MINISTERUL EDUCAȚIEI NAȚIONALE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

PLANIFICATOR PERSONAL PENTRU OPTIMIZARE RUTE ȘI ACTIVITĂȚI

LUCRARE DE LICENŢĂ

Absolvent: **Ioana CRISTEA**

Coordonator ştiinţific: **Asist. Ing. Cosmina IVAN**

2020

MINISTERUL EDUCAȚIEI NAȚIONALE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

MINISTERUL EDUCAȚIEI NAȚIONALE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

DECAN,
Prof. dr. ing. Liviu MICLEA

DIRECTOR DEPARTAMENT,
Prof. dr. ing. Rodica POTOLEA

Absolvent: **Ioana CRISTEA**

Planificator personal pentru optimizare rute si activitati

1. **Enunţul temei:** Proiectul își propune implementarea unui platforme web pentru planificarea și managementul locațiilor și generarea de rute favorabile pentru utilizatorii care sunt în continuă mișcare. Sistemul a fost gândit atât pentru planificare zilnică, cât și pentru planificare săptămânală.

2. **Conţinutul lucrării:** Curpins, Introducere, Obiectivele Proiectului, Studiu Bibliografic, Analiză și Fundamentare Teoretică, Proiectare de Detaliu și Implementare, Testare și Validare, Manual de Instalare și Utilizare, Concluzii, Bibliografie și Anexe

3. **Locul documentării**: Universitatea Tehnică din Cluj-Napoca, Departamentul Calculatoare

4. **Consultanţi**: Asist. Ing. Cosmina Ivan

5. **Data emiterii temei:** 1 noiembrie 2019

6. **Data predării:** 8 iulie 2020

Absolvent: _____

Coordonator ştiinţific: _____

MINISTERUL EDUCAȚIEI NAȚIONALE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

MINISTERUL EDUCAȚIEI NAȚIONALE

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

Declaraţie pe proprie răspundere privind autenticitatea lucrării de licenţă

Subsemnatul(a)_____
_____,
legitimat(ă) cu _____ seria _____ nr. _____
CNP _____, autorul lucrării
_____
_____
_____elaborată în vederea susţinerii examenului de finalizare a studiilor de licență la Facultatea de Automatică și Calculatoare, Specializarea _____ din cadrul Universităţii Tehnice din Cluj-Napoca, sesiunea _____ a anului universitar _____, declar pe proprie răspundere, că această lucrare este rezultatul propriei activităţi intelectuale, pe baza cercetărilor mele şi pe baza informaţiilor obţinute din surse care au fost citate, în textul lucrării, şi în bibliografie.

Declar, că această lucrare nu conţine porţiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislaţiei române şi a convenţiilor internaţionale privind drepturile de autor.

Declar, de asemenea, că această lucrare nu a mai fost prezentată în faţa unei alte comisii de examen de licenţă.

In cazul constatării ulterioare a unor declaraţii false, voi suporta sancţiunile administrative, respectiv, *anularea examenului de licenţă*.

Data

_____

Nume, Prenume

_____

Semnătura

Cuprins

Capitolul 1. Introducere 1
1.1. Contextul proiectului 1
1.2. Motivația 1
1.3. Conținutul lucrării 2
Capitolul 2. Obiectivele Proiectului 3
2.1. Obiectivul principal 3
2.2. Obiectivele specifice 3
2.3. Obiectivele generale 4
Capitolul 3. Studiu Bibliografic 5
3.1. Planificare și optimizare 5
3.2. Evoluția tehnologică. Necesități și consecințe 6
3.3. Hărți digitale 6
3.3.1. Google Maps 6
3.3.2. OpenStreetMap 8
3.3.3. Comparație între Google Maps și OpenStreetMap 9
3.4. Sisteme similare 10
3.4.1. RouteXL 10
3.4.2. ViaMichelin 11
3.4.3. Lyft 11
3.4.4. Concluzii 12
Capitolul 4. Analiză şi Fundamentare Teoretică 13
4.1. Cerințe funcționale 13
4.2. Cerințe non-funcționale 14
4.3. Arhitectura conceptuală 14
4.3.1. Arhitectura sistemului de rutare 15
4.4. Cazuri de utilizare 16
4.4.1. Diagrame de utilizare 17
4.4.2. Descriere cazuri de utilizare 19
4.5. Tehnologiile client 25
4.5.1. React JS 25
4.5.2. HTML, CSS, SCSS 26
4.5.3. Node Package Manager 27

4.5.4. JSON Web Token 27
4.5.5. Axios 28
4.6. Tehnologiile server 28
4.6.1. Spring Boot 28
4.6.2. JPA , Hibernate 29
4.6.3. Servicii REST 29
4.6.4. SMTP 30
4.7. Tehnologii storage 30
4.7.1. MySQL 30
4.8. Tehnologii de management 31
4.8.1. Apache Maven 31
4.9. Servicii Google Maps 32
Capitolul 5. Proiectare de Detaliu si Implementare 33
5.1. Arhitectura componentei backend 33
5.1.1. Descriere generală 33
5.1.2. Descrierea componentelor 34
5.2. Arhitectura componentei frontend 35
5.2.1. Descriere generală 35
5.2.2. Descrierea componentelor 36
5.3. Structura bazei de date 42
5.3.1. Descrierea tabelelor 43
5.4. Mecanismul de optimizare activități 45
5.4.1. Notații și valori de referință 45
5.4.2. Sugestiile aplicației 46
5.4.3. Pseudocod 47
Capitolul 6. Testare şi Validare 49
6.1. Testare manuală 49
6.2. Testare automată 51
Capitolul 7. Manual de Instalare si Utilizare 52
7.1. Resurse necesare 52
7.2. Instalare și rulare 52
7.3. Instrucțiuni de utilizare 54
7.3.1. Pagina principală 55
7.3.2. Înregistrare și autentificare 56

7.3.3. Planificare activități 57
7.3.4. Selectarea rutei optime 59
7.3.5. Vizualizarea planurilor salvate 60
Capitolul 8. Concluzii 61
8.1. Analiza rezultatelor obținute 61
8.2. Dezvoltări ulterioare 62
Bibliografie 63
Anexa 1
8.2.1. Lista de tabele 64
8.2.2. Lista de figuri 64

Capitolul 1. Introducere

”Plans are of little importance, but planning is essential” – Winston Churchill, former British Prime Minister

”Lack of direction, not lack of time, is the problem. We all have twenty-four hours days.” -Zig Ziglar

”The bad news is time flies. The good news is you’re the pilot” – Michael Altshuler

Cele trei citate de mai sus subliniază cele trei concepte esențiale care stau la temelia elaborării acestei teme , mai exact : planificare , managementul timpului și control asupra activităților .

Zig Ziglar , în citatul de mai sus pune accent pe ideea că de cele mai multe ori suntem copleșiți de atribuțiile pe care trebuie să le ducem la bun sfârșit, și de fapt cea mai mare problemă nu este incapacitatea de a le îndeplini ci faptul că suntem invadați de numere și de estimarea timpului total pe care trebuie să îl investim. Acești factori psihologici care ne fac să avem blocaje deoarece ne împiedicăm de propriile temeri a lipsei de timp și picăm în plasa sentimentului de incapacitate de a controla ce se întâmplă în jurul nostru . De aceea, așa cum sugerează Michael Altshuler, trebuie să fim conștienți că nu putem controla trecerea timpului însă ce putem controla este modul în care ne organizăm activitățile astfel încât să nu fie un timp pierdut. Ținând cont de cele rezumate , cel mai bun mod de a ne organiza astfel încât să fim învingători în raport cu gestionarea timpului este planificarea .

1.1. Contextul proiectului

Proiectul își propune implementarea unui platforme web pentru planificarea și managementul activităților și generarea de rute favorabile pentru utilizatorii care desfășoară activități diverse în locații diferite. Sistemul a fost gândit atât pentru planificare zilnică, cât și pentru planificare săptămânală.

1.2. Motivația

Motivul alegerii acestei teme, planificare de activități bazate pe coordonate geografice îl reprezintă utilitatea pe care o poate aduce aplicația . Proiectul a fost construit în așa măsură încât să vină în ajutorul utilizatorilor din mai multe tipuri de categorii , fie că este vorba de un agent imobiliar care trebuie să se deplaseze zilnic în mai multe locații , fie că este vorba de un șofer de taxi sau doar un turist într-un oraș nou care are nevoie să-și planifice drumeția . Scopul acestei aplicații este de a oferi o perspectivă asupra timpului consumat prin activități și asupra traseelor alternative din care utilizatorul poate alege .

În majoritatea aplicațiilor, ideea de planificare este foarte mult axată pe traseu, însă, în această aplicație accentul este pus pe activitățile de planificat cu ajutorul traseului . Drumul de parcurs este influențat de mijloacele de transport iar utilizatorul este responsabil în alegerea unui traseu în defavoarea celuilalt în funcție de valorile estimative generate de aplicație în ceea ce privește consumul de timp, energie și buget.

1.3. Conținutul lucrării

În această secțiune vor fi prezentate capitolele acestei lucrări alături de o descriere sumară referitor la conținutul fiecăruia .

- **Capitolul 1. Introducere** prezintă contextul în care este realizat proiectul, motivația de la care s-a pornit și abordarea problemei de planificare și optimizare a rutelor. De asemenea este prezentată și divizarea lucrării pe capitole
- **Capitolul 2. Obiectivele Proiectului** conține descrierea obiectivului principal urmat de descrierea obiectivelor generale și specifice ce trebuie duse la bun sfârșit pentru consolidarea obiectivului principal
- **Capitolul 3. Studiu Bibliografic** reprezintă un sumar ar informațiilor extrase în partea de cercetare în care sunt explicate conceptele de planificare și optimizare, urmate de descrierea a două celor mai populare hărți digitale iar în final analiza sistemelor similare. Cel din urmă subcapitol este descris cu scopul de a realiza o comparație între sistemul propus și acele sisteme similare .
- **Capitolul 4. Analiză și Fundamentare Teoretică** prezintă cerințele funcționale și non-funcționale, arhitectura conceptuală, cazurile de utilizare cu diagrame și descrierea lor, tehnologiile client, server, persistență și management folosite iar la final descrierea serviciilor Google Maps
- **Capitolul 5. Proiectare de Detaliu și Implementare** conține descrierea în detaliu a componentelor arhitecturii sistemului care este împărțită în trei module : componenta de frontend, componenta de backend și componenta de persistență, urmate de prezentarea mecanismului de optimizare a activităților
- **Capitolul 6. Testare și Validare** prezintă modul în care au fost testate componentele : manual pentru partea de frontend, prin black box și white box și automat pentru partea de backend, printr-o aplicație specializată
- **Capitolul 7. Manual de Instalare și Utilizare** oferă o ghidare pas cu pas pentru instalarea și configurarea resurselor necesare astfel încât proiectul să fie disponibil în forma sa finală . Pe lângă această îndrumare, sunt prezentate instrucțiuni de utilizare, cu imagini captate din aplicație .
- **Capitolul 8. Concluzii** conține observațiile finale extrase din analiza rezultatelor obținute, relativ la obiectivele propuse pentru proiect, dar și idei pentru dezvoltări ulterioare cu soluții posibile pentru implementarea lor .

Capitolul 2. Obiectivele Proiectului

Acest capitol prezintă obiectivul principal al acestei lucrări urmat de două subcapitole ce descriu sumar obiectivele specifice și generale care au dus la realizarea obiectivului principal .

2.1. Obiectivul principal

Scopul principal al acestui proiect este proiectarea și implementarea unei aplicații web care să realizeze planificarea activităților zilnice sau săptămânale în funcție de traseul optim selectat pentru nevoile și cerințele utilizatorului . Proiectul își propune să ofere o imagine de ansamblu, o ordonare cronologică a activităților, un raport vizual și tabelar comparativ pentru cele trei tipuri de traseu : cu mașina, cu bicicleta și mers pe jos plus o sugestie bazată pe distanțe.

2.2. Obiectivele specifice

- → **Înregistrarea de utilizatori noi** : această acțiune are ca efect posibilitatea de creare de planuri, de ale vizualiza; un utilizator neautorizat poate doar să navigheze, să creeze activități, dar nu poate să creeze planuri
- → **Autentificare** : utilizatorii autentificați pot să își personalizeze contul creat modificând date de profil, datele de autentificare , încărcând poze de profil și au acces liber la toate funcționalitățile definite de aplicație pentru un utilizator, spre deosebire de un utilizator neautentificat care are anumite scenarii restricționate
- → **Alegerea zonei de interes** : acest scenariu permite utilizatorilor aplicației să caute un loc (o adresă) și să vizualizeze harta cu locul selectat
- → **Elaborarea unui plan** : reprezintă totalitatea pașilor necesari pentru crearea unui plan printre, de la selectarea tipului de planificare (zilnică sau săptămânală) , adăugarea de markere cu activități ,modificarea sau ștergerea unor activități , până la apăsarea butonului de salvare a planului
- → **Managementul planurilor** : utilizatorii autentificați au posibilitatea de a-și vizualiza planurile create și activitățile asociate fiecărui plan dar și rapoarte grafice cu conținut cronologic; administratorul în schimb poate să vizualizeze și să filtreze toate planurile existente în aplicație
- → **Comunicare și interacțiune** : email-ul este puntea de legătură între administrator și utilizatori , utilizatorii autentificați pot trimite email administratorului pentru a cere detalii sau amănunte despre specificațiile aplicației iar administratorul poate folosi această punte de legătură pentru a transmite oferte , noutăți , etc.
- → **Secțiunea de marketing** : utilizatorii neautentificați pot vedea ultimele evaluări ale utilizatorilor referitor la aplicație iar utilizatorii autentificați, în plus , pot adăuga propriile evaluări

2.3. Obiectivele generale

Pentru a îndeplini cerințele fiecărui obiectiv specific a fost necesar ca proiectul să îndeplinească anumite standarde de calitate . Cel mai important aspect îl reprezintă **securizarea** aplicației prin stocarea datelor de autentificare a utilizatorilor în mod criptat în baza de date , prin separarea rutelor utilizatorilor logați de cei neautentificați și prin setare de sesiuni . Al doilea aspect este **utilizabilitatea** prin ușurința de navigare și de întelegere a aplicației date de o interfață prietenoasă. O altă cerință care trebuie îndeplinită este **performanța** aplicației , dată de timpul de răspuns a request-urilor, care trebuie să fie cât mai mic și ultimul aspect de care trebuie ținut cont : **portabilitatea** ,care este asigurată prin definiție , prin faptul că este o aplicație web care poate rula pe browsere web variate.

Capitolul 3. Studiu Bibliografic

Acest capitol pune în lumină contextul în care se încadrează proiectul ce urmează a fi dezvoltat în capitolele următoare, pornind cu o motivare a necesității implementării unei astfel de teme, raportată la cerințele utilizatorilor țintă, continuând cu beneficiile oferite de cele mai populare hărți digitale și evidențiind caracteristicile sistemelor care au la bază același concept.

Pentru a putea oferi utilizatorilor o experiență cât mai completă în ceea ce privește sistemele bazate pe hărți a fost important de identificat funcționalitățile celor mai populare sisteme similare cu scopul de a le păstra pe cele de bază și pentru a aduce îmbunătățiri și noutăți.

Identificarea celor mai potrivite tehnologii, integrarea și utilizarea acestora a constituit al doilea cel mai important pas, de aceea documentarea și compararea a doua dintre cele mai importante hărți digitale a constituit un factor esențial în decizia de construire a proiectului.

3.1. Planificare și optimizare

Acțiunea de a planifica, conform DEX[1] reprezintă organizarea unei activități întocmind planul după care să se desfășoare diferitele ei faze , a repartiza timpul de muncă pe diferite ramuri de activitate ,a programa . Tipul de planificare abordat în această temă este un mix între este ”Salesperson Plan” și ”Appointment Calendar”, ce reprezintă moduri de planificare explicate și documentate în lucrarea [1]. Pe scurt, primul tip de planificare este un template destinat persoanelor care au multe întâlniri de afaceri în diferite locații și cu diferiți oameni iar al doilea tip de planificare este un șablon pentru activități constrânse de intervale de timp. Pornind de la aceste două tipuri de planificare, pentru proiectul curent a rezultat un nou șablon bazat pe ideea de planificare în funcție de intervale de timp și de constrângerea privind locul de desfășurare a fiecărei activități ,alături de descrierea și etichetarea acestora.

În ceea ce privește conceptul de planificare a rutelor, se poate observa, prin sistemele similare detaliate în subcapitolul 3.5, faptul că a primit o atenție considerabilă din partea dezvoltatorilor de soluții software în ultimii zece ani, prin apariția multor abordări eficiente care derivă din conceptele de bază ale algoritmilor pe grafuri. Din punct de vedere științific, problema de planificare a rutelor poate fi formulată, conform rezultatelor obținute în lucrarea [2] ,ca o variantă a următoarei probleme de optimizare : problema găsirii perechii unice pentru cea mai rapidă cale . Proiectând problematica de optimizare asupra unei rețele de grafuri, funcția construită ar fi reprezentată de alegerea muchiilor cu cost mic în ceea ce privește timpul de călătorie, pornind dintr-un nod etichetat start și încheindu-se într-un nod etichetat destinație. În plus, față de această abordare, în proiectul curent se vor ține cont de criteriile optimale, reprezentând măsurători de calitate precum timp, distanță ,bani consumați pe combustibil și energie consumată prin mișcare. Prioritatea o reprezintă timpul, de cele mai multe ori însă pentru cazuri particulare,

[1] https://www.dex.ro/

prioritatea poate fi reprezentată de economii ale bugetului sau poluare atmosferică cât mai mică.

Optimizarea, la cel mai înalt nivel de abstractizare reprezintă acțiunea de a obține cel mai bun rezultat dintr-o situație sau dintr-o resursă.

3.2. Evoluție prin tehnologie

Odată cu apariția primelor sisteme de calcul, a primului calculator și a primei conexiuni la internet, interesul oamenilor pentru tehnologie a crescut considerabil în ultimele decenii. Acest lucru a fost datorat faptului că prin dezvoltarea tehnologică, calitatea vieții s-a îmbunătățit considerabil, un lucru pe care, fără doar și poate, oricine și-l poate dori. Privind acest aspect, am căutat un domeniu curent, de interes, pentru care omul modern să aibă nevoie de ajutor.

Conform Biroului de Statistica din SUA[2], locurile de muncă cele mai populare din ultima decadă includ : jurnalist, arhitect, manager de vânzări, șofer Uber. Ce au aceste locuri de muncă în comun este nevoia de deplasare constantă între ședințe din diferite locații, între întâlniri de afaceri, dar și întercalarea vieții personale cu cea profesională.

Volumul imens de informații și nevoia organizatorică a adus pe piața media o suită de aplicații cu scopul de a planifica, eficientiza și organiza rutele pe care le au de parcurs utilizatorii. Dintre aceste aplicații cele mai remarcante sunt: RouteXL[3], viamichelin[4], Lyft[5]. Acestea au la bază hărți virtuale ce își au originile din sisteme precum Google Maps[6] sau OSM[7].

3.3. Hărți digitale

Hărțile digitale cu funcționalitățile oferite, ce comportă o evoluție permanentă, constituie un factor determinant în evoluția economică , conform articolului din [3] , apariția acestora a eficientizat rutele de călătorie, reducând cu 12% timpul de călătorie și a creat un sentiment de siguranță în rândul populației care a resimțit utilitatea lor prin utilizarea lor directă sau în diverse aplicații .

Prin acest mod dar și prin alte aplicații care utilizează servicii geospațiale, s-a observat că din numărul total de utilizatori din mediul online din lume, 90% dintre aceștia folosesc hărți digitale.

3.3.1. Google Maps

Google Maps este un serviciu de mapare web dezvoltat de Google. Oferă imagini prin satelit, fotografiere aeriană, hărți stradale, vedere panoramică interactivă la 360 ° a

[2] Biroul de Statistică din SUA : https://www.census.gov/
[3] RouteXL : https://www.routexl.com/
[4] viamichelin : https://www.viamichelin.com/
[5] Lyft : https://www.lyft.com/
[6] Google Maps : https://www.google.ro/maps
[7] OSM : https://www.openstreetmap.org/

străzilor, condiții de trafic în timp real și planificare a rutelor pentru călătorii pe jos, mașină, bicicletă, cu avionul sau transport în comun. Conform articolului din sursa [4] se poate remarca faptul că în 2020 Google Maps a fost utilizat de peste un miliard de oameni în fiecare lună .

- **Google Maps Platform**

Platforma ce favorizează interacțiunea dintre developeri și serviciile oferite de Google Maps se numește Google Maps Platform[8] și este o extensie de la Google Cloud . Această platformă oferă ghidare în ceea ce privește dezvoltarea aplicațiilor software prin tutoriale, ca de exemplu : adăugarea unui marker într-o hartă pe o platformă Web sau identificarea locației curente într-o platformă Android. Oferă la dispoziție librării, documentație, exemple, servicii și asistență, conferind stabilitate și încredere în ceea ce privește construirea unei aplicații pe bază de hărți digitale.

- **Google Maps API**

O caracteristică importanta a Google Maps este faptul că oferă un API prin care este posibilă încorporarea hărților pe site-uri web terțe . Acest aspect, în mod mai concret, se referă la faptul că permite dezvoltatorilor de site-uri și aplicații web sau mobile să construiască aplicații bazate pe locații prin servicii și tool-uri Google, să creeze hărți pentru aplicații mobile, să vizualizeze date geospațiale și să își customizeze hărțile construite.

În ceea ce privește disponibilitatea acestui sistem, din iunie 2005 până în 21 iunie 2018 serviciul a fost gratuit, însă din 16 iulie 2018, pentru a putea accesa API-ul de Google Maps este necesară asocierea unei chei API asociată la un cont Google Cloud cu funcție de facturare activată.

- **Elemente de structură**

Elementele componente ale hărților Google sunt descrise și exemplificate prin tutoriale și limbaje de programare în platforma Google Maps. Mai jos sunt enumerate și descrise cele mai importante patru elemente de structură :

- **Coordonatele logitudine și latitudine**

Longitudinea și latitudinea sunt elemente ce stochează traducerea unei locații de pe harta grafică în coordonate digitale corespunzătoare acesteia. Pentru a se efectua această translatare se utilizează proiecția Mercator[9] .

În acest mod, coordonatele mondiale din Google Maps sunt măsurate de la originea proiecției Mercator (colțul de nord-vest al hărții la 180 de grade longitudine și aproximativ 85 de grade latitudine) și cresc în direcția x spre est (dreapta) și crește în direcția y spre sud (jos).Deoarece placa de bază Google Maps de la Mercator este de 256 x 256 pixeli, spațiul de coordonate din lume utilizabil este {0-256}pentru longitudine, {0-256}pentru latitudine.

[8] Google Maps Platform : https://cloud.google.com/maps-platform
[9] Proiecția Mercator : https://www.britannica.com/science/Mercator_Projection

- **Markere statice și dinamice**

Marker-ul identifică o locație pe o hartă. Aceștia pot fi customizați prin modificarea formei, adăugare de label-uri, animații, setarea lor ca și markeri statici sau dinamici și prin oferirea posibilității utilizatorului de a-i așeza în mod draggable pe hartă.

- **Fereastra de informații**

Acest element de structură afișează conținut (de obicei text sau imagini) într-o fereastră pop-up deasupra hărții, la o anumită locație. Fereastra de informații are o zonă de conținut descriptiv așezată într-o reprezentare de tipul unei tulpini conice. Vârful tulpinii este atașat la o locație specificată pe hartă.
În mod obișnuit, o fereastră de informație este atașată la un marker, dar se poate atașa o fereastră de informație și anumitor coordonate.

- **Bara de căutare**

Bara de căutare automată oferă aplicațiilor comportamentul de căutare avansat al câmpului de căutare Google Maps. Când un utilizator începe să tasteze o adresă, completarea automată va completa restul oferindu-i utilizatorului cele mai bune locații care au începutul denumirilor identice cu primele litere tastate de acesta.

3.3.2. OpenStreetMap

Conform capitolului din [5] , OpenStreetMap (prescurtat OSM) este descris ca un proiect colectiv, în 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 dispozitive GPS.

Datele de la OSM pot fi utilizate în diferite moduri [6], inclusiv producția de hărți de hârtie și hărți electronice (similare cu Google Maps), geocodarea adreselor și a locurilor precum și planificarea rutelor.

- **Portalul de dezvoltare wiki**

Asemeni Google Maps Platform, OSM pune la dispoziție utilizatorilor și dezvoltatorilor de aplicații portalul wiki[10]. Acesta este o platformă web care prezintă caracteristicile produsului OSM, pune la dispoziție un ghid introductiv al aplicației JOSM[11] dar și referințe cu informații despre modul în care OSM poate fi intregat în aplicații de development. Poate fi intregrat atât în aplicații mobile cât și in aplicații web, este flexibil pentru orice platformă iar faptul că este open source oferă un plus de incredere programatorilor.

- **API-ul OSM**

[10] Platforma de informare OSM : https://wiki.openstreetmap.org/wiki/
[11] JOSM : https://josm.openstreetmap.de/

Modul de integrare a sistemului de hărți OSM în aplicații de development se realizează, asemeni ca și în cazul Google Maps, printr-un API denumit API v0.6 care este versiunea actuală a API-ului de editare OSM.
Această editare API se bazează pe structura API-ului REST. Cererile REST au forma de mesaje HTTP GET, PUT, POST și DELETE. Orice sarcină utilă este în formă XML, folosind tipul MIME „text / xml" și codarea caracterelor UTF-8 și poate fi comprimată pe stratul HTTP dacă clientul indică prin antetul „Acceptare" HTTP că poate trata mesaje comprimate.

- **Structura OSM**

OSM folosește o structură de date topologică cu patru elemente de bază : noduri, căi, relații și tag-uri. Nodurile sunt puncte ce reprezintă poziția geografică, stocate sub forma unor perechi de forma latitudine și longitudine conform Sistemului Global de Geodetecție[12] (WGS 84) . Căile sunt liste ordonate de noduri .Relațiile sunt folosite pentru reprezentarea conexiunii dintre un nod curent existent și calea din care diverge.Tag-urile sunt perechi cheie-valoare folosite în stocarea metadatelor despre obiectele hărților.

- **Implementare și integrare**

Actuala implementare completă a serverului este portul Rails. Portul Rails este versiunea actuală de producție a codului serverului OSM - API, web frontend și tot ceea ce rulează în aplicația web OSM[13].
Portul Rails este aplicația web care asigură suport pentru aplicația OSM prin API și contextul pentru care sunt oferite paginile hărților.

Cele mai indicate limbaje de programare și tool-uri prin care se poate realiza integrarea aplicației OSM prin APIv06 sunt :

- Java – in special pentru aplicații desktop
- JavaScript – aplicații web
- Nominatim – tool de căutare în datele OSM prin nume și adresă (geocodată)
- Leaflet – librărie JavaScript, in general folosită pentru aplicații mobile interactive
- C++ - utilizat pentru translatări,randări sau geocodări

3.3.3. Comparație între Google Maps și OpenStreetMap

Așa cum am menționat în introducerea acestui capitol o importanță majoră o reprezintă tipul de hartă digitală ce urmează a fi integrat în proiect . Mai jos am construit un tabel comparativ între două cele mai populare hărți virtuale : Google Maps și OpenStreetMap pentru care am pus în evidență caracteristicile ce le definesc pe fiecare în parte.

[12] Definiție WGS 84 : https://gisgeography.com/wgs84-world-geodetic-system/
[13] OpenStreetMap : https://www.openstreetmap.org/

Table 3.1 Comparație Google Maps și OpenStreet

	Google Maps	**OpenStreetMap**
Suport pe browseri web	6 browseri	4 browseri
Nivelul de zoom maxim	22	19
Tipul de date	imagini satelitate	imagini spațiale
Direcții de orientare	integrate în serviciu	integrate prin third parties
Flexibilitatea de personalizare	salvare și printare locații	salvarea locației de acasă
Nivelul de intregare în aplicații	dezvoltat	dezvoltat
Integrare pe dispozitive mobile	preferință pentru sistemele android	predominant cu ajutorul OsmAnd
Geocodarea	Geocoding API	Nominatim (plus servicii terțiare)
Tipuri de hărți	6 tipuri	5 tipuri
Informații despre trafic în timp real	Da	Posibil prin servicii terțiare
Street View	Da	Nu
Puncte forte	Funcționalități multiple, construit pentru arii extinse	Rapid , construit pentru arii restrânse
Ghiduri de integrare	Tutoriale video , text , exemple	Ghid online , tutoriale mai puțin numeroase

În articolul [7] redactat de Universitatea de Științe Aplicate din Beuth, Berlin sunt conturate diferențele dintre aceste două hărți digitale. Concluzia extrasă din acest articol este că ambele hărți au funcționalitățile de bază integrate, ambele au aceeași structură de bază însă ce le diferențiază este domeniul de aplicabilitate și modul de interacțiune cu dezvoltatorii de aplicații .

Ținând cont că Google Maps are un domeniu de aplicabilitate mai extins iar ghidurile de utilizare și îndrumare sunt mai numeroase decât cele oferite de OSM, plus ușurința de integrare a serviciilor cu care are loc manipularea hărților, am ales ca acest tip de hartă ca fiind cea pe care voi construi aplicația curentă .

3.4. Sisteme similare

3.4.1. RouteXL

RouteXL este un planificator de rute pentru mai multe destinații. Acesta găsește cea mai bună rută multi-opriri pentru livrări, preluări și servicii, folosind un algoritm inteligent de sortare a adreselor cu scopul de a minimiza durata generală a traseului. Permite adaptarea la mai multe tipuri de transport : mers pe jos, mers cu bicicletă , mers cu autoturism sau autobuz.Interfața de utilizator și algoritmii de rutare ai aplicației utilizează date din OpenStreetMap .

Conform informațiilor expuse publicului prin secțiunea de blog a aplicației[14] se poate remarca faptul că adresele sunt localizate pe hartă folosind serviciile de geocodare Mapbox, Photon, Bing, Google, Here, Mapquest și Nominatim. Rutele sunt calculate cu Graphhopper, OSRM și Gosmore iar frontend-ul este construit cu Leaflet și jQuery. Optimizarea rutelor se realizează prin implementarea algoritmului *The Travelling Salesmen Problem* (TSP) [8] .

Problema vânzătorului călător (TSP) pune următoarea întrebare: „Având în vedere o listă de orașe și distanțele dintre fiecare pereche de orașe, care este cea mai scurtă rută posibilă care vizitează fiecare oraș și se întoarce la orașul de origine? ".TSP poate fi modelat sub forma unui graf ponderat nedirectat, astfel încât orașele sunt vârfurile grafului, căile sunt marginile grafului, iar distanța unei căi este greutatea muchiei. Este o problemă de minimizare care începe și se termină la un nod specificat după ce a fost vizitat reciproc exact o dată.

3.4.2. ViaMichelin

ViaMichelin oferă servicii concepute atât pentru publicul larg, cât și pentru afaceri. Compania își folosește expertiza tehnologică pentru a oferi o ofertă completă de servicii (hărți, planuri de rute, listări la hoteluri și restaurante, trafic și informații turistice etc.), pe o gamă largă de mijloace de comunicare, inclusiv internet, telefoane mobile, asistenți digitali personali (PDA) și sisteme de navigație GPS.

Opțiunile de routing și costurile de transport sunt determinate de utilizator. Utilizatorul poate alege tipul de optimizare al traseului dorit : cel mai scurt traseu, cel mai rapid, cel mai economic, cel mai amplu din punct de vedere al monumentelor turistice și recomandarea Michelin .

Aplicația web oferă posibilitatea de a alege între 15 limbi de traducere și identificarea dinamică pe hartă a locurilor de parcare, a benzinăriilor dar și a traficului în timp real în zona de interes a utilizatorului.

Avantajul acestui sistem de ghidare îl reprezintă portabilitatea sa. Acest sistem poate fi integrat atat pe aplicații desktop, dispozitive GPS, dispozitive mobile cât și în modul deja prezentat, aplicație web.

Diferența față de aplicațiile web de planificare a traseului și această aplicație o reprezintă domeniul de interes. ViaMichelin este focusată mai mult pe ramura turistică îmbinată cu planificarea de traseu și optimizarea acestuia .

3.4.3. Lyft

Lyft, Inc. este o companie americană de călătorie cu sediul în San Francisco, California și care operează în 644 de orașe din Statele Unite și 12 orașe din Canada. Aceștia dețin aplicația mobilă Lyft, prin care oferă curse cu mașina, cu scuterul, un sistem de schimb de biciclete dar și un serviciu de livrare a alimentelor.

Cei care doresc să utilizeze serviciile companiei trebuie să descarce aplicația mobilă Lyft pe telefonul lor iOS sau Android, să se înscrie introducând un număr de telefon valid și să introducă o formă de plată valabilă (fie un card de credit, un card cadou Lyft, fie un

[14] routexl.com/blog/about/

link către un Apple Pay, Google Portofel sau cont PayPal). Odată ce călătoria este finalizată, fondurile sunt extrase automat din sursa de finanțare setată de utilizator.

Lyft oferă cinci tipuri de călătorii predefinite în aplicație:

- Shared Ride, care nu este disponibil în toate orașele, este cea mai ieftină opțiune și se va potrivi pasagerilor cu alți călători, dacă aceștia merg în aceeași direcție
- Lyft este oferta de bază și cea mai populară care se potrivește pasagerilor cu șoferii din apropiere
- Lyft XL se potrivește pasagerilor cu un vehicul care poate găzdui cel puțin șase pasageri
- Lux, Lux Black, Lux Black XL se potrivește pasagerilor cu o plimbare cu un vehicul exterior de culoare neagră de lux

Aplicația are la bază sistemul de hărți digitale de la Google și utilizează serviciile acestora pentru identificarea locației pasagerilor, a rutelor optime și pentru vizualizarea vehiculelor în apropiere sau în mișcare, participând activ la estimările de durată conform rutelor alese .

3.4.4. Concluzii

Prin prisma analizei celor mai populare sistemele similare existente pe piață din punct de vedere a funcționalităților dar și a resurselor pe care le folosesc se pot evidenția conceptele care stau la baza unor aplicații ce folosesc hărți, modalitățile de abordare a problematicii și popularitatea pe care o au pentru publicul țintă .

RouteXL este complex, deține multe funcționalități astfel încât să rezolve cerințele unui număr cât mai diversificat de utilizatori . Punctul slab îl reprezintă interfața utilizator . Pentru a putea folosi aplicația este necesară parcurgerea obligatorie a ghidului de utilizare , tocmai datorita numărului mare de funcționalități.

ViaMichelin are o particularitate aparte față de alte sisteme de ghidare prin hartă . Acesta își ascunde foarte bine complexitatea printr-o interfață prietenoasă și ușor de folosit. Este focusată pe un public cât mai larg, dat faptul că are conținutul translatabil în 15 limbi.

Lyft este un sistem portabil focusat pe distanță și calcularea celor mai optime trasee, însă cel mai important lucru pe lângă strânsa legătură cu serviciile de hărți este sistemul de notificare și chat, care transformă ideea ce lucru cu hărți într-o aplicație dinamică cu mai multe tipuri de utilizatori cu diferite interfețe ce interacționează constant .

Pentru proiectul ce urmează a fi elaborat am extras ideea de îmbinare a unui sistem de hărți cu o aplicație web a cărei funcționalitate este planificarea dar și importanța eficientizării rutelor, dat fiind faptul că, indiferent de ce mijloc de transport ar avea la indemână un utilizator, interesul său principal este să obțină cel mai scurt traseu sau cel mai scurt timp de deplasare .

Capitolul 4. Analiză şi Fundamentare Teoretică

4.1. Cerințe funcționale

Cerințele funcționale reprezintă o descriere a comportamentului sistemului software . Aceste cerințe sunt create într-un mod raportat la experiența pe care trebuie să o aiba utilizatorul. Tabelul 4.1 prezentă cerințele funcționale raportate la utilizatori.

Nr	Cerință funcțională	Utilizator
1	**Autentificare / creare utilizator**	
1.1	Înregistrare în aplicație	Registered User, Guest, Admin
1.2	Autentificare pentru utilizatorii înregistrați în aplicație	Registered User, Guest, Admin
1.3	Solicitarea unei parole noi dacă cea veche este uitată	Registered User, Admin
1.4	Editarea informațiilor de profil	Registered User
1.5	Încărcare imagini în format .jpeg pentru propriul cont	Registered User
1.6	Editarea parolei contului	Registered User
2	**Alegerea zonei de interes**	
2.1	Căutarea unui loc	Registered User, Guest, Admin
2.2	Vizualizarea hărții pentru locul căutat	Registered User, Guest, Admin
3	**Elaborarea unui plan**	
3.1	Selectarea unei săptămâni sau o zi pentru operația de planificare	Registered User, Guest, Admin
3.2	Adăugarea unui marker de activitate	Registered User, Guest, Admin
3.3	Ștergerea unei activități selectate	Registered User, Guest, Admin
3.4	Editarea unei activitäti selectate	Registered User, Guest, Admin
3.5	Salvarea planului elaborat	Registered User
4	**Managementul planului**	
4.1	Vizualizarea planurilor asociate propriului cont	Registered User
4.2	Vizualizarea graficelor de activitate pentru planurile asociate propriului cont	Registered User, Admin
4.3	Vizualizarea recomandărilor date de aplicației cu privire la traiectorie	Registered User
4.4	Vizualizarea scanării inteligente date de aplicație pe baza planurilor de activitate	Registered User
4.5	Vizualizarea tuturor planurilor înregistrate în aplicație	Admin
4.6	Filtrarea tuturor planurilor înregistrate în aplicație	Admin
5	**Interfața și marketing-ul aplicației**	
5.1	Utilizatorul poate alege tema de vizualizare dorită	Registered User, Guest, Admin
5.2	Utilizatorul poate evalua aplicația	Registered User
5.3	Utilizatorii pot vedea cele mai recente evaluări ale aplicației	Registered User, Guest, Admin
6	**Managementul comunicării**	
6.1	Trimitere email	Registered User,Admin

Table 4.1 Cerintele funcționale ale sistemului

4.2. Cerințe non-funcționale

Cerințele non-funcționale desemnează constrângerile asupra serviciilor oferite de un sistem software . Aceste cerințe se focusează în principiu pe caracteristici ce ar trebui introduse sau elemente ce ar trebui ascunse astfel încât sistemul să atingă un potențial maxim în ceea ce privește performanța de calcul, abstractizarea și securizarea la un nivel cât mai ridicat iar experiența utilizatorului să nu fie alterată de erori sau alte imperimente.

În tabelul 4.2 de mai jos sunt redate caracteristici esențiale ale sistemului astfel încât aplicația să funcționeze optim.

1	**Securitate**
1.1	Parola stocată în baza de date este criptată
1.2	Securitatea rutelor
1.3	Sesiuni de cookies
2	**Usability**
2.1	Interfață prietenoasă
2.2	Utilizatorul se poate autentifica în aplicație și îl poate utiliza ca membru sau poate naviga ca invitat și poate efectua / vedea o demo
3	**Performanța**
3.1	Proiectat arhitectural pentru a menține o performanță de înaltă calitate pentru experiența utilizatorului
3.2	Request-uri simple care includ timp de răspuns mic
4	Portabilitate
4.1	Compatibilitatea cu browsere web variabile (Chrome, Safari, Opera)

Table 4.2 Cerințele non-funcționale ale sistemului

4.3. Arhitectura conceptuală

În figura 4.1 este prezentată arhitecura conceptuală a sistemului . Se pot observa elementele componente ale unui sistem distribuit :

→ Client (Front-end)
→ Server (Back-end)
→ Baza de Date (Persistența)

Prima componentă (cea din stânga) reprezintă componenta back-end a sistemului, componentă care se ocupă cu gestionarea request-urilor primite de la client și care oferă un răspuns la acestea . Portul pe care rulează această componentă este 8080 . Design-ul arhitecural aplicat este Layered, mai exact, structură pe mai multe nivele :

→ Primul nivel este cel de controller-e. Aici sunt definite metodele RESTful asociate endpoint-urilor create.
→ Al doilea nivel comunică prin Data Transfer Objects (DTOS) .Acest nivel conține logica aplicației .
→ Al treilea nivel este cel care face translatarea informațiilor extrase din baza de date. Acest nivel comunică cu cel de servicii prin entități, punându-i acestuia la dispoziție toate datele necesare pentru construirea informației de transmis către nivelele superioare .

A doua componentă (cea din dreapta) reprezintă componenta front-end a sistemului . Aceasta pune la dispoziție utilizatorului aplicației interfața grafică și ascunde toată complexitatea acesteia . Portul pe care rulează aceasta componentă este 3000. Design-ul arhitectural abordat în această componentă este MVC, astfel că :

→ View-urile sunt reprezentate de componentele HTML integrate in fiecare clasă de tip JS .
→ Modelele sunt proprietățile definite prin stările constructorilor din fiecare clasă.
→ Controller-ele sunt reprezentate de clasele de ApiService , care , prin intermediul serviciului Axios creează request-uri la endpoint-urile serverului.

A treia componentă este baza de date unde are loc persistența datelor din aplicație. Componenta de back-end comunică cu aceasta prin nivelul de Repositories. Portul pe care rulează serverul MySQL al acestei componente este 3306 .

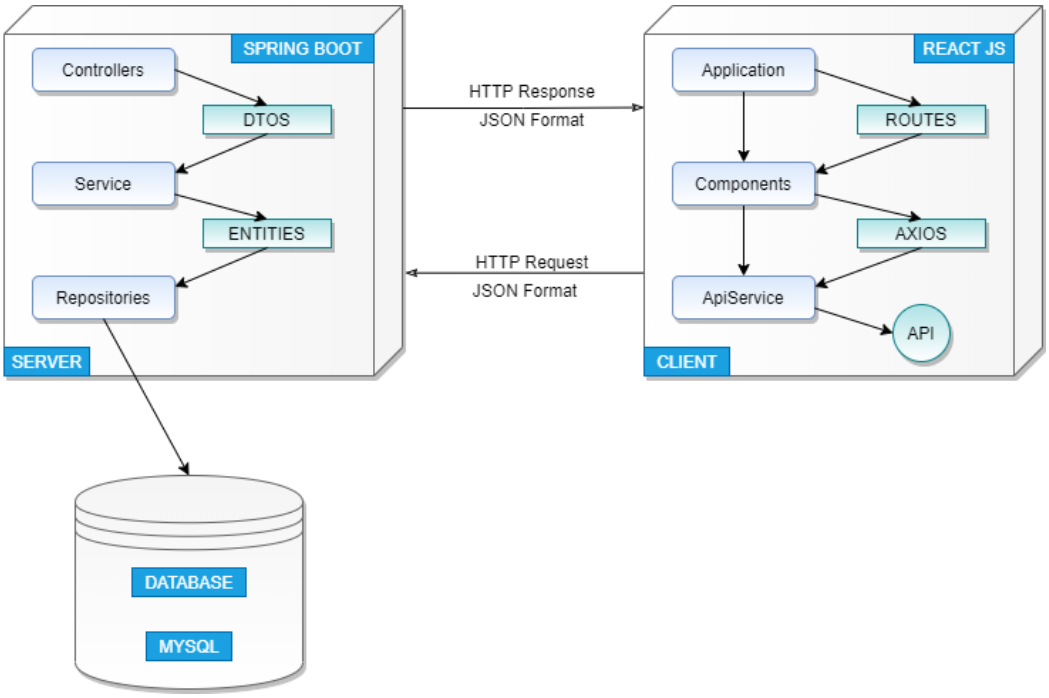

Figure 4.1 Arhitecura sistemului

4.3.1. Arhitectura sistemului de rutare

Cea mai semnificativă parte din acest proiect o reprezintă mecanismul de planificare și rutare a traseului utilizatorului conform activităților definite. Pentru a explica într-o manieră mai descriptivă, figura 4.2 prezintă legăturile dintre componentele arhitecturii sistemului reprezentate în figura anterioară și în plus conturează modulele care comunică cu serverul extern (Google Maps) .

Modulele care comunică direct cu serverul Google Map sunt :

- Location Search : modul care comunică cu serviciul PlacesAPI
- Car Route , Bicycle Route și Walking Route : module care comunică cu serviciul DirectionsService
- Show Map : modul care comunică cu serviciul Geocoding API

Componentele Google Map alături de Waypoints Transport reprezintă date care intră în componența algoritmului de optimizare a rutelor pentru planul curent al utilizatorului . În modulele Car route, Bicycle route, Walking route sunt analizate fiecare

două coordonate succesive din cadrul traseului extras și pentru fiecare sunt calculate costurile în energie , timp și buget . În plus , dacă toate punctele de legătură din traseele extrase au distanțe sub o limită impusă de aplicație , atunci , algoritmul va rezulta printr-o sugestie de recomandare pentru unul din cele trei tipuri de transport și traseul asociat acestuia.

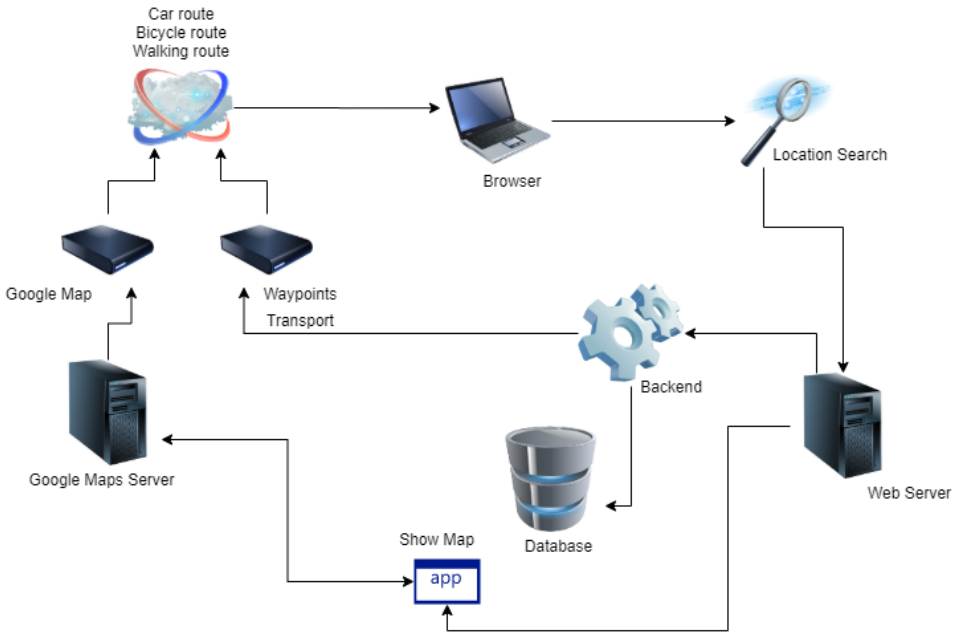

Figure 4.2 Arhitectura sistemului de rutare

4.4. Cazuri de utilizare

Aplicația curentă implică trei tipuri de utilizatori :

- ↳ utilizator neautentificat
- ↳ utilizator autentificat
- ↳ administrator

Cele mai multe funcționalități sunt aplicate asupra utilizatorului autentificat, deoarece acesta este actorul principal, urmat de administrator care trebuie să gestioneze întregul sistem și utilizatorul neautentificat căruia îi este permisă navigarea până în punctul în care dorește să salveze planul de activitate construit. Pentru ca un utilizator neautentificat să devină interesat de aplicație și de funcționalitățile pe care le poate oferi, acestuia îi este permisă navigarea doar până într-un anumit punct, după aceea fiind redirectat către o pagină de autentificare .

Pentru a fi mai ușor de înțeles fluxul de evenimente, fiecare nivel de funcționalitate este identificat printr-o culoare distinctă iar tipurile de acțiuni se diferențiază prin două tipuri de săgeți :

- *include* - reprezintă faptul că o acțiune asupra nivelului anterior va rezulta în apariția unei noi acțiuni la nivelul următor al acesteia .

- *extends* - reprezintă opțiunile pe care le poate avea utilizatorul rezultând din nivelul curent, acestea sunt acțiuni pe care utilizatorul poate alege să le realizeze

Astfel că, pentru a ilustra funcționalitățile sistemului, în raport cu utilizatorii aplicației, figurile următoare înfățișează posibilele interacțiuni și rezultate ca urmare a acțiunilor pe care aceștia le realizează.

4.4.1. Diagrame de utilizare

➢ Utilizator autentificat

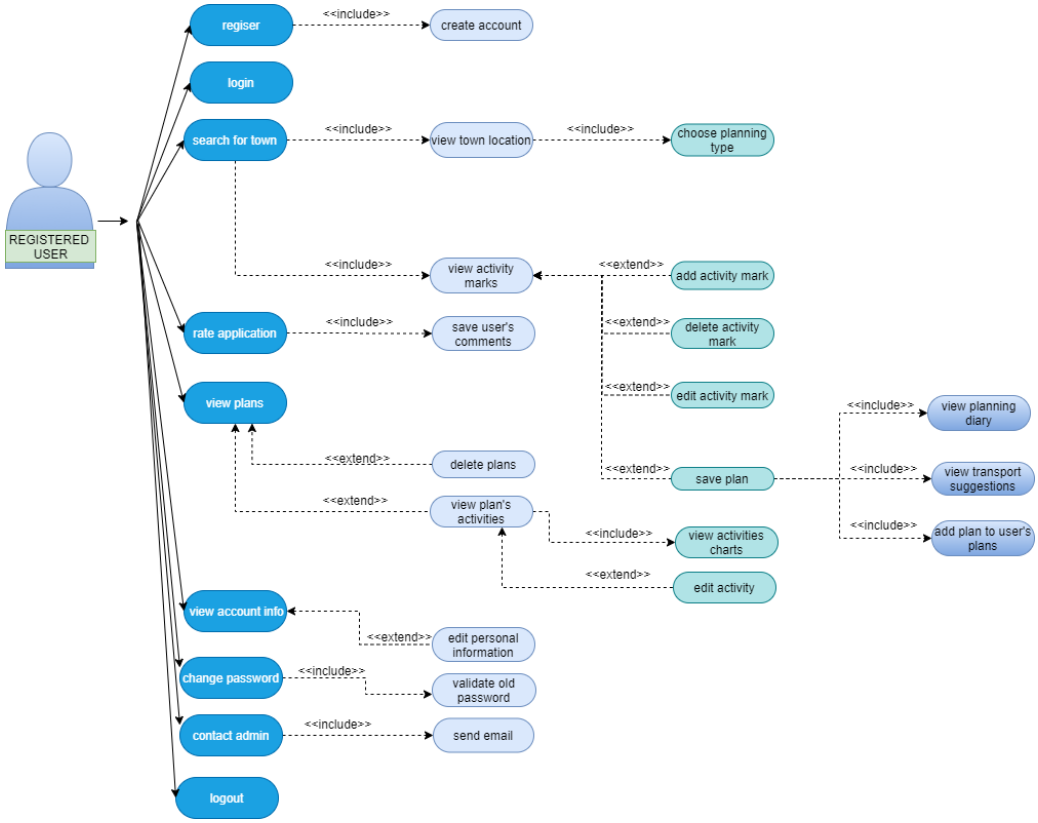

Figure 4.3 Diagrama UML Use Case Utilizator

Utilizatorul autentificat are principalele acțiuni : înregistrare, autentificare, adăugare de recenzii, căutarea unui loc de interes , vizualizarea planurilor create , vizualizarea informațiilor despre profil și cont , schimbare de parola, contactarea administratorului printr-un email și deconectarea de la aplicație . Acțiunile secundare sunt cele care derivă din acțiunile menționate mai sus .

- Utilizator neautentificat (guest)

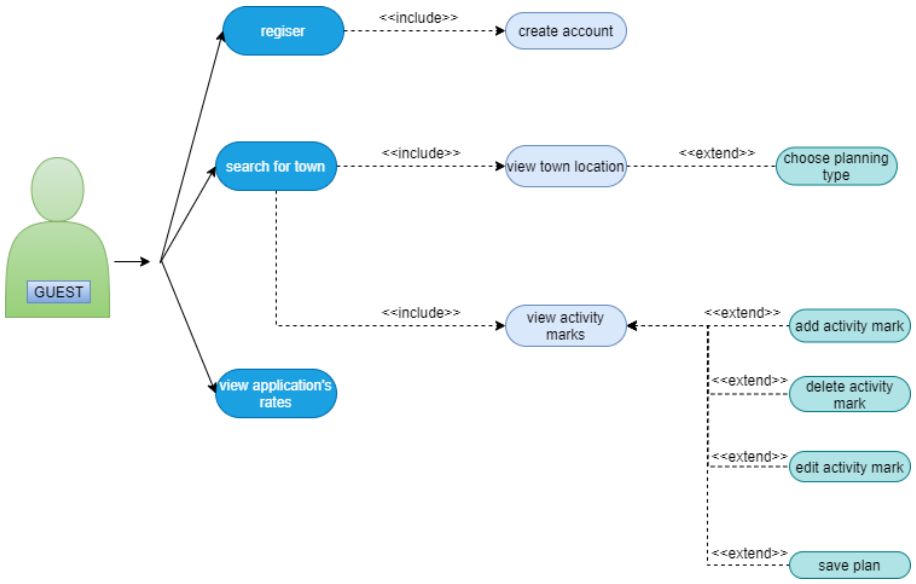

Figure 4.4 Diagrama UML Use Case Guest

Utilizatorului neautentificat i se permite accesul în aplicație însă nu are acces la funcționalitatea principală a aplicației, cea de a salva planurile create ci doar oferă experiența de a planifica. Așadar principalele operațiuni pe care le poate realiza sunt înregistrarea , căutarea unei locații și vizualizarea recenziilor aplicației

- Admin

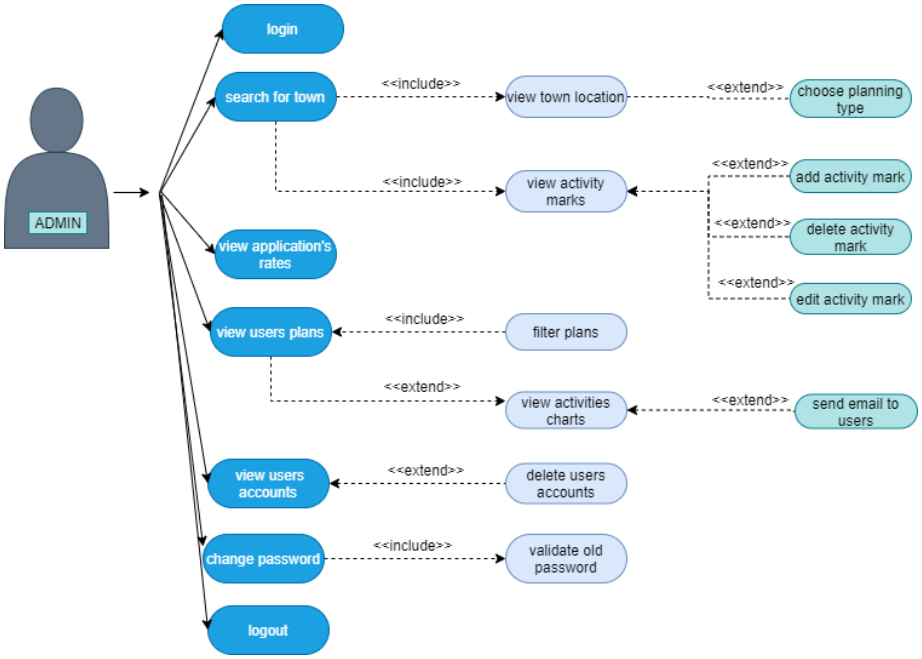

Figure 4.5 Diagrama UML Use Case Admin

Administratorul este persoana care gestionează aplicația, de aceea principalele sale atribuții sunt de a vizualiza și monitoriza activitățile utilizatorilor și de a realiza modificări, dacă sunt necesare .

4.4.2. Descriere cazuri de utilizare

4.4.2.1.Scenariul I

Nume : Înregistrare în aplicație
Actor principal : Utilizator neautorizat
Descriere : Utilizatorii care nu au un cont înregistrat în baza de date a aplicației pot să realizeze doar anumite operațiuni, printre care : căutarea unei locații dorite, vizualizarea locației căutate, adăugarea de pinuri de activitate, stergerea și editatea pinurilor de activitate, vizualizarea notelor utilizatorilor despre aplicație și posibilitatea de înregistrare .

Precondiții:
- Unicitatea email-ului : să nu existe în baza de date a aplicației un cont cu același email cu care se dorește înregistrarea

Postcondiții:
- Contul de utilizator este creat
- Informațiile oferite de utilizator la înregistrare sunt salvate
- Utilizatorul este redirectat către pagina principală

Fluxul de evenimente :
1. Actorul alege optiunea de înregistrare
2. Sistemul afișează un formular de înregistrare prezentând câmpurile de date necesare de completatat : nume, prenume, email , username, parolă, data nașterii
3. Actorul completează toate câmpurile din formular
4. Actorul alege opțiunea de salvare a formularului de înregistrare
5. Sistemul redirectează noul utilizator înregistrat către pagina principală

Fluxuri alternative de evenimente :
- ↺ Completarea formularului de înregistrare
  3. Actorul nu completează unul sau mai multe câmpuri din formular sau introduce valori incompatibile cu tipul de date cerut de unul sau mai multe câmpuri
  4. Sistemul sesizează câmpurile lăsate goale și prezintă mesaje în care se indică faptul că acele câmpuri trebuie să fie completate
  5.1 Actorul completează toate câmpurile rămase libere
  5.2 Actorul abandonează operația

Fluxul de evenimente pentru înregistrarea în aplicație a unui utilizator este prezentat în figura 4.6

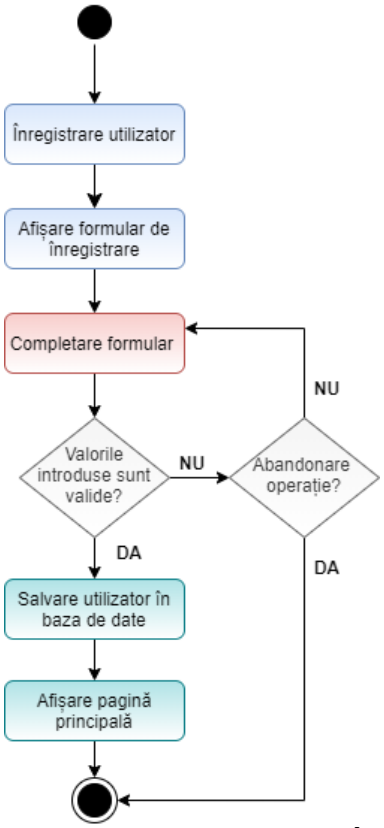

Figure 4.6 Diagrama de evenimente Înregistrare

4.4.2.2. Scenariul II

Nume : Adăugarea unui plan

Actor principal : Utilizator înregistrat

Descriere : Principalul obiectiv al utilizatorului înregistrat este cel de a-și crea propriile planuri săptămânale sau zilnice și să poată manipula activitățile componente ale acestora după bunul său plac . Așadar, pentru a putea crea un plan , un utilizator înregistrat în aplicație trebuie să realizeze anumiți pași .

Precondiții :

- Actorul este autentificat în aplicație
- Actorul se află pe pagina principală

Postcondiții :

- Actorul poate viziona rutele indicate de aplicație dintre punctele setate de acesta în pasul anterior
- Actorul poate viziona indicațiile și un story log al activităților pentru planul creat

Fluxul de evenimente :

1. Actorul caută un oraș în care să-și planifice activitățile
2. Actorul selectează un oraș
3. Sistemul afișează locația selectată , un formular de adăugare de activitate (adăugare pin pe harta) și activitățile curente din aria selectată (pinii de pe hartă)

4. Actorul selectează tipul de planificare
5. Actorul selectează un punct de pe hartă
6. Sistemul identifică coordonatele punctului selectat și le afișează în câmpurile latitudine și logitudine din formulatul de adăugare a activității
7. Actorul completează restul câmpurilor rămase libere cu datele corespunzătoare
8. Actorul alege optiunea de salvare a activității iar in continuare poate repeta pașii anteriori pentru crearea mai multor activități sau poate efectua pașii următori
9. Sistemul semnalează stocarea locală a activităților afișându-le în pagină și reactualizând harta cu pinii locațiilor corespunzătoare activităților
10. Actorul selectează salvarea planului
11. Sistemul redirectează actorul către pagina de vizualizare a rutelor între punctele de activitate selectate pe hartă

Fluxuri alternative de evenimente :

- Căutare oraș
  1. Actorul introduce o locație de căutare greșită sau inexistentă
  2. Sistemul semnalează eroarea prin afișarea unui mesaj corespunzător
- Adăugare activitate

  8. Actorul completează câmpurile din formularul de adăugare unei activități cu valori incompatibile tipurilor de date indicate sau lasă unul sau mai multe câmpuri goale

  9. Sistemul semnalează erorile prin afișarea unor mesaje de ghidare prin care evidențiază user-ului ce a greșit și modul în care trebuie să repete pasul
- Număr minim de activități

  11. Actorul adauga doar o activitate și solicită crearea planului

  12. Sistemul va notifica utilizatorul faptul că este necesară existența a cel puțin două activități pentru crearea unui plan
- Planificare în funcție de timp

  12. Utilizatorul adaugă intervale de timp suprapuse cu alte activități sau intervalele de timp între una sau mai multe activități sunt prea mici pentru a realiza deplasarea dintr-un loc în altul

  13. Sistemul identifică eroarea și generează un mesaj corespunzător pentru a ghida utilizatorul despre modul în care poate remedia eroarea

  14. Utilizatorul poate aborda operația sau poate edita intervalul de timp dintre activitățile care reprezintă o problemă

Fluxul de evenimentul pentru adăugarea unui plan este prezentat în figura 4.7

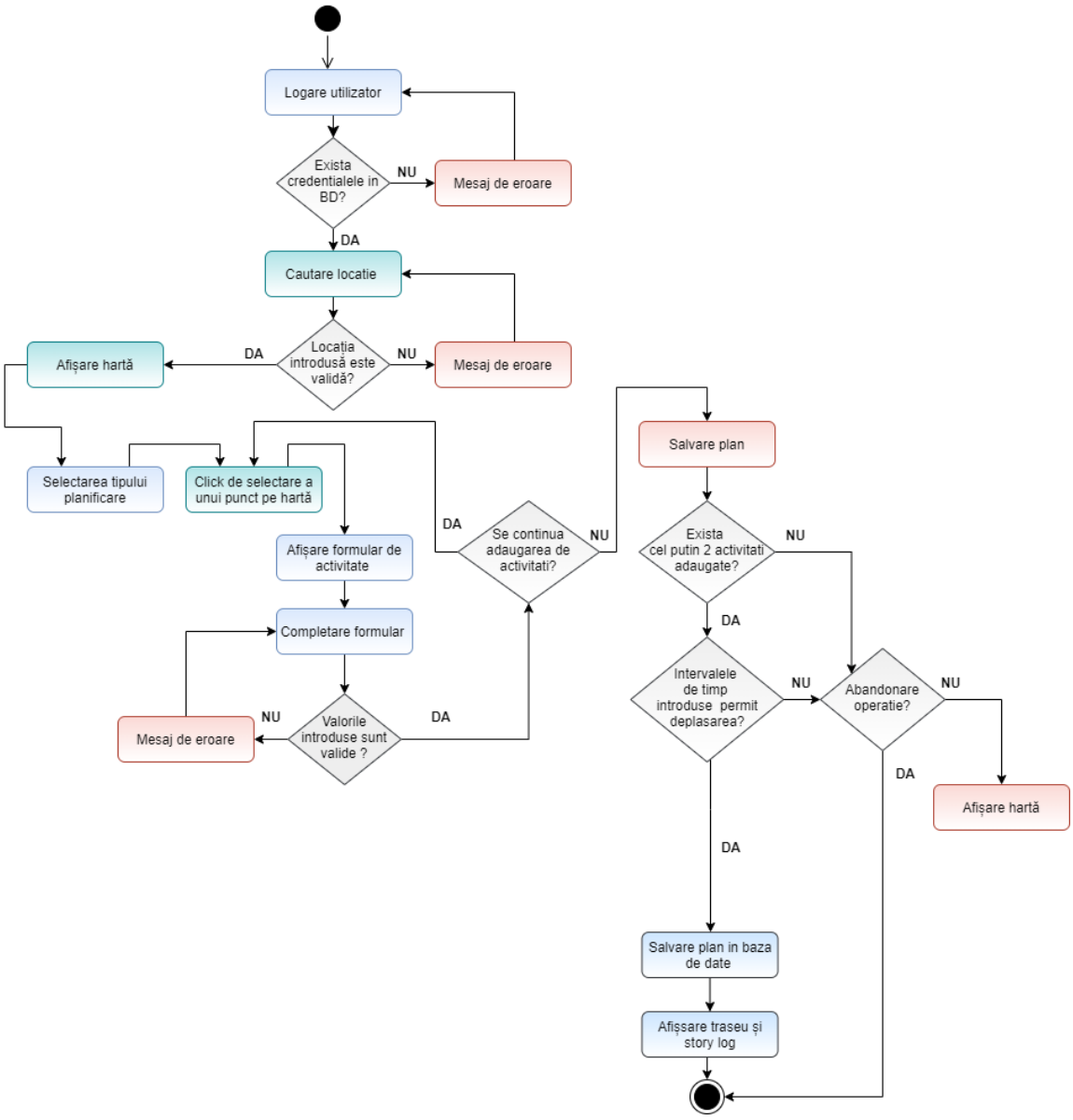

Figure 4.7 Diagrama de evenimente Adaugare Plan

4.4.2.3. Scenariul III

Nume : Trimitere email către utilizatori

Actor principal : Admin

Descriere : Toate conturile înregistrate în aplicație pot conține planurile de activitate ale utilizatorilor. Aceste planuri pot fi vizualizate , filtrate și analizate de către administrator. În acest mod , administratorul poate comunica cu utilizatorii prin transmitere de noutăți despre locurile selectate de aceștia, notificări cu privire la evenimentele ce urmează a avea loc sau diverse oferte .

Precondiții:

- Actorul este autentificat
- Actorul este localizat pe pagina de vizualizare a planurilor utilizatorilor

- Utilizatorii selectați au cel puțin un plan de activitate creat

Postcondiții:

- Sistemul redirectează utilizatorul către o pagină care semnalează succesul sau eșuarea operației de transmitere a mesajului

Fluxul de evenimente:

1. Actorul selectează un plan din cele afișate în baza de date
2. Actorul compune mesajul de transmis prin email utilizatorului a cărui plan îi aparține cel selectat
3. Actorul selectează transmiterea email-ului
4. Sistemul afișează un mesaj corespunzător succesului realizării operațiunii de trimitere email

Flux alternativ de evenimente:

↺ Completare conținut email

3. Actorul selectează transmiterea unui email fără conținut
4. Sistemul detectează eroarea și notifică utilizatorul printr-un mesaj

5.1 Actorul renunță la trimiterea email-ului

5.2 Actorul revine la compunerea mesajului

4.4.2.4. Scenariul IV

Nume : Editare informații de profil

Actor principal : Utilizator înregistrat

Descriere : După crearea contului, utilizatorul are posibilitatea de a-și actualiza informațiile personale : nume, prenume, număr de telefon, adresă , email și încărcare de imagine de profil .

Precondiții:

- Actorul este autentificat

Postcondiții:

- Actorul poate vizualiza profilul contului cu noile informații actualizate

Fluxul de evenimente :

1. Actorul selectează opțiunea de vizualizare a profilului personal
2. Actorul completează câmpurile din formularul de profil cu datele ce dorește a le modifica
3. Actorul selectează opțiunea de salvare a modificărilor făcute
4. Sistemul afișează noile modificări

Flux alternativ de evenimente :

↺ Completare formular de editare profil

2. Actorul lasă câmpuri necompletate sau cu valori diferite de tipul de dată cerut în formularul de editare a profilului
3. Sistemul notifică utilizatorul despre eroarea detectată

4.4.2.5. Scenariul V

Nume : Editarea activităților unui plan

Actor principal : Utilizator înregistrat

Descriere : După crearea unui sau mai multor planuri, utilizatorul poate să își vadă planurile și să realizeze editări la cele curente .

Precondiții :

- Actorul este autentificat

Postcondiții :

- Actorul poate vizualiza planurile actualizate

Fluxul de evenimente :

1. Actorul selectează opțiunea de vizualizare a planurilor sale
2. Sistemul afișează planurile curente și cele trecute
3. Actorul selectează un plan activ
4. Sistemul afișează o pagină cu descrierea planului
5. Actorul selectează opțiunea de editare a planului selectat anterior
6. Sistemul afișează un formular de editare
7. Actorul completează formularul cu noile date
8. Actorul selectează salvarea modificărilor
9. Sistemul afișează descrierea planului actualizat

Flux alternativ de evenimente :

↺ Completare formular de editare plan

7. Actorul completează câmpurile din formularul de editare a planului selectat cu valori incompatibile tipurilor de date indicate sau lasă unul sau mai multe câmpuri goale
8. Sistemul notifică utilizatorul despre eroarea detectată

Fluxul de evenimentul pentru editarea activităților unui plan este prezentat în figura 4.8

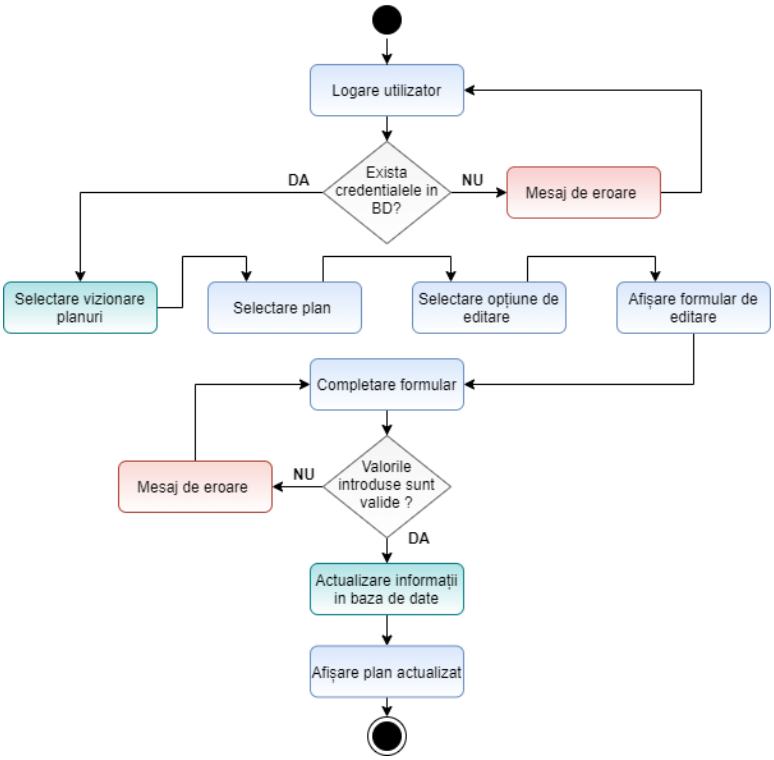

Figure 4.8 Diagrama de evenimente Editare Plan

4.5. Tehnologiile client

4.5.1. React JS

ReactJS[15] este o biblioteca JavaScript utilizată pentru construirea de componente UI reutilizabile. Este întreținută de Facebook și de o comunitate de dezvoltatori și companii individuale.

Principalele concepte, enumerate și explicate în [8] despre React JS sunt:

- JSX - este extensia de sintaxă JavaScript. Asemănător cu aspectul HTML, aceasta oferă o modalitate de structurare a redării componentelor folosind sintaxa familiară multor dezvoltatori.
- Components - codul React este format din entități numite componente ce pot fi redate unui anumit element din DOM folosind biblioteca React DOM.
- Class-based components - componentele bazate pe clase sunt declarate folosind clase ES6. Ele sunt, de asemenea, cunoscute sub numele de componente "stateful", deoarece starea lor poate păstra valori pe întreaga componentă și poate fi transmisă componentelor „copii” prin proprietăți (props)
- Lifecycle methods – sunt cârlige care permit executarea codului la anumite puncte setate din timpul de viață a componentei . Printre cele mai populare , se enumeră :
  - shouldComponentUpdate care permite dezvoltatorului să prevină redarea inutilă a unei componente prin returnarea valorii false dacă nu este necesară o redare.
  - componentDidMount este apelată odată ce componenta a fost „montată” (componenta a fost creată în interfața cu utilizatorul, adesea prin asocierea acesteia cu un nod DOM). Aceasta este utilizată în mod obișnuit pentru a declanșa încărcarea datelor de la o sursă de la distanță prin intermediul unei API.
  - componentWillUnmount este apelat imediat înainte ca descompunerea componentei să aibă loc. Această metodă este utilizată în mod obișnuit pentru a șterge dependențele care necesită resurse de componentă care nu vor fi eliminate pur și simplu cu demontarea componentei
  - render este cea mai importantă metodă de ciclu de viață și singura necesară în orice componentă. Acesta este de obicei apelată la fiecare actualizare a stării componentei, ceea ce ar trebui să fie reflectat în interfața cu utilizatorul.
- Fluxul de date unidirecțional și Flux - React implementează fluxul de date unidirecțional. Flux este un pattern care ajută la păstrarea datelor unidirecționale.

[15] https://en.wikipedia.org/wiki/React_(web_framework)

4.5.2. HTML, CSS, SCSS

HTML [16]este prescurtarea de la Hyper Text Mark-up Language si este codul care sta la baza paginilor web definind sensul și structura conținutului web. Alte tehnologii, în afară de HTML, sunt utilizate în general pentru a descrie aspectul / prezentarea unei pagini web (CSS,SCSS,SASS) sau funcționalitatea / comportamentul (JavaScript).

Avantajele[17] folosirii HTML sunt multiple și enumerate în rândurile următoare :

- Ușor de învățat și utilizat - nu aruncă nicio eroare și nu creează nicio problemă precum alte limbaje de programare. Are etichete care servesc unui scop specific și este ușor de citit chiar și de un utilizator fără experiență în domeniu
- Gratis - nu necesita software specializat sau alte plugin-uri pentru a funcționa, este în totalitate gratis, ceea ce reduce foarte mult din costurile de proiectare ale unei aplicații web
- Susținut de toate Browser-ele
- Se poate integra ușor cu alte limbaje - de exemplu : Javascript, Php, node.js, CSS

CSS[18] (Cascading Style Sheets) descrie modul în care elementele HTML trebuie să fie afișate. Este utilizat pentru a defini stiluri pentru paginile web, incluzând designul, aspectul și variațiile de afișare pentru diferite dispozitive și dimensiuni de ecran. Poate controla design-ul mai multor pagini web printr-un singur fișier extern cu extensia .css .

Cele mai importante avantajele folosirii CSS sunt :

- Economia de timp – permite reutilizarea aceeași foaie CSS în mai multe pagini HTML.
- Întreținere ușoară - Pentru a face o schimbare globală se poate schimba doar stilul iar toate elementele din toate paginile web vor fi actualizate automat.
- Independența platformei - Scriptul oferă o independență constantă a platformei și poate susține și cele mai recente browsere.
- Timp mai rapid de descărcare a paginii web - utilizarea CSS duce la reducerea codului din spatele paginilor web, ceea ce înseamnă timp de descărcare mai rapid. Codul CSS este păstrat în cache-ul din browser după cererea inițială, dacă este separat de codul HTML

SCSS[19] (Syntactically Awesome Style Sheet) este versiunea mai avansată a CSS . Acesta oferă variabile, ce permite diminuarea numărului de linii de cod scrise, în comparație cu CSS convențional.

[16] https://ro.wikipedia.org/wiki/HyperText_Markup_Language
[17] https://www.educba.com/advantages-of-html/
[18] https://www.w3schools.com/css/css_intro.asp
[19] https://www.geeksforgeeks.org/what-is-the-difference-between-css-and-scss/

4.5.3. Node Package Manager

Npm[20] este managerul de pachete pentru JavaScript, utilizat pentru instalare, partajare și cod de distribuire. Npm este scris în întregime în JavaScript și scopul său este de a gestiona dependențe în cadrul proiectelor.

Când este folosit ca manager de dependență pentru un proiect local, acesta instalează toate dependențele necesare prin fișierul package.json. Conținutul package.json trebuie să fie scris în JSON și cel puțin două câmpuri trebuie să fie prezente în fișierul de definiție: nume și versiune. Npm este instalat cu Node.js, ceea ce înseamnă că pentru a obține un npm instalat pe computer este necesară prima dată instalarea Node.js .

Npm este format din trei componente distincte:

- site-ul web - de unde se pot găsi pachete , se pot gestiona și de unde se poate crea un cont personal
- interfața liniei de comandă (CLI) - rulează de la un terminal și este modul în care majoritatea dezvoltatorilor interacționează cu npm
- registrul - este o mare bază de date publică ce conține software JavaScript și meta-informațiile care îl înconjoară

4.5.4. JSON Web Token

JSON Web Token (JWT)[21] este un standard deschis care definește o modalitate compactă și de sine stătătoare pentru a transmite în siguranță informații între părți prin obiecte JSON ce conțin jetoane de acces pentru o aplicație. Jetoanele sunt marcate fie folosind un o cheie secretă privată fie o cheie publică sau privată.

Contextele în care este folosit JWT sunt:

- Autorizare - cel mai frecvent scenariu pentru utilizarea JWT. După ce utilizatorul este conectat, fiecare solicitare ulterioară va include JWT, permițând utilizatorului să acceseze rutele, serviciile și resursele care sunt permise cu acel simbol.
- Schimb de informații – prin jetoanele de acces Web JSON. Acestea sunt o modalitate excelentă de a transmite în siguranță informații între părți deoarece JWT-urile pot fi marcate cu anumite nivele de confidențialitate și criptare. În plus, deoarece semnătura este calculată folosind antetul și payload-ul,se poate verifica ușor dacă conținutul a fost modificat.

Avantajele folosirii JWT[22] :

- Nu necesită gestionarea sesiunii (stateless): JWT este un token autonom care conține informații de autentificare, expiră în timp și poate conține alte revendicări definite de utilizator.
- Portabil: un singur jeton poate fi utilizat cu mai multe aplicații de backend.
- Nu sunt necesare cookie-uri, deci este foarte mobil
- Performanță bună: reduce timpul de întoarcere al rețelei.

[20] https://www.w3schools.com/whatis/whatis_npm.asp
[21] https://jwt.io/introduction/
[22] https://dzone.com/articles/jwtjson-web-tokens-are-better-than-session-cookies

- ✯ Decuplat, descentralizat: jetonul de acces poate fi generat oriunde. Autentificarea se poate întâmpla pe serverul de resurse sau poate fi separată în propriul server.

4.5.5. Axios

Axios este un client HTTP bazat pe serviciul XMLHttpRequests. Este similar cu Fetch API și este utilizat pentru a efectua solicitări HTTP.

Principalele caracteristici[23] ale lui axios (în comparație cu Fetch) sunt :

- ↳ trimite automat cookie-uri înapoi la server atunci când realizează o solicitare
- ↳ are o modalitate încorporată de a aborda request-urile astfel încât dacă răspunsul nu este o reușită, este capturat și gestionat
- ↳ capabil să înregistreze funcții de apelare pentru onUploadProgress și onDownloadProgress pentru a afișa procentul de completare în interfața de utilizare a aplicației

Instalarea pachetului se poate realiza cu ajutorul CLI-ului de la npm, prin comanda npm install axios.

4.6. Tehnologiile server

4.6.1. Spring Boot

Spring Boot[24] este un framework open source bazat pe Java și care oferă o platformă pentru dezvoltarea de aplicații Spring independente și de înaltă calitate.

Spring Boot configurează automat aplicația pe baza dependențelor adăugate în proiect folosind adnotarea @EnableAutoConfiguration. De exemplu, dacă baza de date MySQL se află pe calea de clasă, dar nu este configurată nicio conexiune la baza de date, atunci Spring Boot configurează automat o bază de date în memorie.

Punctul de intrare al aplicației este clasa care conține adnotarea @SpringBootApplication și metoda principală main.Spring Boot scanează automat toate componentele incluse în proiect folosind adnotarea @ComponentScan.

Cele mai importante caracteristici și avantajele oferite de Spring Boot reprezintă și motivele pentru care a fost ales acest framework în construirea aplicației de backend pentru proiectul curent :

- ✯ Oferă un mod flexibil de a configura Java Beans, configurații XML și tranzacții de baze de date.
- ✯ Oferă o procesare și gestionare puternică a endpoint-urilor REST.
- ✯ În Spring Boot, totul este configurat automat; nu sunt necesare configurații manuale.
- ✯ Oferă aplicații Spring bazate pe adnotări

[23] https://www.sitepoint.com/axios-beginner-guide/
[24] https://www.tutorialspoint.com/spring_boot/spring_boot_introduction.htm

- ✯ Usurează managementul dependențelor, de exemplu, folosind dependența „spring-boot-starter-data-jpa”, Spring Boot va atrage toate dependențele „spring-data-jpa”
- ✯ Reduce timpul de dezvoltare și rulare a aplicațiilor
- ✯ Încorporarea serverului Tomcat, ceea ce exclude necesitatea implementării fișierelor WAR

4.6.2. JPA , Hibernate

Java Persistence[25] API (JPA) este o specificație Java care este utilizată pentru a accesa, gestiona și persista date între obiecte Java și baze de date relaționale. Este considerată o abordare standard pentru ORM[26] (Object Relational Mapping).

JPA și poate fi privit ca o punte de legătură între modelele orientate pe obiect și sistemele de baze de date relaționale. Fiind o specificație, JPA nu efectuează nicio operație de unul singur de aceea necesită implementare.

Pe de altă parte, Hibernate[27] este un framework Java care este utilizat pentru a stoca obiectele Java în sistemul de baze de date relaționale. Hibernate este o implementare a JPA. Prin urmare, respectă standardele comune oferite de JPA.

De exemplu, în java , Clasa javax.persistence.Entity definește ce obiecte ar trebui persistate în baza de date, prin adnotarea @entity. Pentru fiecare entitate persistentă, JPA creează un nou tabel în baza de date aleasă. Rolul Hibernate în acest context este să implementeze toate clasele javax.persistence declaratate alături de funcționalitățile lor : căutare, validare, inserare, modificare, alterare, ștergere,etc .

4.6.3. Servicii REST

Serviciile web RESTful sunt, în principiu, servicii web bazate pe arhitectura REST. Pot fi scalabile și ușor de întreținut și sunt foarte frecvent utilizate pentru a crea API-uri pentru aplicații web. Scopul principal al acestui unui serviciu web este de a oferi acces la resurse și de a permite comunicarea dintre client și server .

Protocolul de bază pentru REST este HTTP[28] (Hypertext Transfer Protocol). API-ul REST este stateless, ceea ce înseamnă că fiecare request de la client trebuie sa conțină toată informația necesară pentru a fi înțeleasă de server , din moment ce stocarea stării de sesiune pe server nu este permisă .

Elementele cheie ale unei implementări RESTful sunt următoarele:

- ↳ Resurse - Primul element cheie este resursa în sine.
- ↳ Verbe de request – Acestea descriu acțiunea ce se dorește a se realiza asupra resursei. De exemplu , pentru un browser care emite un request GET , se poate translata faptul că dorește să primească date referitoare la resursa asupra căreia este aplicat request-ul . Pe lângă GET , există mai mult tipuri de acțiuni corespunzătoare verbelor : POST, PUT, DELETE , acestea fiind

[25] https://www.tutorialspoint.com/jpa/
[26] https://en.wikipedia.org/wiki/Object-relational_mapping
[27] https://www.javatpoint.com/jpa-vs-hibernate
[28] https://ro.wikipedia.org/wiki/Hypertext_Transfer_Protocol

cele mai frecvent folosite purtând numele de acțiuni CRUD (Create , Read , Update, Delete) .

- ↳ Anteturi de request-uri - reprezină instrucțiuni suplimentare trimise odată cu solicitarea. Acestea pot defini tipul de răspuns necesar sau detaliile de autorizare.
- ↳ Corpul request-ului - datele sunt trimise odată cu solicitarea. În mod normal, datele sunt trimise în cerere atunci când se face o solicitare POST către serviciul web REST.Într-un apel POST, clientul spune de fapt serviciului web că dorește să adauge o resursă la server, prin urmare, corpul de solicitare trebuie să conțină detaliile resursei care trebuie să fie adăugate pe server.
- ↳ Corpul de răspuns - Acesta este principalul corp al răspunsului. De exemplu , pentru o acțiune GET, serverul este interogat pentru resursa transmisă prin URL și în cazul în care există date, acestea sunt transmise în corpul de răspuns , fie în format XML, JSON, TXT sau orice format specificat în antet
- ↳ Coduri de răspuns - Aceste coduri sunt codurile generale care sunt returnate împreună cu răspunsul de la serverul web. Un exemplu este codul 200 care este returnat în mod normal dacă nu există nicio eroare la returnarea unui răspuns către client.

În proiectul curent acest tip de servicii va fi utilizat pentru comunicarea dintre componenta frontend a aplicației (client) și componenta backend (server), folosind un format recunoscut de ambele părți : JSON.

4.6.4. SMTP

SMTP[29] (Simple Mail Transfer Protocol) este un protocol simplu, folosit pentru transmiterea mesajelor în format electronic pe Internet.

Protocolul SMTP specifică modul în care mesajele de poştă electronică sunt transferate între procese SMTP aflate pe sisteme diferite. Procesul SMTP care transmite un mesaj este numit client SMTP, iar procesul SMTP care primeşte mesajul este numit server SMTP.

SMTP foloseşte următorul model de comunicaţie: transmiţătorul, ca urmare a unei cereri de transmisie a mail-ului, stabileşte o legătură bidirecţională cu receptorul, care poate fi destinatarul final al mail-ului sau doar un intermediar. De aceea este necesar să se precizeze numele de host al destinaţiei finale precum şi utilizatorul căruia îi este destinat mesajul.

4.7. Tehnologii storage

4.7.1. MySQL

MySQL este un sistem relațional de gestionare a bazelor de date relaționale (RDBMS), disponibil gratuit, care folosește limbajul de interogare structurat (SQL).[30]

[29] https://ro.wikipedia.org/wiki/SMTP
[30] https://www.siteground.com/tutorials/php-mysql/mysql/

Cea mai frecventă utilizare pentru MySQL este în scopul unei baze de date web. Poate fi folosit pentru a stoca orice, de la o singură înregistrare de informații la un întreg inventar de produse disponibile pentru un magazin online.

Cele mai importante caracteristici[31] care vin în completarea popularității de care se bucură MySQL sunt următoarele :

- ✯ Arie vastă de compatibilitate – MySQL a fost conceput pentru a fi compatibil pe o varietate de tehnologii și arhitecturi. RDBMS rulează pe toate platformele de calcul majore, inclusiv sisteme de operare bazate pe Unix sau Mac OS și Windows.
- ✯ Bazele de date MySQL sunt relaționale -Factorul principal care diferențiază bazele de date relaționale de alte stocări digitale constă în modul în care datele sunt organizate la un nivel ridicat. Bazele de date precum MySQL conțin înregistrări în mai multe tabele separate, separate și puternic codificate, spre deosebire de un singur depozit integral, sau de colecții de documente semi-sau nestructurate.
- ✯ MySQL este open-source - Orice persoană fizică sau întreprindere poate utiliza în mod liber, modifica, publica și extinde pe baza codului MySQL open-source.

Datorită necesității de conectivitate între tabele prin relații și prin compatibilitatea atât la nivel de sistem cât și din punct de vedere al integrării cu Java Spring Boot, acest model relațional a fost ales ca și componentă de persistență pentru proiectul curent.

4.8. Tehnologii de management

4.8.1. Apache Maven

Apache Maven este un instrument de gestionare și înțelegere a proiectelor software. Bazat pe conceptul de model de obiect de proiect (POM), Maven poate gestiona construirea unui proiect,raportarea și documentarea dintr-o informație centrală.[32]

POM este reprezentat printr-un fișier pom.xml, unde numele proiectului, proprietarul său și dependențele sale sunt specificate. Când este creat un proiect Maven, Maven creează o structură de proiect default .

Cele mai utilizate comenzi din maven sunt maven clear urmată de maven install . Prima golește pachetul de descărcări a dependințelor declarate în fișierul pom.xml iar a doua realizează descărcarea, testarea și construirea lor .

Mai multe IDE-uri (inclusiv IntelliJ IDEA) asigură integrarea Maven, ceea ce înseamnă că Maven este capabil să compileze proiecte din cadrul IDE. Mai mult, suplimentele prin care Maven este integrat în IDE-uri, furnizează capacitatea de a edita POM sau de a folosi POM pentru a determina setul complet de proiecte dependențe, direct în cadrul IDE.

Maven este un instrument adecvat pentru structurarea și gestionarea părții din spate a proiectului curent, deoarece adaugă automat dependențe în proiect, fără a fi nevoie de

[31] https://www.talend.com/resources/what-is-mysql/
[32] https://maven.apache.org/

specificarea acestora manual. Un alt avantaj este ușurința pe care o oferă în partea de management de construire (ciclul de viață) al proiectului.

4.9. Servicii Google Maps

Google Maps oferă pune la dispoziție dezvoltatorilor de aplicații șase servicii pentru lucrul cu hărți digitale : Directions, Distance Matrix, Elevation, Geocoding, Maximum Zoom Imagery și Street View, dintre care următoarele sunt cele utilizate în lucrare :

- **Directions Service**

Serviciul DirectionsService oferă o modalitate de calcul a direcțiilor ce variază în funcție de metodele de transport implicate. Obiectele declarate prin instanțierea acestui serviciu din librăriile Google Maps comunică cu Directions Service, care primește solicitări de direcție și returnează o cale alcătuită din noduri, reprezentând punctele intermediare, distanța și timpul între nodurile succesive și rutele alternative.

Timpul de călătorie este factorul principal care este optimizat, dar pot fi luați în considerare și alți factori precum distanța, numărul de viraje, trafic și multe altele. Rezultatul returnat poate fi deasemeni gestionat, constrâns sau afișat utilizând obiectul DirectionsRenderer.

Directions Service poate returna direcții în mai multe părți folosind o serie de puncte de referință. Instrucțiunile sunt afișate ca o polilinie care desenează ruta pe o hartă sau, în plus, poate pune la dispoziție instrucțiuni printr-o serie de descrieri textuale sau voce.

- **Google Matrix Distance Service**

Serviciul Google Matrix Distance calculează distanța și durata călătoriei între mai multe origini și destinații folosind un anumit mod de călătorie. Acest serviciu nu returnează informații detaliate despre traseu. Informațiile de rută, inclusiv polilinele și indicațiile textuale, pot fi obținute trecând originea și destinația dorite către obiectul Direction Service.

Modul Travel (transport) permite utilizatorului să selecteze modul de transport între oricare destinații și să calculeze direcțiile pentru fiecare tip de transport identificat.

Cele patru moduri de transport puse la dispoziție utilizatorului de către API-ul Google Maps sunt următoarele: google.maps.TravelMode

- .TRANSIT – transport în comun
- .DRIVING – transport utilizând rețeaua rutieră
- .BICYCLING – transport cu bicicleta
- .WALKING – transport în mers prin căi pietonale

- **Geocoding Service**

Un alt serviciu esențial pentru dezvoltarea unei aplicații web dinamice este serviciul de geocodare care funcționează prin apelarea unui server extern.

Geocodarea este procesul de transformare a adreselor în coordonate geografice care pot fi folosite pentru a plasa markeri, a poziționa hărți sau pentru a memora locații. De exemplu, adresa “1600 Amphitheater Parkway, Mountain View, CA” supusă procesului de geocodare va rezulta în doua coordonate, mai exact, latitudine 37.423021 și longitudine -122.083739 .

Capitolul 5. Proiectare de Detaliu si Implementare

Acest capitol prezintă și explică pas cu pas deciziile luate din momentul proiectării până în momentul implementării soluției alese .

Sistemul ce urmează a fi detaliat , menționat și în secțiunea 4.1 este compus din trei subsisteme : client (frontend) , server (backend) și baza de date . Fiecare dintre acestea joacă un rol esențial în reprezentarea finală a aplicației astfel că în următoarele secțiuni vor fi analizate și discutate aspecte reprezentative ale acestora.

5.1. Arhitectura componentei backend

5.1.1. Descriere generală

În cadrul dezvoltării aplicației de backend, tehnologia utilizata face parte din gama de module Spring, descrisă în secțiunea 4.6.1, iar în ceea ce privește modelul arhitectural a fost implementat urmărind structura conceptuală descrisă în Figura 4.1 .

Drept urmare, în implementarea conceptului descris, s-au utilizat ca șabloane arhitecturale și de implementare paradigma arhitecturii pe nivele (Three Layers), delimitând nivelurile de prezentare (HTTP endpoints) de cele de business logic și de acces la date.

→ **Architectural pattern**

O arhitectură software este fundamentul unui sistem software. Designul arhitectural este semnificativ pentru calitatea și succesul pe termen lung al software-ului . Ținând cont de aceste aspecte, componenta de backend este structurată folosind arhitectura pe trei nivele (Three Layer) : Presentation Layer, Business Logic Layer și Data Access Layer sau Persistence Layer . În figura 5.1 este prezentat modelul arhitectural și modul în care fiecare nivel comunică .

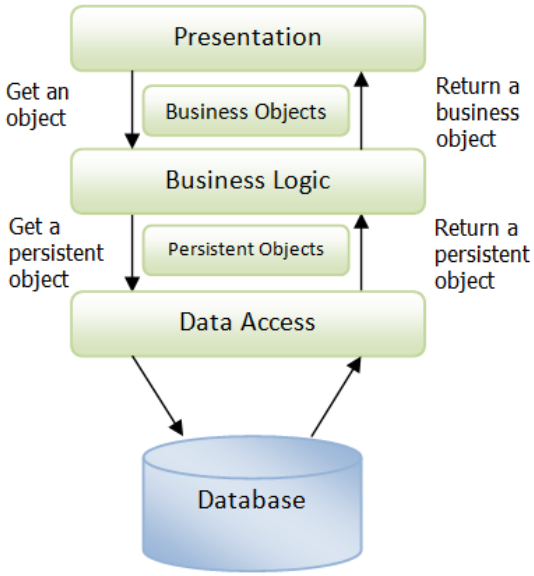

Figure 5.1 Architectural pattern Three Layer pentru backend

Acest tip de arhitectură separă componentele din backend în funcție de responsabilități și acțiunile generale pe care trebuie să le realizeze, astfel că :

- Primul strat, Presentation este cel care în proiect poartă denumirea pachetului Controller și este specializat cu prezentarea datelor către User Interface. În clasele definite aici sunt definite request-urile la care aplicația de frontend face apel . În definitoriu, acest nivel este nivelul care reprezintă componenta de backend .
- Al doilea strat, Bussiness Logic reprezintă logica aplicației, aici sunt implementați algoritmi, sunt filtrate datele și se realizează legătura către următorul strat. Cele mai reprezentative clase pentru acest nivel sunt cele din pachetul Service.
- Ultimul strat, Data Access este specializat cu apelul către serverul bazei de date și extragerea datelor, pe baza specificațiilor aduse din straturile anterioare. În acest caz, pachetul Repository este cel a cărui clase sunt responsabile de acțiunile menționate.

Principalul motiv în alegerea acestui model arhitectural îl reprezintă mecanismul de management și întreținere . De exemplu, păstrarea codului de prezentare separat față de cel al logicii aplicației face ca o modificare în stratul logic sa nu afecteze stratul de prezentare . Prin acest mod, erorile sunt depistate la baza nivelului iar structura aplicației este mai ușor de înțeles și de citit de către alți programatori.

5.1.2. Descrierea componentelor

Din cele descrise în secțiunea 4.6, despre tehnologiile server, se poate deduce faptul că serviciile REST reprezintă o componentă specializată ce necesită separare logică, aceeași idee fiind valabilă și pentru JPA și Hibernate . Pornind de la aceste separări de conținut , se poate observa în figura 5.1.2 un exemplu reprezentativ pentru structurarea pe pachete . Pachetele sunt cele existente în proiect însă clasele din ele au rol exemplificativ, astfel că :

- ✓ Controllers – conține clasele de specializare ale serviciilor REST
- ✓ DTO – conține clasele ce încapsulează reprezentarea obiectelor JSON returnate sau accesate prin serviciile REST
- ✓ Service – reprezintă componenta de procesare, numită și componenta bussiness logic, aici se află interfețele care expun metode specializate cu prelucrarea, procesarea și transmiterea datelor
- ✓ Service.Implementation – conține clasele care vor implementa metodele din interfețele definite în pachetul Service
- ✓ DAO – este pachetul care conține clasele cu ajutorul cărora se creează entități de date interceptate de JPA și Hibernate și mai apoi sunt translatate în SQL. În cadrul acestor clase sunt definite modelele de tabele și relațiile dintre ele folosind adnotări Spring
- ✓ Repository – conține interfețele care implementează serviciile oferite de JPA pentru modelul de date reprezentat prin clasele pachetului DAO

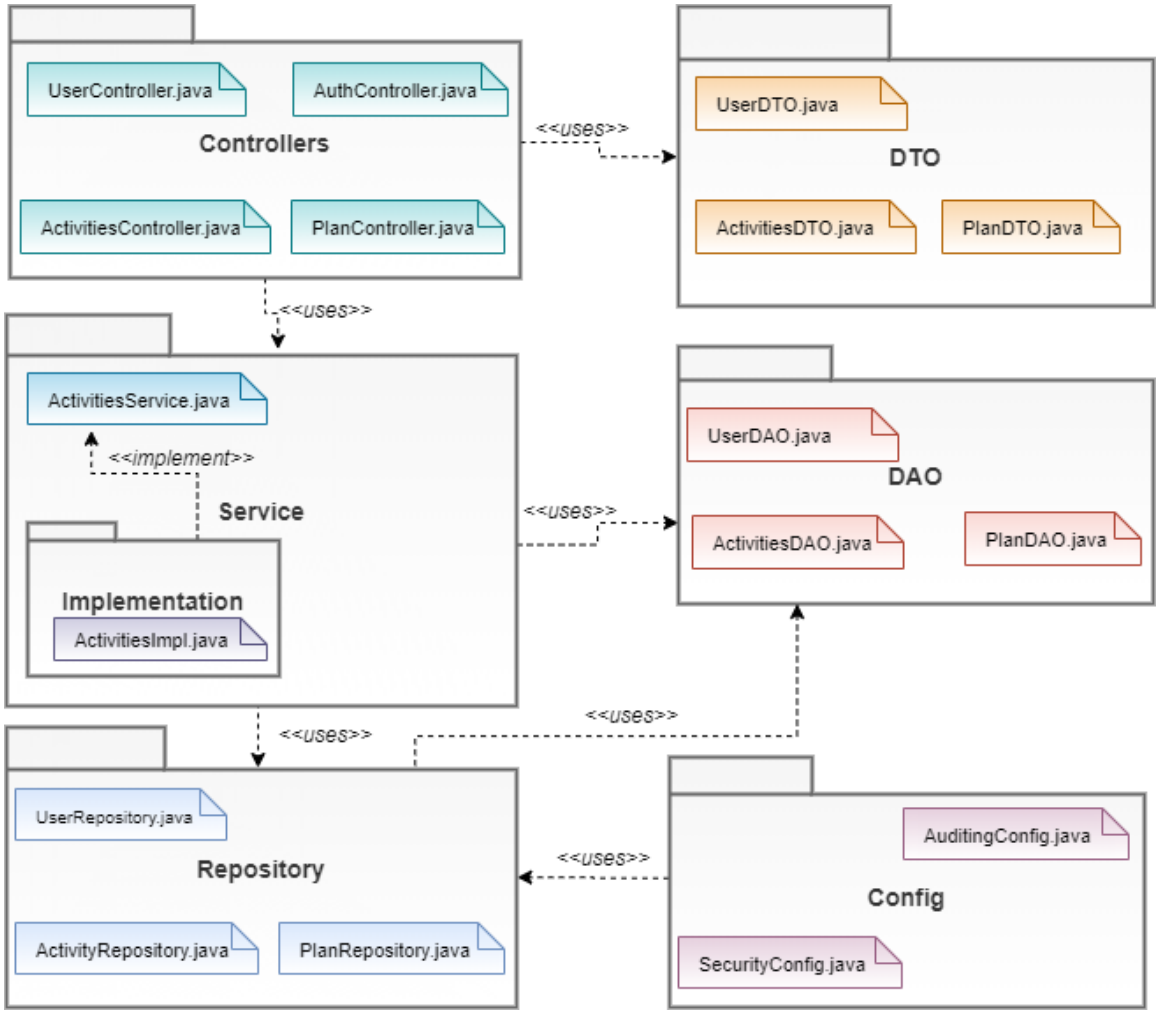

Figure 5.2 Diagrama de pachete a componentei backend

5.2. Arhitectura componentei frontend

5.2.1. Descriere generală

Framework-ul folosit pentru construirea părții de client a aplicației este ReactJS . Pentru mai multa claritate, ușurintă în citirea codului dar și segmentare logică a componentelor interne, partea de client are la bază ca și pattern arhitectural MVC. Alături de ReactJS, înfățișarea este responsabilitatea HTML, CSS, SCSS iar JavaScript creează și partajează datele, menținând dinamica aplicației.

→ **Architectural pattern**

Pattern-ul arhitectural MVC este compus din M (Model) unde sunt reprezentate datele , de cele mai multe ori câmpurile corespunzătoare atributelor din clasele subsistemului server , V (View) conține componente HTML , care ,prin ajutorul metodelor reprezentative din JavaScript , sunt constant reîmprospătate pentru a fi posibila reprezentarea dinamică a datelor , iar C (Controller) care constituie logica de trecere de la o pagină la alta, practic , navigarea în aplicație .

În figura alăturată se poate observa design-ul arhitectural menționat anterior, ce conține modul de reprezentare a fluxului de date și legăturile dintre acestea .

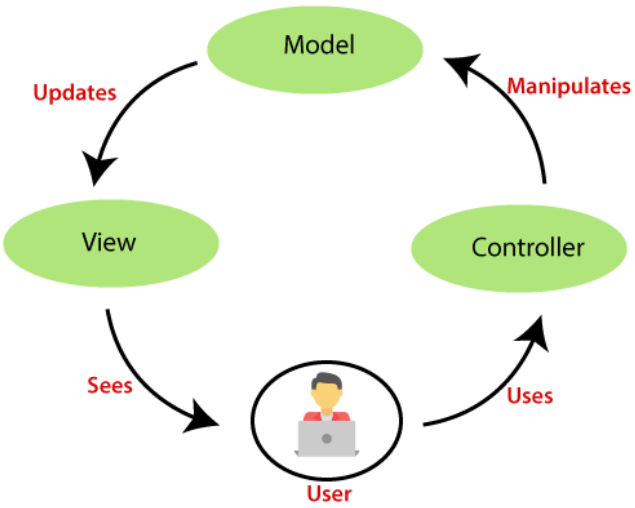

Figure 5.3 Design pattern MVC pentru frontend

Prin definiția oferită de MVC , alegerea acestui pattern a fost mai mult decat necesară datorită separării evidente a elementelor ce construiesc această parte a aplicației. Separarea aceasta nu este doar pentru modularizare cât și pentru împărțirea în grupuri de responsabilități . În acest mod excepțiile sau erorile rămân intr-un anumit nivel, cel care le-a generat, iar detecția lor este mai rapidă.
Un alt beneficiu îl reprezintă standardul pe care îl impune ce face aplicația mai ușor de înțeles și de citit.

5.2.2. Descrierea componentelor

Figura alăturată , 5.4 , arată modul în care componentele din frontend comunică între ele astfel încât împreună să gestioneze comenzile și să ofere utilizatorului o experiență de înaltă calitate.

Se poate observa influența pe care o are componenta State asupra componentei View . Orice modificare în View este sesizabilă și stocată în stările (proprietățile) asociate din vedere iar orice modificare in stare este observabilă și în vedere .

Prin vedere (View) utilizatorul poate realiza acțiuni. Aceste acțiuni pot avea drept consecință un request la subsistemul de backend prin metode de API la care se face apel sau acțiuni din interior reprezentate de apel a unor metode cu scopul de a manipula sau transmite datele mai departe printre componentele frontend .

Ca și consecință a acțiunilor utilizatorului acesta poate rămâne pe pagina de unde a invocat acțiunile sau , în funcție de tipul de acțiune realizată, poate evolua către o altă pagină . Această evoluție este monitorizată de Controller , prin rute, de unde se reîncepe un alt ciclu al experienței utilizator.

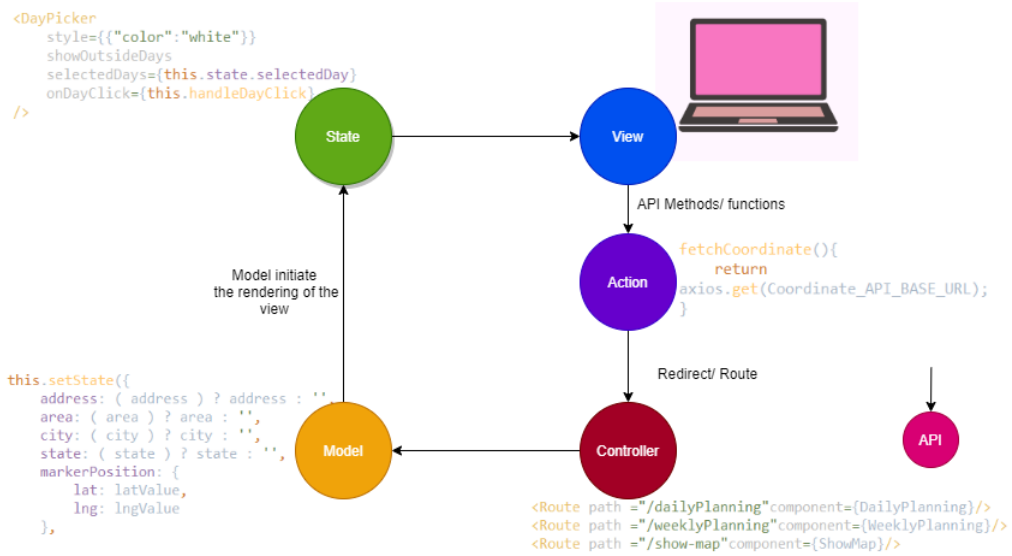

Figure 5.4 Arhitectura detaliată a componentelor din frontend

Pentru a menține o evidență și o separare logică a elementelor componente a fost necesară organizarea acestora în pachete . Un exemplu reprezentativ de organizare este prezentat in figura de mai jos .

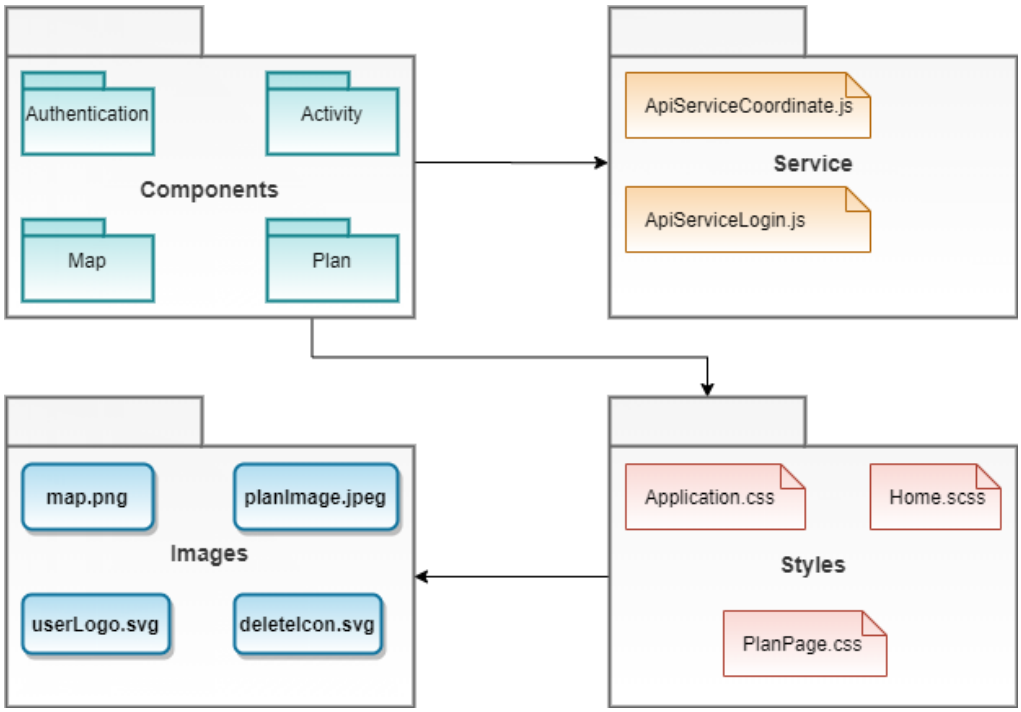

Figure 5.5 Diagrama de pachete pentru componenta frontend

Din cele descrise în capitolul 4, despre React JS, se poate deduce faptul că structura centrală o reprezintă Componenta, iar acest lucru se poate observa și în figura de mai sus .
În pachetul Components se află mai multe pachete de componente, fiecare specializate pe activitățile pe care trebuie să le realizeze. De exemplu, în subdirectorul Activity există patru fișiere de tip jsx cu denumirea „EditActivityComponent", „DeleteActivityComponent", „ViewActivityComponent", "SaveActivityComponent". Întregul subdirector conține componente pentru realizarea acțiunilor CRUD. Acestea vor apela clasa serviciului ApiServiceActivity.js din directorul Service, iar pentru fiecare funcționalitate se va apela funcția corespunzătoare care va trimite un request la backend . Pe lângă posibilitatea de a crea request-uri, componentele pot integra conținut de reprezentare grafică de tip HTML care comunică cu pachetul Styles pentru manipularea structurilor, iar fiecare fișier de tip css sau scss poate conține imagini ce sunt accesibile din pachetul Images .

5.2.2.1. Controller și Route

Acestă componentă este specializată în managementul direcției fluxului de date din aplicație . Pentru a ajunge dintr-o pagină în alta este necesar mai întâi declararea acestora prin Route , așa cum se poate observa în figura 5.6

```
<Route path ="/dailyPlanning"component={DailyPlanning}/>
<Route path ="/weeklyPlanning"component={WeeklyPlanning}/>
<Route path ="/show-map"component={ShowMap}/>
```

Figure 5.6 Definirea rutelor prin Route în aplicația frontend

Odată definire rutele, pentru a putea naviga dintr-o pagină în alta este necesar doar un "push" din pagina curentă către calea ("path") definite, prin redirectarea către o cale sau prin importarea componentelor în variabile și utilizarea lor ca și elemente grafice .

→ push

```
dailyPlanningChoice = (e) => {
    e.preventDefault();
    this.props.history.push('/dailyPlanning');
}
```

Figure 5.7 Navigare prin push în aplicația frontend

→ redirect

```
renderRedirect = () => {
    if (this.state.redirect) {
        return <Redirect to='/direction' />
    }
}
```

Figure 5.8 Navigare prin redirect în aplicația frontend

→ import

```
import SearchBar from "./SearchBar";
class SelectedLocation extends Component {

    render() {
        return(
            <div style={{ margin: '100px' }}>
                <SearchBar/>
            </div>
        );
    }
}
export default SelectedLocation;
```

Figure 5.9 Navigare prin import în aplicația de frontend

Fiecare mod de navigare dintr-o pagină în altă are facilitățile și atuurile sale.

De exemplu , Redirect este folosit în predominanță când un utilizator neautorizat dorește să realizeze o acțiune neautorizată, fiind automat redirectat la o pagină de login sau când este nevoie de o distincție între utilizatori, astfel că, în funcție de tipul de utilizatori, aceștia sunt redirectați către pagina corespunzătoare rolului .

Push este o metodă de navigare în pagină care memorează întreaga cale de navigare a utilizatorului, permițându-i să navigheze cu ușurință înapoi în paginile prin care a trecut .

Import se folosește de cele mai multe ori atunci când componentă importată este o funcție sau o constantă care returnează conținut html .

Figura 5.10 prezintă modul în care se poate naviga în aplicație în pentru un utilizator neautorizat iar Figura 5.11 , pentru un utilizator autorizat.

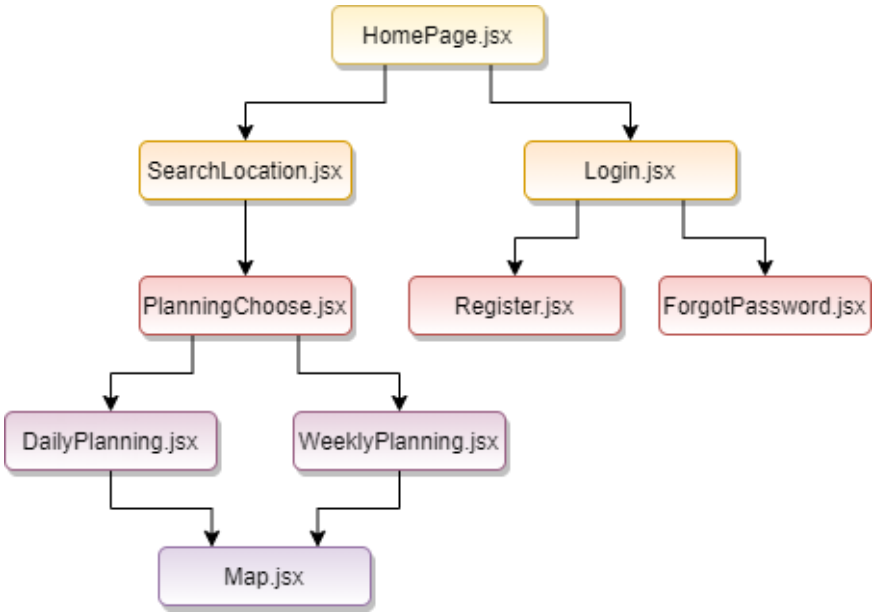

Figure 5.10 Diagrama de navigare a utilizatorului neautorizat

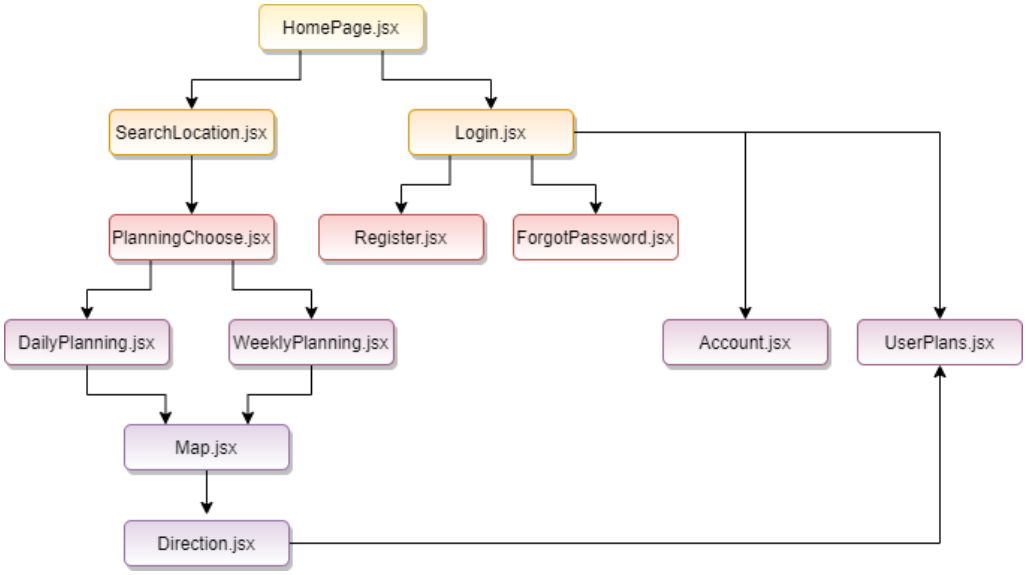

Figure 5.11 Diagrama de navigare a utilizatorului autorizat

5.2.2.2. Services

Serviciile din partea de frontend au rolul de a emite și a intercepta solicitări http de la backend . Așa cum se poate observa din figura 5.3 de mai sus , din diagrama de pachete, componentele comunică cu partea de servicii .

Modul în care se realizează aceste funcționalități este gestionat printr-o librărie pusă la dispoziție în react , anume , Axios . Principalele caracteristici ale acestei librării sunt interceptarea de date, transformarea automata a acestora în format JSON, anulare de request-uri și suport pentru Promise API. Aceasta din urmă permite realizarea unor acțiuni asincrone, asigurându-le un comportament sincron până când este îndeplinită “promisiunea” îndeplinirii acțiunii în curs. De exemplu :

```
ApiServiceCoordinate.addCoordinate(coordinate)
    .then(res => {
        this.setState({message : 'Coordinate added successfully.'});
        this.props.history.push('/planning');
    });
```

Figure 5.12 Apel funcție de request în frontend

În acest caz, se dorește realizarea acțiunii de salvare a coordonatelor unei activități. Pentru a realiza acest lucru se apelează metoda din serviciul de coordonate și se promite că, în cazul în care operațiunea s-a realizat cu succes, se va seta starea message cu un mesaj corespunzător și se va trece la o altă pagină de navigare . Ținând cont că realizarea apelului la backend implică un anumit timp, această acțiune este considerată asincronă, și în acest timp se trece mai departe în cod, revenindu-se la această metodă în momentul în care backend-ul recepționează și transmite un răspuns.
Clasa de serviciu prin care se face apelul este următoarea :

```
import axios from 'axios';
const Coordinate_API_BASE_URL = 'http://localhost:8091/coordinate';
class ApiServiceCoordinate {
    addCoordinate(coordinate){
        return axios.post(""+Coordinate_API_BASE_URL, coordinate);
    }
```

Figure 5.13 Request folosind axios în frontend

Se realizează requestul de post prin pasarea configurațiilor specifice metodei instanță corespunzătoare, în acest caz , metoda post .

5.2.2.3. Integrarea API-urilor de la Google

Nucleul central al acestei aplicații îl reprezintă serviciul de hărți oferit de Google. Pentru manipularea acestora este necesară existența următoarelor API-uri de servicii web :

→ **Directions API** : un serviciu care calculează direcțiile între locații. Pune la dispoziție căutarea indicațiilor pentru mai multe moduri de transport, inclusiv tranzit, conducere, mers pe jos sau mers cu bicicleta. În exemplul de mai jos este definit un nou obiect de tip DirectionsService și se apelează metoda instanță route pasând parametrii cu punctul de început (originea) , punctele intermediare (vector de coordonate) , punctul final (destinația) și tipul de transport , care , în acest caz este DRIVING, adică mers cu mașina .

```
const DirectionsService = new window.google.maps.DirectionsService();
DirectionsService.route({
    origin: new window.google.maps.LatLng(waypoints[0].coords.lat,
waypoints[0].coords.lng),
    destination : waypoints[waypoints.length-1].coords,
    waypoints: waypts,
    optimizeWaypoints: true,
    travelMode: window.google.maps.TravelMode.DRIVING,
```

Figure 5.14 Apel serviciu DirectionsService în frontend

→ **Geocoding API** : un serviciu care oferă geocodare și geocodare inversă a adreselor. Este utilizat pentru crearea de activități, prin selecția click pe hartă și returnarea coordonatelor în format latitudine, longitudine dar și invers, pentru serviciul Directions, din coordonate sunt transformate în adrese .

- Geocodare inversă

```
import PlacesAutocomplete, {
    geocodeByAddress,
    getLatLng
} from "react-places-autocomplete";

const handleSelect = async value => {
    const results = await geocodeByAddress(value);
    const latLng = await getLatLng(results[0]);
    setAddress(value);
    setCoordinates(latLng);
```

Figure 5.15 Exemplu de geocodare inversă în aplicația de frontend

- Geocodare directă

```
import Geocode from "react-geocode";

Geocode.fromLatLng( this.state.mapPosition.lat ,
this.state.mapPosition.lng ).then(
    response => {
        const address = response.results[0].formatted_address,
            addressArray =  response.results[0].address_components,
            city = this.getCity( addressArray ),
```

Figure 5.16 Exemplu de geocodare directă în aplicația de frontend

→ **Places API** : un serviciu care returnează informații despre locuri folosind solicitări HTTP. Locurile sunt definite în cadrul acestui API ca unități, locații geografice sau puncte de interes proeminente.În cadrul acestui proiect request-ul care se apelează este cel de Place Autocomplete care completează automat numele și / sau adresa unui loc.

```
<PlacesAutocomplete
    value={address}
    onChange={setAddress}
    onSelect={handleSelect}
>
```

Figure 5.17 Exemplu de utilizare places API în aplicația de frontend

5.3. Structura bazei de date

Pentru a stoca datele aplicației, se folosește baza de date MySQL, deoarece este cea mai populară bază de date open source din lume și prezintă câteva avantaje subliniate în capitolul anterior, secțiunea 4.7.1 . Baza de date este generată automat din aplicația de backend prin adnotări ce indică entități și prin relațiile declarate în aceste clase. Datele de identificare a bazei de date sunt descrise în fișierul de resurse, astfel că identificarea și crearea tabelelor are loc cu ușurință.

În figura alăturată este prezentată baza de date a proiectului ce conține un număr de șapte tabele și relațiile dintre acestea: one-to-one, one-to-many , many-to-one sau many-to-many .

Baza de date este normalizată și respectă forma normală Boyce-Codd care susține faptul că o relație este în forma normală BC dacă orice determinant din relație este cheie candidat. Această formă asigură că în baza de date nu există relații de dependență parțială și tranzitivă.

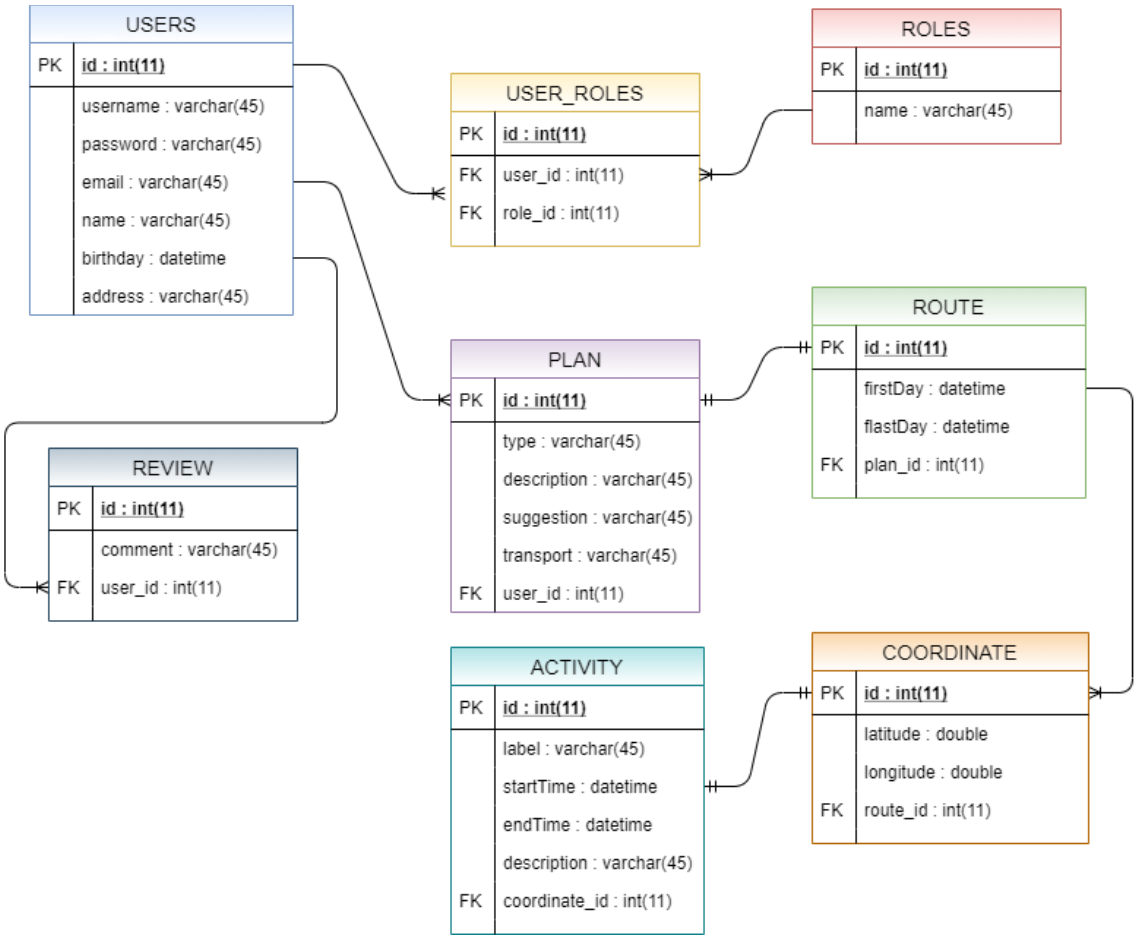

Figure 5.18 Diagrama bazei de date

5.3.1. Descrierea tabelelor

Baza de date a proiectului se numește “PlanningDataBase” și conține opt tabele asupra cărora se vor realiza diferite operații : interogare, extragere, adăugare, modificare, stergere de conținut .

Tabelele Users, User_Roles și Roles reprezintă partea de autentificare a aplicației . Între tabela Users și Roles există o relației many-to-many, ceea de a dus la crearea tabelei User_Roles . În acest mod utilizatorii cu roluri diferite sunt mai ușor de identificat și în același timp oferă o flexibilitate în ceea ce privește adăugarea de roluri multiple pentru un singur utilizator . Deși aplicația este alcătuită din două tipuri de utilizatori, implementarea acestei abordări a fost o decizie luată deoarece s-a ținut cont de dezvoltările ulterioare ale aplicației și ușurința în gestiunea și managementul utilizatorilor.

Tabela User este alcătuită din următoarele câmpuri :

- → id : cheie primară, de tip int, prin care se realizează identificarea tabelei
- → username : identificator unic,de tip varchar, care face parte din credențialele de autentificare
- → password : parola user-ului, de tip varchar, este salvată criptată în baza de date, pentru un nivel crescut de securitate

- → email : identificator unic, de tip varchar, alături de username, face pare din credențialele de autentificare
- → name : câmp de tip varchar pentru stocarea numelui user-ului
- → birthday : câmp de tip datetime folosit pentru stocarea datei de naștere
- → address : stochează un tip de dată varchar cu adresa user-ului

Tabela User_Roles conține identificatorul tabelei, id, de tip int, care este cheia primară și modul unic de identificare al tabelei plus alte două câmpuri ce reprezintă cheile străine ale tabelelor users și roles : user_id, respectiv, role_id.
Tabela Roles conține cheia primară, de tip int și un câmp pentru numele rolului ce urmează a fi asignat user-ului.

Tabela Plan este tabela centrală a aplicației datorită faptului că realizează conexiunea utilizatorului cu activitățile acestuia. Conține legături către două tabele : Users și Route. Între tabela Plan și Users există o relație many-to-one iar între Plan și Route există o relație one-to-one . Ca și elemente componente, tabela este alcătuită din :

- → id : cheie primară de tip int
- → type : câmp de tip varchar, stochează tipul de plan, care poate fi : săptămânal (weekly) sau zilnic (daily)
- → description : conține descrierea rutelor planului prin indicații folosind tipul de date varchar
- → suggestion : câmp de tip varchar, reprezintă sugestia oferită de aplicație pentru planul utilizatorului
- → transport : stochează un tip de dată varchar cu tipul de transport selectat de user (car, walking, bicycle)
- → user_id : cheia străină de tip int, reprezintă identificarea user-ului

Tabela Route reprezintă descrierea traseului dintre mai multe coordonate iar ca și structură prezintă :

- → id : cheie primară de identificare a tabelei, de tip int
- → firstDay : câmp de tip Date ce reprezintă prima zi din planificarea activităților
- → lastDay : câmp de tip Date ce reprezintă ultima zi din planificarea activităților
- → plan_id : cheie străină , de tip int prin care se realizează identificarea planului asociat

Tabela Coordinate stochează coordonate (latitudine și longitudine), cărora le este asignată o activitate, fapt reprezentat din relația one-to-one cu tabela Activities . Pe de altă parte, este si elementul de construcție a unei rute, însă , datorită faptului că o rută necesită minim două coordonate, relația dintre tabela Coordinate și Route este de tip many-to-one .

Tabela Activity conține elementele descriptive ale unei activități, acestea fiind :

- → id : identificatorul tabelei , cheie primară de tip int
- → label : câmp de tip varchar ce reprezintă o descriere scurtă, de dimensiunea unui titlu
- → startTime : data și ora pentru începutul unei activități , de tip datetime
- → endTime : data și ora pentru încheierea unei activități, de tip datetime

→ description : descriere, de tip varchar , într-un număr mai mare de cuvinte comparativ cu label-ul, ce caracterizează activitatea desfășurată
→ coordinate_id : cheie primară, de tip int , ce identifică coordonata căreia îi este asociată activitatea

Tabela Review stochează comentariile despre aplicație a utilizatorilor și conține cheia străină prin care este identificată apartenența la user-ul care a adăugat comentariul și textul (comentariul) adăugat.

5.4. Mecanismul de optimizare activități

Tema centrală a proiectului o reprezintă modalitatea de optimizare a activităților pe bază de rute, de aceea a fost necesară proiectarea și implementarea unui algoritm care să satisfacă cerințele mai multor tipuri de utilizatori .

Cel mai important aspect în determinarea unui mecanism de optimizare în cazul unor trasee dintre diferite coordonate îl reprezintă mijlocul de transport, în cazul acestui proiect :

→ Walking (mers pe jos)
→ Bicycling (mers cu bicicleta)
→ Car (mers cu autoturismul)

Fiecare tip de transport selectat va rezulta într-un traseu cu distanțe, timpi, consum de energie(calorii) și consum de bani .

5.4.1. Notații și valori de referință

Următorul tabel este reprezentativ pentru determinarea avantajelor și dezavantajelor fiecărui tip de transport :

	Time	Distance	Energy Cost	Budget Cost
Walking	Wt	Wd	Wec = EFW x Wt	Wbc = 0
Bicycle	Bt	Bd	Bec = EFB x Bt	Bbc = 0
Car	Ct	Cd	Cec = 0	Cbc = BFC x Cd

Table 5.1 Flaguri pentru tipurile de transport

Unitățile de măsură pentru determinarea variabilelor enumerate sunt:

- timp - minut
- distanță - milă

Unde:

☞ EFW = 4.4
☞ EFB = 10
☞ BFC = 0.084

EFW = energy flag for walking. Acest flag reprezintă valoarea medie de calorii consumată de o persoană cu greutatea medie estimată la 73 de kilograme (160 lb.) a cărei viteze medii de mișcare, pe milă , este de 3 mile pe oră. Conform tabelului din [10] rezultă faptul că un utilizator consumă 4.4 calorii pe minut.

EFB = energy flag for cycling. Acest flag reprezintă valoarea medie de calorii consumată de o persoana cu greutatea medie estimată la 73 de kilograme (160 lb.) a cărei viteze medii de mișcare, pe milă este de 16 mile pe ora. Conform tabelului din [11], pe oră, un utilizator consumă 600 de calorii, raportând aceste valori pe minut, va rezulta că un utilizator arde 10 calorii pe minut .

BFC = budget flag for driving. Acest flag reprezintă o valoare medie estimativă care vizează suma de bani consumată pe combustibil pentru o milă . Din catalogul de prezentare al unui autoturism Golf[33], considerat model de referință, valoarea medie de consum pentru 100 de kilometri (aproximativ 62 de mile) , calculată ca un raport mediu între combustibil benzină (4.9 litri) și motorină (3.8 litri) , este de 4.4 litri . Următoarea valoare care determină costul mediu între litrul de benzină și motorină în Europa, în momentul prezent, cu valori extrase online[34] ,a rezultat un cost mediu (în euro) pe litru de 1.2 euro.
Folosind regula de trei simplă :

Figure 5.19 Calcul unitate de măsură euro

Am concluzionat că valoarea medie de consum în euro pe milă este de 0.084 de euro.

Valorile variabilelor Wt (walking time), Wd (walking distance), Bt (bicycle time), Bd (bicycle distance), Ct (car time), Cd (car distance) vor fi extrase pentru fiecare tip de transport selectat, din rezultatul apelului serviciului Google, DirectionsService, asupra coordonatelor activităților planului current.

5.4.2. Sugestiile aplicației

După salvarea planului curent, se vor aplica formulele definite în tabelul de mai sus pe valorile extrase din apelul funcției serviciului DirectionsService și, pentru fiecare tip de transport , vor fi afișate valorile Time, Distance, Energy Cost, Budget Cost . Aceste valori vor fi vizibile utilizatorului cu scopul de a putea face o comparație clară între necesități, fiind în poziția în care acestia decid care este cel mai potrivit traseu .

Pe lângă aceste valori afișate utilizatorului, aplicația vine în suportul conceptului de optimizare prin adăugarea de sugestii, astfel :

- ⇨ Linii de legatura cu distanță mai mică de doi kilometri (1.2 mile) va duce la sugestia ca traseul de parcurs să fie de tip Bicycle
- ⇨ Linii de legatura cu distanță mai mica de un kilometru (0.6 mile) va duce la sugestia ca traseul de parcurs sa fie de tip Walking
- ⇨ Dacă nici una dintre condițiile enumerate mai sus nu este îndeplinită atunci sistemul va sugera tipul de traseu By Car

[33] https://www.auto-motor-und-sport.de/news/2429758/golf_preisliste.pdf
[34] https://www.cargopedia.ro/

5.4.3. Pseudocod

În următoarele două tabele este prezentat pseudocodul pentru implementarea mecanismului de optimizare activități , atât pentru partea de frontend cât și pentru partea de backend .

BACKEND

```
method sort_activities_by_start_date (activities)
Result: Boolean Value
startDateList = orderByStartDate (activities);
for each two succesive activities in startDateList do
    if startDate of current is greater than endDate of next then
        return false;
    else
        continue;
    end
end
return true;
method time_availability_between_activities (routes , activities)
Result: Boolean Value
for for each leg route in routes do
    variable fT = time of succesive activity;
    variable sT = time of current activity;
    if (fT – sT) is smaller than leg route time then
        return false;
    else
        continue;
    end
end
return true;
```

Table 5.2 Pseudocodul de optimizare activități - frontend

FRONTEND

function generateMapOptimisations(routes, transport)

```
Result: String Value
declare flags for each transport type;
for each transport type map do
    declare suggestion flags ;
    initialize suggestion flags with False;
    calculateTransportFlags();
    for each leg route do
        if leg route distance bigger than reference value for bicycle then
            jump to next if;
        else
            continue;
        end
        set suggestion flag for bicycle to True ;
        exit for ;
        if leg route distance bigger than reference value for walking then
            exit for;
        else
            continue;
        end
        set suggestion flag for walking to True ;
        exit for ;
    end
    if all flags are false then set suggestion flag Car to True;
    generate suggestion message based on suggestion flags and transport flags;
end
return suggestion messages ;
```

Table 5.3 Pseudocodul de optimizare activități - backend

Capitolul 6. Testare şi Validare

Acest capitol prezintă metodele de testare și validare utilizate pentru a verifica dacă comportamentul sistemului este cel așteptat. În general, testarea nu garantează funcționarea completă și corectă în toate condițiile sistemului, dar poate fi utilă pentru a identifica problemele și pentru a oferi soluții de rezolvare.

Ca și tehnici de testare au fost folosite testele automate și manuale. În testarea manuală testele sunt executate manual, rezultând rapoarte de test fără ajutorul unui software de testare. În testarea automată testele sunt executate folosindu-se framework-uri sau aplicații specializate în testare.

6.1. Testare manuală

Majoritatea testării pe perioada de dezvoltare pentru partea de frontend a fost făcută manual testând pas cu pas cerințele sistemului, folosindu-se diferite metode de testare, cum ar fi black box și white box .

Pentru metoda de tastere black box sunt create scenarii sau cazuri de test bazate pe cerințele sistemului și pe specificațiile testelor de acceptanță pentru diferiți actori (utilizatori) . În rândurile următoare sunt reprezentate mai multe cazuri de testare de tip black box pentru partea de frontend a proiectului .

Caz de testare 1

Căutarea unui oraș de interes

→ Actor:
- Utilizator autentificat
- Utilizator neutentificat

→ Precondiții:
- Utilizatorul se află pe pagina principal (Home)
  sau
- Utilizatorul a selectat opțiunea Home din bara de navigare și a fost redirectat pe pagina principal

Acțiune	Rezultat așteptat
Introducerea unor valori alfabetice în câmpul de căutare	Apar mai multe opțiuni de selectat generate prin Autocomplete
Selectarea unei locații din cele sugerate	Se recunoaște opțiunea aleasă și se închide bara de opțiuni
Apasă butonul ”Search”	Se părăsește pagina curentă și se continuă navigarea către o pagină nouă

Table 6.1 Testare Căutarea unui oraș de interes

Caz de testare 2

Adăugarea de activități

→ Actor :
- Utilizator autentificat
- Utilizator neautentificat

→ Precondiții:

- Utilizatorul a selectat o locație
- Utilizatorul a selectat tipul de planificare dorit (weekly sau daily)

Acțiune	Rezultat așteptat
Selectarea prin click a unei locații pe harta afișată	Apar coordonatele (longitudine și latitudine) în formă flotantă
Completarea câmpurilor de activitate corespunzătoare coordonatelor selectate	Se analizează conținutul și se generează mesaj de confirmare/infirmare a corectitudinii datelor
Apasă butonul ”Add activity”	Se adaugă local activitatea, se afișează în partea dreapta a ecranului, alături de celelalte activități adăugate

Table 6.2 Testare Adăugarea de activități

Metoda de tastere white box se bazează pe structura codului intern al aplicației de frontend. În testarea white box se pune accent asupra perspectivei interne a sistemului. Această testare a fost făcută la nivel de unitate, prin cazuri de testare pozitivă și negativă.

Caz de testare pozitivă : Înregistrare utilizator

Intrare	Descrierea cazului de test	Remarci
Email	• Trebuie să contină forma unui input de tip email:[utilizator]@[domeniu].[TLD]. • Formatul nu conține caractere speciale precum # $ %	✓ Inserarea unor valori care respectă formatul de input email, de exemplu: ioanacristea5@yahoo.com ✓ Introducerea unor valori ce nu include caractere speciale
Parola	• Conține minim un caracter cu majuscule • Conține minim un caracter litere mici • Conține cel puțin caracter numeric • Dimensiunea minima este de 8 caractere și dimensiunea maximă de 32 de caractere	✓ Inserarea unei valori de input cu cel putin un caracter litere mici, un caracter majuscule, o cifră , a cărei dimensiune finală să fie între 8 și 32 de caractere, de exemplu: Par0laS3f3
Birthday	• Trebuie să conțină forma unui input de tip date sau o formatare de tipul MM/DD/YYYY unde : → MM - valoare între 1 și 12 → DD - valoare între 1 și 28/29/30/31, în funcție de MM → YYYY - valoare între 1900 și anul curent	✓ Inserarea unei valori de input care respectă structura de formatare pentru tipul date , de exemplu : 03/06/2002
Name	• Trebuie să conțina doar caractere alfabetice	✓ Inserarea unei valori de input care respect condiția din cazul de test, de exemplu : Ioana

Table 6.3 Testare Înregistrare utilizator

După execuția acestor teste, definite în tabelul de mai sus, sistemul ar trebui să nu detecteze nicio neregulă, dat faptul că se respectă specificațiile definite, în caz contrar, este detectat un comportament neașteptat iar programatorul are sarcina să revină asupra codului scris și să testeze din nou.

Datorită faptului că proiectul de licență a fost construit treptat, pe parcursul unui an universitar, aproximativ, adăugarea de noi funcționalități și modificarea celor existente s-a realizat progresiv .În momentul adăugării de funcționalități noi , în funcție de complexitatea și de nivelul de integrare cu celelalte componente , a fost necesară repetarea sau modificarea unor teste , tehnică denumită testare de tip regression .

6.2. Testare automată

Partea de testare pentru componenta de backend a fost realizată automat folosindu-se Postman, un instrument de testare a API-urilor. Cu ajutorul Postman au fost testate și monitorizate toate URL-urile serviciilor REST din aplicatie. În figurele alăturată este prezentată interfața Postman și un exemplu de testare - apelul request-ului de extragere a tuturor planurilor din aplicație :

Figure 6.1 Secțiunea URL și metode de apel ale aplicației Postman

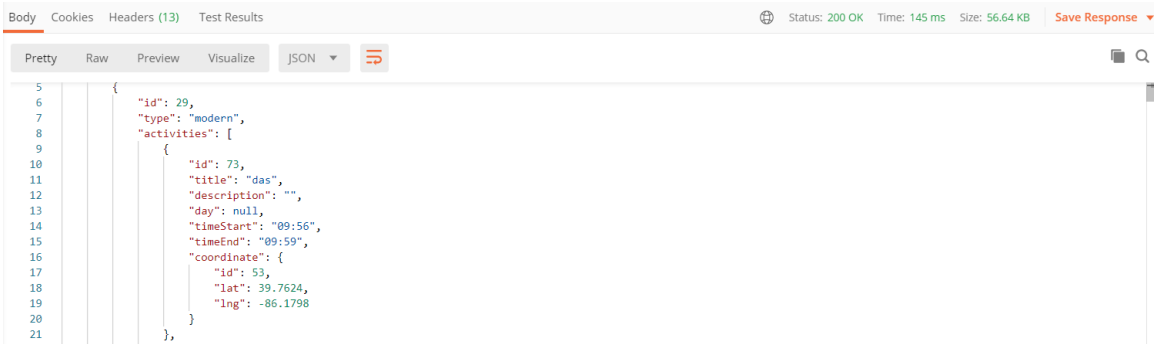

Figure 6.2 Secțiunea Response Body a aplicației Postman

Prin acest tip de testare s-a putut verifica cu ușurință modul în care comunică cele trei componente ale sistemului . Prin request-uri get se putea observa dacă datele au fost persistate corect în baza de date iar prin request-uri post se verifica corectitudinea cu care ajung obiectele structurate în format JSON în baza de date .

Capitolul 7. Manual de Instalare si Utilizare

Acest capitol prezintă resursele hardware și software necesare pentru configurarea și execuția proiectului dezvoltat, cât și pașii necesari de parcurs pentru instalarea fiecărei componente fără de care execuția aplicației nu ar fi posibilă, urmat de un manual de utilizare.

7.1. Resurse necesare

Fiind o aplicație web, sistemul poate fi accesat prin intermediul unui browser web de către utilizatori, de aceea , ca și resurse hardware sunt necesare cele pentru rularea unui astfel de browser web, cum ar fi : Google Chrome, Microsoft Edge, Mozilla Firefox, Safari, Opera, Internet Explorer .

Deployment-ul aplicației se poate face pe mașina locală utilizând următoarele componente:

- Java 1.8
- MySQL 8.0.16
- MySQL Workbench 8.0 CE
- NodeJS 10.16.3
- Apache Maven 3.6.3
- IntelliJ IDEA 2019.2.3
- WebStorm 2019.3.3

7.2. Instalare și rulare

Pentru rularea corectă a aplicației trebuie descărcate, instalate si configurate instrumentele software specificate mai sus, urmărind următorii pași:

- **Descărcarea proiectului**
  1. Accesarea paginii de github https://github.com/ioanac977
  2. Selectarea repository-ului cu numele ”Licenta_2020” care conține două subdirectoare ”Frontend”, respectiv, ”Backend” și descărcarea repository-ului local

- **Configurări pentru baza de date :**
  1. Descărcarea și instalarea MySQL Workbench[35]
  2. Descărcarea serverului MySQL[36]
  3. Se deschide aplicația MySQL Workbench și se creează o conexiune nouă (folosind portul default 3306) și un utilizator nou
  4. Crearea unei noi scheme pentru baza de date :

[35] https://dev.mysql.com/downloads/workbench/
[36] https://dev.mysql.com/downloads/installer/

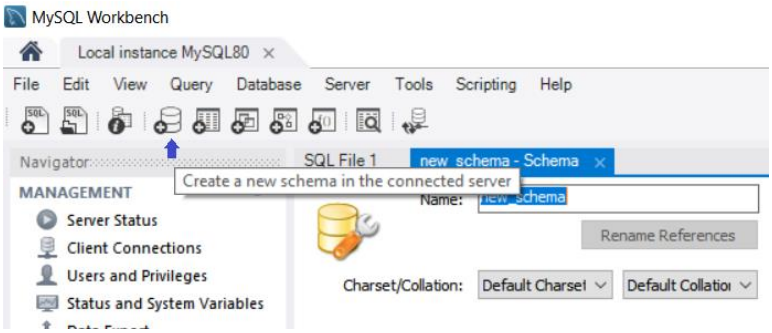

Figure 7.1 Crearea unei scheme în MySQL Workbench

→ în căsuța Name se va completa cu numele bazei de date ”licenta” și se va apăsa butonul ”Apply” din partea dreaptă , jos, a ferestrei

❖ **Configurări pentru backend :**

1. Descărcarea și instalarea kit-ului de dezvoltare Java (Java Development Kit[37])
2. Setarea variabilei JAVA_HOME[38] pe sistemul de operare
3. Descărcare și instalare Maven[39]
4. Descărcarea și instalarea Java IDE – Intellij IDEA [40] , ediția Comunity
5. Importarea proiectului în Intellij : File → Open → se selectează fișierul Pom.xml care se află în primul director al aplicației → click pe butonul ”OK” → va apărea o căsută cu trei opțiuni de deschidere a fișierului Pom.xml :

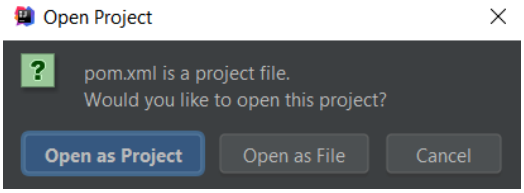

Figure 7.2 Fereastră de opțiuni în IntelliJ pentru deschiderea fișierului pom.xml

→ click pe opțiunea Open as Project → proiectul se va încărca și în același timp va descărca dependințele de care are nevoie pentru a putea fi rulat, apoi va apărea, în partea de sus dreapta, denumirea aplicației

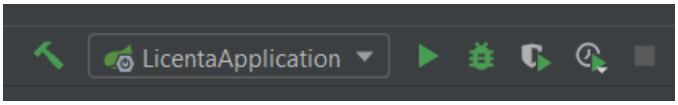

Figure 7.3 Bara de rulare a aplicației în IntelliJ

→ înainte de a porni aplicația este necesară modificarea conținului fișierului application.properties care se află în src/main/resources,

[37] https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html
[38] https://confluence.atlassian.com/doc/setting-the-java_home-variable-in-windows-8895.html
[39] https://maven.apache.org/download.cgi
[40] https://www.jetbrains.com/idea/download/#section=windows

cu datele de identificare a utilizatorului bazei de date înregistrate în pasul de configurare a bazei de date
→ aplicația va porni pe portul localhost:8091

❖ **Configurări pentru frontend :**

1. Descărcare și instalare NodeJS[41]
2. Descărcare și instalare WebStorm[42]
3. Importarea proiectului în WebStorm : File → Open → Selectarea directorului cu aplicația de frontend → Click pe butonul ”OK” → proiectul se va încărca și indexa, urmând ca pachetele de dependințe din package.json să fie descărcate și instalate iar acest lucru se va realiza manual accesând căsuta cu o săgeată îndreptată spre dreapta din bara de rulare de sus în partea dreaptă :

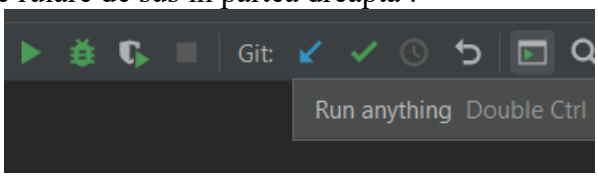

Figure 7.4 Bara de rulare a aplicației în WebStorm

→ va apărea o fereastra în care se introduce comanda “npm install” urmată de tasta Enter → se revine la pasul 3 și de data aceasta se introduce comanda ”npm start” urmată de tasta Enter ceea ce va rezulta prin pornirea aplicației → aplicația va putea fi accesată prin http://localhost:3000/

7.3. Instrucțiuni de utilizare

Pentru utilizarea aplicației vor fi reluate anumite scenarii descrise în subcapitolul 4.4.1 și vor fi explicate prin imagini și descriere text . Scenariile prezentate în paginile următoare sunt printre cele mai reprezentative pentru acest proiect.

[41] https://nodejs.org/en/download/
[42] https://www.jetbrains.com/webstorm/download/#section=windows

7.3.1. Pagina principală

Pagina principală și întreaga navigare în aplicație este diferită pentru fiecare tip de utilizator iar acest lucru este evidențiat chiar din pagina principală, astfel că :

- Utilizator neautentificat

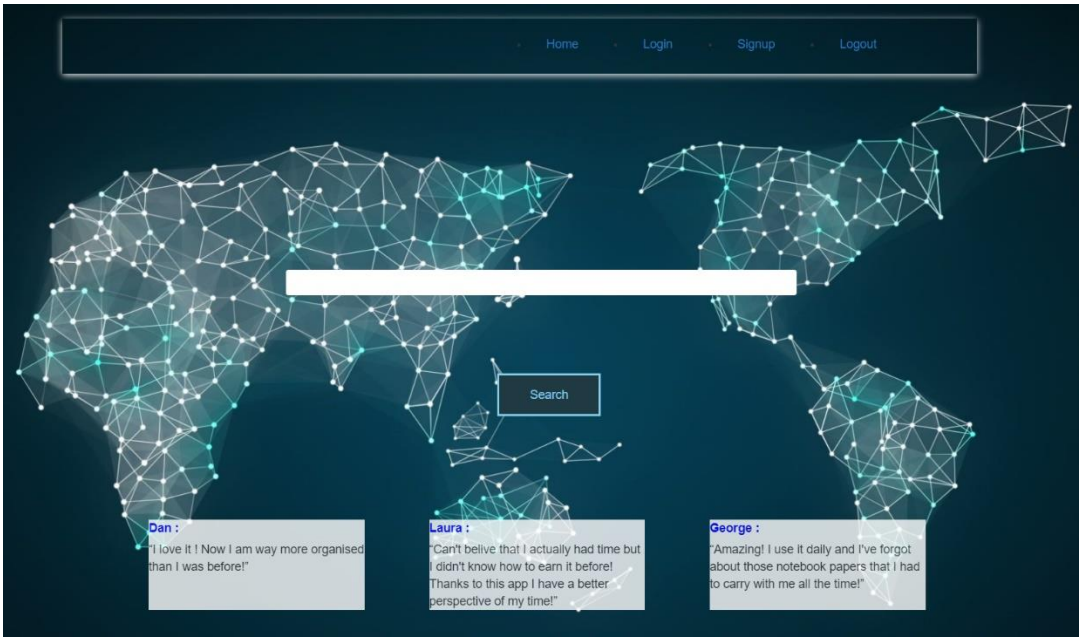

Figure 7.5 Pagina principală pentru un utilizator neautentificat

- Utilizator autentificat

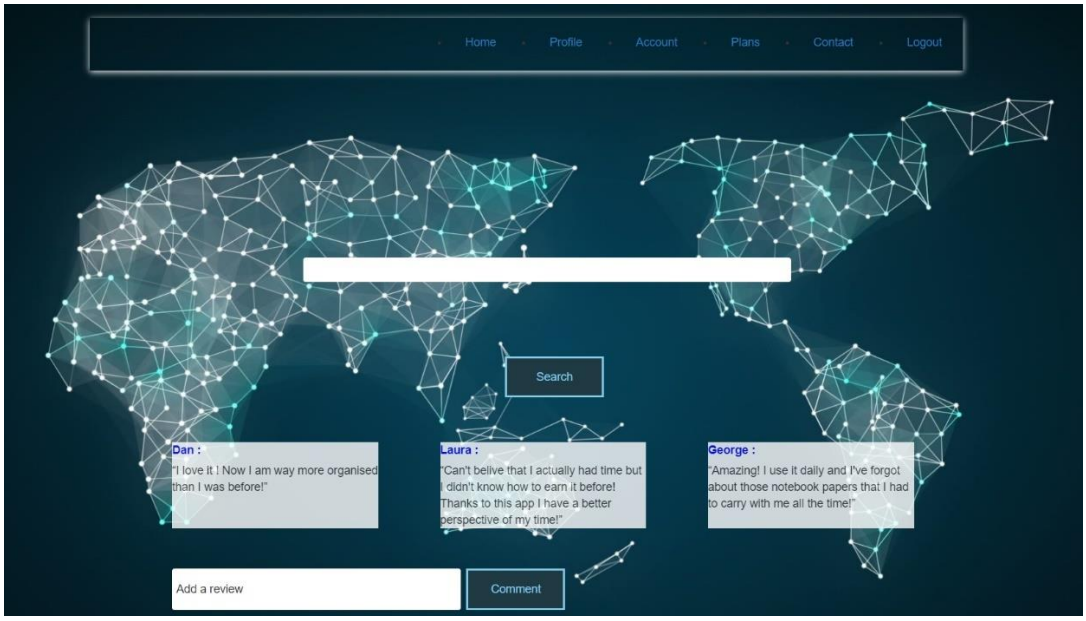

Figure 7.6 Pagina principală pentru un utilizator autentificat

7.3.2. Înregistrare și autentificare

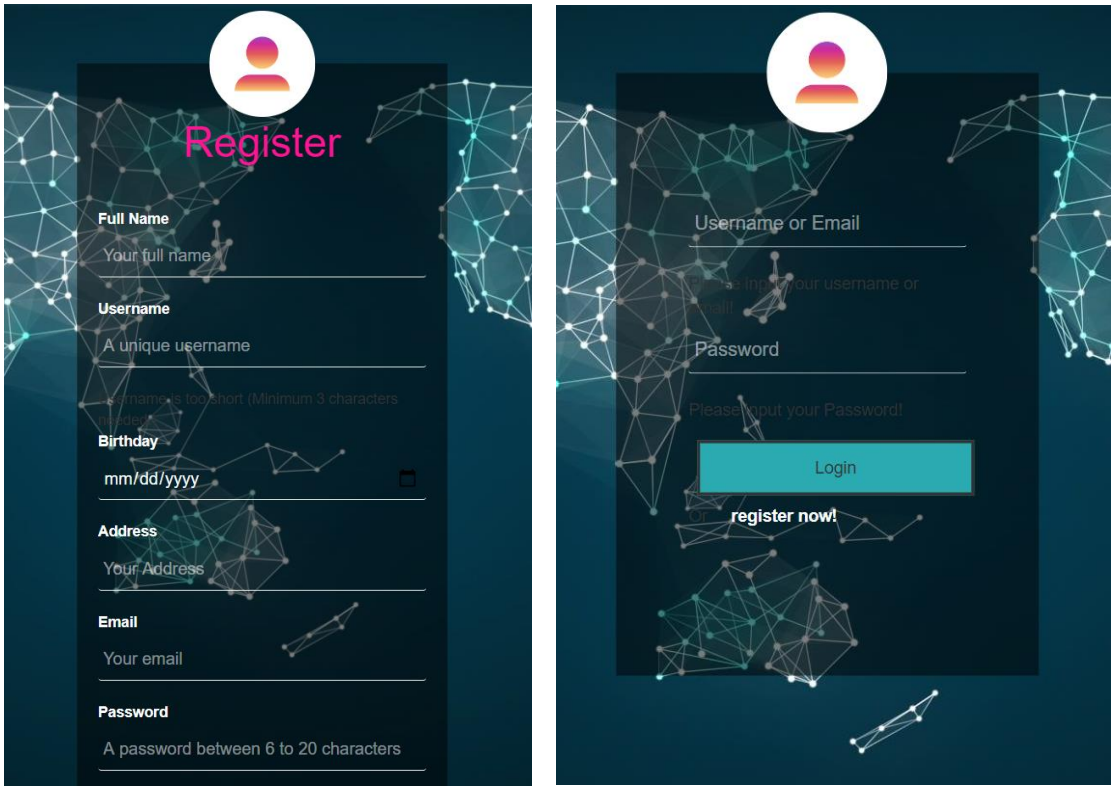

Figure 7.7 Pagina de înregistrare și pagina de autentificare

Pagina de autentificare conține câmpurile definite în baza de date pentru tabela Users de aceea, înainte de a fi stocate datele unui nou utilizator, acestea trebuie validate . În momentul în care câmpurile sunt valide utilizatorului i se deblochează butonul de Register și va putea să își creeze contul .

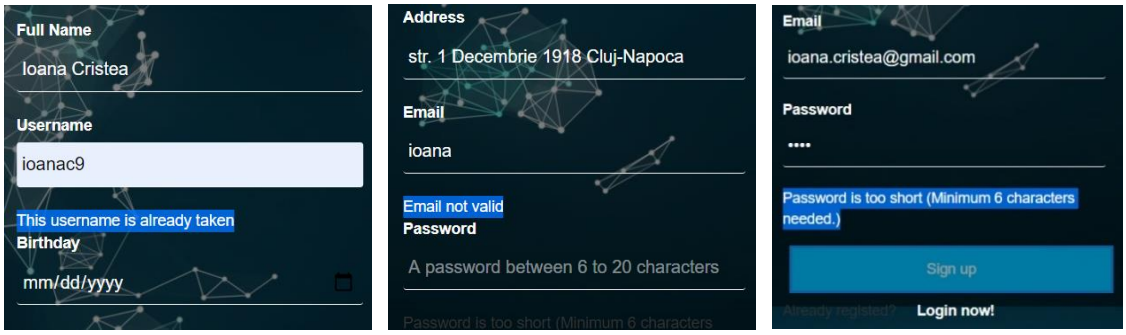

Figure 7.8 Validări de câmpuri pentru pagina de înregistrare

După cum se poate observa în capturile anterioare, butonul de Sign Up este albastru deschis, ceea ce înseamnă că încă mai sunt câmpuri invalide în formular. După completarea formulatorului, în mod corespunzător, culoarea butonului va fi verde , după cum urmează în Figura 7.10 .

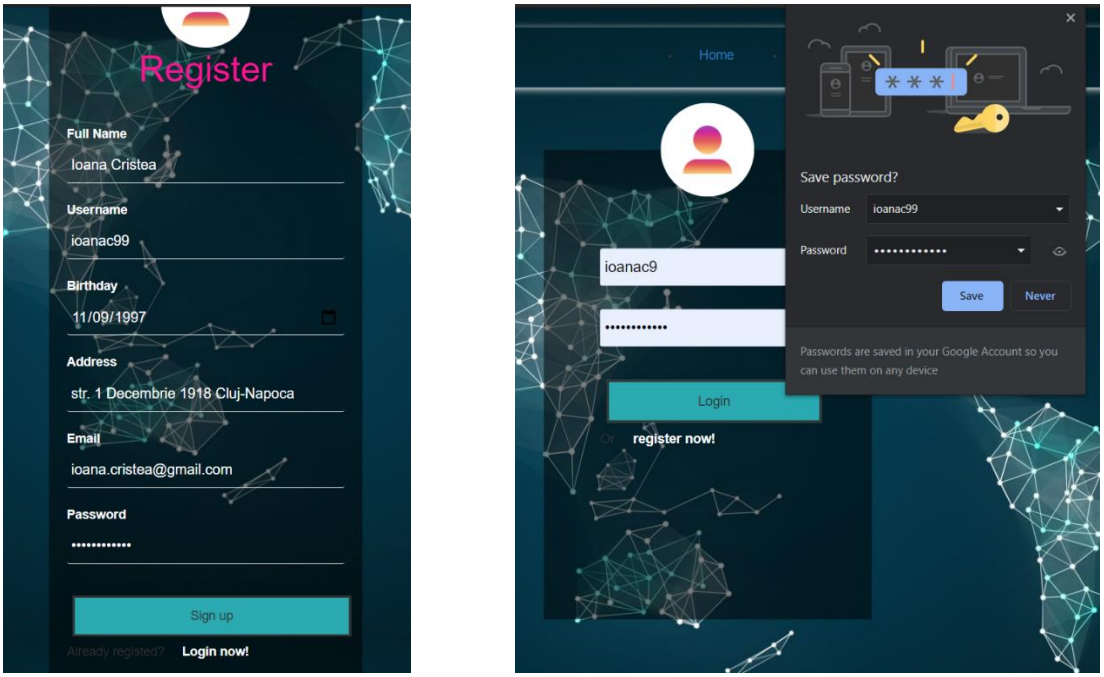

Figure 7.9 Formul valid de înregistrare și formular de salvare date

După ce utilizatorul apasă butonul ”Sign up” este redirectat către pagina de login de unde va apărea o fereastră prin care utilizatorul își poate păstra credențialele salvate asociate contului de Google. Dacă apasă butonul ”Save” formularul de login se va completa automat cu noile date de autentificare, dacă apasă butonul ”Never” atunci formularul de login va trebui completat manual de către utilizator .

7.3.3. Planificare activități

- Pasul 1 : Alegerea unei locații prin căutarea după denumire

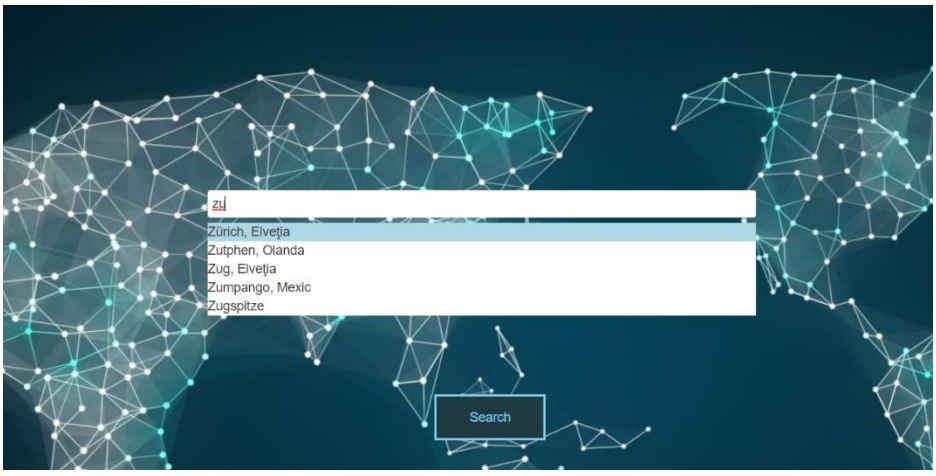

Figure 7.10 Căutare locație pentru planificare

- Pasul 2 : Alegerea tipului de planificare care va rezulta în două rute alternative

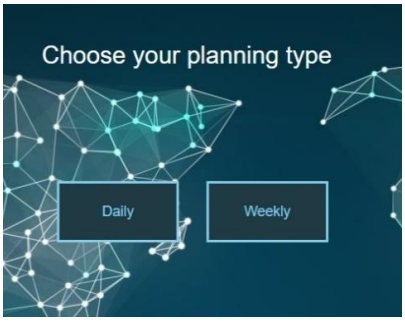

Figure 7.11 Pagina de alegere a tipului de planificare

- Pasul 3 : În cazul în care la pasul anterior utilizatorul dă click pe butonul „Daily” atunci va fi redirectat către pagina de selectare a unei zile, reprezentată în figura 7.14 , în caz contrat, pentru ”Weekly” va fi redirecționat către o pagină pentru selectarea unei săptămâni.

Figure 7.12 Pagină de selectare zi

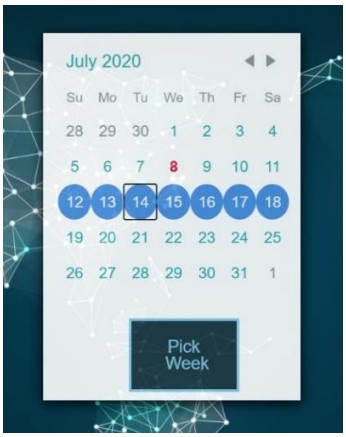

Figure 7.13 Pagină de selectare săptămână

- Pasul 4 : Adăugarea de activități pentru tipul de activitate selectat
- Pasul 4.1 : Adăugarea de activități pentru tipul de planificare zilnică

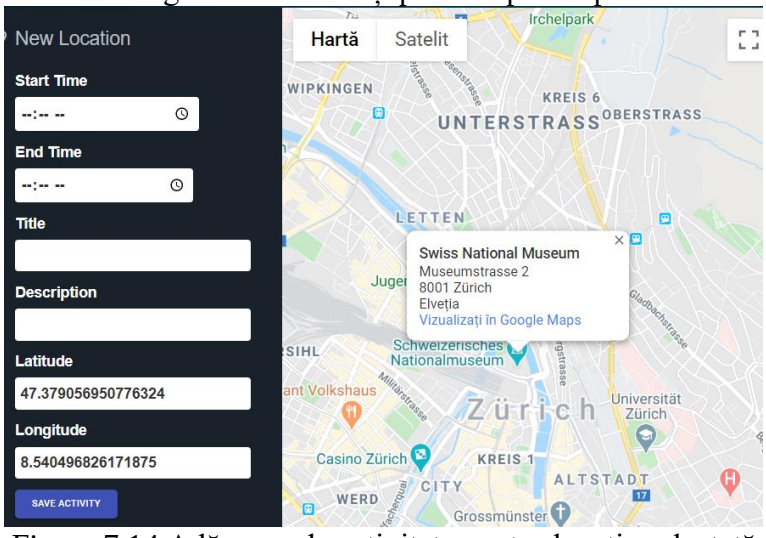

Figure 7.14 Adăugare de activitate pentru locația selectată

- Pasul 4.2 : Vizualizarea activităților adăugate pentru tipul de planificare daily. În momentul în care utilizatorul dă click pe hartă, coordonatele de latitudine și longitudine pentru acea locație se vor completa automat în formularul din stânga. După completarea tuturor câmpurilor de activitate și apăsarea butonului „Save activity”, activitatea va apărea alături de celelalte activități în panoul din dreapta paginii .

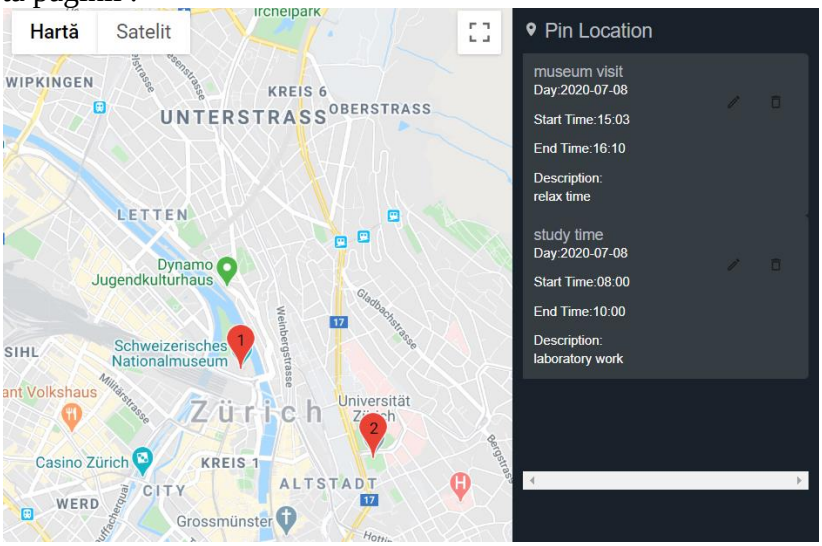

Figure 7.15 Vizualizarea activităților adăugate pentru planul curent

- Pasul 4.3 Salvarea planului curent este simplă , prin click pe butonul„Save plan” care se găsește sub secțiunea de adăugare activități. În momentul apăsării butonului, se verifică dacă utilizatorul este autentificat iar apoi creează un request către backend de unde sunt ordonate și validate activitățile, în caz contrar un utilizator neautentificat va fi redirectat către o pagină de login

7.3.4. Selectarea rutei optime

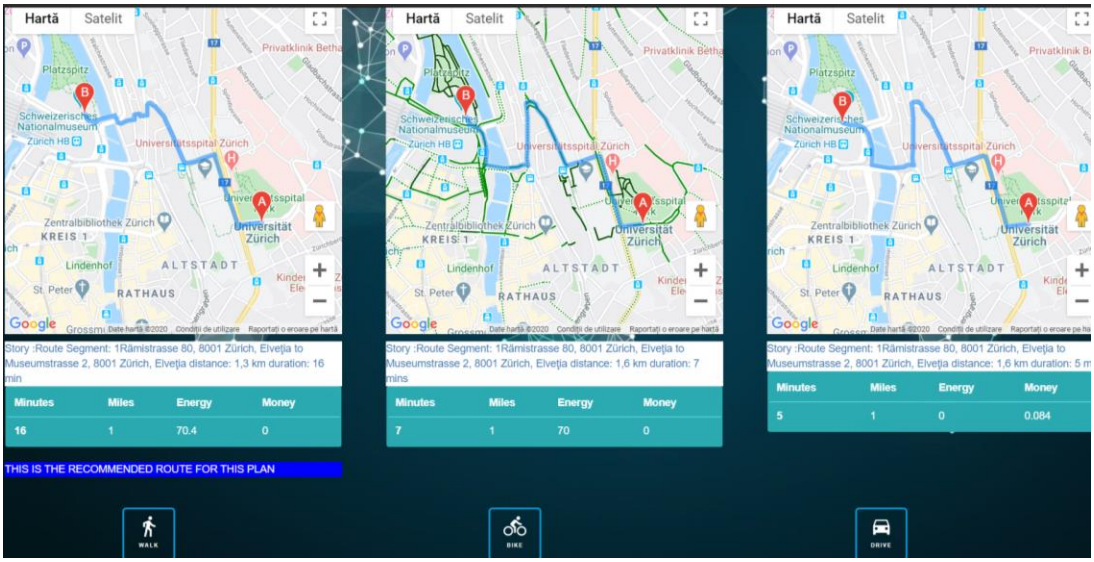

Figure 7.16 Alegerea tipului de traseu

7.3.5. Vizualizarea planurilor salvate

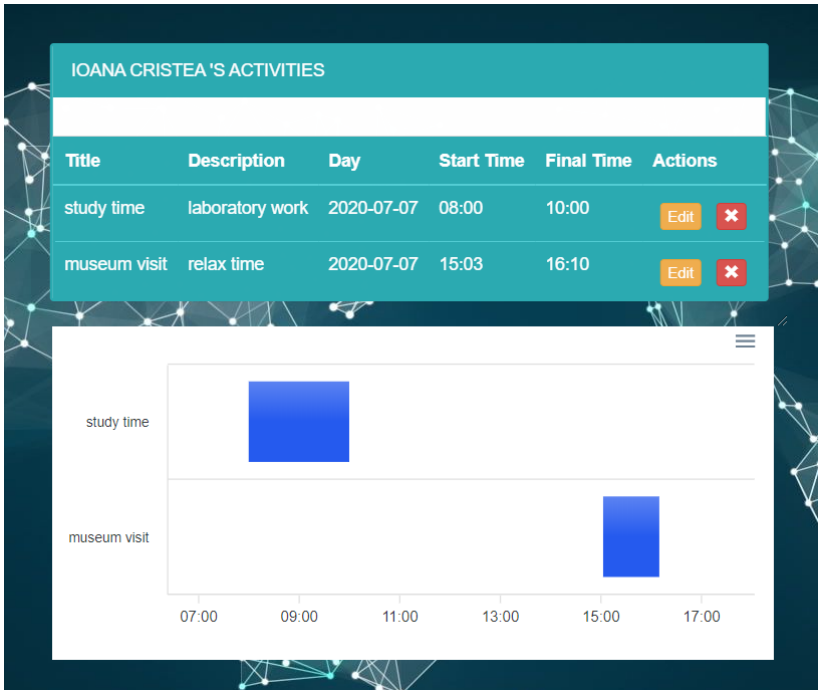

Figure 7.17 Vizualizare date și raport grafic pentru activitățile planului selectat

Capitolul 8. Concluzii

În cadrul acestui capitol sunt captate realizările proiectului, reprezentate de obiectivele ce au fost atinse și viitoarele dezvoltări posibile ale aplicației.

8.1. Analiza rezultatelor obținute

Sistemul care a fost implementat și-a îndeplinit obiectivul principal de a fi o aplicație care să realizeze planificarea activităților utilizatorilor în funcție de traseul optim selectat pentru nevoile și cerințele fiecăruia oferind o imagine de ansamblu, o ordonare cronologică a activităților , un raport vizual și tabelar comparativ pentru cele trei tipuri de traseu : cu mașina, cu bicicleta și mers pe jos alături de sugestiile sistemului.

Utilizatorii pot să își adauge activități pentru oricare din cele două tipuri de planuri : săptămânal sau zilnic , fără să se preocupe de ordinea lor cronologică , deoarece acest lucru intră în sarcina aplicației. Activitățile adăugate sunt ordonate după timpul de început al fiecărei activități iar trasarea celor trei hărți corespunzătoare mijloacelor de transport : autoturism, bicicletă și mers pe jos țin cont de coordonatele activităților ordonate .Pe lângă afișarea grafică a celor trei hărți, prin aplicarea algoritmului de optimizare al activităților , hărțile vor avea asociate valorile de cost al energiei (măsurată în calorii), al bugetului (monedă euro) plus distanța totală alături de timpul total și sugestiile pentru un anumit tip de transport .

Un alt obiectiv îndeplinit îl reprezintă vizualizarea tuturor planurilor salvate și a tuturor activităților asociate acestora . Pentru planurile zilnice utilizatorii pot vedea o diagrama de activitate pentru orele ocupate din ziua planificată. Din aceste diagrame utilizatorii pot vedea ferestrele rămase libere între minutele sau orele ocupate ale zilei și pot folosi acel timp cum doresc . Pentru planurile săptămânale, utilizatorii pot vedea diagrame cu numărul de activități realizate pe zile . În acest mod, utilizatorii pot face o diferență între săptămâni, luni sau chiar perioade și pot lua măsuri extra de a anticipa organizarea activităților ulterioare .

Următorul obiectiv îndeplinit îl reprezintă gestionarea datelor de profil și de cont . Utilizatorul poate cere modificarea datelor de cont iar datele de profil pot fi modificate în timp real chiar de către utilizator. Utilizatorul autentificat poate chiar să contacteze administratorul, în cazul în care sunt probleme cu aplicația sau are orice alte adăugări de transmis .

Pentru a îndeplini obiectivul de a atrage cât mai mulți utilizatori sistemul pune la dispoziție ultimele comentarii ale utilizatorilor și permite navigarea celor neautentificați în aplicație până în momentul salvării planului construit .

Cerințele non-funcționale au fost și ele îndeplinite , astfel că securitatea aplicației este asigurată de roluri, setare de sesiuni, JWT și salvarea parolei criptată în baza de date. Utilizabilitatea aplicației a fost asigurată prin indicațiile label-urilor alături de Icon-urile sugestive iar din punct de vedere a portabilității, natura aplicației (web) permite rularea pe mai multe browsere, performanța aplicației fiind asigurată prin request-uri simple.

8.2. Dezvoltări ulterioare

Sistemul propus este deschis modificărilor și extinderilor viitoare, conceptele și ideile pe care le expune sunt destul de generale, îndreptate către un public cât mai larg, iar îmbunătățirile ce pot fi aduse sunt numeroase, dintre care :

- Portarea aplicației pe dispozitive mobile, dat faptului că utilizatorii sunt în continuă mișcare iar accesarea aplicației web de pe telefonul mobil poate fi îngreunată de traficul de date și de nevoia de a accesa browser-ul de fiecare dată când este necesară navigarea prin aplicație
- Sincronizarea planurilor cu timpul real și notificarea utilizatorilor prin reamintirea începerii unei activități
- Îmbunătățirea algoritmului de optimizare a rutelor prin implementarea unei funcționalități de alegere a tipului de mașină, a tipului de carburant și extragerea costului pe combustibil în funcție de locația selectată de utilizator și a tipului de autoturism . Îmbunătățirea parametrilor de generare a energiei consumate prin mersul pe bicicletă sau pe jos prin completarea unui formular cu descrierea fizică a utilizatorului : greutate, înălțime, vârstă, sex și rezistență fizică (nivelul sedentarisului)
- Implementarea unei statistici cu privire la categoriile de activități desfășurate (divertisment, studiu, mâncare)

Bibliografie

[1] J. D. Ferner, Successful Time Management , Volumul 75 din Wiley Self-Teaching Guides, 1980.

[2] T. Chondrogiannis, „Efficient Algorithms for Route Planning Problems on Road Networks,”,teză 2017,
Available: https://pro.unibz.it/library/thesis/00010851P_30362.pdf.

[3] Globema, „Câteva date despre geospatial,” Noiembrie 2017. Available:https://www.globema.ro/o-industrie-de-400-miliarde-geospatial/.

[4] D. Rijo, „Google Maps now used by over 1 billion people every month,” PPC Land,secțiunea Marketing News, 15 Februarie 2020.
Available:https://ppc.land/google-maps-now-used-by-over-1-billion-people-every-month/.

[5] J. Bennett, „OpenStreetMap,” *What is OpenStreetMap,* editura Packt Publishing, septembrie 2010.

[6] G. Maier, „OpenStreetMap, the Wikipedia Map,” vol. 1, nr. 1, pp. R3-R10, 11 decembrie 2014,
DOI:10.18335/region.v1i1.70.

[7] Beuth University of Applied Sciences, „Google vs OpenStreetMap – Which one is better,” 2019.

[8] Harshad B. Prajapati, Vipul K. Dabhi „Stochastic Local Search,” *Foundations and Applications,The Morgan Kaufmann Series in Artificial Intelligence,* 2005,
DOI: 10.1016/B978-155860872-6/50018-4 .

[9] Harshad B. Prajapati, Vipul K. Dabhi, „High Quality Web-Application Development,” *Advance Computing Conference,* 2009,
DOI: 10.1109/IADCC.2009.4809267.

[10] Rebus Inc, în *Walking and running : the complete guide*, Alexandria, Va, editura Time-Life Books, 1989, pp. 19-20.

[11] P. D. Roni Sarig, în *The bike to work guide : save gas, go green, get fit*, 2009, pp. 88-89
Available: https://archive.org/details/biketoworkguides0000sari/.

Anexa 1

8.2.1. Lista de tabele

Table 3.1 Comparație Google Maps și OpenStreet 10
Table 4.1 Cerintele funcționale ale sistemului 13
Table 4.2 Cerințele non-funcționale ale sistemului 14
Table 5.1 Flaguri pentru tipurile de transport 45
Table 5.2 Pseudocodul de optimizare activități - frontend 47
Table 5.3 Pseudocodul de optimizare activități - backend 48
Table 6.1 Testare Căutarea unui oraș de interes 49
Table 6.2 Testare Adăugarea de activități 50
Table 6.3 Testare Înregistrare utilizator 50

8.2.2. Lista de figuri

Figure 4.1 Arhitecura sistemului 15
Figure 4.2 Arhitectura sistemului de rutare 16
Figure 4.3 Diagrama UML Use Case Utilizator 17
Figure 4.4 Diagrama UML Use Case Guest 18
Figure 4.5 Diagrama UML Use Case Admin 18
Figure 4.6 Diagrama de evenimente Înregistrare 20
Figure 4.7 Diagrama de evenimente Adaugare Plan 22
Figure 4.8 Diagrama de evenimente Editare Plan 24
Figure 5.1 Architectural pattern Three Layer pentru backend 33
Figure 5.2 Diagrama de pachete a componentei backend 35
Figure 5.3 Design pattern MVC pentru frontend 36
Figure 5.4 Arhitectura detaliată a componentelor din frontend 37
Figure 5.5 Diagrama de pachete pentru componenta frontend 37
Figure 5.6 Definirea rutelor prin Route în aplicația frontend 38
Figure 5.7 Navigare prin push în aplicația frontend 38
Figure 5.8 Navigare prin redirect în aplicația frontend 38
Figure 5.9 Navigare prin import în aplicația de frontend 39
Figure 5.10 Diagrama de navigare a utilizatorului neautorizat 39
Figure 5.11 Diagrama de navigare a utilizatorului autorizat 40
Figure 5.12 Apel funcție de request în frontend 40
Figure 5.13 Request folosind axios în frontend 41
Figure 5.14 Apel serviciu DirectionsService în frontend 41
Figure 5.15 Exemplu de geocodare inversă în aplicația de frontend 41
Figure 5.16 Exemplu de geocodare directă în aplicația de frontend 42
Figure 5.17 Exemplu de utilizare places API în aplicația de frontend 42
Figure 5.18 Diagrama bazei de date 43
Figure 5.19 Calcul unitate de măsură euro 46
Figure 6.1 Secțiunea URL și metode de apel ale aplicației Postman 51
Figure 6.2 Secțiunea Response Body a aplicației Postman 51
Figure 7.1 Crearea unei scheme în MySQL Workbench 53

Figure 7.2 Fereastră de opțiuni în IntelliJ pentru deschiderea fișierului pom.xml 53
Figure 7.3 Bara de rulare a aplicației în IntelliJ 53
Figure 7.4 Bara de rulare a aplicației în WebStorm 54
Figure 7.5 Pagina principală pentru un utilizator neautentificat 55
Figure 7.6 Pagina principală pentru un utilizator autentificat 55
Figure 7.7 Pagina de înregistrare și pagina de autentificare 56
Figure 7.9 Validări de câmpuri pentru pagina de înregistrare 56
Figure 7.10 Formul valid de înregistrare și formular de salvare date 57
Figure 7.12 Căutare locație pentru planificare 57
Figure 7.13 Pagina de alegere a tipului de planificare 58
Figure 7.14 Pagină de selectare zi Figure 7.15 Pagină de selectare săptămână . 58
Figure 7.16 Adăugare de activitate pentru locația selectată 58
Figure 7.17 Vizualizarea activităților adăugate pentru planul curent 59
Figure 7.18 Alegerea tipului de traseu 59
Figure 7.19 Vizualizare date și raport grafic pentru activitățile planului selectat 60