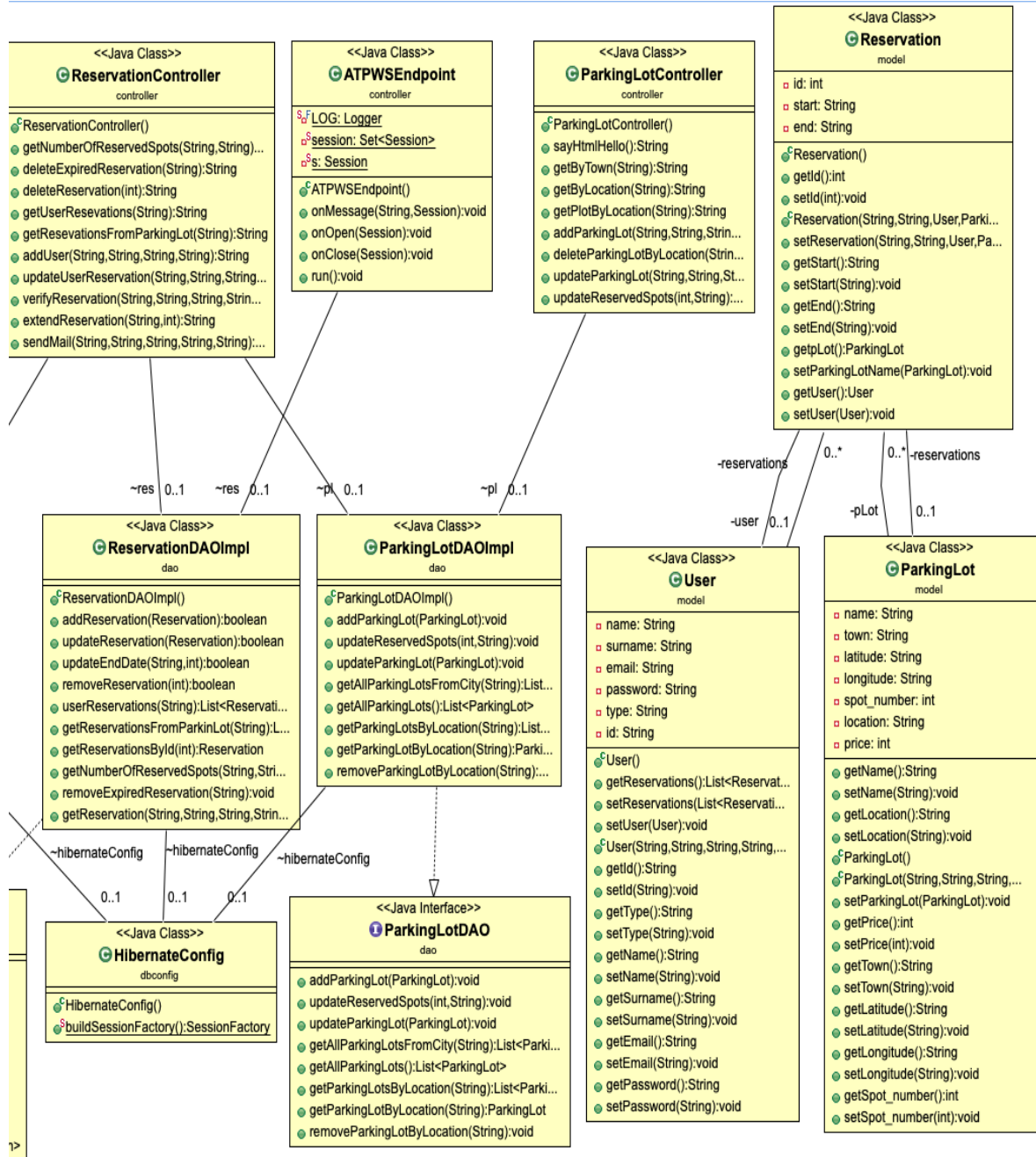
- Diagrama de pachete

Diagrama de pachete este o diagramă structurală care reprezintă modul în care sunt aranjate și organizate elementele de tip model din interiorul unui pachet de dimensiuni mari sau medii. Poate să descrie atât structura cât și dependențele dintre subsisteme sau module.[6]

### 3.1 Diagrama de clase

Diagrama de clase conține toate clasele de pe partea de backend din Java și relațiile dintre acestea.





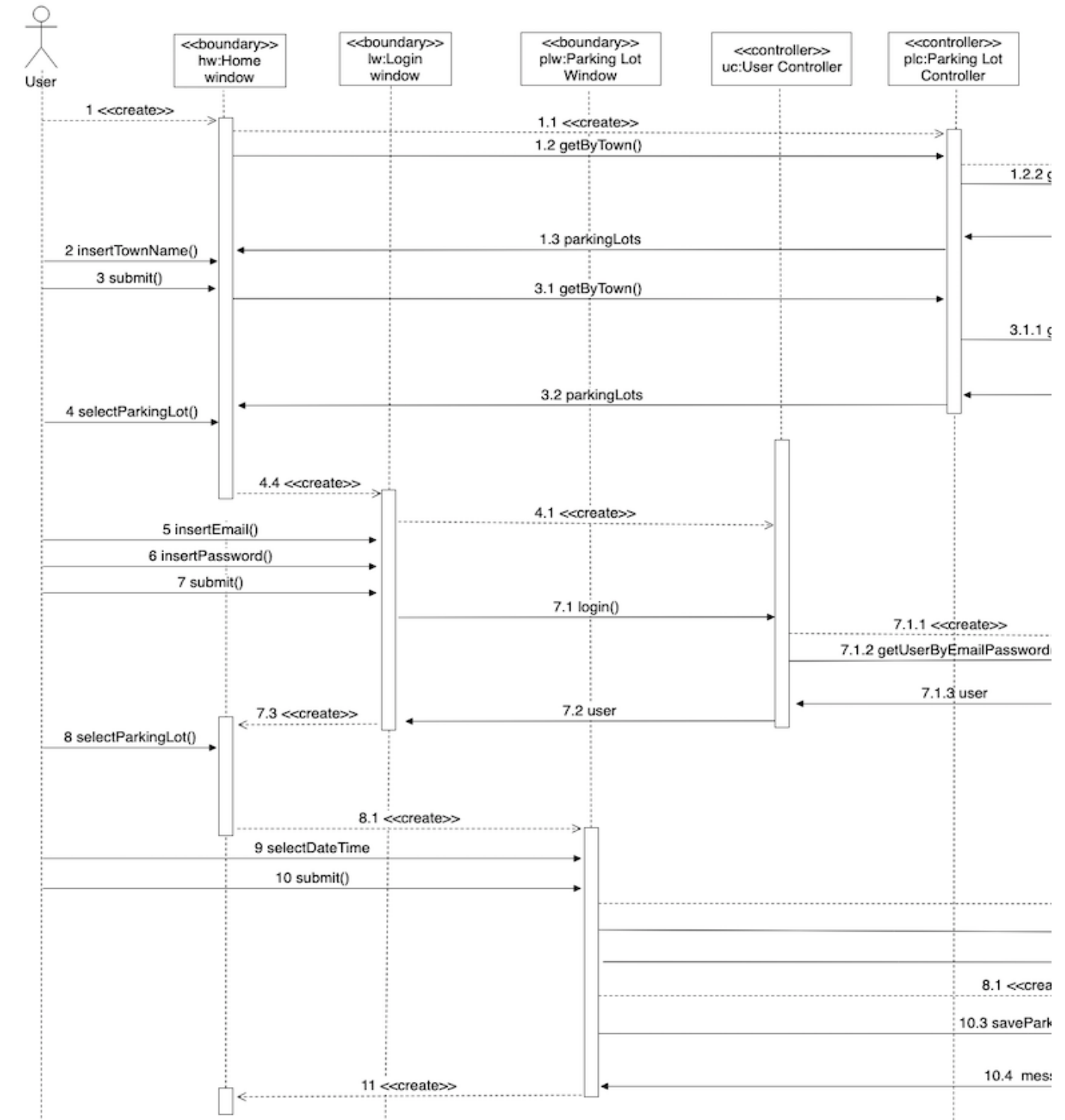
## 3.2 Diagrama cazurilor de utilizare

În această diagramă am inclus principalele funcționalitățile pe care le poate realiza un utilizator într-o ordine logică.

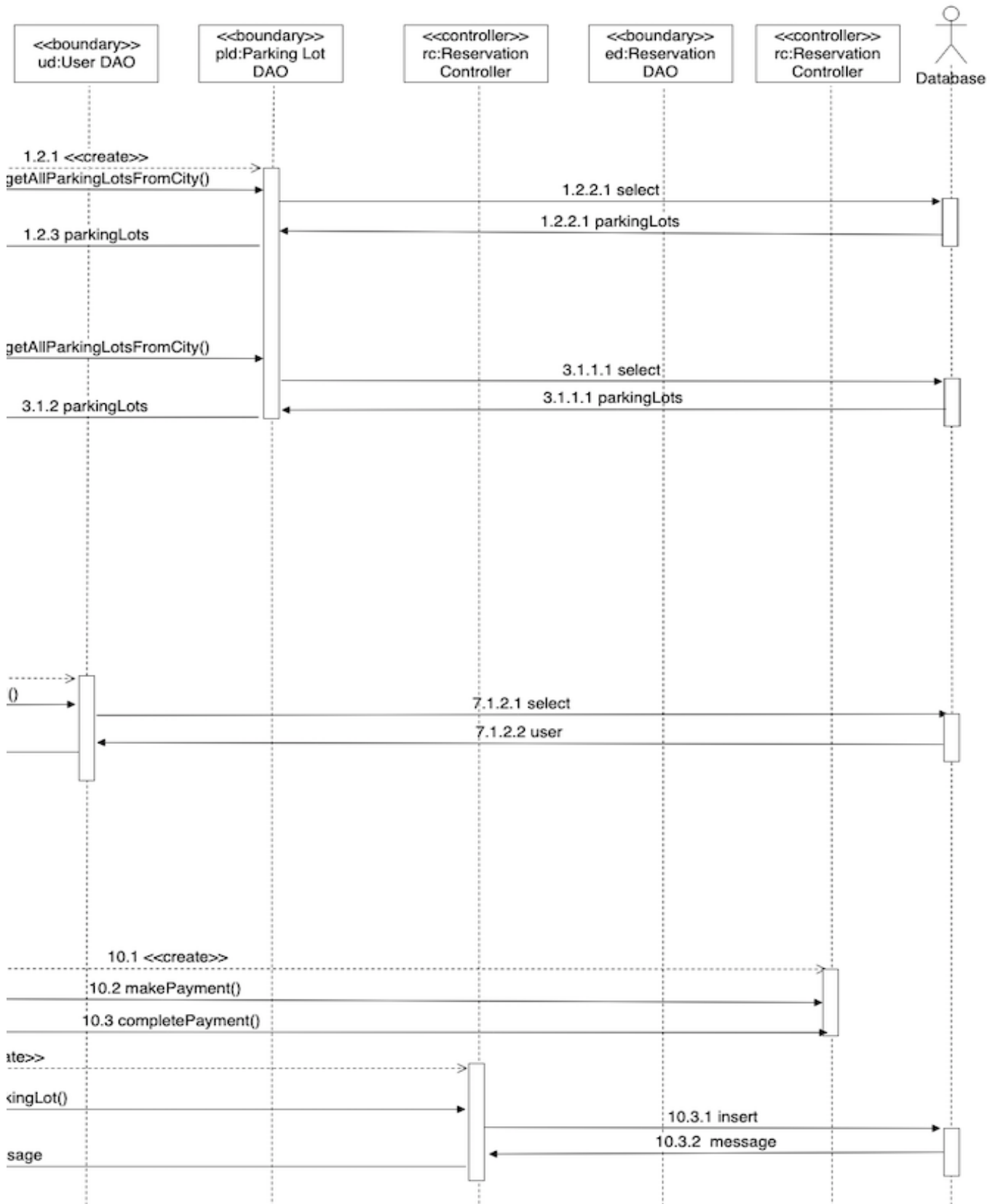


### 3.3 Diagrama de interacțiune

Diagrama de interacțiune descrie interacțiunile care au loc între utilizator și aplicație și între componente de la momentul înregistrării până la realizare unui rezervări.

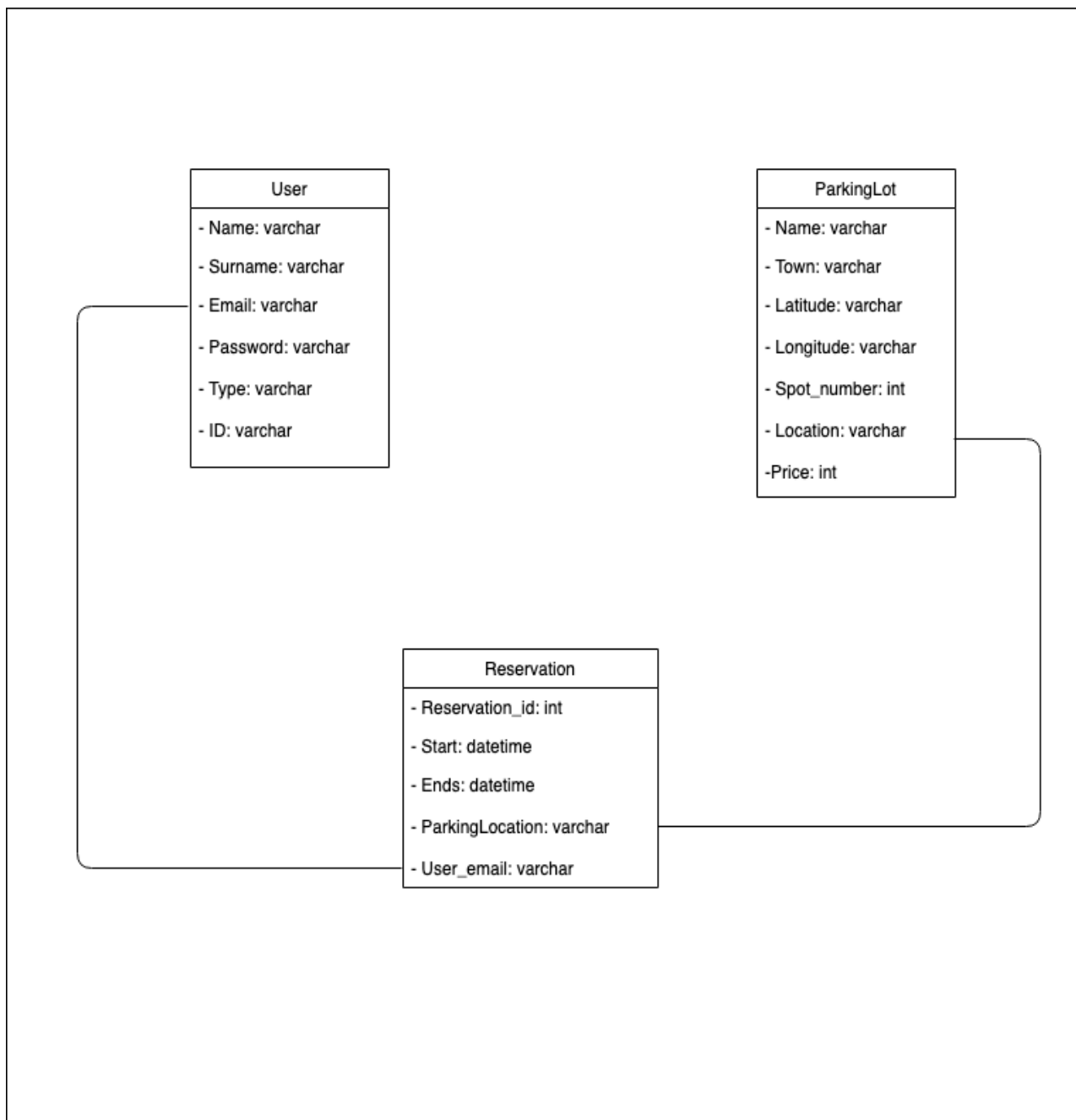






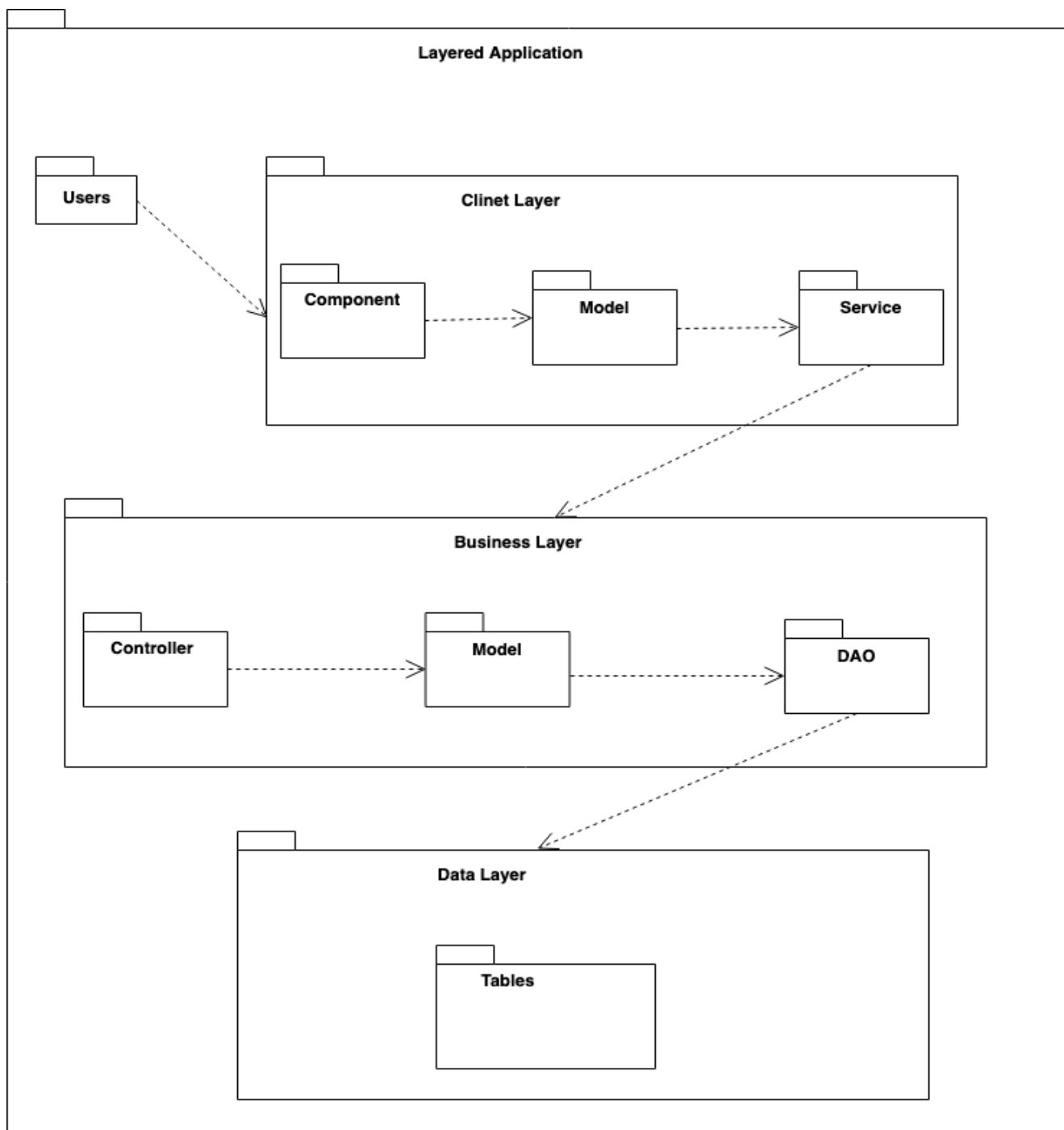
### 3.4 Diagrama bazei de date.

Conține entitățile din baza de date cu atributele lor și legăturile dintre acestea.



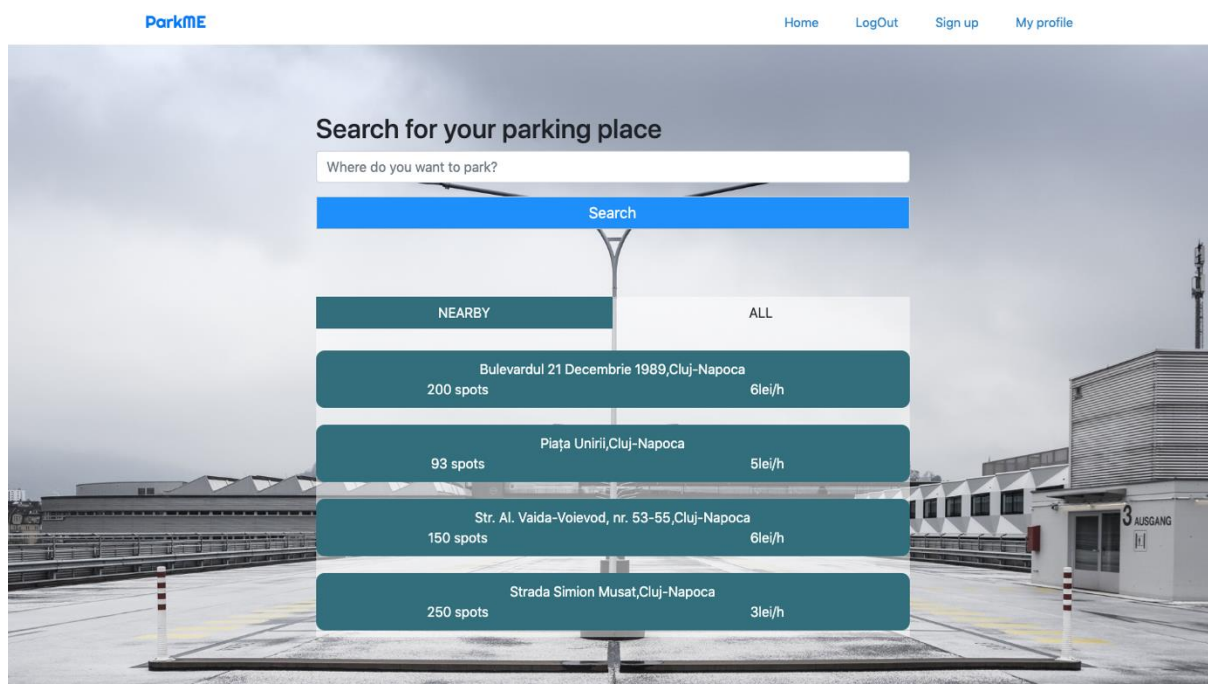
### 3.5 Diagrama de pachete

Diagrama descrie structura aplicației care este împărțită în trei programe care comunică între ele, și conține pachetele din fiecare program împreună cu legăturile dintre acestea.

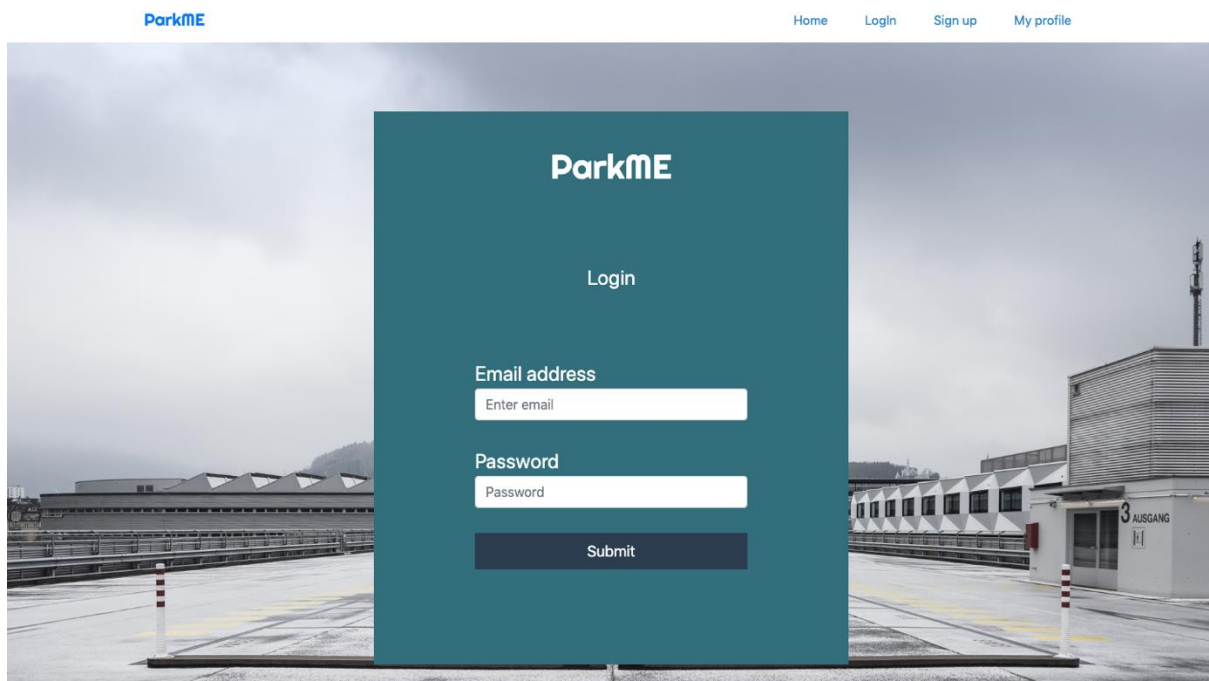


## Cap 4. Utilizarea aplicației

În momentul în care un utilizator care nu este logat deschide aplicația, se va deschide pagina principală pentru clienți. Acesta poate să vizualizeze lista locurilor de parcare cu informații general despre acestea, să caute un loc de parcare după un oraș sau să aleagă conținutul listei. Inițial lista conține parcarile din orașul în care se afla utilizatorul specificat de butonul „nearby” dar poate să ceară și toate parcarile prin apăsarea butonului „all”. Însă utilizatorul nu vor putea să selecteze locul de parcare pentru mai multe informații și pentru a crea o rezervare.

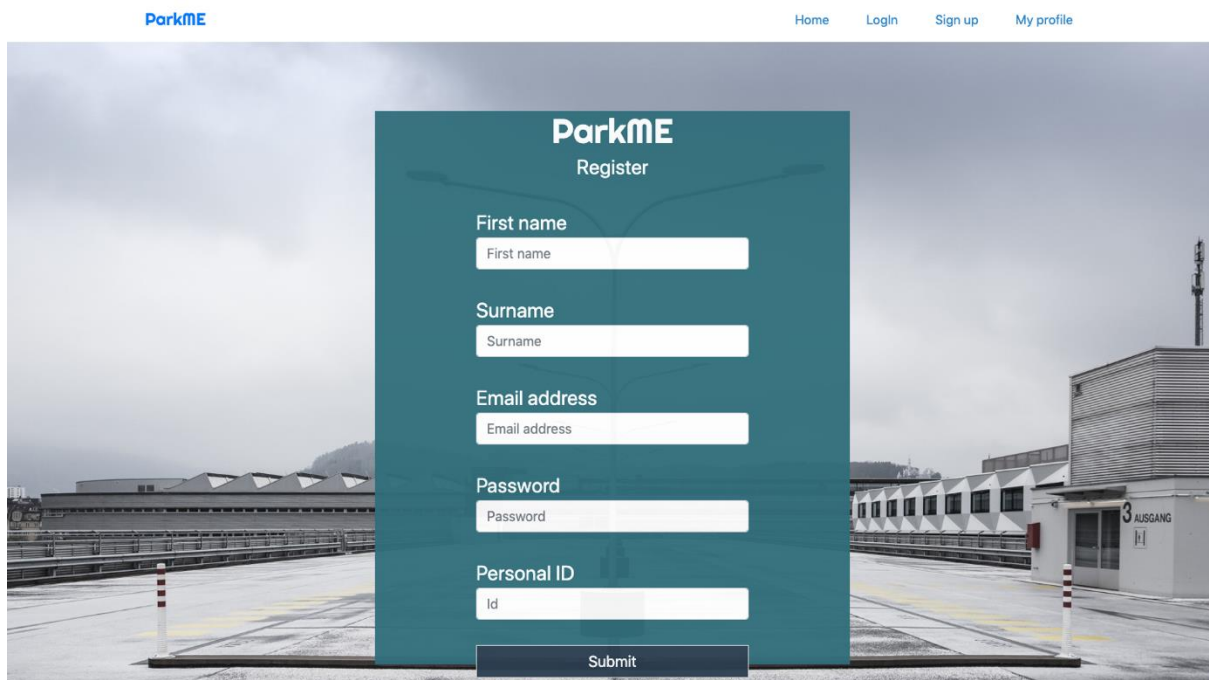


După un click pe link-ul „Login” din meniul principal sau selectarea unui loc de parcare utilizatorul va fi redirecționat la pagina de login. Aceasta conține un formular pe care utilizatorul trebuie să-l completeze cu adresa de email și parola și să apese pe butonul de login pentru a se loga. În acel moment datele sunt trimise la server și verificate, iar dacă utilizatorul există atunci în funcție de rolul pe care-l are va fi redirecționat fie la pagina de „adminhome” fie la pagina „userhome”.



The image shows the 'ParkME' login page. At the top, there is a navigation bar with the 'ParkME' logo on the left and links for 'Home', 'Login', 'Sign up', and 'My profile' on the right. The main content area features a teal-colored login form centered over a background image of a parking lot. The form includes the 'ParkME' logo, the title 'Login', and two input fields: 'Email address' with a placeholder 'Enter email' and 'Password' with a placeholder 'Password'. A dark blue 'Submit' button is located at the bottom of the form.

Dacă utilizatorul nu are cont atunci poate să acceseze pagina de înregistrare cu ajutorul link-ului „Register” din meniu, unde, introducând corect datele cerute va putea să-și creeze un cont.

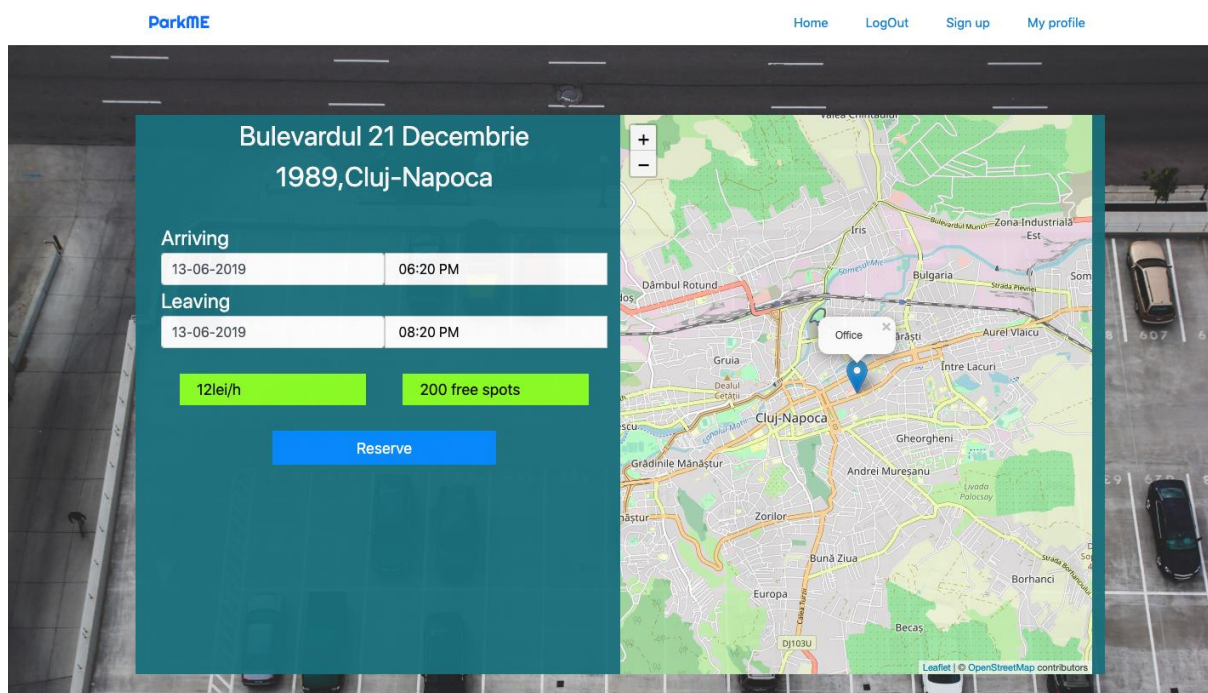


The image shows the 'ParkME' register page. It has the same navigation bar as the login page. The main content area features a teal-colored register form centered over the same parking lot background. The form includes the 'ParkME' logo, the title 'Register', and five input fields: 'First name' (placeholder 'First name'), 'Surname' (placeholder 'Surname'), 'Email address' (placeholder 'Email address'), 'Password' (placeholder 'Password'), and 'Personal ID' (placeholder 'Id'). A dark blue 'Submit' button is at the bottom of the form.

Odată ce utilizatorul este logat, aplicația se împarte ca interfață și funcționalități în două părți, în funcție de rolul pe care utilizatorul îl are.

## 4.1 Utilizare aplicației de către client

Clientul în momentul în care este logat, după ce caută și alege un loc de parcare este redirecționat la pagina „parkingSpot” unde poate să aleagă intervalul de timp pentru care vrea să rezerve un loc de parcare, să vadă prețul și numărul de locuri libere, iar în final să realizeze rezervarea.



Dacă datele introduse sunt corecte și clientul apasă pe butonul de rezervare, va fi direcționat spre pagina de la PayPal unde va trebui să plătească pentru locul de parcare.



## Pay with PayPal

☐ Stay logged in for faster purchases [?](#)

**Log In**

Having trouble logging in?

or


**Pay with Debit or Credit Card**

[Cancel and return to test facilitator's Test Store](#)

În pagina de mai sus utilizatorul poate să se logheze într-un cont de PayPal sau să plătească cu card de debit sau credit.

La pasul următor, în cazul în care utilizatorul alege să se logheze, trebuie să aleagă modalitatea în care va plăti suma necesară.

## test facilitator's Test Store



---

Hi, test!


Ship to

test buyer


1 Main St, San Jose, CA 95131 United States


[Change >](#)

Pay with

☒  Balance

☐ Make my balance my preferred way to pay

☐  CREDIT UNION 1 x-1900

☐  Visa x-2530

[+ Add a debit or credit card](#)

[+ !\[\]\(2fe42fa2b52b33cdb60ff42dfea9aa0b\_img.jpg\) Apply for PayPal Credit](#)


Pay over time for your purchase of \$6.00 with PayPal Credit.

Subject to credit approval. [See terms](#)

[Manage >](#)

PayPal is the safer, easier way to pay

No matter where you shop, we keep your financial information secure.



[View \*\*PayPal Policies\*\* and your payment method rights.](#)

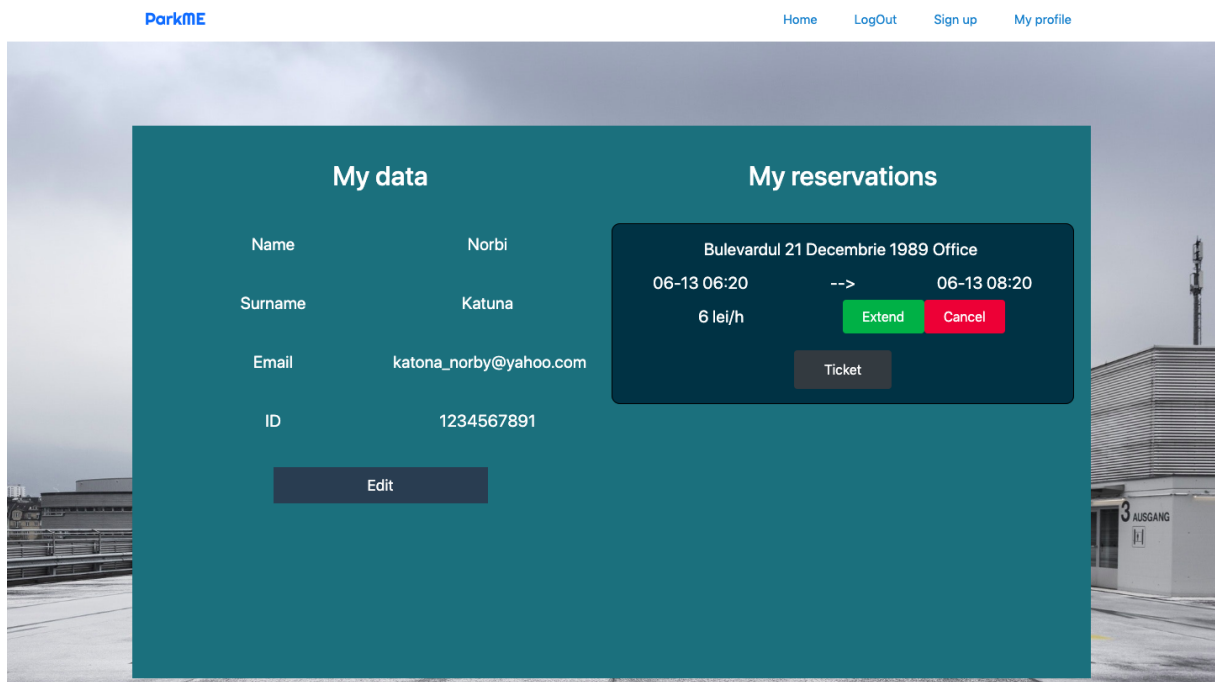
Continue

You'll be able to review your order before you complete your purchase.

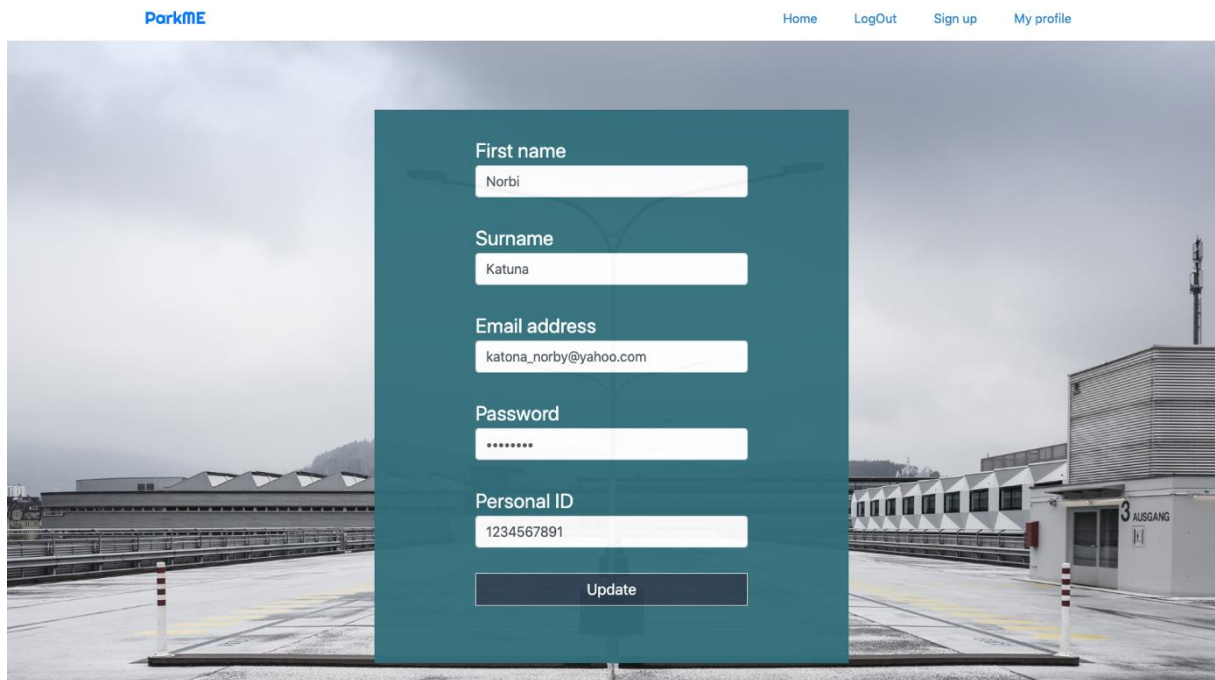
După ce acesta plătește, este redirecționat la pagina „home” și primește un mesaj de succes. În același timp aplicația trimite un email cu datele rezervării și codul QR pe baza căruia va putea să acceseze locul de parcare.

Clientul poate să-și consulte rezervările și profilul la pagina „userInformations” pe care o poate accesa cu ajutorul link-ului „MyProfile” din meniul principal. El poate să șteargă sau să prelungească o rezervare. Dacă alege să șteargă rezervarea atunci lista de rezervări se va actualiza, iar dacă alege să o prelungească va fi dus la pagina în care poate să creeze o rezervare nouă.

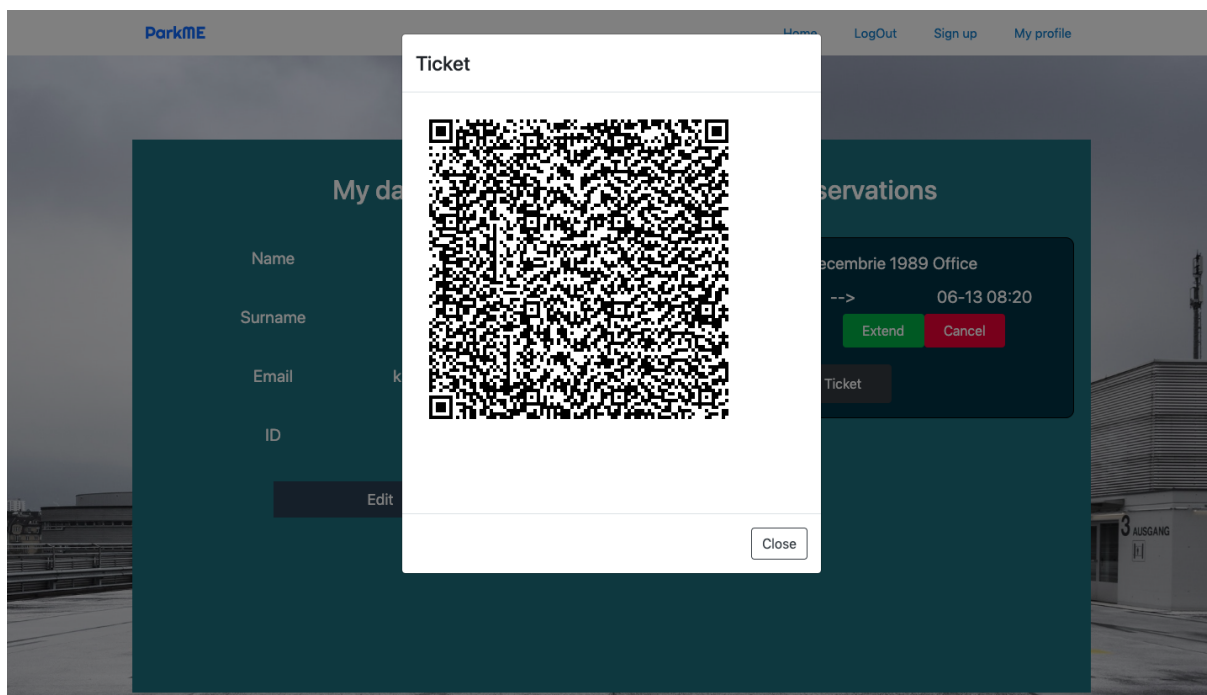




Tot aici acesta poate să schimbe datele personale prin apăsarea butonului „edit” din secțiune de date. Acesta va fi redirecționat la pagina „editProfile” care conține input-uri cu datele clientului unde poate să schimbe datele dorite apoi să salveze schimbarea.



Și în final, în momentul în care dorește să intre în locul de parcare, va trebui să prezinte biletul primit în formă de cod QR pe care-l poate accesa prin apăsarea butonului „Ticket” din lista de rezervări. Biletul va apărea astfel:



Pentru ca biletul să poată fi accesat și mai repede în momentul în care se dorește intrarea în locul de parcare, în momentul efectuării unei rezervări utilizatorul primește un mail în care este informat despre realizarea rezervării, datele despre rezervare precum și un cod QR care reprezintă biletul și pe care-l poate arăta în momentul în care dorește să acceseze locul de parcare.

 **ParkME** <katunanorbert@gmail.com>  
To: katona\_norby@yahoo.com

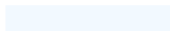
  Jun 12 at 9:55 PM 

**Hello Norbi**

**You have a reservation at Office with location Bulevardul 21 Decembrie 1989.**

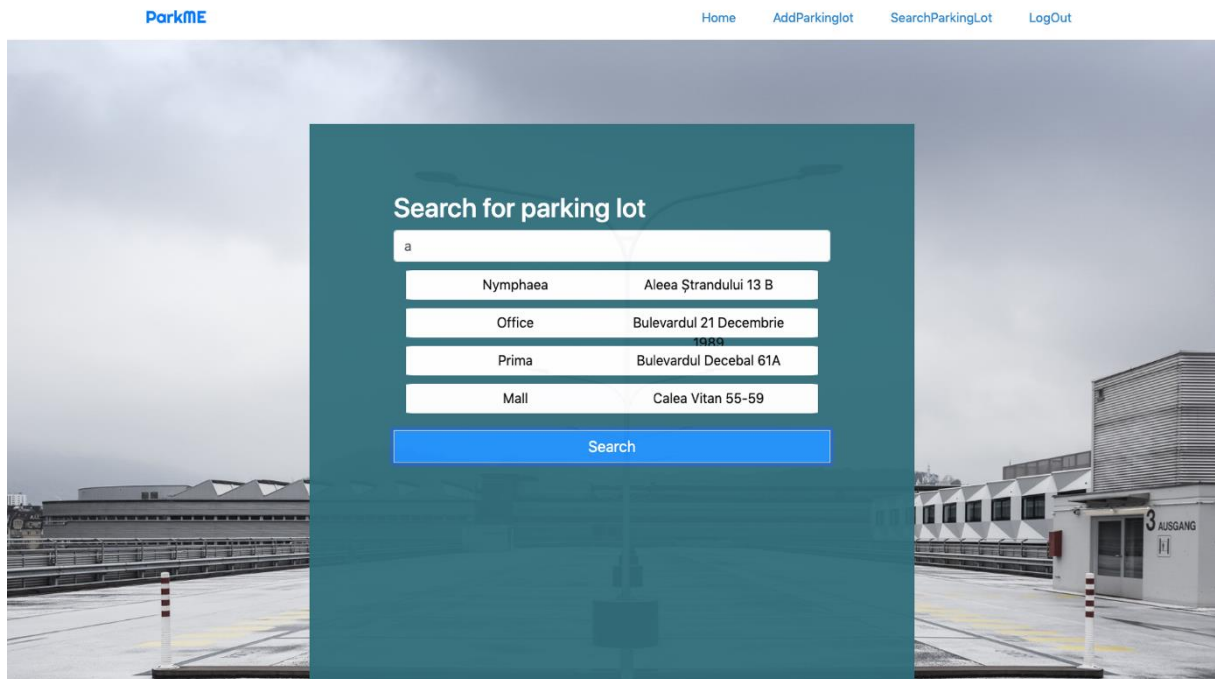
**Your reservation starts at 2019-06-12 09:54:27 and ends at 2019-06-12 10:54:27.**

**Use the QRCode from bellow to acces your parking spot.**



## 4.2 Utilizare aplicației de către administrator

În momentul în care este logat, administratorul primește pagina „searchParkingLot” unde acesta trebuie să caute un loc de parcare după locație și să selecteze locul de parcare pentru a trece mai departe:



The screenshot shows the 'ParkME' application interface. At the top, there is a navigation bar with links: Home, AddParkinglot, SearchParkingLot, and LogOut. The main content area displays a 'Search for parking lot' form. The form has a search bar with the text 'a' entered. Below the search bar, there is a table of suggestions:

Nymphaea	Aleea Ștrandului 13 B
Office	Bulevardul 21 Decembrie
Prima	Bulevardul Decebal 61A
Mall	Calea Vitan 55-59

Below the table is a blue 'Search' button. The background of the application is a photograph of a parking lot with a building in the distance.

După ce administratorul va selecta un loc de parcare, parcare va fi salvată în memorie și va fi redirectionat la pagina „adminHome” unde primește două opțiuni:

- Să modifice un loc de parcare.
- Să obțină statistici despre locul de parcare.

Dacă alege să modifice un loc de parcare va fi redirectionat la pagina „editParkingLot” unde poate să modifice date ale parării și să actualizeze locul de parcare cu modificările făcute sau să șteargă locul de parcare.

ParkME [Home](#) [AddParkinglot](#) [SearchParkingLot](#) [LogOut](#)

Name  
Nymphaea

Town  
Oradea

Latitude  
47.054006

Longitude  
21.951648

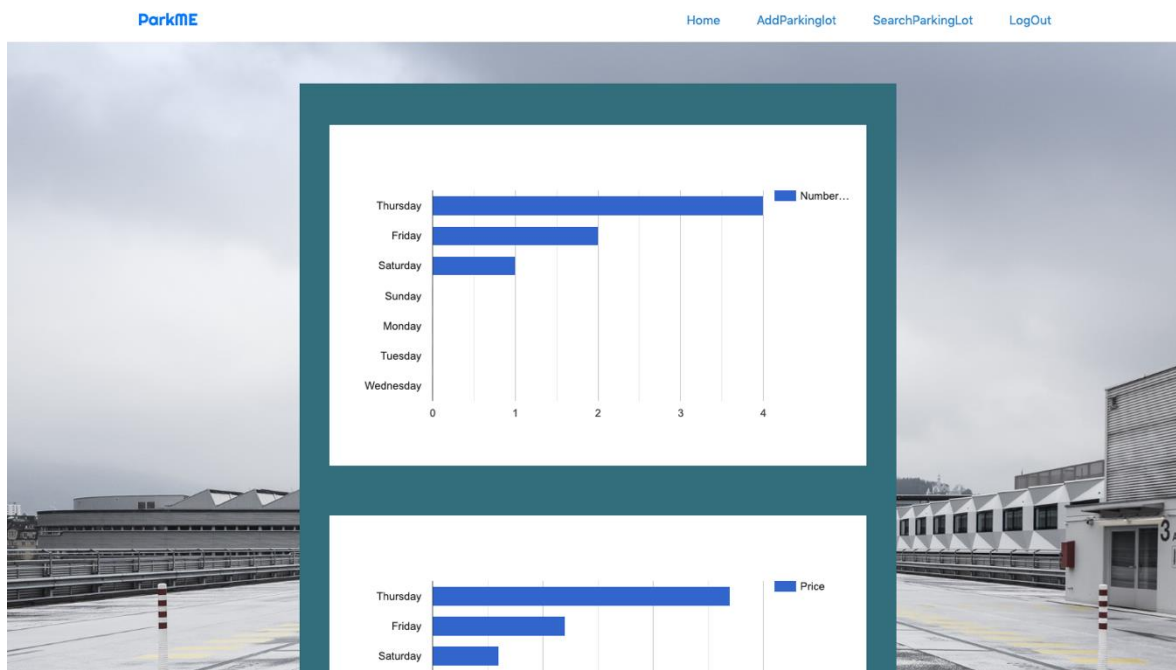
Number of spots  
200

Price  
6

Location  
Aleea Ștrandului 13 B

[Update](#) [Delete](#)

Dacă alege să vizualizeze statistici despre locul de parcare va fi direcționat la pagina „charts” unde se găsesc două grafice, unul cu date despre numărul de rezervări din fiecare zi, iar celălalt conține câștigul pe zi, într-un interval de o săptămână.



De asemenea administratorul poate să adauge un nou loc de parcare printr-un click pe link-ul „AddParkinglot” din meniu. Acesta va fi redirectionat la o pagină în care pentru a adăuga locul de parcare trebuie introducă datele necesare și să dea în cele din urmă click pe butonul „Add”.

**ParkME** Home AddParkinglot SearchParkingLot LogOut

**Name**

**Town**

**Latitude**

**Longitude**

**Number of spots**

**Price**

**Location**

**Add**

https://172.20.10.2:4200/addParkingLot

## Cap 5. Concluzii

În cele din urmă aplicația abordează problema găsirii și rezervării unui loc de parcare prin identificarea și rezolvarea principalelor subprobleme din acest domeniu. Subproblemele abordate prin intermediul acestei aplicații sunt găsirea, obținerea de informații și rezervarea unui loc de parcare, respectiv plata și obținerea unui tichet de parcare în mediu online.

Precum aplicațiile menționate care reprezintă în momentul actual o soluție a problemei abordate, această aplicație oferă următoarele facilități:

- găsirii unui loc de parcare utilizând locația curentă sau folosind căutarea după numele orașului.
- plata online pentru o rezervare.
- obținerea de date despre locul de parcare precum locația specificată atât prin adresa acestuia, cât și cu ajutorul unei hărți, prin utilizarea coordonatelor și prețul acestuia.

Spre deosebire de acestea, a fost implementat un sistem care să permită accesul în interiorul parcărilor bazat pe utilizarea unor bilete generate folosind mecanisme de tip cod QR. Însă raportându-ne la aplicațiile de la noi din țară, care presupun doar vizualizarea locurilor libere din cadrul parării, sistemul prezentat oferă posibilitatea găsirii și rezervării unui loc pe parcare.

Complexitatea acestei probleme este destul de ridicată, motiv pentru care prototipul implementat nu a reușit să aducă o soluție întregii probleme. Astfel aplicația poate să conțină unele îmbunătățiri care se vor putea implementa pentru versiunile ulterioare.

Aplicația poate să fie îmbunătățită prin următoarele dezvoltări ulterioare:

- În primul rând se pot crea grafice mai complexe care să poată fi accesate și de către utilizatori. Un exemplu de astfel de grafic ar fi unul care să conțină date despre numărul de persoane care folosesc o anumită parcare într-un anumit interval de timp. Cu ajutorul acestuia atât utilizatorii cât și administratorii pot să vadă momentul zilei în care parcare este cea mai aglomerată. Acest lucru ar ajuta administratorii în

utilizarea unui preț dinamic pentru acea perioadă astfel încât să reducă numărul excesiv de persoane care doresc să folosească parcare și totodată să crească câștigul obținut. Acest grafic ar ajuta și utilizatori deoarece știind în care moment al zilei o parcare este căutată pot să facă rezervarea din timp pentru a se asigura de faptul că vor obține un loc de parcare.

- Se poate adăuga un algoritm de învățare automată care să îi sugereze utilizatorului un loc de parcare din apropiere mai puțin solicitat, astfel încât mașina să fie parcată cât mai repede posibil. Această implementare ar ușura atât căutarea cât și deplasarea spre un loc de parcare.
- În al treilea rând se pot crea administratori pentru unele locuri de parcare, astfel încât fiecare administrator să poată întreține doar parcarile care îi sunt atribuite, un lucru care ar face mai ușoară întreținerea locurilor de parcare mai ales în momentul în care numărul lor crește.



## Bibliografie

- [1] Angular, <https://angular.io/guide/architecture>
- [2] Arhitectura Client-server, <https://ro.wikipedia.org/wiki/Client-server>
- [3] Bootstrap, [https://en.wikipedia.org/wiki/Bootstrap\\_\(front-end\\_framework\)](https://en.wikipedia.org/wiki/Bootstrap_(front-end_framework))
- [4] CSS, [https://en.wikipedia.org/wiki/Cascading\\_Style\\_Sheets](https://en.wikipedia.org/wiki/Cascading_Style_Sheets)
- [5] Diagrame UML, <https://tallyfy.com/uml-diagram/>
- [6] Diagrame UML, <http://www.cs.ubbcluj.ro/~vladi/Teaching/Didactic/ISS%202018-2019/mate-info%20romana/curs/curs2.pdf>
- [7] Hibernate, [https://en.wikipedia.org/wiki/Hibernate\\_\(framework\)](https://en.wikipedia.org/wiki/Hibernate_(framework))
- [8] HTML, <https://en.wikipedia.org/wiki/HTML>
- [9] Imagine Angular MVC, <https://medium.com/@colinsygiel/the-heart-of-angularjs-a687a153c024>
- [10] Imagine CSS, [https://developer.mozilla.org/en-US/docs/Learn/CSS/Introduction\\_to\\_CSS/How\\_CSS\\_works](https://developer.mozilla.org/en-US/docs/Learn/CSS/Introduction_to_CSS/How_CSS_works)
- [11] Imagine design pattern MVC, <https://stackoverflow.com/questions/29325038/convertng-java-code-to-model-view-controller-architecture-pattern>
- [12] Imagine PayPal, <https://developer.paypal.com/docs/classic/paypal-payments-standard/integration-guide/paymentdatatransfer/>
- [13] Imagine compilare TypeScript, <https://www.manejandodatos.es/2015/08/preparing-the-field-for-typescript-programming-by-microsoft/>
- [14] Java, [https://ro.wikipedia.org/wiki/Java\\_\(limbaj\\_de\\_programare\)](https://ro.wikipedia.org/wiki/Java_(limbaj_de_programare))
- [15] Jersey, [https://en.wikipedia.org/wiki/Java\\_API\\_for\\_RESTful\\_Web\\_Services](https://en.wikipedia.org/wiki/Java_API_for_RESTful_Web_Services)
- [16] Maven, <https://maven.apache.org/>
- [17] Maven, [https://ro.wikipedia.org/wiki/Apache\\_Maven](https://ro.wikipedia.org/wiki/Apache_Maven)
- [18] MySQL, <https://ro.wikipedia.org/wiki/MySQL>
- [19] OpenStreetMap, <https://ro.wikipedia.org/wiki/OpenStreetMap>
- [20] PayPal, <https://developer.paypal.com/docs/integration/direct/payments/>
- [21] Tomcat, [https://en.wikipedia.org/wiki/Apache\\_Tomcat](https://en.wikipedia.org/wiki/Apache_Tomcat)
- [22] TypeScript, <https://en.wikipedia.org/wiki/TypeScript>