UNIVERSITATEA POLITEHNICA BUCUREȘTI

FACULTATEA DE AUTOMATICA ȘI CALCULATOARE

DEPARTAMENTUL CALCULATOARE

|  |  |
| --- | --- |
| upb | cs |

PROIECT DE DIPLOMA

Sistem de BikeSharing

versiunea 2018

Nicoleta Mirela Grigore

**Coordonator știintific:**

Prof. Dr. Ing. Razvan RUGHINIS

Ş.l. dr. ing. Gavril GODZA

BUCUREŞTI

2018

# CUPRINS

[CUPRINS 1](#_Toc517133374)

[Sinopsis 4](#_Toc517133375)

[Abstract 4](#_Toc517133376)

[1 Introducere 5](#_Toc517133377)

[1.1 Context. Motivatie 5](#_Toc517133378)

[1.2 Problema 6](#_Toc517133379)

[1.3 Obiective 7](#_Toc517133380)

[1.4 Solutia propusa 7](#_Toc517133381)

[1.5 Structura lucrarii 7](#_Toc517133382)

[2 Analiza cerintelor 9](#_Toc517133383)

[2.1 Modelele sistemului 9](#_Toc517133384)

[2.2 Cerinte functionale 10](#_Toc517133385)

[2.3 Cerinte nefunctionale 11](#_Toc517133386)

[2.4 Cazuri de utilizare – exemple 12](#_Toc517133387)

[2.4.1 Sign up 12](#_Toc517133388)

[2.4.2 Login 13](#_Toc517133389)

[3 Studiu de piata / Metode existente 14](#_Toc517133390)

[3.1 In lume 14](#_Toc517133391)

[3.1.1 Londra, Anglia 14](#_Toc517133392)

[3.1.2 New York City, U.S. 15](#_Toc517133393)

[3.1.3 Monreal, Canada 15](#_Toc517133394)

[3.2 In Romania 15](#_Toc517133395)

[3.2.1 Cluj 15](#_Toc517133396)

[3.2.2 Timisoara 16](#_Toc517133397)

[3.2.3 Bucuresti 16](#_Toc517133398)

[4 Solutia propusa 17](#_Toc517133399)

[4.1 Prezentarea generala a arhitecturii 17](#_Toc517133400)

[4.1.1 Pagina web 17](#_Toc517133401)

[4.1.2 Aplicatie single-page 17](#_Toc517133402)

[4.1.3 Arhitectura MVC 18](#_Toc517133403)

[4.1.4 Web API 19](#_Toc517133404)

[4.2 Tehnologii folosite 19](#_Toc517133405)

[4.2.1 Front-end 19](#_Toc517133406)

[4.2.2 Back-end 23](#_Toc517133407)

[4.2.3 Baza de date 25](#_Toc517133408)

[5 Detalii de implementare 26](#_Toc517133409)

[5.1 Structura bazei de date 26](#_Toc517133410)

[5.1.1 User 26](#_Toc517133411)

[5.1.2 UserType 27](#_Toc517133412)

[5.1.3 Place 27](#_Toc517133413)

[5.1.4 Bike 27](#_Toc517133414)

[5.1.5 BikeState 28](#_Toc517133415)

[5.1.6 Transaction 28](#_Toc517133416)

[5.1.7 TransactionState 28](#_Toc517133417)

[5.1.8 Price 29](#_Toc517133418)

[5.2 Back-end 29](#_Toc517133419)

[5.2.1 Modele 29](#_Toc517133420)

[5.2.2 Clase auxiliare 29](#_Toc517133421)

[5.2.3 Controlere 31](#_Toc517133422)

[5.3 Front-end 32](#_Toc517133423)

[5.3.1 Modele 32](#_Toc517133424)

[5.3.2 Clase auxiliare 32](#_Toc517133425)

[5.3.3 Componente 33](#_Toc517133426)

[5.3.4 AppModule 35](#_Toc517133427)

[6 Evaluare 37](#_Toc517133428)

[6.1 Scenarii 37](#_Toc517133429)

[6.1.1 Login 37](#_Toc517133430)

[6.1.2 Sign up 38](#_Toc517133431)

[6.1.3 Start transaction 39](#_Toc517133432)

[7 Concluzii 41](#_Toc517133433)

[7.1 Dezvoltari ulterioare 41](#_Toc517133434)

[7.1.1 Device-uri specializate 41](#_Toc517133435)

[7.1.2 Aplicatie mobile 42](#_Toc517133436)

[8 Bibliografie 43](#_Toc517133437)

[9 Anexe 45](#_Toc517133438)

# Sinopsis

Exista o serie de probleme legate de transport, securitate si mobilitate pe care acest sistem le rezolva foarte simplu. Se stie ca in lumea contemporana tot mai multi oameni sunt predispusi la sedentarism, motiv pentru care, in multe state, cetatenii folosesc bicicleta ca unul din principalele mijloace de transport.

Pentru rezolvarea acestei probleme oferim o solutie rapida, user-friendly care va pune la dispozitie resursele necesare pentru a eficientiza si facilita procesul de bike-sharing, fata de solutiile actuale unde procesul este ingreunat din anumite cauze.

Modul de inregistrare ofera posiblitatea, atat locuitorilor cat si turistilor de a accesa resursele disponibile fara a fi necesar in prealabil de alte actiuni, acte sau documente.

# Abstract

There is a series of problems related to transport, security and mobility which this system solves very easily. It is known that nowadays more and more people are predesposed to sedentarism, which is why, in many states, residents use bike as a means of transport.

For solving this problem, we offer a fast and user-friendly solution which will provide the necessary resources that will facilitate and will make the bike-sharing system more efficient, compared to the actual solutions where the process is weighted down out of different causes.

The sign-up mode gives the possibility, for both residents and tourists, to access the available resources without any requirements needed. such as actions or documents.

# Introducere

## Context. Motivatie

Lucrarea de fata trateaza subiectul de bicycle-sharing system, un sistem originar din Viena. Acest sistem presupune mai multe puncte (centre) de inchiriere de biciclete mapate pe o anumita harta a orasului. Clientul va putea inchiria o anumita bicicleta, prima ora a utilizarii acestea fiind gratuita. Sistemul va recunoaste clientul dupa user si parola, afland astfel daca beneficiul de gratuitate al primei ore a fost ori nu consumat.

Principala problema intr-un oras mare sau in acest caz o capitala, este aglomeratia si traficul. Bucurestiul, spre exemplu, este cel mai aglomarat oras al Romaniei, fiind orasul cu cel mai mare numar de locuitori din tara.

Numarul de masini este in crestere, inca din anul 1990, iar straziile nu mai au capacitatea si buna starea pentru a face fata acestui numar mare de masini. Pe langa gropiile care s-au format de-a lungul timpului, din cauza numarului mare de masini, o banda este folosita drept loc de parcare.

Nici transportul public nu este de mare ajutor. Autobuzele si trolebuzele sunt in aceleasi situatii ca masiniile, travaiele sunt o variant mai sigura, dar si in cazul acestora pot fi probleme (masini care folosesc banda alocata tramvaiului si dupa la intarzierea acestuia). La ora actuala, metroul este cea mai rapida modalitate de transport din Bucuresti, dar acesta nu este accesibil in toate zonele (ex Drumul Taberei, sector 6, etc).

Pe langa problema aglomeratiei, din cauza evolutiei tehnologiei, tot mai multi oameni lucreaza la birou, stau cel putin 8 ore in fata unui calculator si dupa terminarea programului de munca, cand ajung acasa, prefera sa stea tot in fata electronicelor. Sedentarism este stilul de viata, gresit, pe care tot mai multi oameni il adopta si acesta este motivul pentru care tot mai multi ajung sa faca bolii grave, cum ar fi obezitatea, atrofierea muschilor si a oaselor, etc.

Acestea sunt principalele motive pentru care consider ca un sistem de bike-sharing este necesar intr-o societate dezvoltata. Este o metoda rapida de transport, sanatoasa, relaxanta si ieftina.

Exemplu, din zona Vatra Luminoasa, sector 2, pentru a ajunge la Universitatea Politehnica Bucuresti, exista 4 modalitati: troleu 90, care vine la un interval de 20-30 de min si timpul aproximativ in care ajunge este de 45 min – 2 ore, in functie de ora. Alta modalitate este tramvaiul 1, care vine la un interval de 10-15 min si timpul este de 50 min – 1 ora si 15 min. A treia modalitate este metroul, calatoria cu metroul este de aproximativ 30-35 min, drumul pana la/de la metrou aproximativ 15 min, deci un total de 45-50 de min.

Ultima modalitate si cea mai rapida, este bicicleta, indiferent de ora, timpul aproximativ este de 30-40 min. Fata de mijloacele de transport in comun, care au un program, ajung in anumite statii la anumite intervale, in functie de ora, transportul cu bicicleta este liber, nu trebuie sa astepti, nu stai in aglomeratie, etc.

## Problema

Exista o serie de probleme legate de transport, securitate si mobilitate pe care acest sistem le rezolva foarte simplu. Se stie ca in lumea contemporana tot mai multi oameni sunt predispusi la sedentarism, motiv pentru care, in multe state, cetatenii folosesc bicicleta ca unul din principalele mijloace de transport.

Cel mai socant adevar este ca aproape jumatate din toti ciclistii ajung ca bicicletele lor sa fie furate. Cei mai afectati sunt ciclistii ce locuiesc in oras iar in majoritatea situatiilor acestia se trezesc cu bicicletele furate nu doar o singura data. Cercetatorii de la McGill au gasit 961 de astfel de pagubiti cu un total de 1890 de biciclete furate.

Foarte trist este faptul ca doar 2,4% din bicicletele furate au fost recuperate. Cu cat oamenii circula mai des cu aceste vehicule, cu atat mai mult sansa furtului creste.

76% din bicicletele furate sunt in valoare mai mica de 500$. In general, persoanele cu biciclete mai scumpe iau masuri de precautie mai ridicate (aschizitioneaza lacate mai performante sau inchiriaza spatii special pentru parcarea lor). Aici insa apare o alta problema. Cu cat masurile de precautie sunt mai mare, cu atat ciclismul devine mai costisitor, in consecinta reducerea acestuia.

7% dintre victime nu si-au mai inlocuit niciodata bicicletele. Aceasta este una din cele mai alarmante statistici, insemnand faptul ca acesti oameni au revenit complet la transportul in comun sau la simplul mers pe jos.

## Obiective

Cumpararea unei biciclete este o investitie buna, dar sunt riscuri. Prima fiind investitia initiala, pretul unei biciclete decente, nu este mic, iar pentru o familie de 2, poate fi o problema. Pe langa investitia initiala, pe parcurs mai pot aparea costuri mici, in cazul unor probleme minore. Insa cel mai mare risc, este furtul acesteia. Este un numar mare de cazuri de furturi ale bicicletelor in Bucuresti, multe dintre ele fiind inca nerezolvate.

Acest sistem este menit sa incerce sa resolve aceste probleme. Centrele din care se pot inchiria bicicletele vor fii in toate partiile Bucurestiului, pretul scazut, posibilitatea inchirierii atat pe perioade scurte dar si lungi, etc.

## Solutia propusa

In sistemele clasice de inchiriere a unor astfel de vehicule, clientul este obligat sa il returneze de la acelasi centru de unde a fost inchiriat, acesta fiind obligat sa revina in locul respectiv, parcurgand astfel o distanta suplimentara la costul unui anumit timp.

Noutatea pe care o introduce acest sistem este faptul ca bicicleta inchiriata de la centrul x, va putea fi lasata in oricare alt centru y (unde y este centru ce face parte din cele mapate peste o zona / oras / harta comuna). De asemenea, sistemul va face economie la resursele umane, deoarece, la fiecare centru se va afla maxim un angajat (administrator). O astfel de persoana poate administra unul sau mai multe centre simultan, fara a fi nevoie de prezenta fizica a acestuia la fiecare dintre ele.

Fiecare centru va tine o evidenta a numarului de bicliclete disponibile pentru inchiriere si a numarului de locul libere pentru lasarea acelor inchiriate.

## Structura lucrarii

Documentul de fata are urmatoarea structura: se va incepe cu un sinopsis unde se va descrie succinct problema abordata, impreuna cu o enumerare sumara a rezultatelor și concluziilor.

In continuare, primul capitol al documentul, reprezentand introducerea propriu-zisa, va descrie problematica abordata impreuna cu exemple, concrete ce vor reflecta importanta solutionarii acesteia, solutia propusa pentru o rezolvare eficienta și rezultatele obtinute in urma aplicarii acesteia.

In urmatorul capitol se va face o analiza a cerintelor produsului din prisma potentialilor clienti si a scenariilor de utilizare preconizate, urmand a fi generata o lista de functionalitati.

Al treilea capitol va trata studiul de piata a solutiilor similare vorbindu-se despre limitarile acestora si pentru ce cazuri de utilizare ori pentru ce tip de clienti produsele existente pe piata nu raspund cerintelor. Tot aici gasim si metodele existente care se refera, de regula, la nivelul curent de dezvoltare (starea curenta a domeniului, unde ne regasim, care este contextul).

Se va continua apoi cu solutia propusa ce rezolva problema, avand inclusa și structura sau arhitectura acesteia.

Capitolul 5 contine detaliile de implementare a sistemului, adica elementele specifice ale rezolvarii problemei care au presupus dificultati deosebite din punct de vedere tehnic. Vor fi incluse configuratii, secvente de cod, pseudo-cod, implementari ale unor algoritmi, analize ale unor date, scripturi de testare.

Va urma apoi un capitol in care se va regasi evaluarea aplicatiei, și anume vom vedea cat de bine merge sistemul propus și cum se compara acesta cu solutiile existente pe baza unor metrici clare.

Incheierea documentului va preciza concluziile proiectului precum si dezvoltarile ulterioarele. La final se vor gasi bibliografia, impreuna cu anexele.

# Analiza cerintelor

Acesta aplicatie este folosita de toti clientii pentru a permite deplasarea dintr-un punct in altul.

## Modelele sistemului

Exista 2 tipuri de actori, fiecare avand un anumit set de drepturi:

1. Clientul

* Are dreptul de a-si face cont („Sign-up”)
* Are dreptul de a se loga in aplicatie („Login”)
* Are dreptul de a vizualiza locurile din care poate inchiria o bicicleta (ecran „Place”)
* Are dreptul de a vizualiza bicicletele existente in fiecare locatie (ecranul „Bike”)
* Are dreptul a crea o tranzactie (ecranul „Transaction”, butonul „Start transaction”)
* Are dreptul de a incheia o tranzactie (ecranul „Transaction”, butonul „Finish transaction”)
* Are dreptul de a vizualiza un istoric al tranzactiilor (ecranul „Transaction”)
* Are dreptul de a modifica informatiile din contul acestuia (ecranul „Edit Details”)
* Are dreptul de a adauga sau modifica date de cont (ecran „Payment details”)
* Are dreptul de a se deloga din aplicatie

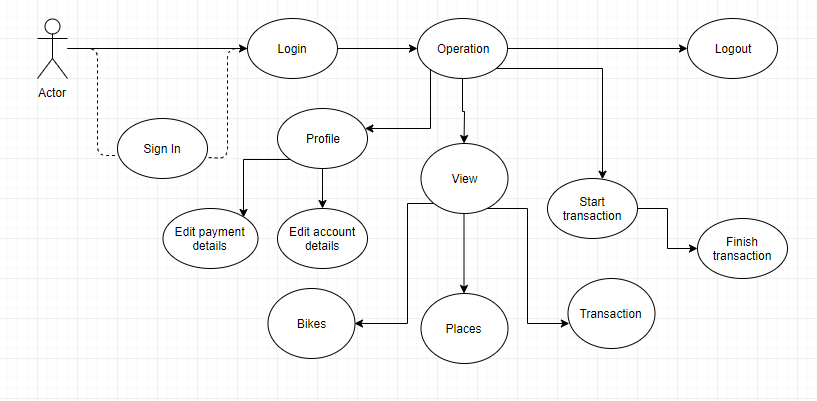


Figura 1 Workflow client

1. Administratorul

* Are dreptul de a-si face cont („Sign-up”)
* Are dreptul de a se loga in aplicatie („Login”)
* Are dreptul de a face alti utilizatori administratori
* Are dreptul de a adauga/edita/sterge o locatie (ecran „Place”)
* Are dreptul de a adauga/edita/sterge o bicicleta (ecran „Bike”)
* Are dreptul de a adauga un nou pret (ecran „Price”)
* Are dreptul de a edita/sterge un pret, doar daca acesta nu a intrat in vigoare (ecran „Price”)
* Are dreptul a crea o tranzactie (ecranul „Transaction”, butonul „Add transaction”)
* Are dreptul de a incheia o tranzactie (ecranul „Transaction”, butonul „Finish transaction”)
* Are dreptul de a vizualiza un istoric al tranzactiilor (ecranul „Transaction”)
* Are dreptul de a modifica informatiile din contul acestuia (ecranul „Edit Details”)
* Are dreptul de a adauga sau modifica date de cont (ecran „Payment details”)
* Are dreptul de a se deloga din aplicatie

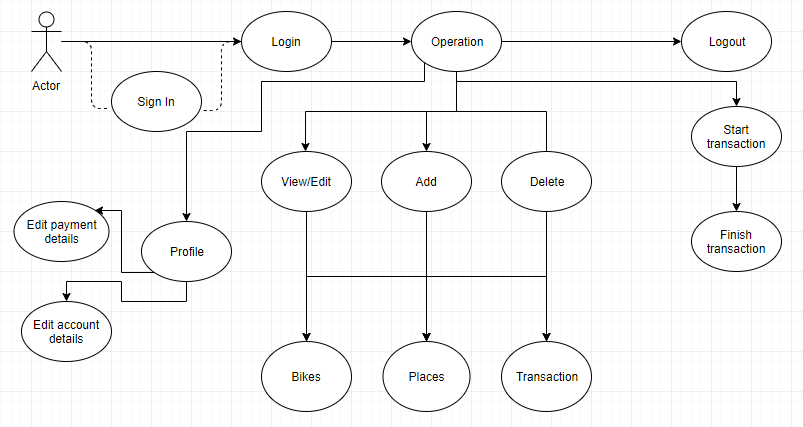


Figura 2 Workflow administrator

## Cerinte functionale

Aplicatia indeplineste urmatoarele functii:

1. comunicarea intre utilizator si mediul online prin intermediul unei interefete grafice;
2. accesul utilizatorului la informatiile disponibile despre centrele si bicicletele existente
3. posibilitatea utilizatorului de a inchiria bicicletele disponibile
4. posbilitatea adminului de a adauga / sterge / edita inregistrari

Utilizatorul introduce credentialele si se logheaza. In functie de tipul utilizatorului, aplicatia pune la dispozitie interfete diferite cu butoanele specifice:

* administratorul va avea o interfata care ii va permite crearea / stergerea / editarea inregistrarilor si a informatiilor;
* utilizatorul va avea o interfata care ii va permite sa vizualizeze si sa execute functiile disponibile ale aplicatiei

In caz de introducere de date eronate la logare (nume de utilizator sau parola), de indisponibilitatea site-ului (stare de mentenanta) sau de crearea unui utilizator deja existent, acestea se vor semnala prin afisarea de mesaje.

Logarea in cadrul contului conditioneaza accesul la date si la actiunile permise. Singurii care pot manipula datele existente sunt adminii.

Cand adminul efectueaza o operatie de creare / stergere / editare a informatiilor sau a inregistrarilor in sistem, aceasta operatie este verificata pana sa ajunga la server, iar in cazul in care aceasta nu este valida este intors un mesaj de eroare.

## Cerinte nefunctionale

Aplicatia va functiona doar pe browserele Google Chrone si Mozilla Firefox (orice versiune actuala).

Sistemul va avea interactivitate sporita, interfata fiind user-friendly.

Limitari:

* Fiecare cont va avea un email si o parola (campuri unice).
* Accesul la site este conditionat de o retea de internet stabila si relativ rapida
* Securitatea conturilor este conditionata de combinatia nume + parola personala
* Numarul resurselor fizicie (biciclete, centre)
* Ecranele care folosesc Google Maps / Braintree vor avea load-time mai mare decat celelalte

## Cazuri de utilizare – exemple

### Sign up

Pas 1: Utilizatorul introduce data necesare pentru crearea unui cont nou

Pas 2: Prima validare se face in partea de client-side ( daca adresa de email/numarul de telefon este in formatul corect, etc)

Pas 2.1: In cazul in care date nu sunt corecte, utilizatorul v-a primi mesaj de eroare

Pas 3: Transmiterea datelor catre server-side

Pas 4: Se genereaza un hash pe baza parolei introduse, aplicand algoritmul SHA256

Pas 5: Transmiterea datelor catre server-ul de Braintree unde vor fi tinute datele de card, din acest motiv este nevoie de o legatura intre baza de date a aplicatiei si Braintree.

Pas 6: Server-ul de Braintree trimite un id unic pentru client

Pas 7: Trimiterea datelor catre baza de date, pentru a le salva

Pas 8: Salvarea datelor

Pas 9: Trimiterea mesajului de succes catre client side

Pas 10: Afisarea interfetei corespunzatoare

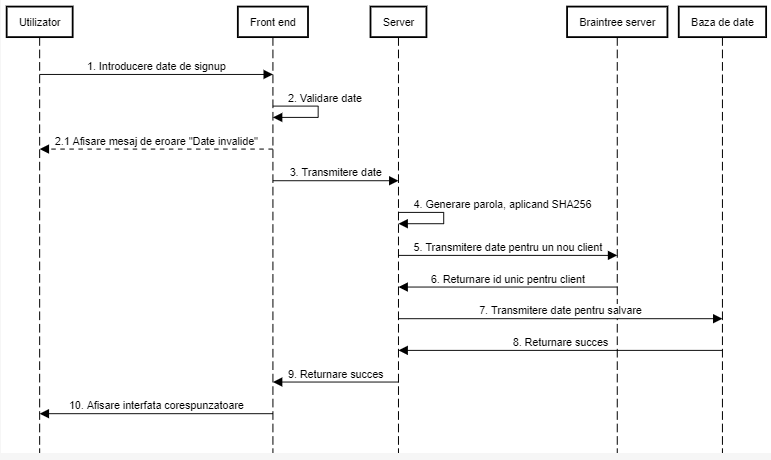


Figura 3 Diagrama secvential Signup

### Login

Pas 1: Introducerea datelor de login

Pas 2: Validarea datelor

Pas 2.1: In cazul in care datele sunt invalide, afisarea unui mesaj de eroare

Pas 3: Transmiterea datelor catre server-side

Pas 4: Verificarea existentei clientului in baza de date

Pas 5: Returnare date de login

Pas 5.1: Daca clientul nu exista, nu se returneaza nici o inregistrare din baza de date

Pas 5.2: Se returneaza fail catre client side pentru a fi returnat un mesaj de eroare

Pas 5.3: Se afiseaza mesajul de eroare

Pas 6: Se genereaza hash-ul corespunzator parolei introduse si se compara cu cea din baza de date

Pas 7: In cazul in care cele 2 parole sunt identice, se retuneaza succes

Pas 7.1: In caz contrar, se returneaza fail

Pas 7.2: Se afiseaza mesaj de eroare

Pas 8: Afisare interfata corespunzatoare

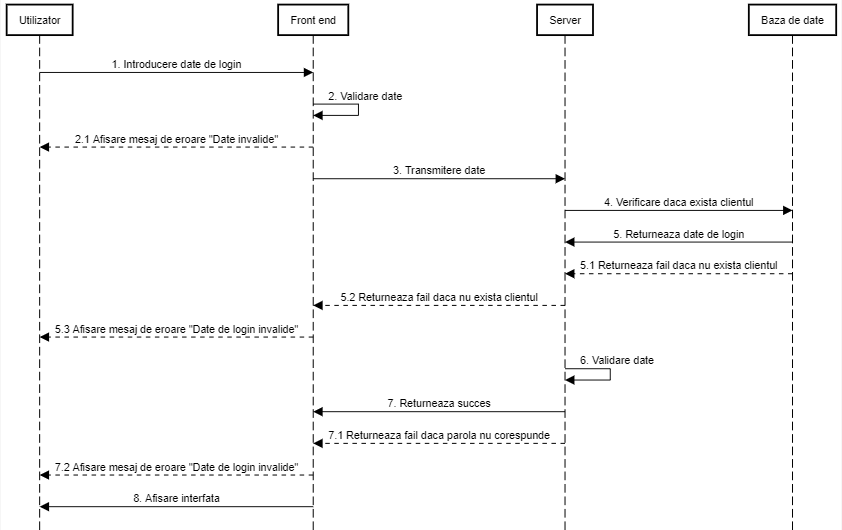


Figura 4 Diagrama secventiala Login

# Studiu de piata / Metode existente

## In lume

### Londra, Anglia

Intr-un oras extrem de aglomerat cum este Londra, un sistem de bike-sharing este vital pentru transportul din oras. Programul a inceput in 2010 si este unul dintre cele mai cunoscute sistem de bike-sharing din lume.

Proiectul a inceput cu doar 5000 de biciclete, dar acum a ajuns la peste 11000 si peste 750 de statii de inchiriere a acestuia.

Totusi, sistemul a avut parte de multe critici, deoarece foarte multi utilizatori au raportat probleme ale sistemului. Acestia au criticat sistemul pentru probleme de functionare (durate mai lungi al inchirierii, sistemul de blocare/deblocare a bicicletelor nefunctional, etc), dar acestea au fost rezolve de companie, dupa upgrate-ul facut sistemului.

O alta problema, au fost bicicletele de start, care au aveau o greutate destul de mare si facand transportul in zonele aglomerate sau cele de deal, destul de dificile. [1][7]

### New York City, U.S.

New York City a fost unul dintre primele orase din U.S. care au adoptat un sistem de bike-sharing la un nivel larg. Programul a inceput in anul 2013, cu 6000 de biciclete si 330 de locatii, dar in anul 2017, a ajuns sa isi dubleze numarul bicicletelor.

Facut dupa modelul sistemului din Montreal, sistemul a fost foarte bine primit de utilizatori. Dar, ca in alte orase, sistemul a avut de suferit din cauza vandalismului si a furturilor, cauzate de probleme ale sistemul de blocare. Pe langa problema aceasta, utilizatorii au devenit nemultumiti dupa o perioada, datorita preturilor. [2][7]

### Monreal, Canada

Acest sistem este un model pentru foarte multe oras din lume (New York City, etc) si este considerat unul dintre cele mai bune sisteme de bike-sharing din lume.

Desi numarul bicicletelor 5200 si al locatiilor 460 este relativ mic, fata de alte orase, sistemul este foarte folosit, avand incasari anulale de 3 milioane de dolari.

Ca si cel din New York City, si acest sistem a avut parte de probleme cu vandalismul si furturile, dar si , din cauza folosiri neglijente si excesive, bicicletele au trebuit reparate sau inlocuite dupa un timp relativ scurt. [3][7]

## In Romania

### Cluj

Sistemul de bike-sharing de aici, a fost inaugurat recent, pornind cu 540 de biciclete si 42 de locatii. Scopul acestui sistem a fost reducerea traficului si protejarea mediului si pentru primele 30 de minute, este gratis.

Numarul mic de biciclete si al locatiilor fata de alte sisteme, a fost un motiv de controverse, dar organizatorii au asigurat ca vor creste numarul acestora, o data cu cererea. [4]

### Timisoara

Primul sistem de bike-sharng implementat in Romania (1 iulie 2015), contine 25 de locatii si 300 de biciclete.

Desi sistemul a fost foarte bine primit de utilizatori, foarte multi dintre acestia au avut experiente neplacute cu aplicatia.

Una dintre principalele probleme cu acest sisem a fost timpul de incarcare destul de mare, interfata incarcata si foarte inceata.

Alte probleme intampinate de utilizatori au fost: bicicletele standard, nu exista diversificare, biciclete mici sau prea mari, grele. [5]

### Bucuresti

Acest sistem a fost deschis pe 27 septembrie 2016, modelul fiind ca cel din Timisoara. Sistemul dispune de 210 biciclete si 12 statii amplasate in zonele cele mai importante ale orasului.

Sistemul a fost criticat din cauza numarului mic de locatii, pentru capitala Romaniei si pentru inexistenta diversificarii tipurilor de biciclete. [6]

# Solutia propusa

## Prezentarea generala a arhitecturii

### Pagina web

O pagina web sau „webpage” este un document scris in HyperText Markup Language (HTML) care este accesibila folosind internetul. Aceasta este accesata folosind o adresa URL.

De obicei furnizorul de informatii iși grupeaza și organizeaza paginile in cadrul unui site web, care la cerere, și pentru a putea fi regasit, primește un identificator Internet unic numit URI.

O pagina web poate contine:

• texte in diferite stiluri (culori, fonturi,etc.)

• imagini

• audio

• continut multimedia - jocuri

• miniaplicatii

[8]

### Aplicatie single-page

O aplicatie single-page (SPA – Single Page Application )reprezinta o aplicatie web sau un site web care se incadreaza pe o singura pagina web cu scopul de a oferi utilizatorului experienta similara cu cea a aplicatiilor desktop. Intr-un SPA, tot codul necesar (HTML, JavaScript si CSS) este fie extras cu o singura incarcare de pagina ori, resursele sunt incarcate dinamic si adaugate paginii doar cand este necesar (de obicei la raspunderea actiunilor utilizatorului). Pagina nu se reincarca, in niciun moment al procesului, nici nu controleaza transferul catre alta pagina, desi in location hash sau in istricul HTML5 poate fi folosit pentru a oferi perceptia si navigabilitatea a paginilor logice separate din cadrul aplicatiei. Interactiunea cu aplicatiile single-page implica deseori comunicarea dinamica cu serverul web din „spatele scenei”. [9]

### Arhitectura MVC

Arhitectura generala a proiectului va fi descrisa de Modelul View Controller (MVC). Acest model arhitectural are rolul de a separa programul in trei sectiuni principale: Modelul, Controller-ul si View-ul.

Ca si celelalte arhitecturi software, MVC exprima „miezul solutiei” la o problema, oferind posibilitatea acesteia sa se adapteze fiecarui sistem. Arhitecturile MVC particulare pot varia semnificativ de la modelul original.

Majoritatea aplicatiilor web moderne folosesc cereri de tip AJAX. De aceea, odata cu evolutia limbajului JavaScript, acest model este tot mai des intalnit si utilizat pentru astfel de proiecte. Elementele prezentate in acest model sunt compatibile cu aceste tehnologii.

Modelul este responsabil cu manipularea operatiilor logice si cu utilizarea informatiei (trimisa dinainte de catre rangul sau superior) pentru a rezulta de o forma usor de inteles.

View-ul are sarcina reprezentarii grafice, adica a ultimei forme a datelor (interfata grafica ce interactioneaza cu utilizatorul final). Evidentiaza informatia obtinuta pana ce ajunge la controller.

Controller-ul permite accesul la aplicatie. Pot fi fisiere, scripturi sau programe (in general orice tip de informatie permisa de interfata). In acest fel putem diversifica continutul de o forma dinamica si statica in acelas timp.



Figura 5 Schema modelului arhitectular MVC

Pentru o aplicatie web este necesar sa stabilim urmatoarele elemente:

• O baza MCV: alcatuia din cele 3 elemente cu mentiunea ca: controller-ul trebuie sa fie capabil de a manipula rute, fisiere, clase, metode si functii; model-ul este asemanator unui script obisnuit intr-un server, doar ca regrupat sub un model reutilizabil; view-ul este asemanator includerii unui fisier in aplicatia noastra.

• Un sistem format din: Router (pentru a putea partaja cerintele fara multe conditionale) si Loader.

Desi MVC-ul a fost initial dezvoltate pentru desktop computing, acesta a fost adoptat la o scara larga ca si arhitectura pentru aplicatii World Wide Web in limbaje mari de programare. [10]

### Web API

Web API este o interfata de program fie pentru un server web sau un browser web. Este un concept de dezvoltare web, de obicei limitat la aplicatiile din partea clientului (incluzand orice framework web folosit), si astfel de obicei nu include server web sau detalii de implementare a browserului cum ar fi SAPI-uri sau API-uri, daca nu sunt acesibile in mod public de catre o aplicatie web de la distanta. [11]

## Tehnologii folosite

### Front-end

#### HTML5

HTML5 este ultima evolutie a standardului HTML. Este cea mai noua versiune a limbajului HTML, care aduce noi elemente, atribute si un numar mare de tehnologi care permit procesului de dezvoltare a aplicatiilor sa fie mai diversificat.

[12]

#### CSS3

CSS (Cascading Style Sheets) este un standard pentru formatarea elementelor unui document HTML. Stilurile sunt atasate documentului HTML prin fisiere externe, scripturi, sau tag-uri de tip style

CSS3 reprezinta ultima versiune a standardului CSS ce aduce ca noutate atribute mult mai diversificate care ii ofera dezvoltatorului mai multa libertate.

Mai jos sunt prezente in lista cele mai importante modulele adaugate in CSS3:

• Selectors

• Box Model

• Backgrounds and Borders

• Image Values and Replaced Content

• Text Effects

• 2D/3D Transformations

• Animations

• Multiple Column Layout

• User Interface

Deși au aparut unele deficiente de compatibilitate intre browsere, majoritatea proprietatilor CSS3 au fost implementate cu succes in variantele browserelor noi.

**Exemplu CSS3 border-radius:**

div {

border: 2px solid #333333;

padding: 10px 40px;

background: #dddddd;

width: 300px;

border-radius:25px;

}

[13]

#### JavaScript

JavaScript (JS) este un limbaj interpretat, orientat pe obiect cu „first-class functions”. Este cel mai cunoscut limbaj de scripting pentru pagini web, dar poate fi folosit si in multe aplcatii non-browser.

Este un limbaj dinamic ce ruleaza pe partea de client side a unei aplicatii si este folosit pentru design, pentru interactivitatea aplicatiei si pentru a programa comportamentul aplicatiei la aparitia unui eveniment.

Fiind un limbaj multi-paradigm, JavaScript permita mai multe stiluri de programare bazate pe evenimente(poate fi folosit si ca limbaj orientat pe obiect, dar si ca limbaj functional).

[14]

#### JSON

JSON (JavaScript Object Notation) este un format ușor de transmitere a datelor. Este un limbaj usor de interpretat si parsat de catre calculator.

JSON este un limbaj independent si este o alternativa a XML-ului pentru transmiterea datelor de la server la aplicatie web.

Este un tip de date universal, avand posibilitatea integrarii acestuia cu toate limbajele de programare. [16]

#### Angular 6

Angular (referit in general ca “Angular 2+” sau “Angular 2) reprezinta o aplicatie web open-source de tip platforma front-end, bazata pe scripting. Este dezvoltata de Angular Team la google si de o comunitate de persoane si corporatii pentru a adresa toate partile workflow-ului dezvoltatorului in timp ce acesta creaza aplicatii web complexe. Angular este o rescriere completa a aceleiasi echipa ce a fabricat AngularJS.

Angular 6 a fost lansat pe 4 mai 2018. Aceasta a fost o lansare majora si se focuseaza mai putin pe frameworkul de baza, si mai mult pe toolchain si pe a-l face mai usor de mutat cu Angular pe viitor, cum ar fi: ng update, ng add, Angular Elementa, Angular Material + CDK Components, Angular Material Starter Components, CLI Workspaces, Library Support, Tree Shakable Providers, Animations Performance Improvements, si RxJS v6. [17]

#### Node.js (NPM)

Node.js este un mediu run-time, open-source, cross-platform, Javascript pentru executare de cod JavaScript pe partea de server. Node.js face posibila scriptarea pe partea deserver a lui JavaScript si ruleaza scripturile pe partea de server pentru a produce continut de pagina web dinamica inainte ca pagina sa fie trimisa la browserul web al utilizatorului. Consecvent, Node.js a devenit una din elementele de baza ale paradigmei “JavaScript everywhere”, permitand dezvoltarilor de aplicatie web sa se unifie in jurul unui singur limbaj de programare, in loc sa se bazeze pe un limbaj diferit pentru a scrie scripturi partiale de server.

Desi .js este conventia extensiei de fisier pentru cod JavaScript, numele “Node.js” nu se refera la un tip de fisier particular in acest context, ci este pur si simplu numele produsului. Node.js are o arhitectura bazata pe event-uri capabil de asincronicitate I/O. Aceste alegeri de design au ca scop optimizarea output-ului si scapabilitatii in aplicatii web cu multe operatii input/output, la fel ca si aplicatiile web real-time (cum arfi programele de comunicatie real-time si jocurile browser).

Proiectul Node.js distributed development, condus de Fundatia Node.js este facilitat de programul proiectelor colaborative a fundatiei Linux.

Node.js a fost construit de Google V8 JavaScript engine de cand a devenit open-source sub DSB license, extrem de rapid si competent cu fundamentele internetului cum ar fi HTTP, DNS, TCP. De asemenea, JavaScript a fost un limbaj binecunoscut, facand imediat Node.js acesibil intregii comunitati de dezvoltare web.

Node.js poate fi combinat cu un browser, o baza de date care suporta JSON (cum ar fi Postgres, MongoDB, CouchDB) pentru o stiva unificata de JavaScript development. Odata cu adaptarea a ceea ce era esential pe partea de patternuri de dezvoltare a serverului (cum ar fi MVC, MVP, MVVM, etc), Node.js permite refolosirea aceluias model de servicii de interfata intre partea de client si partea de server. [18]

npm este un gestionar de pachete pentru programarea in JavaScript. Reprezinta gestionarul de pachete general pentru JavaScript runtime environment Node.js. Este format dntr-o linie de comanda client, de asemenea nuimta npm, si o baza de date online avand atat pachete publice cat si pachete platite si private numit registrul npm. Registrul este accesat prin client, si pachetele disponibile pot fi cautate prin website-ul npm. Gestionarul de pachete si registrul sunt controlate de npm Inc. [19]

#### Google Maps API

Google Maps este un serviciu de web mapping dezvoltat de Google. Ofera imagini de satelit, harti de strada, vederi panoramice de 360 de grade ale strazilor (Street View), conditii de trafic real-time (Goodle Traffic), si planificari de rute pentru traseeele pe jos, cu masina, bicicleta , sau cu transportul public.

Google Maps a inceput ca un program desktop C++ creat de Lars si Jens Elisturp Rasmussen. In Octombrie 2004, compania a fost cumparata de Google, care a convertit aplicatia sa fie web. Serviciile de front end utilizeaza JavaScript, XML si Ajax. Google Maps ofera un API care permite hartilor sa fie embedded pe site-uri third-party, si ofera un localizator pentru afacerile urbane si alte organizatii in numeroase tari din jurul lumii. Google Map Maker a permis utilizatorilor sa extinda in mod colaborativ si sa updateze serviziile de mapping din toata lumea, dar acest lucru a fost intrerupt in martie 2017.

Vederea satelit a lui Google Map este una “top-down” sau “birds eye”. Majoritatea imaginilor de rezolutie inalta a oraselor reprezinta fotografii din avion luate in timpul zborului la o altitudine de 240 pana la 460m, in timp ce majoritatea celorlalte imagini sunt facute din satelit. Majoritatea imagisticii din saltelit nu are mai mult de 3 ani si este updatata regulat. Google Maps foloseste un varianta apropiata a proiectarii Mecrator, si astfel nu poate arata cu precizie zonele din jurul polilor. [20]

### Back-end

#### .Net

.NET Framework este un framework software dezvoltat de Microsoft care ruleaza, in principal, pe Microsoft Windows. Include un mare class library numit Framework Class Library (FCL) si ofera interoperabilitate de limbaj (fiecare limbaj poate folosi cod scris in alte limbaje) peste cateva limbaje de programare. Programele scris pentru .NET Framework executa, intr-un mediu software (in contrast cu mediul hardware) numit Common Language Runtime (CLR), o aplicatie masina virtuala care ofera servicii precum securitatea, gestionarea de memorie si gestionarea executiei.

FCL ofera interfata utilizator, acces de date, conectivitate la baza de date, criptografie, dezvoltare de aplicatii web, algoritmi numerici si comunicatii de retea. Programatorii produc software combinand codul lor sursa cu frameworkul .NET si alte librarii. Frameworkul este intentionat de a fi folosit de cele mai noi aplicatii create pentru Windows. [21]

#### Entity Framework

Entity Framework (EF) este un framework object-relational mapping (ORM) open source pentru ADO.NET. A fost parte din framwrokul .NET, dar de cand a aparut versiunea 6 a lui Entity framework, acesta a fost separat de .NET.

Entity Framework reprezinta un set de tehnologii in ADO.NET care ofera suport pentru dezvoltarea orientata pe date a aplicatiilor software. Dezvoltatorii de aplicatiile orientate pe date s-au confruntat, de regula, cu nevoia de a atinge doua obiective foarte diferite. Ei trebuie sa modeleze entitatile, relatiile si logica problemelor de afaceri pe care o rezolva, si ei trebuie de asemenea sa lucreze cu datele enginurilor folosite pentru stocarea si colectarea acestora. Datele ar putea sa acopere mai multe sisteme de stocare, fiecare cu propriile protocoale; chiar si aplicatiile care lucreaza cu un sistem de stocare de oe singura pagina trebuie sa echilibreze cererile sistemului de stocare, in ciuda cererilor de scriere si eficienta si mentenanta codului aplicatiilor. [22]

#### Braintree API

Braintree, o subdivizie a lui PayPal, este o companie specializata in sisteme de plata mobile si web pentru companiile e-commerce.

Braintree API reprezinta un tool care ofera simplicitate cu un set de featureuri complexe, permitand utilizatorilor sa se extinda.

Cum functioneaza:

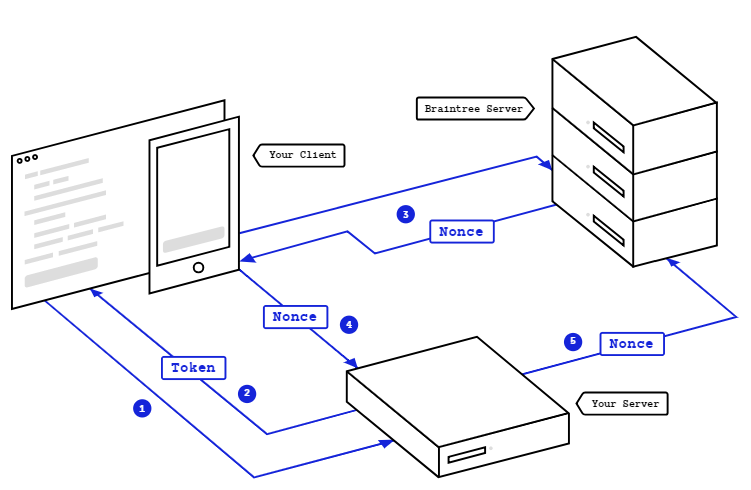


Figura 6 Etape comunicare Braintree

1. Aplicatia sau front-endul tau web trimite o cerere de client token de la serverul tau pentru a initializa SDK client.

2. Serverul tau genereaza si trimite un client token inapoi la clientul tau cu serverul SDK

3. Odata ce clientul SDK este initializat si utilizatorul a submis informatia platii, SDK-ul comunica acea informative si returneaza un payment method nonce

4. Tu apoi vei trimite acest payment method nonce serverului tau

5. Codul tau server primeste payment method nonce de la clientul tau si apoi foloseste serverul SDK pentru a crea o tranzactie sau pentru a apela alte functii [23]

### Baza de date

#### SQL server

Microsoft SQL Server este un sistem de gestionare a bazelor de date relationale, dezvoltat de Microsoft. Ca si server de baza de date, este un produs software cu functia primara de stocare si aducere de date ca urmare a unei cereri a altor aplicatii software, ceea ce ar putea rula fie pe acleasi computer sau pe un altul de-a unei retelee (inclusiv Internetul).

Microsoft vinde cel putin doisprezece editii diferite ale serverului Microsoft SQL, avand ca tinta diferite tipuri de clienti si pentru cantitatile de date de la aplicatii mici pe o singura masina la aplicarii mari cu Internet-facing cu mai multi utilizatori concurenti. [24]

# Detalii de implementare

## Structura bazei de date

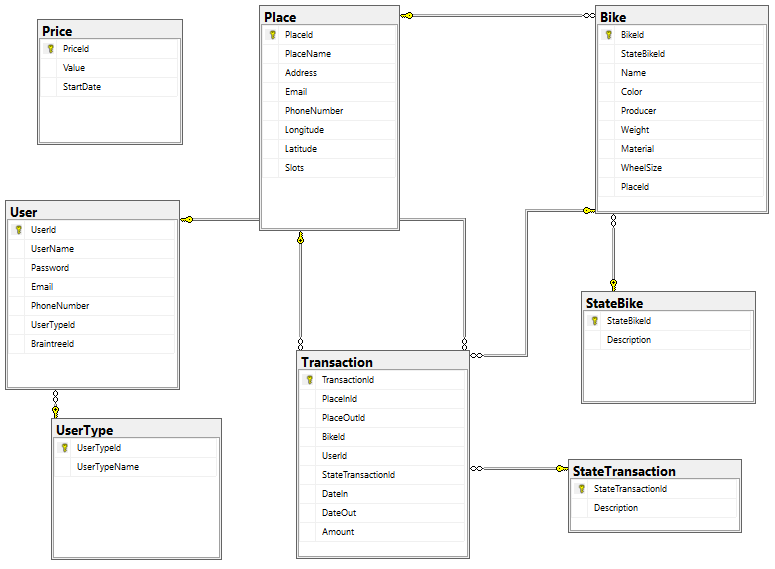


Figura 7 Diagrama baza de date

### User

In tabela User sunt retinute informatiile aferente utilizatorului, administrator sau clienti.

Aceasta tabela contine:

* userId: cheia primara a tabelei
* userName: numele clientului
* password: contine un hash generat pe baza parolei, aplicand SHA256
* email: adresa de email a clientului, necesara pentru login
* phoneNumber: numarul de telefon
* braintreeId: corespondentul customerId generat de Brantree API

### UserType

Tabla UserType este o tabela de tip nomenclator, in care sunt retinute tipurile de clienti existente – administrator sau client. Pe baza acestor informatii, utilizatorul are un anumit nivel de acces.

Tabela contine:

* userTypeId: cheia primara a tabelei
* userTypeName: numele tipului de client

### Place

In tabela Place sunt stocate datele fiecarei locatii de unde se poate lua/duce o bicicleta.

Tabela contine:

* placeId: cheia primara a tabelei
* placeName: nume reprezentativ al locatiei
* adress: adresa locatiei
* email/phone numer: informatii de contact
* longitude/latitude: informatii necesare pentru Google Maps API
* slots: numarul maxim de locuri pentru biciclete

### Bike

In aceasta tabela sunt retinute datele despre bicicletele valabile

Tabela contine:

* bikeId: cheia primara a tabelei
* stateBikeId: cheia de legatura catre tabela StateBike folosita pentru a afla starea bicicletei
* name/color/producer/weight/material/wheelSize: informatii descriptive despre biciclete
* placeId: cheia de legatura catre tabela Place, folosita pentru a afla in ce locatie se afla aceasta

### BikeState

Tabla BikeState este o tabela de tip nomenclator, in care sunt retinute stariile in care pot fi gasite bicicletele. Pe baza acestor informatii, o bicicleta poate fi selectata/ sau nu de un client.

Tabela contine:

* bikeStateId: cheia primara a tabelei
* description: starea in care se afla bicicleta

### Transaction

Tabla Transaction este tabela ce contine cele mai multe informatii. Aici sunt retinute informatiile despre locatie de start sau cea de finish, bicicleta, costul, etc.

Tabela continue:

* transactionId: cheia primara a tabelei
* placeInId: cheia de legatura catre tabela Place folosita pentru a afla locul de start al tranzactiei/locul de unde este luata bicicleta
* placeOutId: cheia de legatura catre tabela Place folosita pentru a afla locul de finish al tranzactiei/locul unde ajunge bicicleta
* userId: cheia de legatura catre tabela User folosita pentru a determina utilizatorul tranzactiei
* stateTransactionId: cheia de legatura catre tabela StateTransaction folosita pentru a afla starea tranzactiei
* dateIn/dateOut: data si ora de start/finish a tranzactiei
* amount: costul tranzactiei

### TransactionState

Tabla TransactionState este o tabela de tip nomenclator, in care sunt retinute stariile in care se pot afla transatiile – activa sau finalizata.

Tabela contine:

* transactionStateId: cheia primara a tabelei
* description: starea in care se afla tranzactia

### Price

In tabela Price sunt tinute preturile per ora a inchirierii unei biciclete. Este sigura tabela care nu are legatura cu celealte, legatura fiind facuta in functie de data in care este introdus noul pret.

Tabela contine:

* priceId: cheia primara a tabelei
* value: valoarea per ora a tranzactiei
* startDate: data de la care incepe noul pret

## Back-end

### Modele

Folosind Entity Framework Database First, au fost generate modele pe baza tabelelor din baza de date. Fiecare tabela are o clasa corespondenta.

### Clase auxiliare

Clase auxiliare au fost create pentru pastra modularitatea aplicatiei si pentru a reutiliza in mod cat mai eficient codul.

#### Helper

Clasa Helper are doua metode ce sunt folosite pentru Sign Up si Login.

* GenerateSHA256String – genereaza un hash pe baza parolei utilizatorului, aplicand SHA256
* GetBraintreeId – obtine proprietatea BraintreeId din tokenul generat la Login

#### ApplicationOAuthProvider

Clasa ApplicationOAuthProvider este folosita pentru autentificare. Autentificarea se face folosind protocolul OAuth2 ce presupune validarea credentialelor utilizatorului si generarea unui token pe baza unor informatii ale utilizatorului: userName, userId, userType, BraintreeId.

var claims = new Claim[] {

new Claim(ClaimTypes.Name, user.UserName),

new Claim(ClaimTypes.Sid, user.UserId.ToString()),

new Claim(ClaimTypes.Role, userTypeName),

new Claim(ClaimTypes.Actor, user.BraintreeId ?? "")

};

ClaimsIdentity oAuthIdentity = new ClaimsIdentity(claims,OAuthDefaults.AuthenticationType);

#### ApplicationRefreshTokenProvider

Clasa ApplicationRefreshTokenProvider este folosita pentru crearea refresh token-ului specific clientului, dar si regenerarea lui dupa expirare. Perioada acestuia de expirare este de 10 min, fata de acces token care are o perioada de expirare de 5 min. Acest token este folosit pentru generarea acces token-ului.

context.Ticket.Properties.ExpiresUtc = new DateTimeOffset(DateTime.Now.AddMinutes(10));

context.SetToken(context.SerializeTicket());

#### CustomBraintreeService

Aceasta clasa este folosita ca legatura intre aplicatie si API-ul de Braintree. Clasa are ca proprietate privata Gateway-ul necesar comunicarii cu API-ul.

static CustomBraintreeService()

{

Gateway = new BraintreeGateway {

Environment = Braintree.Environment.SANDBOX,

MerchantId = "8s7ch7vyr7nzv7ds",

PublicKey = "7gt4k47cfvcbf2df",

PrivateKey = "332d3b2e37dae9e307ae0bc8a7ce7db9"

};

}

Metode:

* createPurchase – creeaza plata in urma terminarii tranzactiei
* createToken – creeaza un token necesar in interfata pentru accesarea metodei de plata
* createCustomerId – genereaza un client unic pe serverul de Braintree folosind informatiile utilizatorului logat si intoarce id-ul acelui client
* updatePaymentMethod – adauga datele de card ale clientului
* customerHasPaymentMethod – verifica daca clientul are datele de card completate

#### WebApiConfig

Aceasta clasa este folosinta pentru a configura rutele catre partea server side.

config.Routes.MapHttpRoute (

name: "NonAction",

routeTemplate: "api/{controller}/{id}",

constraints: new {id = @"\d+" },

defaults: new { id = 0 }

);

config.Routes.MapHttpRoute (

name: "Action",

routeTemplate: "api/{controller}/{action}/{id}",

defaults: new { id = RouteParameter.Optional }

);

### Controlere

Aplicatia fiind de tip Web API, in fiecare controler exista un set de metode, universale, care au rolul de a manipula datele dinspre front-end spre baza de date, sau invers.

Aceste metode sunt:

* save – metoda ce se ocupa cu salvarea datelor, folosing Entity Framework
* delete – metoda ce se ocupa cu stergerea datelor
* getList – metoda ce returneaza toate datele din tabela respectiva, pentru a fi afisate in interfata
* get – metoda folosita pentru aducerea unei singure inregistrari, pentru modului de editare

#### UserController

Acest controller este folosit pentru gestionarea operatiilor efectuate pe tabela User.

Pe langa metodele, standard, acest controller are si metodele specifice pentru sign-up si de creere a metodelor de plata.

Din motive de securitate si pentru a micsora numarul de date retinute pentru client, datele de card ale clientului nu sunt salvate in baza de date, ci sunt salvate in Braintree, corespondenta facandu-se folosind BrainTreeId.

* getClientToken – apeleaza API-ul de Braintree pentru generarea unui token necesar
* getCurrentUser – afla utilizatorul curent
* hasPaymentMethod/setPaymentMethod – se verifica daca utilizatorul are completate datele cardului si daca este nevoie, se actualizeaza
* signup – se salveaza datele introduse de client, parola este salvata in baza de date dupa ce este aplicat protocolul de SHA256 si este creat un Customer pe serverul de BrainTree folosing informatiile utilizatorului.

## Front-end

### Modele

Folosind extensia TypeWriter, au fost generate modele de TypeScript pe baza tabelelor din baza de date. Fiecare tabela are o clasa corespondenta.

### Clase auxiliare

#### CustomHttp

Aceasta clasa este folosita pentru a gestiona apelurile catre server. Apelurile trebuie sa contina Header-ul de HTTP „Authorization” ce contine tokenul de acces OAuth. Fara acest token serverul ar raspunde cu un status code „401 Unauthorized”

constructor(private http: Http, private route: ActivatedRoute, private router: Router) {}

….

this.http.get("/api/bike/getBikeStates")

.catch(error => {let errorMessage = (error.message) ? error.message :

error.status ? `${error.status} - ${error.statusText}` : `Server error`;

console.log(errorMessage);

return Observable.throw(errorMessage);}

)

.subscribe(…)

constructor(private http: CustomHttp, private route: ActivatedRoute, private router: Router) {}

….

this.http.get("/api/bike/getBikeStates")

.subscribe(..)

#### AuthenticationGuard

Clasa AuthenticationGuard este folosita pentru a restrictiona accesul utilizatoriilor.

Pentru a accesa orice cale din aplicatie, utilizatorul trebuie sa fie logat si trebuie sa aiba drepturi asupra acelui ecran. Exceptie fac login si sign up.

#### LoaderService

Clasa LoaderService este folosita pentru afisarea imaginii de load peste tot in aplicatie unde este nevoie.

#### UserService

Clase de UserService este folosita pentru a gestiona operatiile de login, sign-up, logout pe partea de front-end a aplicatiei. Tot aici, sunt gestionate operatiile de token, refresh token si refresh user. Acest serviciu retine utilizatorul curent.

### Componente

Pentru entitatile ce necesita doar operatiile CRUD, au fost create cate o pereche de componente care se ocupa de aceste actiuni. Spre exemplu pentru biciclete exista *BikesComponent* care afiseaza lista de biciclete si permite selectarea unei inregistrari pentru editare sau stergere, si componenta de *BikeDetailsComponent* care este folosita pentru adaugarea sau editarea proprietatilor unei biciclete.

Pentru componentele asociate locatiilor, este folosit API-ul de Google Maps atat pentru afisare cat si pentru introducerea lor.

#### LoginComponent

Componenta de login este cea care este incarcata automat cand utilizatorul acceseaza aplicatia.

In metoda *ngOnInit* mai intai se verifica daca exista deja un token de OAuth in serviciul *CustomHTTP.* Daca exista, asta inseamna ca utilizatorul este deja logat si este redirectionat mai departe in aplicatie.

if (this.http.getToken() && !this.userService.currentUser.value) {

this.http.get("/api/user/getCurrentUser")

.subscribe(response => {

this.userService.currentUser.next(response.json());

let returnUrl = this.route.snapshot.queryParams['returnUrl'] || '/';

this.router.navigateByUrl(returnUrl);

});

}

Metoda de *login* este asociata evenimentului de submit al form-ului. In aceasta metoda mai intai se verifica daca email-ul introdus este valid, si daca nu, este afisat un mesaj de eroare. Daca da, atunci folosind serviciul de *UserService* se face autentificarea in aplicatie.

#### SignupComponent

Componenta este folosita pentru crearea unui utilizator nou sau editarea informatiilor utilizatorului logat.

In metoda *ngOnInit* se stabileste daca este o operatie de editare a utilizatorului current sau de inregistrare a unui utilizator nou.

In metoda de *save* se face mai intai validarea email-ului si se verifica daca parola corespunde cu cea de confirmare. Daca verificarea trece cu success atunci se creaza un obiect cu informatiile introduce si se face un apel de HTTP POST catre metoda corespunzatoare salvarii informatiilor din controllerul *UserController*. Daca se adauga un utilizator nou sau a fost modificata parola, atunci este nevoie sa se faca o autentificare folosind *UserService* pentru a se genera un nou token de OAuth. Asemenea si in cazul in care s-a modificat email-ul pentru utilizatorul curent.

save(event: any): void{

event.preventDefault();

if(!this.validateForm())

return;

let user = {

userName: this.userName,

password: this.password,

email: this.email,

phonenumber: this.phoneNumber

};

this.http.post("/api/user/" + (this.isEdit ? "edit" : "signup"), user)

.subscribe(response => {

//if !isEdit or password has changed

if (!this.isEdit || this.password) {

this.userService.login(this.email, this.password)

.subscribe(response => {

this.router.navigate(['/']);

});

}

//if email has chenged, refresh the user

if (this.email != this.userService.currentUser.value.email) {

this.userService.refreshUser().subscribe(response => {

this.router.navigate(['/']);

});

}

});

}

#### PaymentDetailsComponent

Componenta este folosita pentru introducerea/modificarea cardului folosit pentru tranzactii. Componenta foloseste componenta de *ngx-braintree*, care la randul ei foloseste libraria de “Drop-In UI” oferita de Braintree pentru introducerea/alegerea metodei de plata.

### AppModule

Aceasta clasa reprezinta modulul de Angular pentru intreaga aplicatie.

Sectiuni:

* declarations: Lista de componente ce fac parte din aplicatie
* providers: Lisa de servicii care pot fi oferite prin Dependency Injection componentelor sau altor servicii.
* Imports: lista de module Angular ce contin componente sau servicii necesare functionarii aplicatiei. In aceasta lista se gaseste *RouterModule* in care sunt configurate rutele aplicatiei, componentele care sunt apelate in urma accesarii lor, informatii transmise componentelor, si daca este necesara autentificarea pentru accesarea rutei (*AuthenticationGuard*).

RouterModule.forRoot([

{ path: 'login', component: LoginComponent },

{ path: 'logout', component: LogoutComponent },

{ path: 'signup', component: SignupComponent },

{

path: '',

canActivate: [AuthenticationGuard],

canActivateChild: [AuthenticationGuard],

children: [

{ path: 'account-details', component: SignupComponent, data: { editDetails: true } },

{ path: 'payment-details', component: PaymentDetailsComponent },

{path: 'payment-details/change', component: PaymentDetailsComponent, data: { freshToken: true } },

{ path: 'place', component: PlacesComponent },

{ path: 'place/:id', component: PlaceDetailsComponent, data: { userTypeId: 1 } },

{ path: 'transaction', component: TransactionsComponent },

{ path: 'transaction/:id', component: TransactionDetailsComponent },

{ path: 'bike', component: BikesComponent },

{ path: 'bike/:id', component: BikeDetailsComponent, data: { userTypeId: 1 } },

{ path: 'price/:id', component: PriceDetailsComponent },

{ path: 'price', component: PricesComponent },

{ path: '\*\*', redirectTo: '' }

]

}])

# Evaluare

Fiind o aplicatie web, testarea performantei depinde de foarte multi factori externi, precum viteza internetului, viteza de raspuns a server-ului de Google Maps, respectiv Braintree. Din aceste motive, am decis de a face teste de response time a server-ului.

Am folosit tool-ul Apache JMeter, care simuleaza un numar mare de cereri catre server si analizeaza timpul de raspuns al tuturor cererilor transmise catre server pentru un anumit scenariu. [25]

## Scenarii

### Login

Un utilizator acceseaza aplicatia, isi introduce datele de autentificare, apasa butonul de „Login” si asteapta sa fie autentificat.

Acest scenariu a fost testat pentru 1000 de utilizatori simultani.

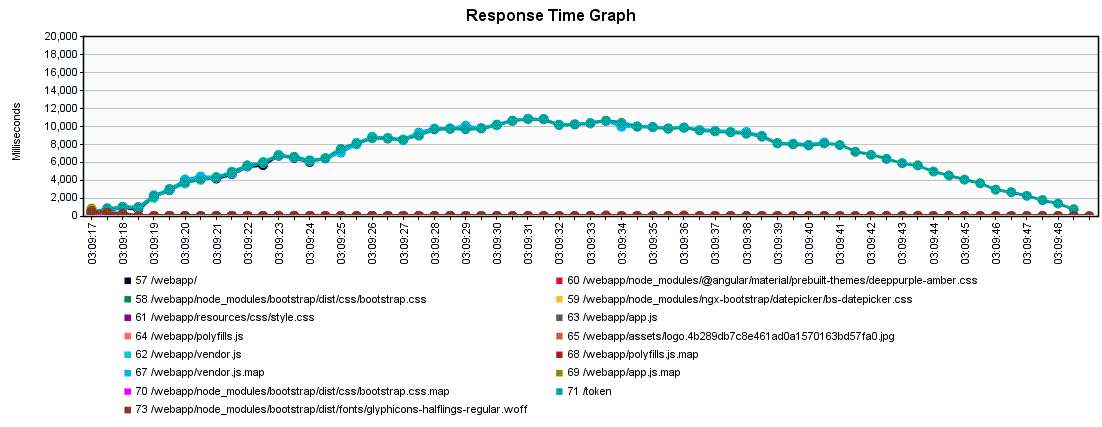


Figura 8 Grafic timp de raspuns „Login”

Tabel 1 – Scenariu de „Login”

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Label | Nr. | Average | Min | Max | Std. Dev. | Err % | Throughput | Received KB/s | Sent KB/s | Avg. Bytes |
|  |  |  |  |  |  |  |  |  |  |  |
| /webapp/ | 1000 | 3308 | 3 | 6899 | 1966.21 | 0.00% | 77.75445 | 151.79 | 29.92 | 1999 |
| /node\_modules/@angular  /material/prebuilt-themes/deeppurple-amber.css | 1000 | 2 | 0 | 71 | 4.72 | 0.00% | 77.84524 | 582.17 | 32.76 | 7658 |
| /node\_modules/bootstrap  /dist/css/bootstrap.css | 1000 | 2 | 0 | 86 | 4.56 | 0.00% | 77.85737 | 1633.56 | 31.1 | 21485 |
| /node\_modules/ngx-bootstrap/datepicker/bs-datepicker.css | 1000 | 1 | 0 | 71 | 4.22 | 0.00% | 77.86343 | 267.35 | 31.86 | 3516 |
| /resources/css/style.css | 1000 | 0 | 0 | 67 | 3.01 | 0.00% | 77.87555 | 173.09 | 29.43 | 2276 |
| /app.js | 1000 | 2 | 0 | 936 | 29.83 | 0.00% | 77.88162 | 1159.55 | 27 | 15246 |
| /polyfills.js | 1000 | 0 | 0 | 60 | 2.31 | 0.00% | 77.89982 | 160.52 | 27.46 | 2110 |
| /assets/logo.4b289db7c  8e461ad0a1570163bd  57fa0.jpg | 1000 | 1 | 0 | 81 | 3.67 | 0.00% | 77.90589 | 5943.75 | 30.2 | 78125 |
| /vendor.js | 1000 | 5987 | 43 | 10464 | 3012.02 | 0.00% | 42.9203 | 43698.12 | 15.01 | 1042557 |
| /polyfills.js.map | 1000 | 3 | 0 | 577 | 26.61 | 0.00% | 43.01075 | 99.04 | 14.03 | 2358 |
| /vendor.js.map | 1000 | 7992 | ## | 10935 | 2811.82 | 0.00% | 31.81167 | 34295.89 | 10.28 | 1103965 |
| /app.js.map | 1000 | 3 | 0 | 446 | 15.99 | 0.00% | 31.97647 | 723.62 | 10.24 | 23173 |
| /node\_modules/  bootstrap/dist/css/  bootstrap.css.map | 1000 | 3 | 1 | 494 | 16.69 | 0.00% | 31.97544 | 2345.11 | 11.46 | 75101 |
| /token | 1000 | 6014 | ## | 10892 | 2835.85 | 0.00% | 31.36566 | 51.83 | 14.4 | 1692 |
| /node\_modules/  bootstrap/dist/fonts  /glyphicons-halflings-regular.woff | 1000 | 2 | 0 | 694 | 22.39 | 0.00% | 32.18953 | 744.23 | 15.5 | 23675 |
|  |  |  |  |  |  |  |  |  |  |  |

### Sign up

Din ecranul de login, utilizatorul apasa pe butonul de „Signup”, introduce datele necesare, apasa pe butonul de „Save” si asteapta sa fie autentificat in aplicatie.

Acest scenariu a fost testat pentru 100 de utilizatori simultani.

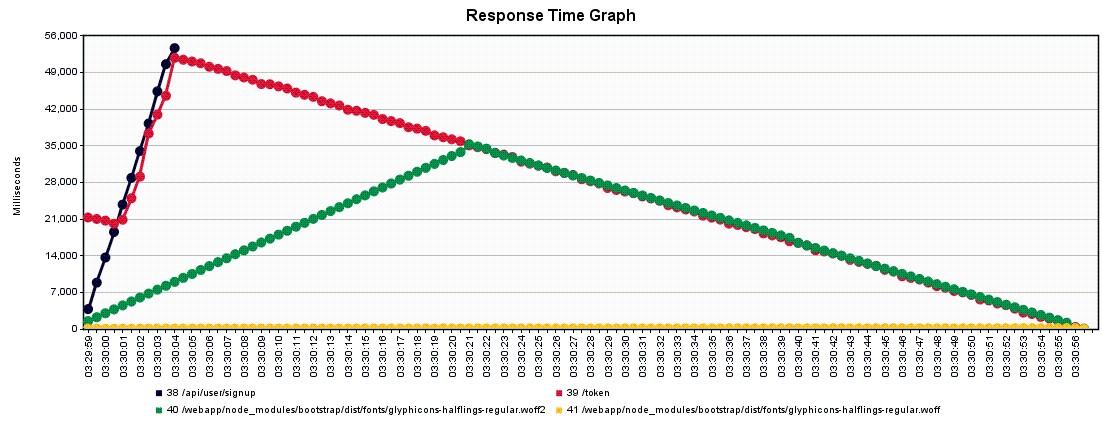


Figura 9 Grafic timp de raspuns „Signup”

Tabel 2 – Scenariu de „Signup”

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Label | Nr. | Average | Min | Max | Std. Dev. | Err % | Throughput | Received KB/s | Sent KB/s | Avg. Bytes |
|  |  |  |  |  |  |  |  |  |  |  |
| /api/user/signup | 100 | 27057 | 1875 | 53628 | 15162.63 | 0.00% | 1.70669 | 0.44 | 0.86 | 261 |
| /token | 100 | 26291 | 3 | 51760 | 15200.54 | 0.00% | 1.76432 | 2.84 | 0.81 | 1648 |
| node\_modules/bootstrap/dist/fonts/glyphicons-halflings-regular.woff | 100 | 31 | 0 | 222 | 47.54 | 0.00% | 76.27765 | 1763.55 | 36.72 | 23675 |
| /api/user/signup | 100 | 27057 | 1875 | 53628 | 15162.63 | 0.00% | 1.70669 | 0.44 | 0.86 | 261 |
|  |  |  |  |  |  |  |  |  |  |  |

### Start transaction

Din ecranul de Transactions, utilizatorul apasa pe butonul de „Start Transaction”, completeaza campurile de „Start place” si „Bike”, apasa pe butonul de save si asteapta raspunsul de la server.

Acest scenariu a fost testat pentru 1000 de utilizatori simultani.

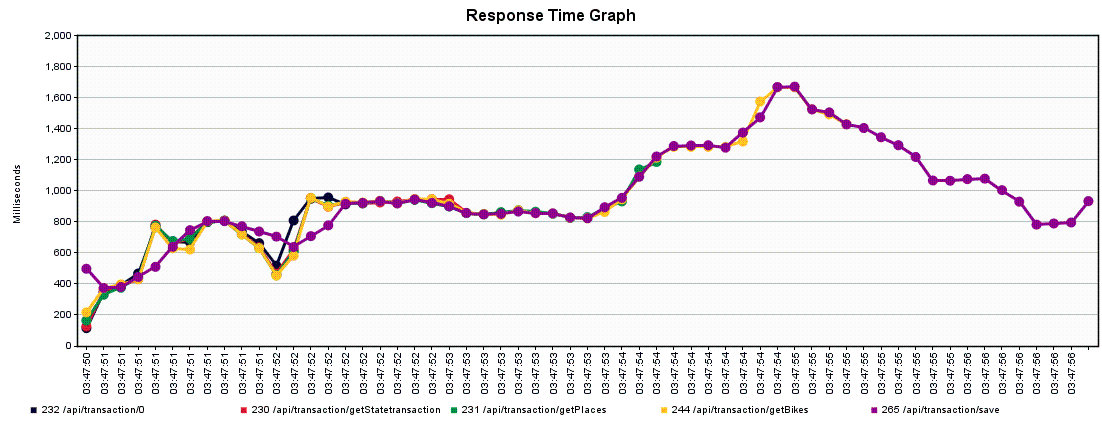


Figura 10 Grafic timp de raspuns „Start Transaction”

Tabel 3 – Scenariu de „Start transaction”

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Label | Nr. | Average | Min | Max | Std. Dev. | Err % | Throughput | Received KB/s | Sent KB/s | Avg. Bytes |
|  |  |  |  |  |  |  |  |  |  |  |
| /api/transaction/0 | 1000 | 684 | 4 | 985 | 230.17 | 0.00% | 412.3711 | 122.38 | 370.89 | 303.9 |
| /api/transaction/  getStatetransaction | 1000 | 794 | 7 | 964 | 184.9 | 0.00% | 304.1363 | 106.03 | 278.89 | 357 |
| /api/transaction/getPlaces | 1000 | 866 | 29 | 1188 | 154.44 | 0.00% | 223.914 | 104.96 | 203.14 | 480 |
| /api/transaction/getBikes | 1000 | 1169 | 127 | 1672 | 323 | 0.00% | 170.6776 | 55.34 | 156.34 | 332 |
| /api/transaction/save | 1000 | 1070 | 373 | 1673 | 245.71 | 0.00% | 150.1953 | 74.26 | 160.46 | 506.3 |
|  |  |  |  |  |  |  |  |  |  |  |

# Concluzii

In aplicatia de fata am oferit posibilitatea gestionarii inchirierii bicicletelor prin intermediul unei aplicatii web.

Scopul a fost creerea unei aplicatii usor de folosit, rapida, ce poate fi folosita de oricine isi doreste inlocuirea mijlocului de transport traditional.

1. arhitectura MVC si single-page sunt foarte importante in dezvoltarea aplicatiei, deoarece sunt un factor important in dezvoltarea unei aplicatii rapide si functionale.

2. desi harta Google Maps intarzie incarcarea paginii, este foarte importanta pentru o aplicatie ce are la baza locatii.

3. cu cat orasele se dezvolta mai mult, traficul si aglomeratia o sa fie in continua crestere, deci astfel de aplicatii sunt necesare.

4. este important sa ne facem timp si pentru miscare, iar mersul cu bicicleta este un mod foarte buna pentru a face asta.

In viitor as dori sa simplific si mai mult interactiunea intre aplicatie si client, creerea unei aplicatii mobile, care sa ofere posibilitatea creearii unei tranzactii mult mai usor, printr-o scanare QR. Sau utilizarea uner device-uri specifice care sa permita autentificarea utilizatorului si efectuarea transactiei fara a fi nevoie de aplicatia web, aceasta fiind folosita doar pentru a afla disponibilitatea bicicletelor in diferite locuri si istoricul tranzactiilor.

## Dezvoltari ulterioare

### Device-uri specializate

Pentru a simplifica utilizarea aplicatiei, o dezvoltare importanta ar fi utilizarea uner device-uri, aflate la fiecare locatie, ce permit utilizatorului sa se autentifice si pentru a selecta bicicleta dorita.

Dupa selectie, sistemul va debloca slotul bicicletei automat si o noua tranzactie va fi creeata.

La aducerea bicicletei, utilizatorul va trebui sa se autentifice din nou si sa selecteleze slotul unde va lasa bicicleta. Dupa selectie, sistemul va bloca slotul si va inchide tranzactia.

Pe tot parcursul acestui proces, aplicatia web va permite utilizatorului sa un istoric al tranzactiilor cat si al tranzactiilor in curs.

Aceste device-uri vor inlocui introducerea manuala a tranzactiilor de catre utilizator.

### Aplicatie mobile

O alta dezvoltare ce ar ajuta la simplificarea procesului de inchiriere este creearea unei aplicatii mobile, care printr-o simpla scanare QR sa creeze/inchida o anumita tranzactie si sa blocheze/deblocheze o bicicleta, fiecare slot de bicicleta avand cate un cod QR asignat.

Pentru inchirierea unei biciclete, utilizatorul va trebui sa intre in aplicatie si sa scaneze codul, iar aplictia deschide o tranzactie noua si deblocheaza slotul respactiv. Pentru lasarea bicicletei, procesul este la fel, clientul scaneaza codul, iar aplicatia inchide tranzactia si blocheaza slotul.

Aplicatia mobile va inlocui introducerea manuala a tranzactiilor de catre client, ca si device-urile, dar, fata de acestea, va simplifica procesul de inchiriere, ne mai fiind nevoie de autentificare pe un alt device de fiecare data.

# Bibliografie

[1] <https://tfl.gov.uk/modes/cycling/santander-cycles>

[2] <https://www.citibikenyc.com/>

[3] <https://montreal.bixi.com/en>

[4] <http://clujbike.eu/>

[5] <http://www.velotm.ro/>

[6] <https://ivelo.ro/urban/>

[7] https://www.icebike.org/bike-share-programs/

[8] https://www.computerhope.com/jargon/w/webpage.htm

[9] https://msdn.microsoft.com/en-us/magazine/dn463786.aspx

[10]https://developer.mozilla.org/en-US/Apps/Fundamentals/Modern\_web\_app\_architecture/MVC\_architecture

[11] https://www.asp.net/web-api

[12] https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5

[13] https://www.w3schools.com/css/

[14] https://www.javascript.com/

[15] https://jquery.com/

[16] https://www.json.org/

[17] https://blog.angular.io/version-6-of-angular-now-available-cc56b0efa7a4

[18] https://nodejs.org/en/

[19] https://www.npmjs.com/

[20] https://cloud.google.com/maps-platform/

[21] https://www.microsoft.com/net/

[22] https://msdn.microsoft.com/en-us/library/aa937723(v=vs.113).aspx

[23] https://developers.braintreepayments.com/

[24] <https://searchsqlserver.techtarget.com/definition/SQL-Server>

[25] https://jmeter.apache.org/

# Anexe

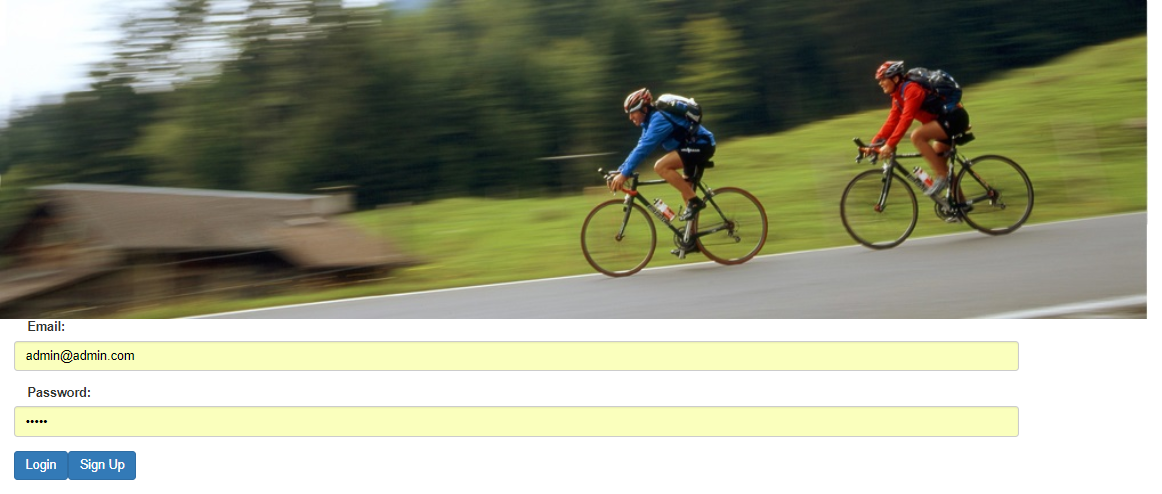


Figura 11 Ecran Login

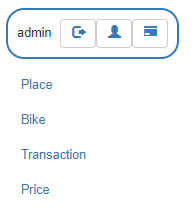


Figura 12 Meniu

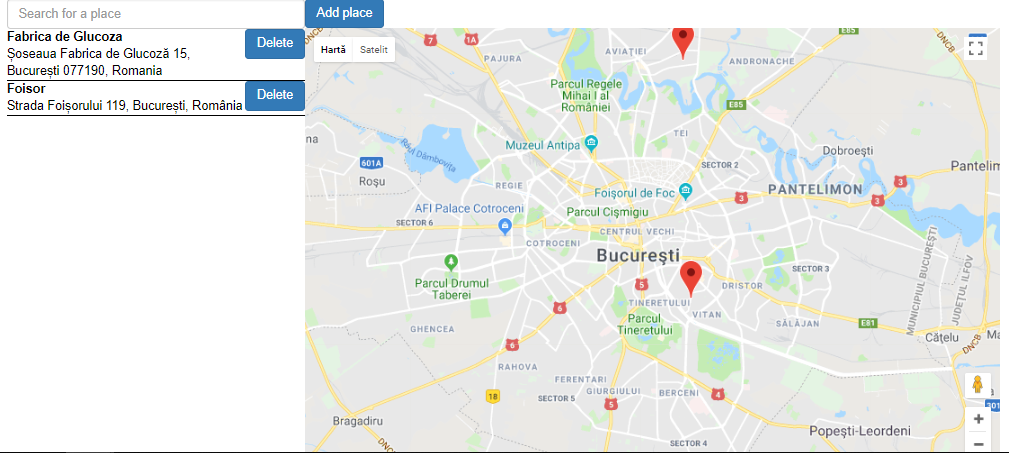


Figura 13 Places

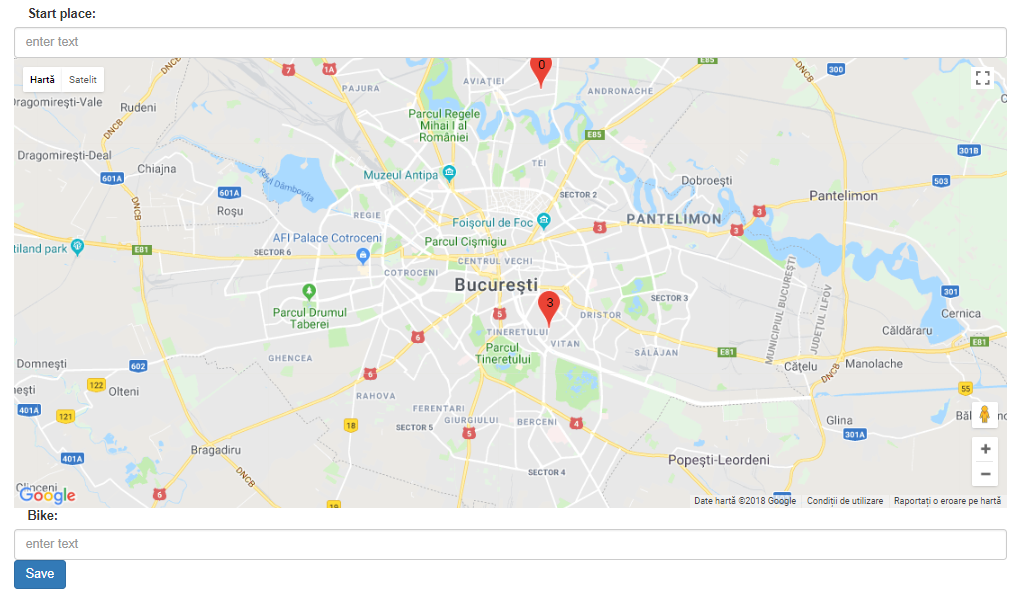


Figura 14 Start transaction



Figura 15 Adaugare date card