

UNIVERSITATEA TEHNICĂ „Gheorghe Asachi” din IAȘI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DOMENIUL: Calculatoare și Tehnologia Informației
SPECIALIZAREA: Tehnologia Informației

Lucrare de diplomă

Coordonator științific:
prof.dr.ing. Florin LEON

Absolvent:
Marian FLĂMÎNZANU-MATEIUC

Iași, 2024

UNIVERSITATEA TEHNICĂ „Gheorghe Asachi” din IAȘI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DOMENIUL: Calculatoare și Tehnologia Informației
SPECIALIZAREA: Tehnologia Informației

Freightbroker
**Platformă web destinată
transporturilor de mărfuri**

LUCRARE DE DIPLOMĂ

Coordonator științific:
prof.dr.ing. Florin LEON

Absolvent:
Marian FLĂMÎNZANU-MATEIUC

Iași, 2024

DECLARAȚIE DE ASUMARE A AUTENTICITĂȚII PROIECTULUI DE DIPLOMĂ

Subsemnatul FLĂMIANU-MATEIU C MARIAN,
legitimat cu C1 seria 17 nr. 089 365, CNP 5010809 22 6700
autorul lucrării

FREIGHT BROKER - PLATFORMĂ WEB DESTINATĂ TRANSPORTURILOR
DE MĂRFURI

elaborată în vederea susținerii examenului de finalizare a studiilor de licență, programul
de studii TEHNOLOGIA INFORMAȚIEI organizat de către Facultatea de
Automatică și Calculatoare din cadrul Universității Tehnice „Gheorghe Asachi” din Iași,
sesiunea JULIE a anului universitar 2023 - 2024, luând în
considerare conținutul Art. 34 din Codul de etică universitară al Universității Tehnice
„Gheorghe Asachi” din Iași (Manualul Procedurilor, UTI.POM.02 - Funcționarea
Comisiei de etică universitară), declar pe proprie răspundere, că această lucrare este
rezultatul propriei activități intelectuale, nu conține porțiuni plagiate, iar sursele
bibliografice au fost folosite cu respectarea legislației române (legea 8/1996) și a
convențiilor internaționale privind drepturile de autor.

Data
27.06.2024

Semnătura


CUPRINS

<i>Capitolul 1. Introducere</i>	1
<i>Capitolul 2. Fundamentarea teoretică și documentarea bibliografică</i>	3
2.1. Analiza soluțiilor existente pentru dezvoltarea de aplicații server.....	4
2.2. Analiza soluțiilor existente pentru dezvoltarea de aplicații client.....	5
2.3. Analiza soluțiilor existente pentru dezvoltarea unei baze de date.....	5
2.4. Analiza algoritmilor ce pot fi utilizati pentru obtinerea unei solutii optime.....	7
2.4.1. Abordarea programarii liniare	8
2.4.2. Optimizare de tip roi de particule.....	8
2.4.3. Metoda/algoritmul evolutiv	9
<i>Capitolul 3. Proiectarea aplicației</i>	11
3.1. Analiza platformei hardware	11
3.2. Proiectarea aplicației - idei generale	12
3.3. Proiectarea propriu-zisă	14
<i>Capitolul 4. Implementarea aplicației</i>	22
4.1. Interfața cu utilizatorul.....	22
4.2. Comunicarea cu alte sisteme și stocarea informațiilor	25
4.2.1. OpenStreetMap Nominatim	25
4.2.2. Paypal Sandbox.....	25
4.2.3. Stocarea informațiilor	26
4.3. Dificultăți întâmpinate și modalități de rezolvare	27
<i>Capitolul 5. Testarea aplicației</i>	28
5.1. Lansarea în execuție a aplicației	28
5.2. Testarea sistemului	28
5.3. Rezultate experimentale	33
5.3.1. Date de test	33
5.3.2. Testarea algoritmului evolutiv.....	34
5.4. Resurse necesare.....	39
5.5. Securitatea sistemului	41
Concluzii	44
Bibliografie	46
Anexe	48

Freightbroker - Platformă web destinată transporturilor de mărfuri

Marian FLĂMÎNZANU-MATEIUC

Rezumat

Transporturile rutiere de marfă reprezintă un domeniu de activitate din ce în ce mai relevant și prezent în viața fiecărei persoane. Acestea constituie unul dintre motoarele economiei și implică o serie de parteneriate, contracte și aranjamente între comercianți, care dispun de marfă, și transportatori, care dețin vehicule capabile să transporte aceste mărfuri. Transporturile rutiere sunt esențiale deoarece permit deplasarea diverselor tipuri de marfă către multiple locații, ținând cont de anumite constrângeri: infrastructură, timp de livrare, dimensiuni ale mărfuii, etc.

În prezent, piața transporturilor rutiere este liberă, ceea ce înseamnă că numeroase societăți comerciale (sau chiar persoane fizice) concurează pentru a fi selectate de către comercianți pentru transportul mărfuii lor. Găsirea unor oferte avantajoase (acorduri între cele două părți) reprezintă o provocare pe care comercianții și transportatorii o întâmpină în mod regulat. Scopul unei oferte este minimizarea costurilor pentru client și maximizarea lor pentru transportator. Soluțiile existente în prezent reușesc doar parțial să automatizeze aceste probleme, astfel că, în majoritatea situațiilor, cele două părți trebuie să găsească manual camionul și marfa care să le aducă cele mai multe avantaje financiare.

Proiectul de față vine în întâmpinarea potențialilor clienți sub forma unei soluții web complete, pe care aceștia o pot folosi pentru a-și eficientiza activitatea. Proiectul include un server web de tip SpringBoot, ce urmează o arhitectură RESTful bazată pe microservicii, utilizate pentru a gestiona diverse componente ale aplicației: autentificarea, adăugarea de conturi noi, adăugarea de mărfuri și camioane, efectuarea de plăți. Acest server implementează, într-unul dintre microserviciile sale, un algoritm evolutiv ce este utilizat pentru a găsi în mod automat cele mai potrivite asocieri între camioane și mărfuri pentru clienți. De asemenea, s-a dezvoltat o aplicație client web pe care transportatorii și comercianții o pot accesa pentru a utiliza metodele expuse în cadrul serverului web, comunicația dintre client și server realizându-se utilizând metode HTTP. Totodată, o bază de date relațională, MariaDB, este utilizată pentru a înmagazina date despre clienți, camioane, mărfuri și achizițiile acestora.

Capitolul 1. Introducere

Transporturile rutiere de marfă reprezintă o problemă de actualitate cu care se confruntă numeroși transportatori și clienți. În contextul creșterii costurilor de exploatare pentru furnizorii de servicii de transport (carburanți, asigurări auto, taxe), necesitatea eficientizării activității de transport de marfă reprezintă un scop pe care cei implicați își doresc cu tot dinadinsul să îl îndeplinească.

Pentru a aborda această problemă complexă, companiile de transport iau în calcul mai multe strategii: alegerea rutelor cele mai profitabile, gestionarea digitalizată a flotelor auto, optimizarea curselor și, implicit, a costurilor de transport. În ceea ce privește clienții, aceștia încearcă să găsească cele mai bune oferte primite din partea transportatorilor, încercând în același timp să găsească un echilibru între cost, timp de livrare, condiții de livrare și.a.m.d.

Putem afirma, deci, că găsirea unui echilibru între profitul realizat de un transportator, costurile pe care un client le suportă și ceilalți parametri ce intervin în cadrul unui transport de marfă este un proces greu de realizat în mod manual. În cadrul acestui proces, pot apărea deficiențe de comunicare, pot avea loc erori în analizarea corectă și pragmatică a ofertelor, pot fi alese oferte mai puțin satisfăcătoare.

În momentul de față, nu există o platformă standard ce oferă servicii automate de generare a unor oferte pentru clienți. Ambele părți sunt nevoie să caute sau să accepte cele mai bune oferte pe mai multe site-uri de profil, fără a putea automatiza acest proces sau fără a cunoaște care este cea mai rentabilă ofertă din punct de vedere economic. În acest sens, proiectul de față vine în întâmpinarea potențialilor clienți cu o platformă web completă, ușor de utilizat și de dezvoltat, ce își propune să ajute părțile interesate în a-și desfășura în mod eficient activitatea și în a le oferi o soluție tehnică completă: gestiunea utilizatorilor, a mărfurilor, a camioanelor, a ofertelor și a plășilor ce pot fi efectuate.

Sistemele distribuite sunt rețele de calculatoare independente ce colaborează pentru a îndeplini un scop comun, oferind utilizatorilor percepția unui sistem unic. Acestea sunt construite pentru a putea fi scalate și pentru a avea o toleranță ridicată la erori (prin faptul că un astfel de sistem poate fi doar parțial afectat).

O arhitectură bazată pe servicii (abreviat SOA) reprezintă o metodă ce utilizează componente software numite servicii pentru a realiza o aplicație de tip business. Prin combinarea acestor servicii, dezvoltatorii au posibilitatea de a construi software-uri ce formează un tot-unitar pentru a îndeplini sarcini complexe. [1]

Arhitectura bazată pe microservicii, asemănător SOA, este formată din mai multe componente software, denumite microservicii, ce sunt concepute pentru a fi slab cuplate (nu depind în mod critic unul de celălalt), pentru a fi îndeplini scopuri specializate și clar definite, pentru a putea fi reutilizate în cadrul unor alte aplicații. O diferență față de SOA constă în modul prin care se poate comunica cu serviciile/microserviciile. Arhitecturile bazate pe servicii utilizează, în cele mai multe cazuri, protocolul de comunicare SOAP (Single Object Access Protocol) și fișiere de tip XML, în timp ce microserviciile utilizează protocoale de comunicare precum HTTP sau cozi de mesaje. Microserviciile sunt concepute pentru a rula într-un mediu de tip cloud, sunt construite pentru a fi operate în containere (de exemplu, containere Docker), avantajele utilizării acestor tehnologii fiind capacitatea de a scala pe orizontală microservicii în funcție de cerere. [2]

Protocolul SOAP (Simple Object Access Protocol) este un protocol standardizat realizat de către Microsoft ce utilizează în mod exclusiv fișiere de tip XML pentru comunicarea dintre client și servicii. Acesta a fost creat pentru a înlocui protocoale mai vechi de comunicație, precum CORBA (Common Object Request Broker Architecture). Cu trecerea timpului,

dezavantajele acestui protocol au devenit vizibile: SOAP este un protocol mai complicat din punct de vedere tehnic, datorită specificațiilor și a extensiilor acestuia. De asemenea, dezvoltatorul trebuie să utilizeze fișiere de tip XML ce sunt mai greu de scris și care sunt mai susceptibile în fața erorilor, iar utilizarea unor fișiere XML complexe duce la o latență mai mare asociată transmiterii acestor fișiere prin rețea. [3]

O alternativă modernă la SOAP o reprezintă arhitectura REST („REpresentational State Transfer”). Aceasta aduce o serie de îmbunătățiri comparativ cu SOAP:

- Comportamentul „fără stare”: paradigmă ce presupune faptul că serverul și clientul nu cunosc informații unul despre celălalt. Orice procesare în cadrul unui microserviciu se produce independent: serverul primește de la client o cerere completă pe care trebuie să o proceseze;

- Separarea dintre client și server: în cadrul acestei paradigmă, cele două entități sunt complet independente și se pot modifica fără a afecta comportamentul celeilalte. Clientul are drept scop afișarea către utilizator a unei interfețe. Prin intermediul acesteia, se comunică cu serverul;

- Interfața de comunicare: REST utilizează protocolul HTTP 1.1 pentru a realiza comunicarea între client și server și pentru a realiza operațiuni de tip CRUD (Creare - Create, Citire - Read, Actualizare - Update, Ștergere - Delete);

- Arhitectura orientată pe resursă: în cadrul protocolului REST, orice mesaj transmis este interpretat ca o resursă, fie că vorbim despre un fișier de tip JSON sau un text simplu sau o cerere HTTP.

Toate aceste avantaje fac REST să fie cel mai utilizat API (Application Protocol Interface) în momentul de față, conform Postman (platformă ce facilitează dezvoltarea și testarea de API-uri. [4]

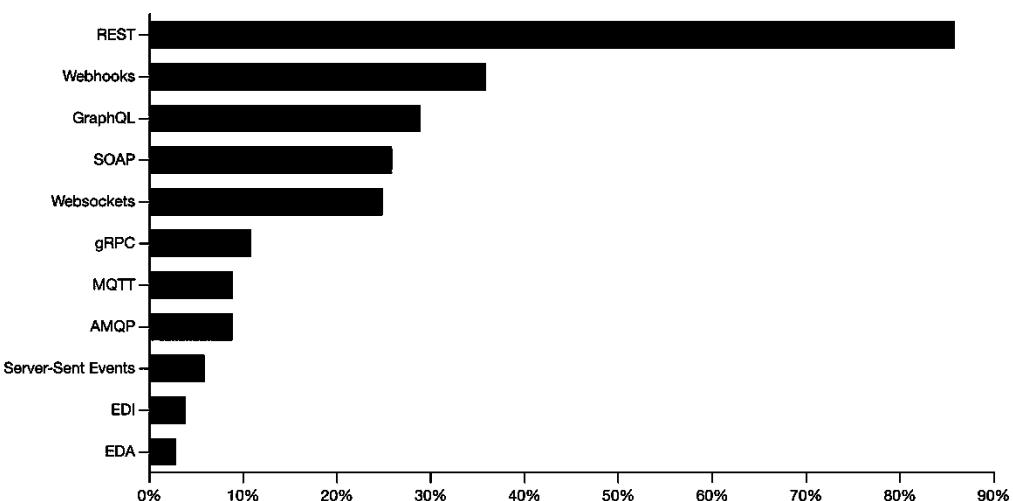


Figura 1. Cele mai utilizate arhitecturi API în 2023, conform Postman

Capitolul 2. Fundamentarea teoretică și documentarea bibliografică

Managementul transportului de mărfuri reprezintă o problemă actuală atât pentru furnizorii de produse și servicii, cât și pentru transportatorii. Întrucât cererea din partea clienților, precum și costurile asociate exploatarii vehiculelor de transport sunt în creștere, este esențială implementarea unei burse de transport mărfuri automatizată. Prin intermediul acesteia, furnizorii și transportatorii pot depune cereri și oferte, facilitând astfel desfășurarea activității de transport. În acest context, proiectul dat vine în sprijinul clienților săi prin automatizarea acestor procese, cu scopul de a eficientiza și optimiza întreaga activitate. Proiectul își propune să ofere o platformă web intuitivă și ușor de utilizat, prin intermediul căreia să fie depuse oferte din partea celor două părți (furnizori și transportatori). Componenta de tip broker a aplicației va analiza datele disponibile și va genera cea mai bună ofertă din punct de vedere al costurilor de transport și al timpilor de livrare a mărfurilor, pe care ambele părți o pot accepta sau, eventual, respinge. De asemenea, oferă un serviciu de plăți și un serviciu de notificări, prin intermediul căruia utilizatorii vor fi înștiințați de progresul cererii/ofertei.

Soluțiile existente ce realizează interacțiunea dintre transportatori și comercianți se bazează, în principal, pe selectarea manuală a ofertelor. Utilizatorii sunt invitați în a-și crea conturi de utilizator, apoi cade în sarcina acestora să găsească cea mai bună ofertă. Acest lucru atrage după sine o serie de consecințe:

- Există posibilitatea de a rata o ofertă care să fie mai profitabilă/mai economică. Există ambiguități în modul de prezentare a gabaritului mărfurilor sau ale camioanelor: există oferte în care dimensiunile sunt trecute în alte măsuri decât cele din Sistemul Internațional (de exemplu, milimetri, tone);

- Pot exista deficiențe în estimarea profitului sau a costului de transport. Un transportator trebuie să ia în calcul costul per kilometru al unei curse, să ia în calcul deplasarea până la punctul de preluare a mărfui și să calculeze manual un preț per kilometru pentru un client pentru a putea calcula profitul.

Este important de precizat faptul că nu există o soluție standardizată de plată în cadrul acestor aplicații. Unele oferte solicită plata în numerar la finalul transportului, ceea ce poate fi dezavantajos pentru unii transportatori: pot avea loc fraude, evidența încasărilor este mai dificilă decât în cazul unor plăți digitalizate. De asemenea, din punct de vedere al interacțiunii cu clienții, unele platforme de acest tip sunt complicate din punct de vedere vizual și pot produce dificultăți în fața unor clienți ce nu sunt obișnuiți cu utilizarea unor astfel de soluții IT.

Având în vedere aceste deficiențe, proiectul de față încearcă să ușureze efortul potențialilor clienți prin automatizarea procesului de brokering dintre clienți și transportatori, prin oferirea unei soluții standardizate și digitalizate de plată și prin oferirea unei interfețe de utilizator (UI) ușor de înțeles și de utilizat.

Spre deosebire de soluțiile deja existente, proiectul de față înregistrează anumite neajunsuri:

- Soluțiile existente prezintă mai multă flexibilitate: se pot introduce cuvinte cheie pentru a se realiza căutări filtrate de camioane sau mărfuri, se pot introduce cereri pentru colaborări pe termen lung între comercianți și transportatori;

- Soluțiile existente implementeză mecanisme interne de chat pentru o comunicare securizată și supravegheată de către deținătorii platformei (prin această metodă se pot combate eventualele fraude);

- Unele soluții pun la dispoziția clienților simulatoare mai avansate de cost: un transportator poate introduce consumul și punctele de încărcare/descărcare ale mărfui, iar simulatorul generează un cost estimativ. În cazul platformei implementate în proiectul de față, cade în responsabilitatea transportatorului de a calcula costul (în euro) pe kilometru.

Din punct de vedere tehnic, soluțiile existente de brokering de mărfuri constituie aproape

exclusiv platforme web ce utilizează arhitectura client-server. Acestea se pretează cel mai bine unei astfel de teme, deoarece aplicațiile web se pot vizualiza și utiliză de pe orice platformă hardware și orice sistem de operare cât timp acestea rulează un browser web modern. O astfel de platformă poate fi utilizată atât de pe PC, de către dispeceri, angajați, proprietari ai unor firme de transport sau de către persoane private, însă, grație tehnologiilor web existente (de exemplu, CSS - Cascading Style Sheets), o astfel de aplicație se poate scala pentru a putea fi rulată cu succes și de pe un dispozitiv mobil: smartphone, tabletă.

Paradigma client-server presupune o separare a celor două componente astfel acestea să existe și să se poată extinde independent unele de altele.

2.1. Analiza soluțiilor existente pentru dezvoltarea de aplicații server

La nivel de server, există diverse framework-uri care permit implementarea unei astfel de componente: SpringBoot, Flask, Django, FastAPI, Node.js, Microsoft ASP.NET.

ASP.NET este un framework al Microsoft folosit pentru a produce pagini web și aplicații dinamice. Este o soluție tehnică open-source (deschisă publicului larg pentru vizualizare și îmbunătățire) și cross-platform (poate fi dezvoltată și rulată pe sistemele de operare Windows, GNU/Linux și MacOS). ASP.NET suportă framework-uri moderne și oferă performanțe bune, însă este mai greu de obținut și de utilizat (este nevoie de anumite medii de programare și sisteme de operare alternative Windows-ului pentru a putea fi folosit), necesită mai multe resurse pentru a putea fi utilizat și dispune de mai puține biblioteci externe care pot fi utilizate. [5]

Node.js este un runtime JavaScript bazat pe motorul JavaScript al Google Chrome. Ca mod de funcționare, una dintre caracteristicile sale principale o reprezintă comportamentul asincron al tratării cererilor (o aplicație Node.js utilizează un singur fir de execuție, însă cererile primite sunt tratate în mod asincron, în fundal). Node.js este un runtime rapid, conceput pentru a implementa microservicii și dispune de un număr impresionant de biblioteci externe, însă randamentul acestuia scade în momentul în care este nevoie de executarea unor operații ce solicită procesorul computerului pe care rulează. De asemenea, din cauza naturii sale asincrone, codul poate fi dificil de urmărit și de menținut, deoarece se poate pierde evidența și logica apelurilor pe care Node.js le tratează separat de firul principal de execuție. [6]

FastAPI este un framework web modern conceput pentru construcția de API-uri de tip RESTful. Bazat pe Python, acest framework oferă performanțe foarte bune comparativ cu alte soluții bazate pe Python (e.g. Django, Flask), suportă cereri asincrone și prezintă o sintaxă clară și concisă. Dezavantajul acestei tehnologii constă în faptul că este o soluție tehnică relativ nouă, ceea ce înseamnă că găsirea unor soluții la anumite probleme particulare poate fi o provocare pentru un dezvoltator aflat la început de drum în dezvoltarea unui server web. [7]

Django este un framework bazat pe limbajul Python ce își propune să faciliteze procesul de dezvoltare a unui server web prin multitudinea de caracteristici pe care le aduce în mod implicit: un sistem pentru autentificare, un ORM (Object-Relational Mapping) ce poate fi accesat printr-un API și o interfață de administrare ce poate fi folosită pentru managementul aplicației. De asemenea, Django dispune de opțiuni pentru scalare atât pe verticală, cât și pe orizontală. Dezavantajele acestui framework constau în faptul că aplicațiile dezvoltate sunt în general monoliți, modularitatea unui proiect și cuplarea slabă a serviciilor sunt mai greu de atins și faptul că dezvoltatorul trebuie să urmeze anumite practici de scriere a codului. [8]

SpringBoot este un framework open-source menit să înlocuiască Java Enterprise Edition, o extensie a limbajului Java pentru dezvoltarea de aplicații enterprise (sisteme distribuite, servicii web). SpringBoot este un framework foarte popular în rândul dezvoltatorilor de servicii web, este un produs software multi-platformă ce poate fi folosit pentru a devolta o multitudine de produse: microservicii, aplicații asincrone, aplicații MVC (Model-View-Controller). SpringBoot permite injectarea dependențelor, facilitează scalarea fișierelor executabile rezultate, facilitează

adăugarea de biblioteci și funcționalități externe prin gestionarele de proiect (de exemplu, Maven, Gradle). Printre dezavantajele SpringBoot, se pot menționa timpul de lansare în execuție mai mare și faptul că SpringBoot nu se pretează pentru proiectele de mari dimensiuni.

2.2. Analiza soluțiilor existente pentru dezvoltarea de aplicații client

În ceea ce privește componenta de client, există mai multe framework-uri ce facilitează crearea acestei componente software, precum Angular, VueJS sau React.

Angular este, din punct de vedere tehnic, unul dintre cele mai puternice framework-uri de acest gen de pe piață. Dezvoltat de Google, Angular este un framework ce permite crearea de aplicații web dinamice (Single Page Applications - SPA), are suport multi-platformă, utilizează TypeScript (îmbunătățire a limbajului de programare JavaScript) și încurajează arhitectura MVC. Cu toate acestea, Angular este un framework dificil de utilizat și de înțeles. Arhitectura și capabilitățile sale extinse fac Angular să fie utilizat în cadrul aplicațiilor și proiectelor complexe. [10]

VueJS este un framework JavaScript open-source folosit pentru dezvoltarea aplicațiilor web cât și mobile. Este un framework, performant, simplu de utilizat și care încurajează programarea reactivă (paradigmă ce presupune utilizarea unor funcții asincrone pentru a actualiza un conținut static). În schimb, Vue nu este un framework foarte des întâlnit, ceea ce poate face depanarea codului să fie mai complicată din lipsă de resurse relevante. De asemenea, Vue nu dispune de un număr la fel de mare de biblioteci externe precum alternativele sale, ceea ce îl face mai puțin atractiv. [11]

React.js este cel mai popular framework în momentul de față. Bazat pe JavaScript, React propune o arhitectură bazată pe componente ce pot fi create, depanate independent una de cealaltă și pot fi reutilizate în cadrul aplicației. React ajută la dezvoltarea de Single Page Applications și vine în întâmpinarea programatorului prin utilizarea unui DOM virtual (Document Object Model - reprezintă structura și conținutul unui document web). Această tehnică ajută la îmbunătățirea performanței aplicației deoarece modificările în cadrul unei pagini web au loc mai întâi în cadrul acestui DOM virtual, în memorie, ele aplicându-se DOM-ului efectiv doar atunci când framework-ul consideră că este necesar. Ca dezavantaje, React este supus unui ciclu rapid de modificare, ceea ce înseamnă că un dezvoltator trebuie să fie la curent și să își adapteze proiectul în momentul în care au loc modificări, ceea ce dăunează suportului tehnic pe termen lung. React nu implementează la fel de multe caracteristici precum alternativele acestuia, însă ușurința cu care acest framework se poate învăța și utiliza îl face să fie foarte utilizat la scară largă. [12]

2.3. Analiza soluțiilor existente pentru dezvoltarea unei baze de date

O platformă web de tipul celei implementate necesită în mod obligatoriu existența unei metode/soluții de stocare eficientă și securizată a datelor. În momentul de față, există mai multe tipuri de baze de date: relaționale (tip SQL - Structured Query Language), ne-relaționale (tip NoSQL - Not Only SQL, sau Non-SQL), bazate de grafuri, bazate pe perechi cheie-valoare, orientate-obiect și altele.

Bazele de date de tip SQL sunt folosite pentru a manipula date din baze de date de tip relational. Utilizând comenzi ce se numesc interogări, utilizatorul/dezvoltatorul poate să realizeze în mod facil diverse operațiuni asupra unei baze de date: adăugări, ștergeri, modificări de date. Acest tip de baze de date folosește colecții denumite tabele, ce conțin atrbute (desfășurate pe coloane) și câmpuri (desfășurate pe linii). În cadrul acestui mod organizat de înmagazinare a datelor există diverse tipuri de constrângeri, precum cheile primare (atribute

unice folosite pentru identificarea datelor) sau chei străine (referințe către chei private din alte tabele). În acest mod, în cadrul bazelor de date de tip relațional se poate implementa cu ușurință modelul entitate-relație. Aceasta presupune existența unor relații bine stabilite („unu la unu”, „mai mulți la unu”, „mai mulți la mai mulți”) între tabelele unei baze de date. Prin urmare, putem concluziona faptul că bazele de date de tip SQL se pretează pentru înmagazinarea „convențională” a datelor - definirea clară a datelor folosite și accentul asupra persistenței acestora. [13]

MariaDB este un server de tip SQL open-source rapid, ușor-scalabil și robust ce cuprinde un număr mare de aplicații, de la pagini de tip web până la aplicații de banking. Este o soluție compatibilă cu MySQL însă, spre deosebire de acesta, cuprinde îmbunătățiri în ceea ce privește interogarea datelor, compresia acestora, scalarea pe mai multe servere pentru a obține performanțe superioare în condiții de încărcare ridicată. [14]

Bazele de date de tip ne-relațional au apărut pe piață la începutul anilor 2000. Odată cu scăderea costului unităților de stocare, a apărut dorința de a se renunța la modelele tradiționale, complexe și greu de gestionat ale SQL în detrimentul unor baze de date ce înmagazinează date în format structurat, semi-structurat sau nestructurat.

MongoDB este un exemplu de bază de date de tip ne-relațional. Este optimizat pentru performanță, cloud-computing, dispune de numeroase funcționalități pentru a realiza operațiuni CRUD simple, cât și interogări complexe. Dezavantajul acestui mod de organizare al datelor constă în faptul că nu se poate asigura în totalitate principiul ACID (Atomicitate, Consistență, Izolare și Durabilitate). Prin urmare, bazele de date de acest tip urmăresc performanța și simplitatea în detrimentul consistenței. [15]

Relational				Non-Relational
student				student.json file body:
id	name	surname	age	
1	John	Brown	19	[
2	Emma	Carly	23	{ "id": 1, "name": "John", "surname": "Brown", "age": 19 }, { "id": 2, "name": "Emma", "surname": "Carly", "age": 23 }]

Figura 2. Diferența de organizare dintre o bază de date de tip SQL și una NoSQL
(în exemplul dat, MongoDB)¹

¹ Sursă imagine:

<https://codefinity.com/courses/v2/5ac24d9d-4a16-45b3-8856-07dec028c5e9/c9717913-3062-46d6-b0ef-164aca862b29/64995190-c55f-4482-a71f-ec51d6ae5b80>

Bazele de date de bazate pe grafuri înmagazinează datele utilizând noduri și muchii. Nodurile stochează proprietățile obiectelor, în timp ce muchiile stochează relațiile dintre entități. Această tehnologie permite construirea unei baze date flexibilă, scalabilă și eficientă, însă este mai dificil de utilizat și se pretează mai bine în aplicații ce pun accentul de relațiile dintre entități/obiecte (de exemplu, rețele de socializare, motoare de recomandare).

Bazele de date de tip cheie-valoare sunt baze de date de tip ne-relațional ce salvează datele într-o colecție de perechi cheie-valoare, unde cheile se folosesc drept identificatori unici. O astfel de bază de date este Redis (Remote Dictionary Server), ce își înmagazinează informațiile în memorie și folosește un mecanism de caching pentru persistența datelor. Deși performanțele acestei baze de date sunt foarte bune, amprenta asupra memoriei este ridicată, iar o eroare de sistem/pierdere de energie/reporningă forțată poate duce la pierdere de informații în lipsa unor snapshot-uri salvate pe disc. [16]

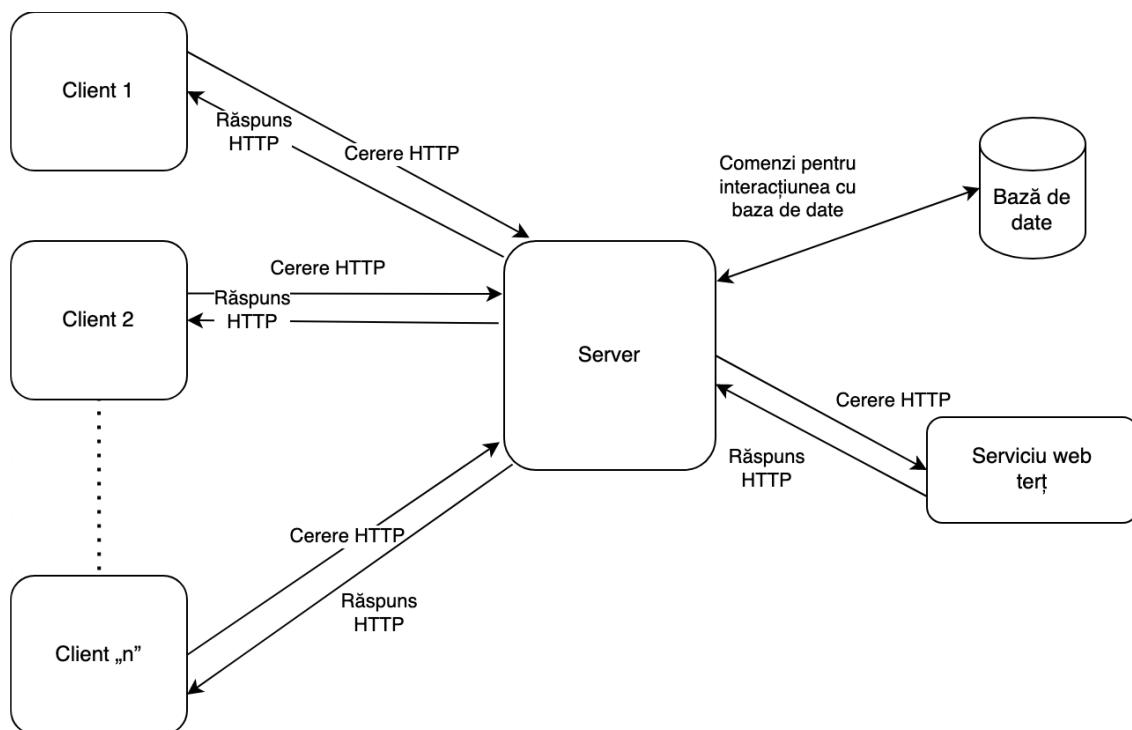


Figura 3. Schemă bloc ce evidențiază comunicarea dintre client, server, bază de date și servicii externe.

2.4. Analiza algoritmilor ce pot fi utilizati pentru obtinerea unei solutii optime

O platformă web dedicată transporturilor de mărfuri conține, la nivel fundamental, două tipuri de obiecte: mărfuri și camioane. Obiectivul acestui proiect este să găsească în mod complet automat cele mai bune potriviri dintre aceste două entități, astfel încât să se urmărească maximizarea profitului pentru transportatorii și minimizarea costurilor pentru un client. Prin urmare, se impune cercetarea/găsirea unui algoritm care să realizeze această optimizare.

2.4.1. Abordarea programării liniare

O metodă simplă de a genera perechi între camioane și mărfuri constă în soluția programării liniare. Problema poate fi exprimată prin formula următoare:

$s = \max (\sum_{i=1}^{nt} \sum_{j=1}^{nc} p(i,j) * x(i,j))$, unde nt reprezintă numărul de transportatori, nc reprezintă numărul de clienți, $p(i,j)$ reprezintă profitul transportatorului i îndeplinind cursa comerciantului j, iar $x(i,j)$ este o valoare binară: $x(i,j) = 1$ dacă transportatorul i va onora comanda comerciantului j și 0 în rest.

De asemenea, se impun următoarele condiții:

$$\sum_{i=1}^{nt} x(i,j) = 1, \forall j \in \{1, \dots, nc\}$$

$x(i,j) \in \{0,1\}, \forall i \in \{1, \dots, nt\}, \forall j \in \{1, \dots, nc\}$: Prin aceste condiții, verificăm ca suma tuturor valorilor lui x să fie egală cu 1, pentru că un transportator să fie atribuit unui singur client și

$$\sum_{i=1}^{nc} w(i,j) * x(i,j) \leq m(i,j), \text{ unde}$$

$w(i,j)$ reprezintă greutatea mărfii transportată de către transportatorul i pentru clientul j și $m(i,j)$ reprezintă masa camionului ce transportă marfa pentru clientul j.

Unul dintre dezavantajele majore ale acestui algoritm constă în faptul că timpul de execuție al acestuia se poate înrăutăti semnificativ pentru un număr mare de camioane și mărfuri înrolate în sistem. [17]

2.4.2. Optimizare de tip roi de particule

Aceasta este un algoritm orientat înspre optimizarea soluțiilor și se inspiră din comportamentul stolurilor de păsări sau a bancurilor de pești.

În cadrul acestui algoritm, o entitate poartă denumirea de particulă și prezintă următoarele caracteristici:

x_i : poziția curentă

v_i : viteza curentă

y_i : cea mai bună poziție personală

\hat{y}_i : cea mai bună poziție a vecinătății

În cadrul problemei de față, algoritmul poate cuprinde următorii 5 pași:

Inițializarea:

Se alege un număr de particule. Pentru fiecare particulă, se inițializează aleatoriu pozițiile x_i . În cadrul problemei de față, poziția xi poate consta într-un vector de dimensiune „nc” de valori binare, fiecare element al vectorului având valoarea 1 dacă transportatorul i onorează comanda clientului j sau 0 în caz contrar. În cele din urmă, se aleg în mod aleatoriu vitezele pentru fiecare particulă în parte.

Stabilirea funcției de fitness/funcției obiectiv:

Asemănător soluției programării liniare, funcția de fitness se poate defini ca

$\sum_{i=1}^{nt} \sum_{j=1}^{nc} p(i,j) * x(i,j))$, unde nt reprezintă numărul de transportatori, nc reprezintă numărul de clienți, $p(i,j)$ reprezintă profitul transportatorului i îndeplinind cursa comer-

ciantului j , iar $x(i,j)$ este o valoare binară: $x(i,j) = 1$, dacă transportatorul i va onora comanda comerciantului j și 0 în rest.

Ajustarea:

- Pentru fiecare particulă, se evaluează funcția obiectiv a particulei $f(x_i)$ și se actualizează optimul personal.

În cazul unei probleme de minimizare,

$$y_i(t+1) = x_i(t+1), \text{ dacă } f(x_i(t+1)) < f(y_i(t)) \text{ și } y_i(t) \text{ în caz contrar.}$$

- Se calculează optimul social al vecinătății:

$$\hat{y}_i(t) = \operatorname{argmin}(f(y_1(t), y_1(t), \dots, y_s(t)))$$

- Pentru fiecare particulă, se actualizează viteza:

$$v_{i,j}(t) = w * v_{i,j}(t) + c_1 * r_{1,j}(t) * (y_{i,j}(t) - x_{i,j}(t)) + c_2 * r_{2,j}(t) * (\hat{y}_i(t) - x_{i,j}(t)),$$

unde w este ponderea inerției, $c1$ și $c2$ sunt constante de acceleratie și $r1$ și $r2$ sunt numere aleatorii cuprinse între 0 și 1.

- Se actualizează poziția curentă:

$$x_i(t+1) = x_i(t) + v_i(t+1)$$

Iterarea

Se repetă acest procedeu pentru un număr finit de iterații sau până când se atinge un criteriu de convergență.

Acest algoritm poate prezenta următoarele deficiențe:

- Convergență prematură: PSO poate să găsească rapid o soluție, însă există posibilitatea ca aceasta să nu se mai îmbunătățească. De asemenea, o particulă poate converge la un punct dintre optimul personal și optimul social, care poate să nu fie nici măcar optim local al problemei.

- Performanțele depind de problemă: parametrii trebuie aleși pentru fiecare problemă în parte. În acest sens, c_1 și c_2 , numiți și „parametri de încredere”, au în general valoarea 2, iar ponderea inerției, w , are valori cuprinse între 0.9 și 0.4. Această valoare poate descrește liniar sau poate să aibă o valoare fixă pe tot parcursul algoritmului. [18]

2.4.3. Metoda/algoritmul evolutiv

Având în vedere faptul că tema proiectului poate avea dimensiuni mari (numeroase camionae și mărfuri ce trebuie alocate), un tip de algoritm ce se poate implementa facil și care poate da rezultate bune este algoritmul evolutiv. Analog vieții reale, acest algoritm presupune existența unor cromozomi ce se pot încrucișa și muta (asemănător unui ADN), iar în urma unei simulări a rezultatelor, procedura poate fi reluată, păstrând cel mai bun dintre cromozomii iterației anterioare. În cazul temei de față, un algoritm evolutiv poate conține următoarele codificări și etape de lucru:

Codificarea

Pentru a putea utiliza date reale în cadrul unui algoritm, este nevoie de o codificare a informației. Pentru acest proiect, se pot folosi un cromozom de forma $c = [id_1, id_2, \dots, id_x]$, unde c are dimensiunea egală cu numărul de mărfuri disponibile, iar id_i reprezintă id-ul unui camion din baza de date.

Selectia

Se generează o listă cu n cromozomi definiți anterior. Din această listă, se aleg la întâmplare doi cromozomi.

Încrucișarea

Acest proces are loc cu o probabilitate pc și constă într-o încrucișare de tip uniform. Aceasta presupune crearea unui singur cromozom combinat din cei doi aleși în pasul anterior. Se stabilește o probabilitate aleatorie, pe baza căreia se generează noul cromozom. În 10% dintre cazuri, se alege un cromozom definit în pasul anterior și se continuă spre pasul următor. Această metodă se utilizează pentru a introduce o diversitate genetică în cadrul algoritmului (pot fi descoperite valori ce nu au fost introduse în cromozomii existenți).

Mutarea

Acest proces are loc cu probabilitatea pm și presupune resetarea unor valori aleatorii din cromozomul obținut anterior la valori din lista de id-uri de camioane existente.

Simularea

Simularea are un caracter euristic și presupune calcularea, pe paza perechilor mărfă-camion, a costurilor de transport pentru un comerciant, pe baza prețului stabilit de către un transportator. Acesta din urmă va scade din profitul total costul de deplasare până la locația de preluare a mărfurii. În această etapă, se impun constrângerile aferente unui transport (respectarea greutății, a volumului și obținerea unui profit pentru transportator).

Acest algoritm cuprinde un număr de na epoci per iteratăie. În cadrul unei iterării, se determină profitul maxim pentru un camion sau costul minim pentru un client. Algoritmul se finalizează când sunt preluate toate mărfurile, toate camioanele sau până când un contorul iterărilor ajunge la valoarea 5.

În general, parametrul pc are valori mari, în timp ce parametrul pm are valori mici (în cazul de față, $pc = 0.9$ și $pm = 0.05$). De asemenea, $n = 100$ de cromozomi și $na = 100$ de epoci. [19]

Algoritmii genetici prezintă, la rândul lor, anumite neajunsuri: pot converge într-un punct de optim local, soluția finală generată nu este întotdeauna cea mai eficientă soluție posibilă, iar păstrarea unei funcții de fitness nemodificate poate împiedica convergerea algoritmului înspre o soluție mai bună. Cu toate acestea, algoritmii genetici au capacitatea de a calcula relativ rapid soluții suficiente de bune pentru o anumită problemă. În cazul particular al temei de față, un astfel de algoritm poate calcula rapid distanțe și diverse costuri, verificând simultan constrângerile.

Capitolul 3. Proiectarea aplicației

3.1. Analiza platformei hardware

Dintr-o perspectivă a numărului de utilizatori, MacOS reprezintă cea mai populară alternativă a sistemului de operare Microsoft Windows, urmat de sistemele de operare din gama Linux. Cu toate acestea, sistemul de operare al companiei Apple reprezintă un produs software complet și coerent, ce deține facilități puternice pentru a putea dezvolta și rula aplicația propusă.

Din punct de vedere software, MacOS este un sistem de operare ce respectă standardul POSIX și care utilizează un nucleu denumit Darwin. Acesta din urmă este bazat pe kernel-ul XNU (abreviere de la „X is Not UNIX” - „X nu este UNIX”), care la rândul său utilizează tehnologii ale sistemului de operare FreeBSD. [20] În acest mod, MacOS poate rula aceleasi comenzi din terminal precum sistemele de operare Linux. Multe dintre bibliotecile externe, plugin-uri și.a.m.d. disponibile pe Linux se pot rula și pe sistemul de operare al Apple prin instalare directă, dacă fișierele sunt deja compilate, sau prin compilare manuală.

Din punct de vedere hardware, Apple a realizat cu succes tranziția către procesoarele bazate pe arhitectura ARM. Motivația generală ce a determinat trecerea de la arhitectura x86 oferită de către compania Intel a reprezentat-o consumul mare de energie al acestor procesoare. Începând cu anul 2015, compania Apple a realizat faptul că arhitectura x86 reprezintă o barieră fizică în drumul spre realizarea unor dispozitive eficiente energetic, fapt echivalent cu o autonomie mai mare a dispozitivelor portabile și temperaturi de lucru mai scăzute la nivel hardware.

Pentru a putea preîntâmpina problemele de compatibilitate software generate de trecerea la o arhitectură hardware nouă, Apple a implementat la nivelul SOC-urilor ARM unele caracteristici menite să păstreze o compatibilitate cu arhitectura x86 (modul prin care procesorul accesează memoria, diverse instrucțiuni la nivel de procesor). Plecând de la această bază, Apple a mai implementat un translator software numit „Rosetta 2” ce permite rularea la performanțe aproape native a programelor și bibliotecilor deja existente pentru computerele ce rulează MacOS de pe un procesor bazat pe arhitectura x86_64. [21]

Având în vedere cele mai sus-menționate, se poate concluziona faptul că sistemul de operare MacOS permite dezvoltarea și utilizarea unor aplicații complexe, asemănător Linux, pe o platformă hardware foarte eficientă.

În acest sens, proiectul de față este implementat și utilizat de pe un laptop Apple MacBook Air cu un SOC Apple M1 ce dispune de 8 nuclee care pot rula la frecvență maximă de 3.2 gigahertz. Dispozitivul este echipat cu 16 gigaocetă de memorie fizică și 256 de gigaocetă de stocare. Sistemul de operare utilizat este MacOS 14 „Sonoma”. Aceasta nu prezintă nicio incompatibilitate față de sistemele de operare MacOS disponibile pentru dispozitivele ce utilizează arhitectura x86_64, prin urmare suita de medii de dezvoltare (de exemplu Visual Studio Code, IntelliJ IDEA) și dependențele externe utilizate (de exemplu Protobuf) pot funcționa fără probleme pe acest dispozitiv.

3.2. Proiectarea aplicației - idei generale

Aplicația de față își propune să implementeze următoarele componente:

- *O aplicație web de tip client*: aceasta presupune construirea unei aplicații de tip frontend care să realizeze componenta UI/UX a aplicației. Acest fapt presupune utilizarea framework-ului React.js pentru a permite utilizatorilor să se autentifice, să creeze conturi de utilizator noi, să modifice conturile existente, să adaugă mărfuri sau camioane, să vizualizeze și să trateze ofertele primite și, după caz, să efectueze plăți în cadrul aplicației.

- *O aplicație de tip server*: aceasta presupune utilizarea tehnologiei SpringBoot pentru a crea un API pe care aplicația de tip client îl va accesa utilizând cereri de tip HTTP.

Server-ul este compus din mai multe microservicii:

- *Microserviciul user_service*: gestionează autentificarea și crearea de noi conturi, precum și modificarea conturilor de utilizator existente;

- *Microserviciul gateway_service*: este utilizat pentru a realiza rutarea cererilor către microserviciile corespunzătoare, implementând un punct de intrare unic pentru toate cererile care vin din partea clientilor, verificând în același timp validitatea token-urilor atașate acestor cereri;

- *Microserviciul broker_service*: este utilizat pentru a adăuga mărfuri și camioane noi în sistem, pentru a genera automat ofertele și pentru a permite utilizatorilor să le accepte sau să le respingă;

- *Microserviciul notifications_service*: este utilizat pentru a trimite notificări prin e-mail utilizatorilor atunci când își creează conturi noi sau când statusul ofertelor primite se modifică;

- *Microserviciul payment_service*: este utilizat a oferi clientilor posibilitatea de a efectua plata către transportatorii în momentul în care ofertele primite de ambele părți sunt acceptate bilateral.

- *O bază de date de tip relational (SQL)*, care să înmagazineze informațiile platformei: transportatori, comercianți, camioane, mărfuri și relațiile dintre aceste ultime două componente. Aceasta conține tabele separate pentru fiecare dintre aceste categorii, iar accesul către informații se face utilizând framework-ul Hibernate ORM disponibil pentru limbajul de programare Java. Hibernate este un framework open-source ce permite accesul ușor către server-ul de tip MariaDB prin maparea claselor existente în codul Java către tabelele existente din baza de date.

Structura tabelelor poate fi observată în figura de mai jos:

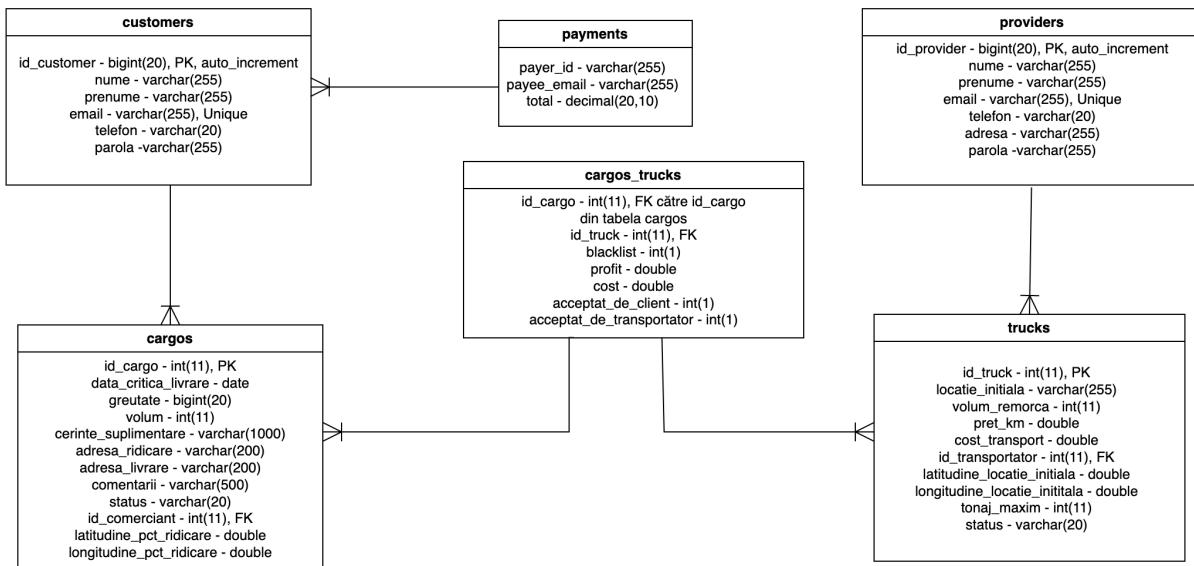


Figura 4: Tabelele aplicației și atribuțile acestora

-*Un mecanism pentru efectuare de plăți:* în contextul acestei aplicații, se va utiliza platforma PayPal Sandbox. Aceasta permite crearea unor conturi cu scop demonstrativ, pentru a putea simula efectuarea plăților de la clienți către transportatorii.

Schela generală a componentelor aplicației și interacțiunile dintre acestea poate fi observată mai jos:

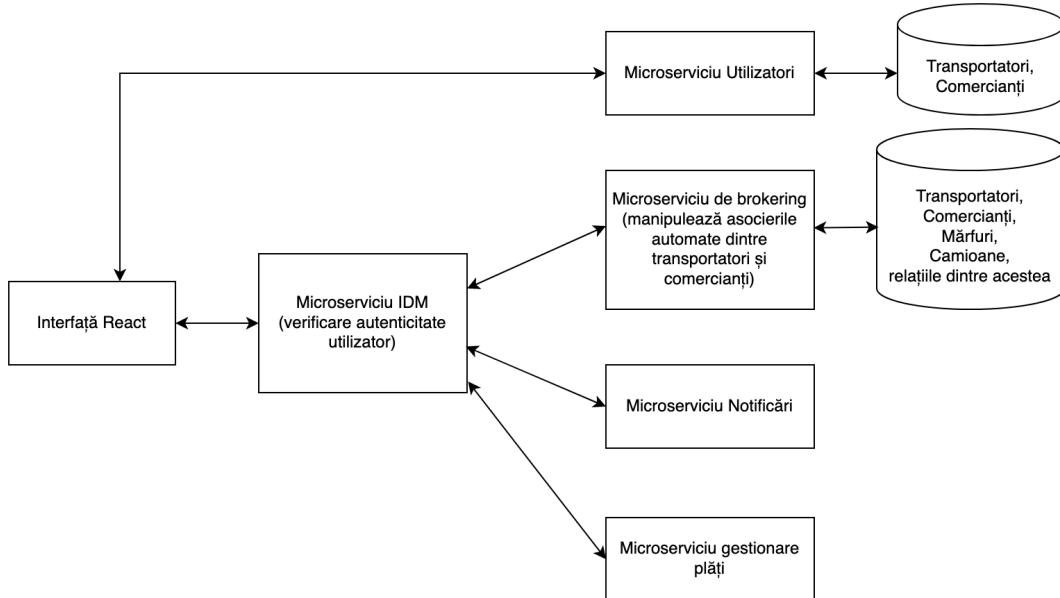


Figura 5: schema bloc a microserviciilor și a tipurilor de obiecte cu care interacționează

Din figura 5, se poate observa faptul că aplicația își propune să realizeze o arhitectură bazată pe microservicii, ceea ce presupune următoarele *avantaje*:

- *Cuplare slabă între componente:* microserviciile din cadrul aplicației sunt independente, ceea ce duce la o toleranță crescută la erori la aplicație. De exemplu, eșuarea unei dintre serviciile de brokerizing sau de notificări sau de gestionare plăți nu conduce la imposibilitatea de a utiliza aplicația, ci doar la imposibilitatea de a utiliza o anumită componentă a acesteia;

- *Modularitate:* aplicația poate fi extinsă ușor prin adăugarea unor noi servicii care să îndeplinească sarcini neimplementate în structura de față;

- *Flexibilitate:* microserviciile pot fi dezvoltate și extinse fără a le afecta pe celelalte;

- *Performanță:* microserviciile pot fi compilate, împachetate sub format executabil și lansate în execuție mult mai rapid comparativ cu aplicațiile de tip monolit deoarece prezintă dimensiuni mult mult mai reduse atunci când complexitatea aplicației crește.

Pe de altă parte, acest tip de implementare prezintă și unele *dezavantaje*:

- *Latență generată de comunicarea prin rețea:* orice interacțiune dintre client și microservicii presupune utilizarea unor cereri HTTP (de exemplu, GET, POST). Acest fapt poate presupune, în funcție de cerere, serializarea datelor la nivel de client, formularea cererii, redirecționarea acesteia de la gateway către microserviciul corespunzător, tratarea cererii, serializarea răspunsului, trimiterea acestuia înapoi către client și deserializarea lui. Acest procedeu poate duce la o suprasolicitare a microserviciilor și a rețelei prin intermediul căreia se face comunicarea, prin urmare, există posibilitatea existenței unor latențe la nivel de client din momentul în care acesta interacționează cu un element de control al aplicației și până când obține răspunsul;

- *Testarea dificilă:* în cadrul unor aplicații de dimensiuni mai mari, poate fi greu de urmărit care microserviciu nu funcționează conform așteptărilor.

Aplicația propusă rulează algoritmul automat de căutare a ofertelor ori de câte ori un camion sau o marfă este adăugată în sistem. Prin urmare, adăugarea unui număr mare de

camioane sau mărfuri poate duce la o suprasolicitare a microserviciului de brokering. Adițional, accesarea unor alte funcționalități din aplicație (acceptare/refuzare de oferte, vizualizare mărfuri/camioane) utilizează endpoint-uri din microserviciul de brokering, ceea ce face ca această componentă să fie cea mai solicitată din cadrul sistemului. Astfel, se propun următoarele soluții:

- Pentru eficiență, se utilizează un algoritm evolutiv pentru generatorul automat de oferte. În acest mod, se poate genera o ofertă care să fie satisfăcătoare pentru ambele părți, chiar dacă aceasta nu este cea mai profitabilă/mai puțin costisitoare.

3.3. Proiectarea propriu-zisă

Componenta de server a acestui proiect cuprinde 5 microservicii:

user_service

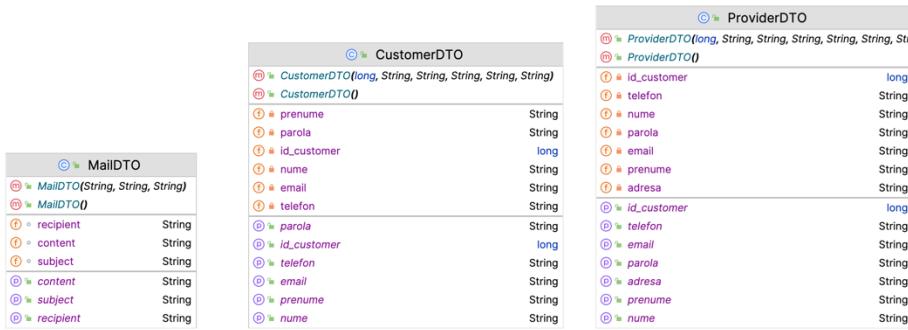


Figura 6. Clasele de tip DTO ale *user_service*

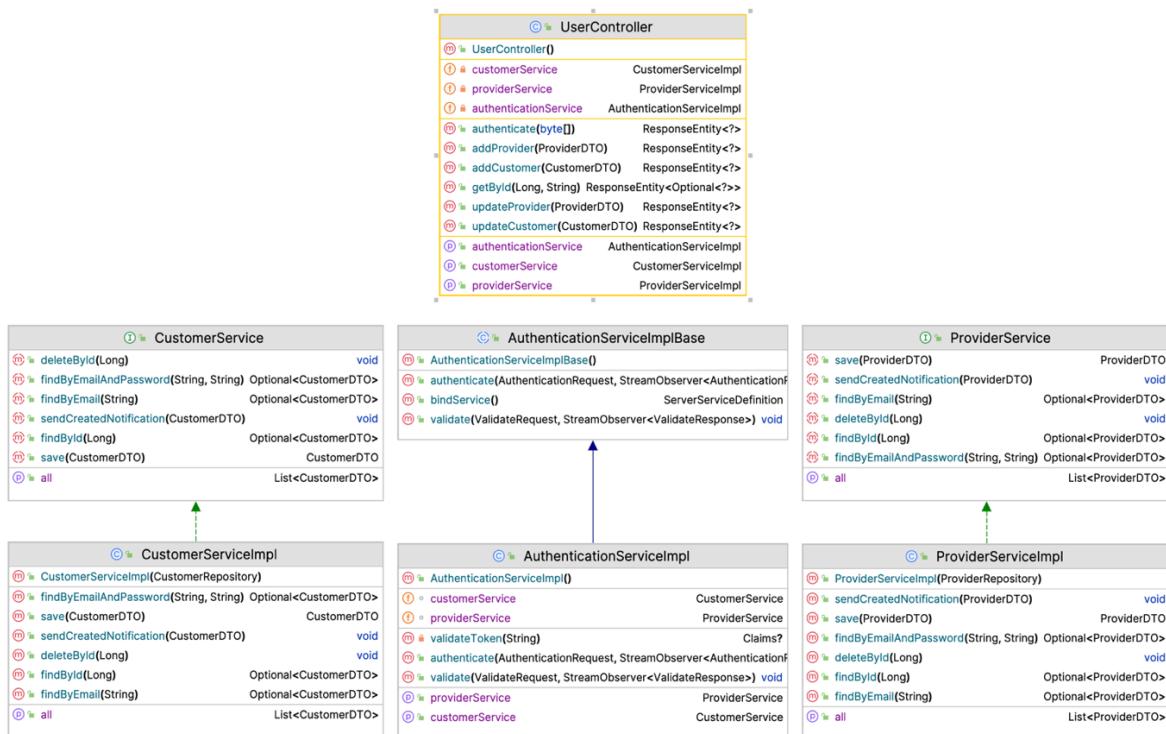


Figura 7. Controllerul și serviciile ce aparțin microserviciului *user_service*

Acum acest microserviciu conține:

Clase de tip DTO (Data Transfer Object).

Acum acestea sunt:

- *CustomerDTO* și *ProviderDTO*: utilizează framework-ul Hibernate pentru a realiza interacțiunea cu baza de date SQL și pentru a reține datele despre un comerciant, respectiv un transportator.
- *MailDTO*: este utilizată pentru a încapsula titlul, subiectul și corpul unui e-mail ce va fi transmis drept confirmare unui utilizator în momentul în care acesta își creează un cont nou.

Servicii pentru gestionarea comercianților și a transportatorilor

Acum aceste servicii, numite *CustomerService* și *ProviderService* constau în două interfețe, ce utilizează la rândul lor alte două interfețe numite *CustomerRepository* și *ProviderRepository*, care la rândul lor extind o interfață numită *CrudRepository* prin intermediul căreia se pot efectua operațiuni de tip CRUD pe tabela *customers*, respectiv *providers*.

CustomerService și *ProviderService* sunt implementate de către două clase, *CustomerServiceImpl* și *ProviderServiceImpl*. Prin intermediul acestora, se pot realiza operațiuni de salvare/actualizare a bazei de date, ștergeri după ID (cheia primară), căutări după ID, căutări după adresa de e-mail. De asemenea, există o metodă, numită *SendCreatedNotification* care apelează un endpoint al microserviciului de notificări pentru a transmite o notificare clientului atunci când își creează un cont nou.

Implementarea metodelor folosite pentru autentificare

În cadrul acestui microserviciu, se utilizează un server gRPC și tehnologia Protobuf pentru a genera un token pentru fiecare client în parte. În acest sens, se generează în mod automat o clasă ce conține metode neimplementate pentru realizarea autentificării, pe baza unui fișier de tip Protobuf deja furnizat. În clasa *AuthenticationServiceImpl*, se realizează implementarea metodelor *authenticate* și *validate* din cadrul clasei *AuthenticationServiceImplBase*.

Astfel, la apelul metodei *authenticate* se preiau datele despre utilizator (adresă de e-mail și parolă) din cererea de autentificare și, dacă utilizatorul există în baza de date, se generează un token care este returnat funcției de autentificare din controllerul microserviciului.

Metoda *validate* este utilizată pentru a verifica dacă token-ul primit ca parametru este valid.

Controller Spring pentru crearea unui endpoint

Acum acest microserviciu conține un controller, denumit *UserController*, folosit cu scopul de a oferi clientilor un endpoint API. Acesta permite autentificarea, crearea de utilizatori și actualizarea datelor acestora.

gateway_service

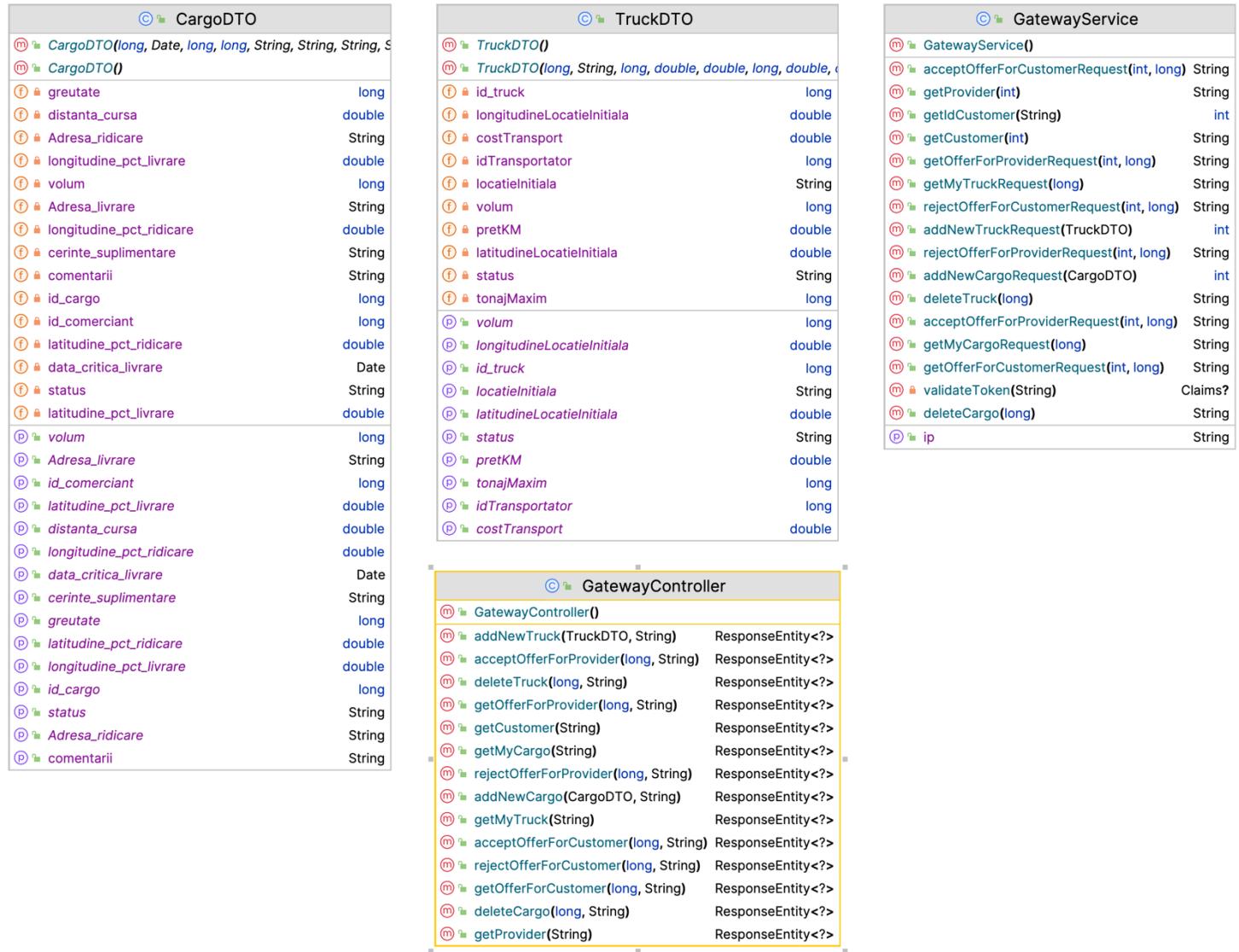


Figura 8. Clasele ce aparțin microserviciului *gateway_service*

Acest microserviciu conține:

Clase de tip DTO (Data Transfer Object).

Acestea sunt:

- *CargoDTO* și *TruckDTO*: aceste clase reprezintă încărcături/mărfuri și sunt folosite la serializarea/deserializarea datelor în contextul primirii de cereri dinspre client și redirecționarea acestora spre microserviciile corespunzătoare

Serviciu pentru redirecționarea cererilor

Acest serviciu, numit `GatewayService`, conține metode pentru a apela celelalte microservicii. Aceste metode trimit cereri către anumite endpoint-uri, cereri cărora le pot atașa obiecte serializeate. Răspunsurile primite sunt redirectionate către client.

Controller Spring pentru crearea unui endpoint

Acest microserviciu conține un controller, denumit *GatewayController*, folosit cu scopul de a oferi clienților un endpoint API. Aceasta este utilizat pentru a redirecționa cererile primite de la client către microserviciile corespunzătoare, verificând în același timp dacă token-ul primit de la client este valid.

notifications_service

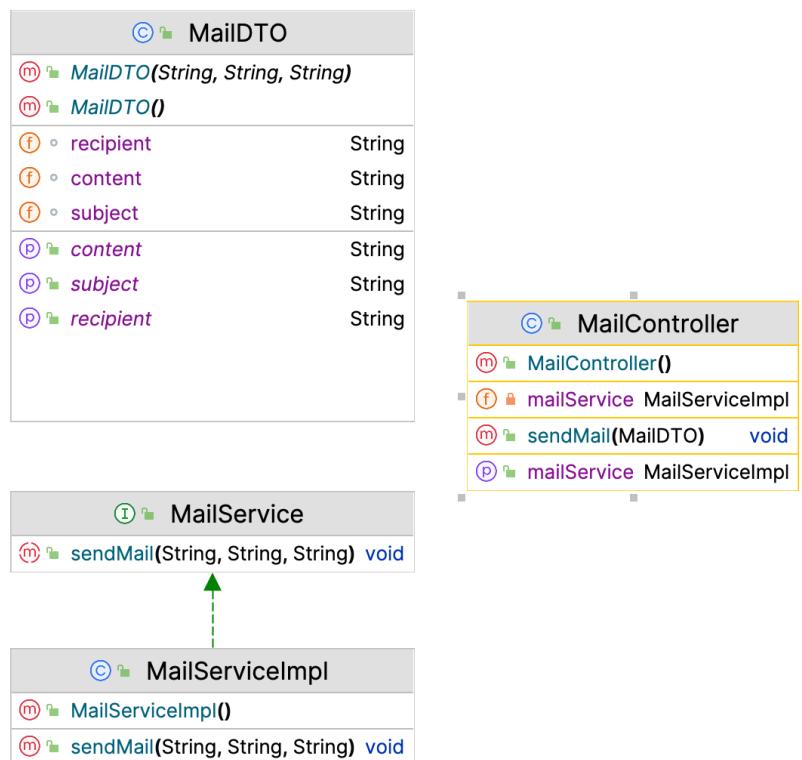


Figura 9. Clasele ce aparțin microserviciului *notifications_service*

Acces microserviciu conține:

Un DTO pentru e-mail:

Acest obiect este utilizat pentru definirea e-mail-urilor (destinatar, subiect, conținut) care vor fi transmise către clienți ori de câte ori este nevoie.

Un serviciu pentru transmiterea de e-mail-uri:

Acest serviciu conține o metodă, numită `sendMail`, ce trimit un e-mail către un anumit destinatar.

Controller Spring pentru crearea unui endpoint

Acest controller este utilizat pentru a apela metoda `sendMail`.

payment_service

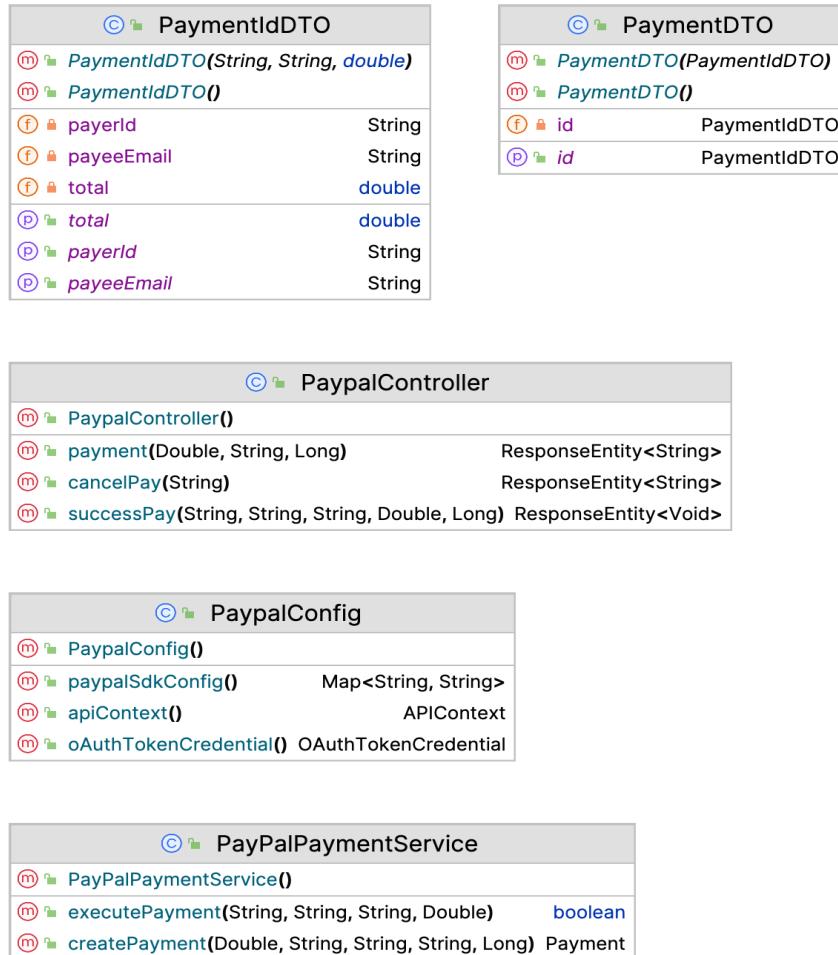


Figura 10. Clasele ce aparțin microserviciului *user_service*

Acum acest microserviciu conține:

Clase de tip DTO pentru definirea unei tranzacții:

Există două clase în cadrul acestui microserviciu: *PaymentIdDTO*, care înmagazinează coloanele ce definesc o tranzacție, ele formând și o cheie primară în cadrul unei tabele SQL, și *PaymentDTO*, ce conține un obiect de tip *PaymentDTO*.

Un serviciu efectuarea de plăți:

Acest serviciu conține două metode: *createPayment* este utilizată pentru a defini o tranzacție (preț, destinatar tranzacție, descriere) și se definesc rutile de redirecționare în cazul în care tranzacția reușește sau eșuează, și *executePayment*, în care se efectuează plata efectivă, verificând în prealabil dacă aceasta a mai fost executată înainte.

Controller Spring pentru crearea unui endpoint

Acest controller este utilizat pentru a permite clientului să inițieze plata și pentru a îl direcționa spre un mesaj de succes/eșec.

Fisier de configurație Paypal Sandbox

Această clasă este utilizată pentru a configura conexiunea cu Paypal Sandox: se setează id-ul administratorului aplicației, tokenul de acces al acestuia și modul de lucru (sandbox).

broker_service

Acest microserviciu reprezintă „motorul” aplicației de față, întrucât este conceput pentru a efectua o multitudine de operații:

- adăugare camioane/mărfuri în baza de date
- generarea ofertelor care să lege aceste tipuri de entități
- acceptarea sau respingerea ofertelor

Acest microserviciu conține două obiecte de tip DTO, *CargoDTO* și *TruckDTO*, care sunt utilizate pentru a defini o marfă, respectiv transport: dimensiuni, locație, descriere etc. Alături de acestea, s-au implementat și două servicii, *CargoService*, respectiv *TruckService*, prin intermediul căror se pot adăuga entități noi în baza de date, sau se pot realiza interogări după id sau e-mail.

Ofertele generate în cadrul algoritmului evolutiv au forma următoare:

CargoTruckAssignmentID		CargoTruckAssignmentDTO	
Ⓜ	CargoTruckAssignmentID()	Ⓜ	CargoTruckAssignmentDTO(CargoTruckAssignmentID, int, double,
Ⓜ	CargoTruckAssignmentID(long, long)	Ⓜ	CargoTruckAssignmentDTO(CargoTruckAssignmentID, int, double,
Ⓕ	id_cargo	Ⓕ	accepted_by_provider
Ⓕ	id_truck	Ⓕ	accepted_by_client
Ⓟ	id_truck	Ⓕ	blacklist
Ⓟ	id_cargo	Ⓕ	profit
		Ⓕ	cost
		Ⓟ	id
		Ⓟ	cost
		Ⓟ	blacklist
		Ⓟ	id
		Ⓟ	accepted_by_client
		Ⓟ	profit
		Ⓟ	accepted_by_provider

Figura 11. Clasele ce definesc o ofertă

Obiectul *CargoTruckAssignmentDTO* conține următoarele câmpuri:

- un obiect de tip *CargoTruckAssignmentID*: utilizat pentru a face legătura cu cheia primară a tabelei SQL *cargos_trucks*; acest obiect înmagazinează id-ul camionului și id-ul mărfui din cadrul ofertei

- Doi flag-uri, *accepted_by_provider* și *accepted_by_client*, care pot avea valoarea 1 dacă transportatorul, respectiv clientul a acceptat oferta.

- Un flag de blacklist, pentru a împiedica realizarea aceleiași oferte;

- O variabilă ce reprezintă profitul transportatorului;

- O variabilă ce reprezintă costul pe care comerciantul trebuie să îl suporte în cadrul ofertei de față.

Alături de acesta, există un serviciu, numit *CargoTruckAssignmentService*, prin intermediul căruia se pot salva ofertele în baza de date, se pot căuta și se pot șterge oferte pe baza id-ului camionului sau a mărfuii.

Întrucât aceeași aplicație de tip client poate fi folosită de un comerciant, dar și de un transportator, a fost necesară definirea a alte două obiecte DTO pentru a facilita obținerea de informații relevante pentru fiecare categorie de utilizator în parte:

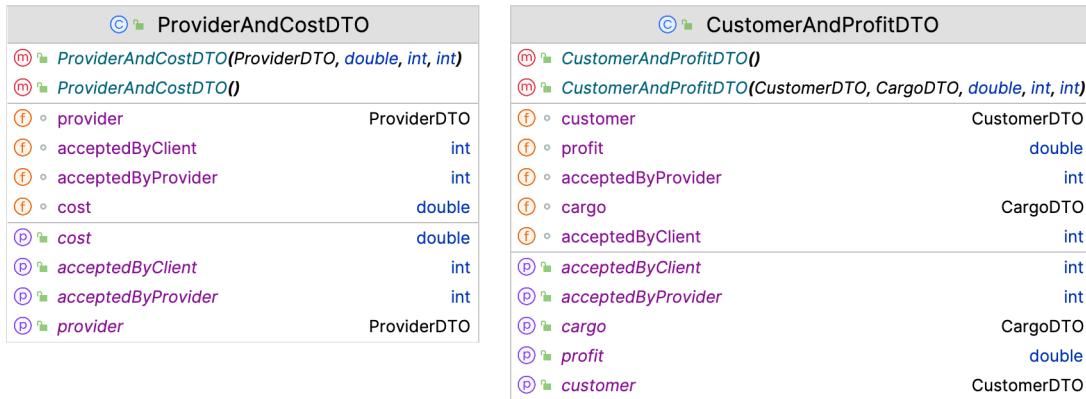


Figura 12. Clase auxiliare folosite pentru obținerea într-o formă serializată a informațiilor

Astfel, un comerciant primește, în momentul vizualizării unei oferte, informații despre transportator, despre costul cursei și dacă oferta a fost deja acceptată de către el sau de către transportator. Analog, transportatorul primește informațiile necesare pentru sine.

Serviciul din cadrul acestui microserviciu care utilizează toate componentele menționate mai sus este prezentat în figura următoare:



Figura 13. Serviciul de broker-ing și dependențele sale

Serviciul *BrokerService* conține următoarele metode și obiecte:

- `getCoordinates` - folosit pentru a apela API-ul Nominatim OpenStreetMap, care furnizează în format Json coordonatele unei locații primite ca parametru;
- `calculateDistance` - calculează distanța dintre două puncte de pe glob utilizând formula Haversine. [22]
- `addTruck`, `addCargo` - utilizate pentru a adăuga camioane/mărfuri în sistem;
- `deleteTruck`, `deleteCargo` - utilizate pentru a șterge camioane/mărfuri din sistem;
- `Chromosome` - clasă ce definește un cromozom din algoritmul evolutiv;
- `pairCargosTrucks` - utilizează un algoritm evolutiv pentru a genera oferte. În cadrul acestui algoritm, initializează cromozomii, definiți în clasa *Chromosome*, și au loc operațiunile de selecție, încrucișare și mutație, apoi are loc etapa de simulare propriu-zisă. În cadrul acesteia, se verifică dacă condițiile de transport sunt îndeplinite (dimensiuni și volum corespunzătoare), iar ofertele corespunzătoare sunt adăugate într-o listă de obiecte de tip *CargoAndDeliveryPrice* (obiecte care, la rândul lor, înmagazinează camionul, marfa și prețul cursei). Apoi, se calculează profiturile pentru fiecare transport în parte. Într-un obiect de tip *Map*, se salvează profitul maxim pentru fiecare camion pe parcursul rulării întregului algoritm (algoritm rulează până când toate camioanele devin ocupate sau toate mărfurile devin ocupate sau până când un contor ajunge la valoarea 5. În cadrul unei iterării, algoritmul propriu-zis rulează de 100 de ori.
- `getOfferForCustomer`, `getOfferForProvider` - folosit pentru a obține ofertele primite de către un comerciant sau transportator;
- `acceptOfferForCustomer`, `rejectOfferForCustomer` - permite comerciantului să accepte sau să respingă o ofertă;
- `acceptOfferForProvider`, `rejectOfferForProvider` - permite transportatorului să accepte sau să respingă o ofertă;

Capitolul 4. Implementarea aplicației

4.1. Interfața cu utilizatorul

La nivel de client, s-a implementat o interfață web utilizând React.js. Aceasta cuprinde mai multe componente:

- Meniu de „acasă” ce permite autentificarea sau crearea unui cont nou;
- Meniu de creare cont pentru comercianți sau transportatori;
- Meniu de autentificare pentru comercianți sau transportatori.

Figura 14. Meniu de creare unui transportator sau comerciant

În momentul în care un utilizator se autentifică, aceste funcționalități nu mai pot fi accesate decât în urma deconectării din aplicație. Accesarea paginii principale a aplicației din meniul superior al paginii web sau introducerea manuală a URL-ului în browser redirecționează către pagina de acasă a utilizatorului autentificat.

Un astfel de utilizator poate accesa următoarele componente:

- Meniu de „acasă” ce oferă posibilitatea de a adăuga un nou camion/o nouă marfă în sistem;
- Meniu de vizualizare a camioanelor sau a mărfurilor existente;
- Meniu de modificare a datelor contului;
- Meniu de deconectare.

În cadrul meniului de vizualizare a camioanelor sau a mărfurilor unui client, acesta are posibilitatea de a vedea în mod intuitiv dacă acestea au fost selectate în cadrul unei oferte, prin colorarea în mod distinct a celor două cazuri posibile. Clientul are posibilitatea de a accepta o ofertă sau de a o refuza, apăsând butoanele corespunzătoare. De asemenea, este notificat dacă oferta a fost deja acceptată.

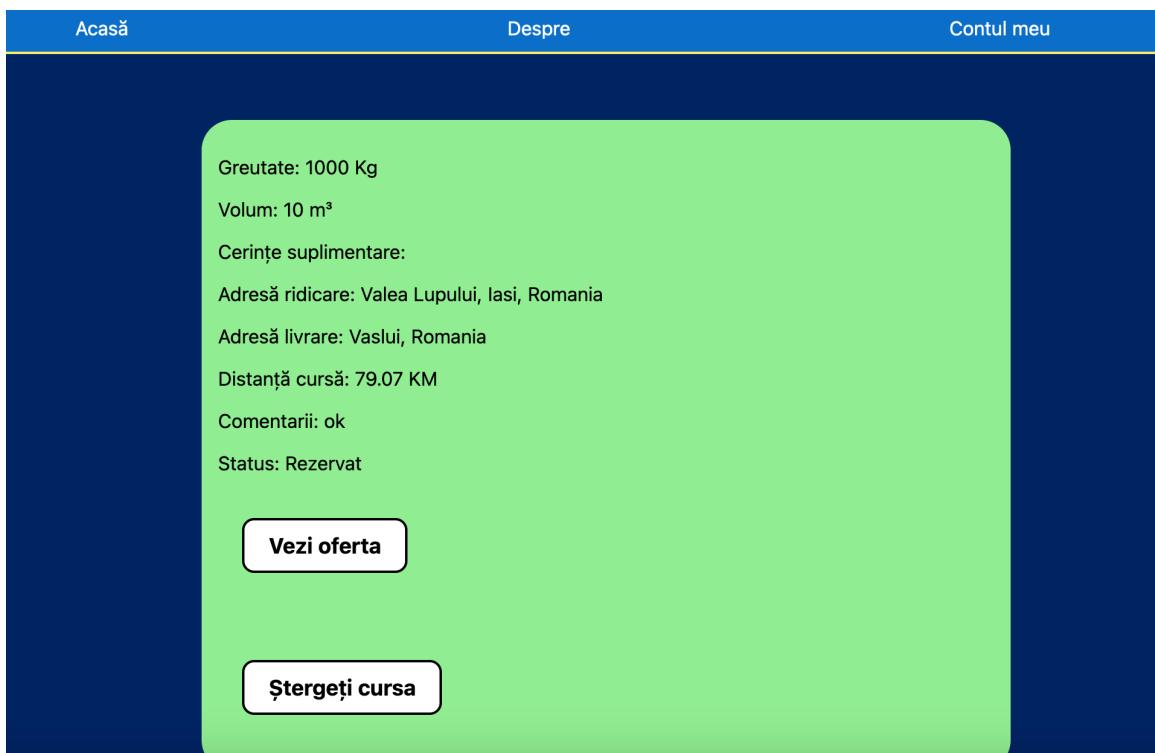


Figura 15



Figura 16

Figurile 15 și 16 prezintă modul de afișare a unei oferte și opțiunile puse la dispoziția clientului.

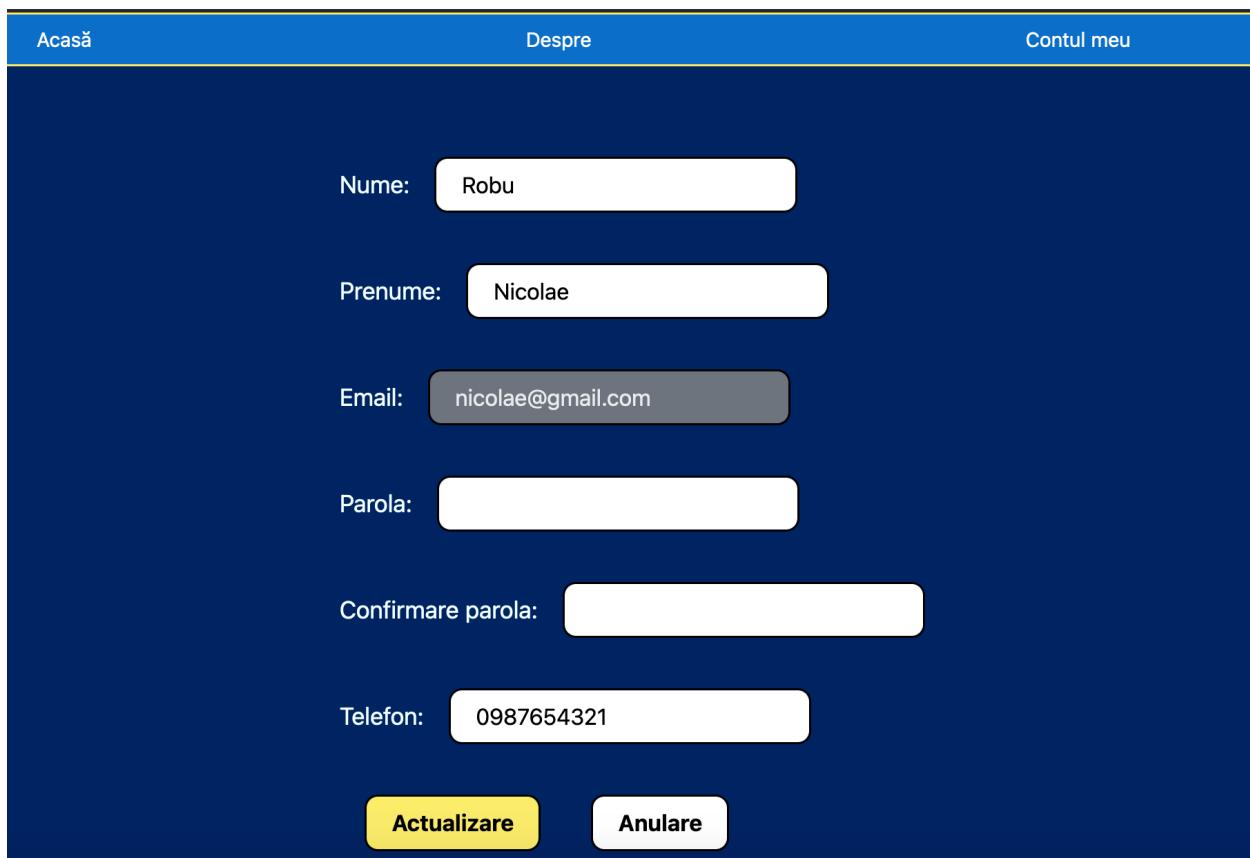


Figura 17. Meniul de modificare a unui cont existent.

Interfața cu utilizatorul a fost realizată pentru a fi ușor de înțeles și de utilizat. Elementele de control sunt dispuse de așa natură încât să ajute utilizatorul în a-și îndeplini obiectivele. De asemenea, diversele mesaje de avertizare din cadrul aplicației îl informează pe acesta în cazul apariției unor erori.

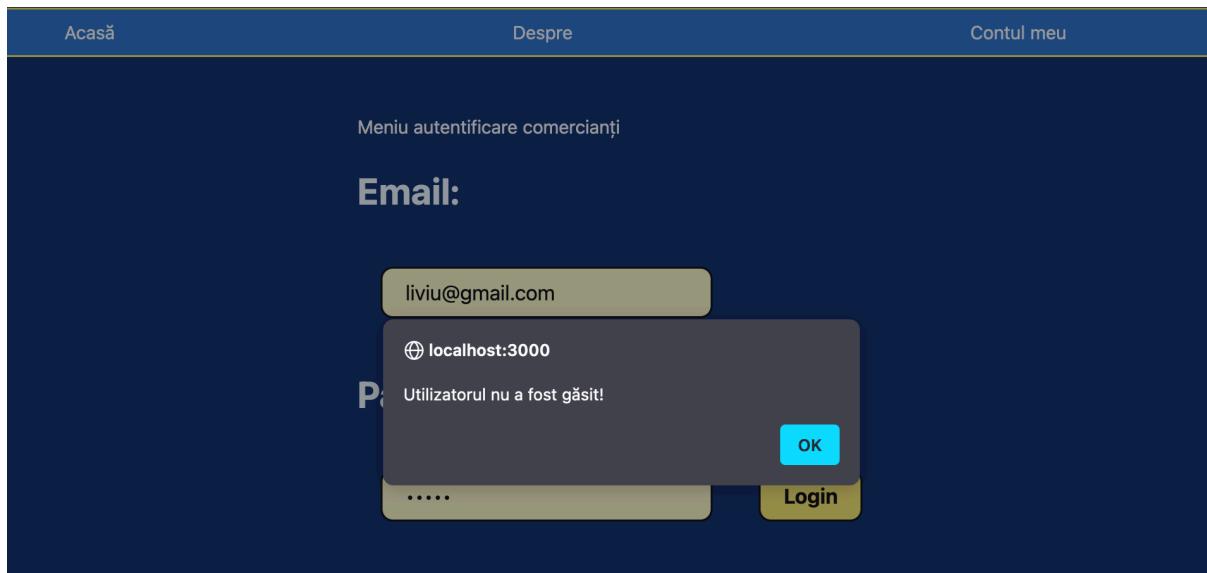


Figura 18. Mesaj de avertizare a clientului.

Prin folosirea unor tokeni, valabili 3 ore de la autentificare, menținerea permanentă a unui tab al aplicației în cadrul browser-ului web nu este necesară.

4.2. Comunicarea cu alte sisteme și stocarea informațiilor

4.2.1. OpenStreetMap Nominatim

În cadrul aplicației, s-a utilizat un API al OpenStreetMap, numit Nominatim. Acesta reprezintă un motor de căutare ce facilitează găsirea coordonatelor geografice ale unei locații fizice.

De exemplu, un apel către adresa:

<https://nominatim.openstreetmap.org/search?format=json&q=Bulevardul%20Tudor%20Vladimir%20escu%20nr.%202,%20Iasi,%20Romania>
returnează un vector de documente Json. Unul dintre acestea este:

```
{  
  "place_id": 51967368,  
  "licence": "Data © OpenStreetMap contributors, ODbL 1.0. http://osm.org/copyright",  
  "osm_type": "way",  
  "osm_id": 676123618,  
  "lat": "47.1618251",  
  "lon": "27.5981467",  
  "class": "highway",  
  "type": "primary",  
  "place_rank": 26,  
  "importance": 0.1000099999999993,  
  "address_type": "road",  
  "name": "Bulevardul Tudor Vladimirescu",  
  "display_name": "Bulevardul Tudor Vladimirescu, Tătărași Nord, Iași, Iași Metropolitan Area, Iași, 700398, Romania",  
  "boundingbox": [  
    "47.1617481",  
    "47.1619528",  
    "27.5980475",  
    "27.5982065"  
  ]  
}
```

Din acesta, se pot extrage valorile corespunzătoare latitudinii („lat”) și longitudinii („lon”) și se pot salva în baza de date.

4.2.2. Paypal Sandbox

Pentru a permite unui comerciant să achite un transport, s-a implementat un sistem de plată bazat pe Paypal Sandbox. Aceasta permite crearea unor conturi cu rol demonstrativ, atât pentru clienți, cât și pentru comercianți. În momentul în care o ofertă a fost acceptată bilateral, utilizatorul are posibilitatea de a efectua plata în contul transportatorului.

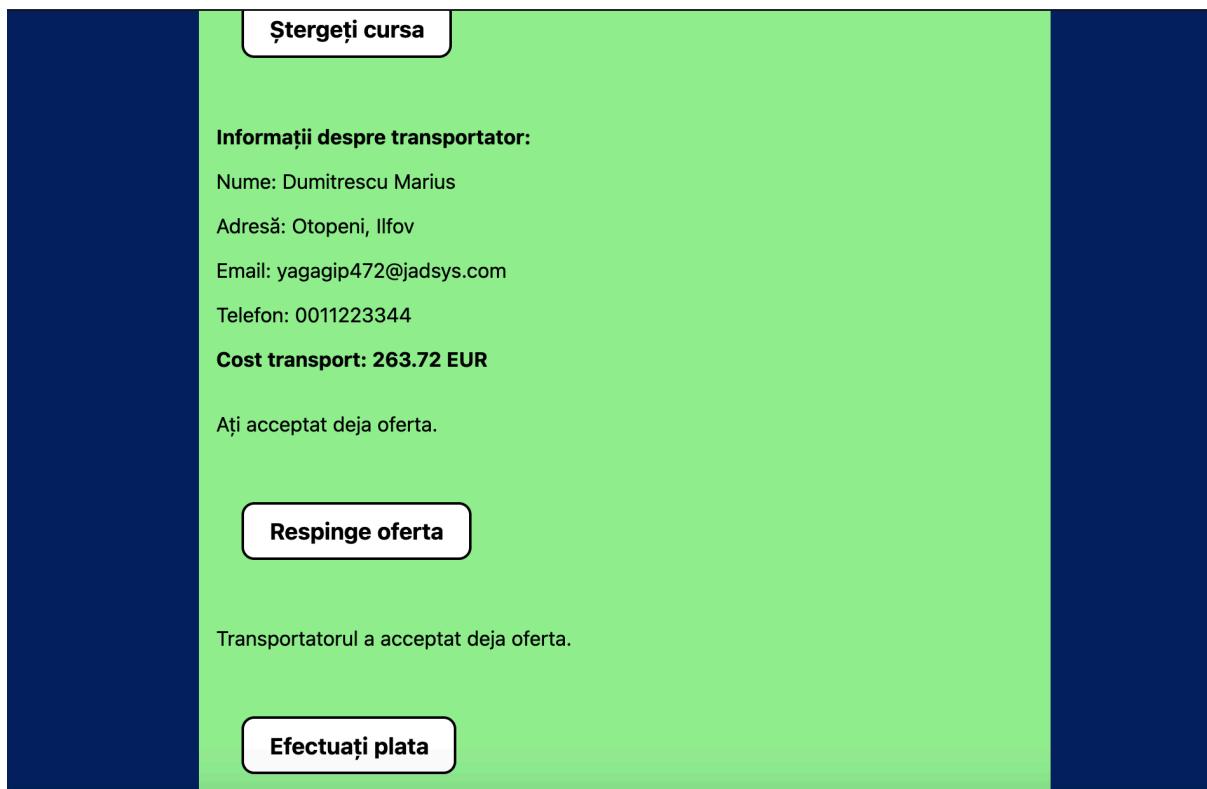


Figura 19. Butonul ce permite efectuarea unei plăți

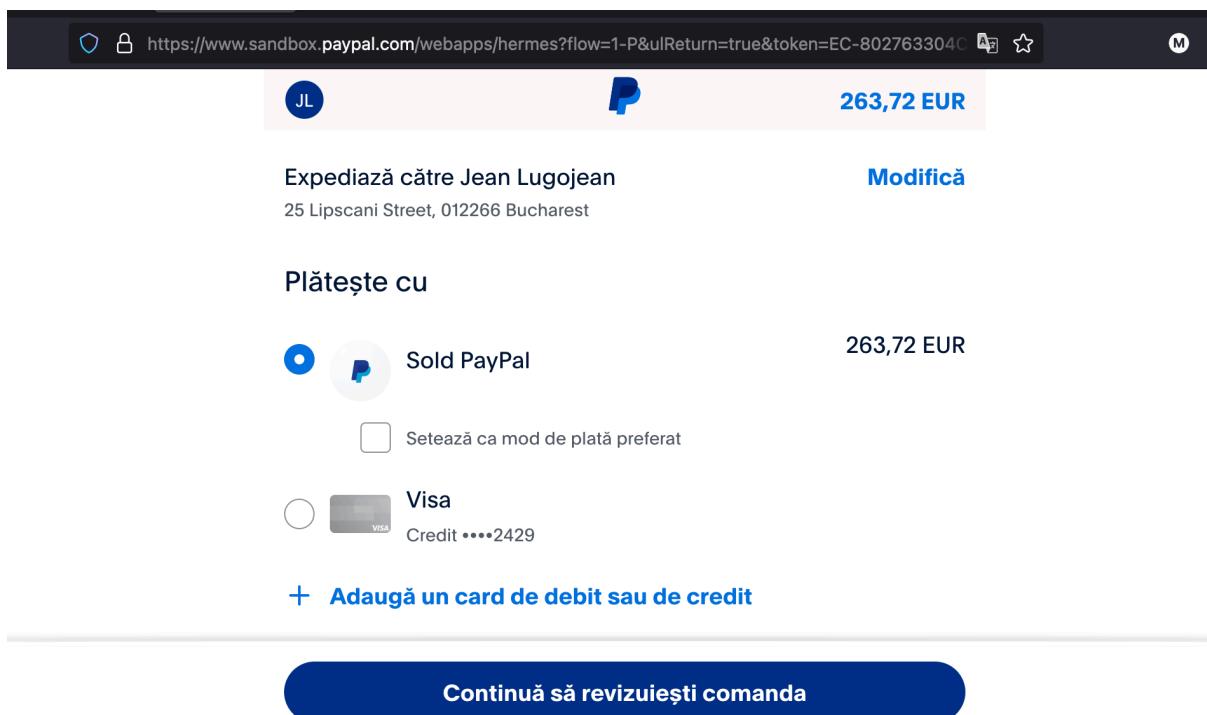


Figura 20. Efectuarea unei plăți.

4.2.3. Stocarea informațiilor

În cadrul acestei aplicații, s-a implementat o bază de date de tip SQL MariaDB. Aceasta reprezintă o soluție open-source ce oferă performanțe ridicate și un nivel de compatibilitate bun.

Serverul MariaDB rulează pe portul 3306, iar baza de date poate fi accesată din microserviciile REST utilizând framework-ul Hibernate ORM și următorul fișier de configurare:

```
spring.datasource.url=jdbc:mariadb://localhost:3306/freightbroker
spring.datasource.username=fmmarian
spring.datasource.password=fmmarian
spring.datasource.driver-class-name=org.mariadb.jdbc.Driver
spring.sql.init.mode=always
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
spring.jpa.hibernate.naming.implicit-
strategy=org.hibernate.boot.model.naming.ImplicitNamingStrategyLegacyHbmImpl
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MariaDBDialect
```

Prin intermediul acestui fișier, se stabilesc calea către baza de date utilizând un driver JDBC, numele de utilizator și parola gestionarului bazei de date, modul de adăugare a datelor, modul de comunicare cu Hibernate, tipul de bază de date utilizat, precum afișarea în consolă a comenzielor pe care Hibernate le execută.

4.3. Dificultăți întâmpinate și modalități de rezolvare

Realizarea autentificării în aplicație

Pentru a realiza o conexiune securizată și persistentă în cadrul aplicației, s-a implementat un server de gRPC ce utilizează fișiere de tip Protobuf. În momentul autentificării unui utilizator, informațiile necesare (nume de utilizator, parolă și tip de utilizator) sunt criptate și transmise microserviciului ce realizează autentificarea. Acesta interpretează mesajul transmis, verifică dacă utilizatorul există, iar în caz afirmativ, generează un token de tip JWT ce conține informații despre utilizator (ID, tip de utilizator, data de expirare token). Acest token va fi folosit ulterior de către utilizator pentru a putea accesa conținutul platformei.

Utilizarea bazei de date în aplicație

Datele platformei sunt salvate într-o bază de tip SQL. Tabelele acesteia sunt accesate de către microservicii prin intermediul framework-ului Hibernate ORM (ORM = (eng.) object relational mapping). În acest sens, au fost necesare unele adaptări în cadrul aplicației: configurarea server-ului MariaDB, crearea unor obiecte ce comunică cu baza de date, implementarea unor metode ce tratează interogări ce nu pot fi rezolvate în mod direct prin intermediul acestui framework.

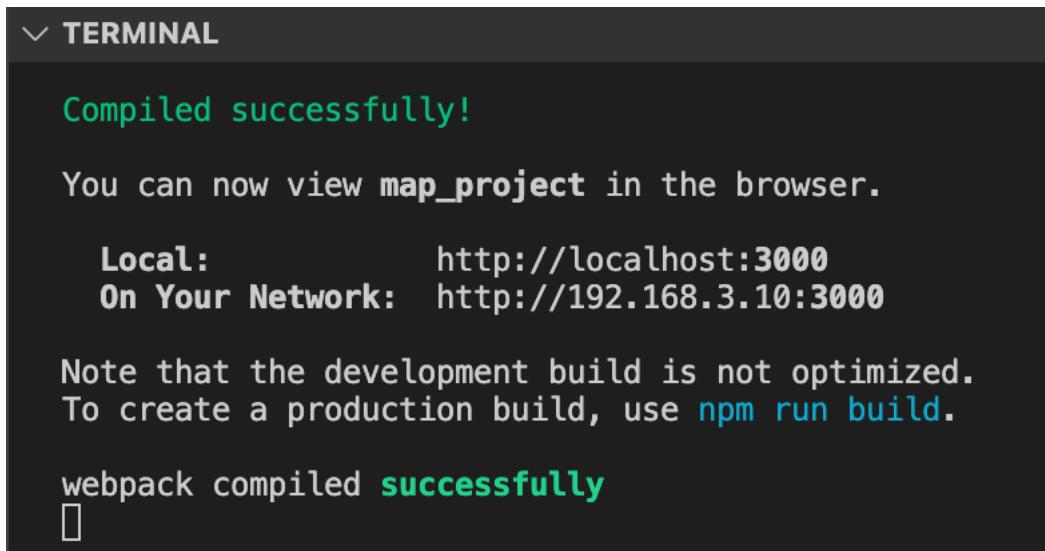
Apelarea unor API-uri în cadrul microserviciilor

Unele dintre microservicii apelează API-uri externe (fie alte microservicii deja implementate în cazul acestui proiect sau API-uri extere). O mare parte dintre aceste API-uri așteaptă să primească, în corpul cererii, un fișier de tip JSON ce reprezintă un obiect. Pentru a putea realiza acest lucru, s-a ales soluția de a transforma obiectul ce va trebui trimis în format binar, pentru a putea fi atașat corpului cererii și trimis către API-ul extern.

Capitolul 5. Testarea aplicației

5.1. Lansarea în execuție a aplicației

Lansarea în execuție a aplicației client se realizează utilizând comanda **npm start**. Rezultatul acesteia este:



```

    ✓ TERMINAL

    Compiled successfully!

    You can now view map_project in the browser.

    Local:          http://localhost:3000
    On Your Network: http://192.168.3.10:3000

    Note that the development build is not optimized.
    To create a production build, use npm run build.

    webpack compiled successfully
    └
  
```

Figura 21. Lansarea în execuție a aplicației client

Astfel, dezvoltatorul este informat că aplicația a fost compilată cu succes și că utilizează portul 3000 al rețelei *localhost*.

Lansarea în execuție a microserviciilor se poate realiza din mediul de dezvoltare IntelliJ IDEA utilizând comanda **mvn spring-boot:run** sau prin compilarea, împachetarea microserviciului respectiv și rularea acestuia în Terminal, prin comanda:

java -jar <nume_aplicație>.jar

Serverul MariaDB este lansat în execuție automat la autentificarea în sistemul de operare, sau poate fi lansată în execuție folosind managerul de pachete Homebrew, disponibil pe MacOS:

brew services start mariadb

5.2. Testarea sistemului

Testarea reprezintă o etapă-cheie în dezvoltarea unui produs software sau hardware și se realizează pentru a putea crește calitatea generală a acestuia, prin verificarea comportamentului său atât în modurile de funcționare prevăzute de programator, cât și în scenarii neprevăzute. Astfel, se asigură că produsul funcționează corect și fiabil în diverse situații, prevenind apariția erorilor și îmbunătățind experiența utilizatorului. Prin testare, se urmărește minimizarea erorilor

de funcționare, însă acest procedeu nu poate fi unul exhaustiv, întrucât proiectele complexe sunt foarte greu de urmărit și de analizat în detaliu.

În cadrul testării manuale, există două categorii de teste: cele de tip White Box, care presupun verificarea funcționalității corecte a programului cunoscând codul scris de către programator, și cele de tip Black Box, în care testerul nu are acces la codul sursă al programului și testează în mod intuitiv programul. Parte a testării de tip Black Box, se disting următoarele tipuri de teste funcționale:

- Unit Testing - se verifică funcționarea unui modul restrâns al aplicației (de exemplu, o funcție/metodă);
 - Integration Testing - se verifică funcționarea corectă și corelată a mai multor module;
 - System Testing - se verifică funcționarea corectă a întregului sistem software;
- și următoarele teste de tip non-funcțional:
- Performance Testing - se verifică funcționarea sistemului în cadrul încărcării rezonabile a acestuia (Load Testing) dar și în cazuri extreme de încărcare (Stress Testing), se verifică scalabilitatea acestuia (Scalability Testing) sau se verifică stabilitatea sistemului (Stability Testing)
 - Usability Testing - este evaluată experiența utilizatorului (UX) în cadrul aplicației: navigarea în sistem, interacțiunea cu acesta etc.
 - Compatibility Testing - este evaluată modul de funcționare al aplicației în cadrul unor platforme hardware/software disponibile: de exemplu, aspectul unui website pe un dispozitiv mobil sau pe PC, aspectul același website pe două browsere diferite, rularea unei aplicații de tip cross-platform pe sisteme de operare diferite.

Testarea automată presupune utilizarea unor programe specializate (e.g. Selenium) pentru a crea scenarii de testare, care se execută și stabilesc succesul/eșecul testului în funcție de răspunsul programului testat.

În cadrul proiectului de față, s-au executat o serie de teste funcționale menite să verifice funcționarea corectă a aplicației:

1. Autentificarea cu un cont valid

Rezultat: succes



Figura 22

2. Autentificarea cu un cont inexistent

Rezultat: eșec

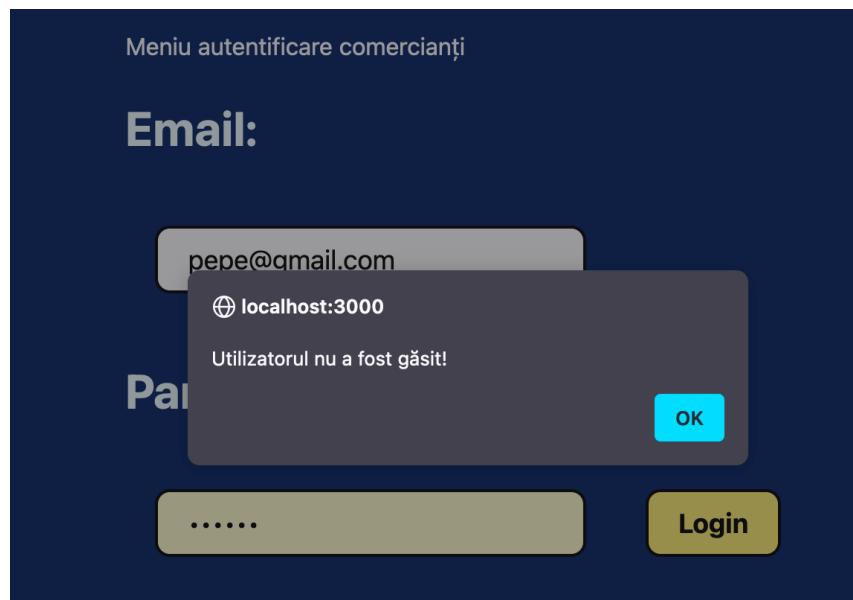


Figura 23

3. Crearea unui cont cu o adresă de e-mail invalidă

Rezultat: eșec

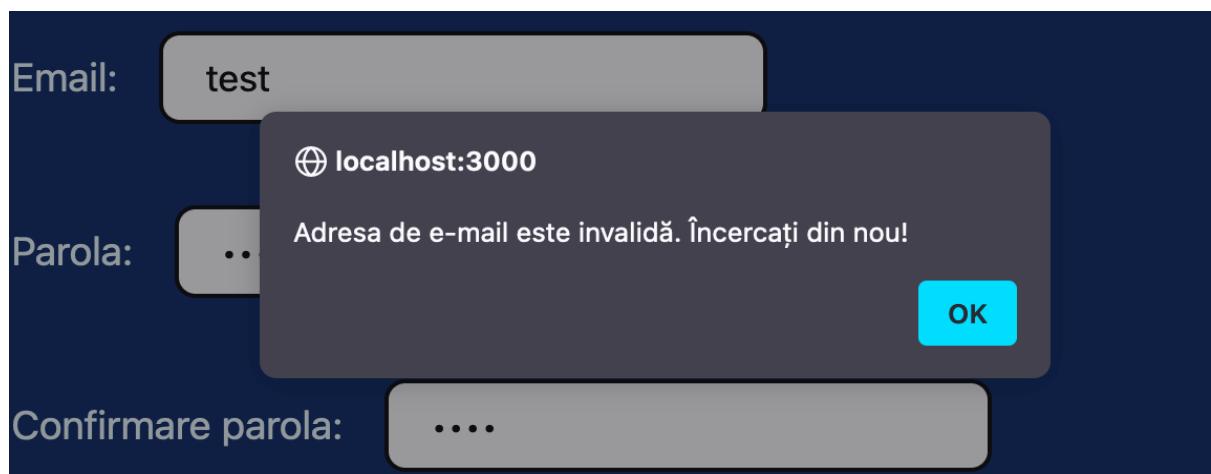


Figura 24

4. Crearea unui cont cu unul dintre câmpurile „Parolă” sau „Confirmare parolă” libere

Rezultat: eșec: apăsarea butonului „Submit” nu produce niciun efect dacă unul dintre aceste câmpuri nu este completat.

5. Crearea unui cont cu o adresă de e-mail validă

Rezultat: succes

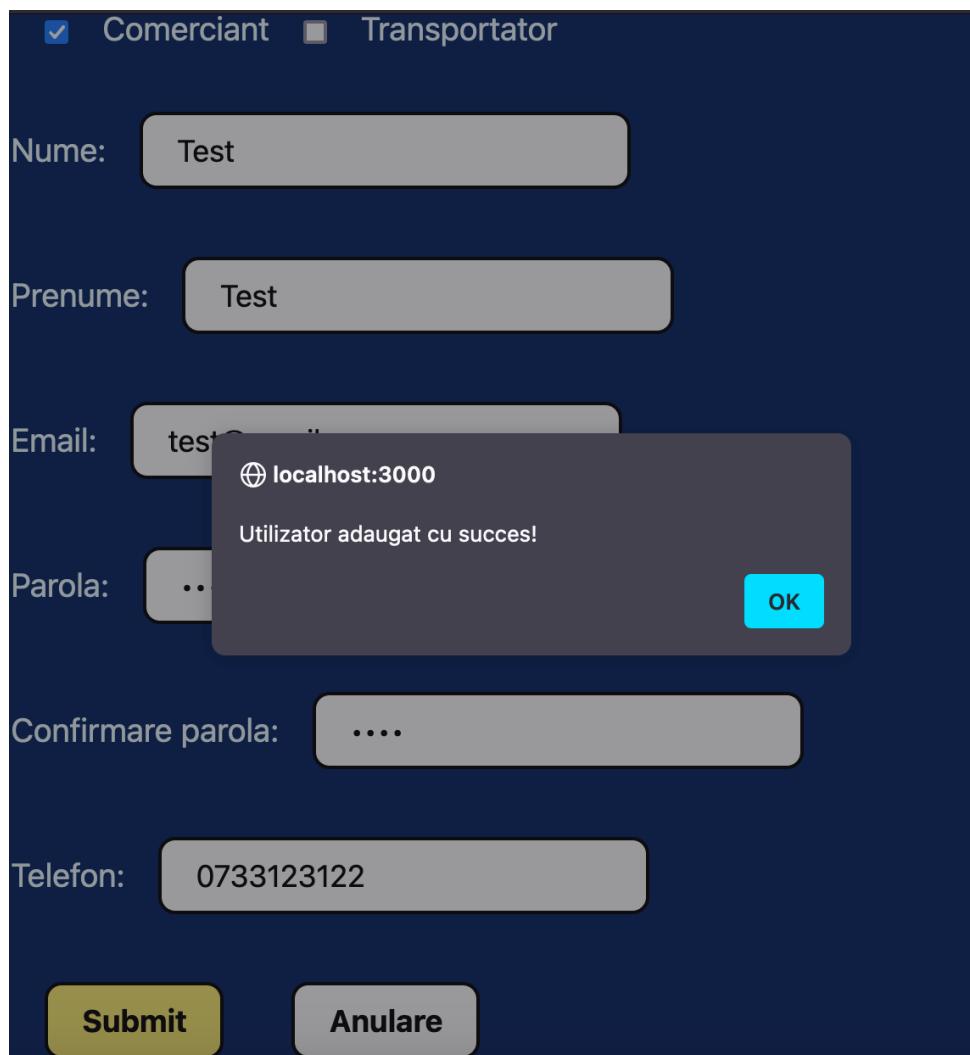


Figura 25

6. Crearea unui cont atunci când câmpurile „Parolă” și „Confirmare Parolă” nu coincid

Rezultat: eșec



Figura 26

7. Crearea unui cont cu o adresă de e-mail deja utilizată

Rezultat: eșec

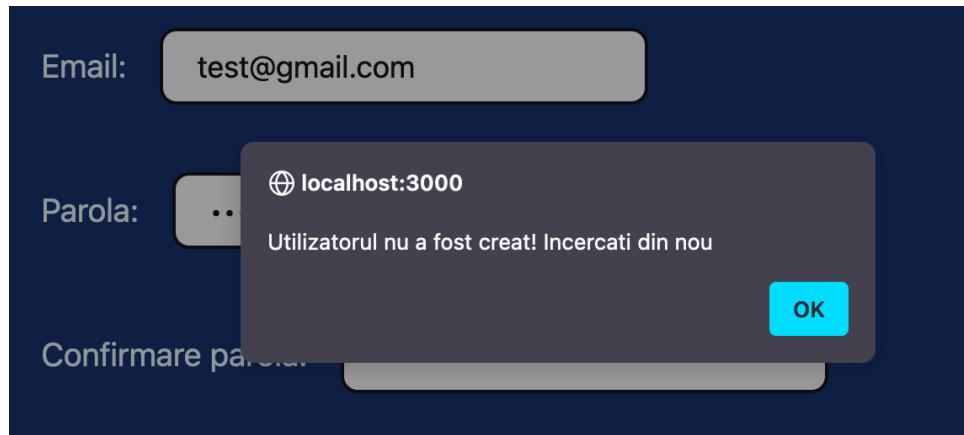
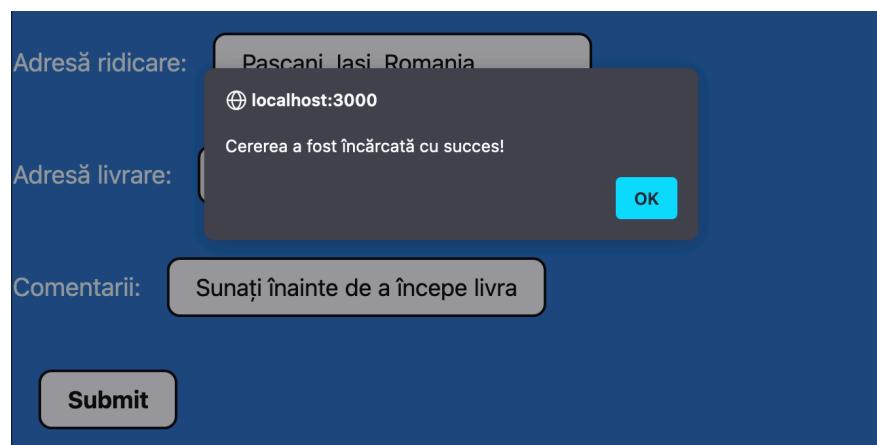


Figura 27

8. Adăugarea unui nou tip de marfă

Rezultat: succes



Figurile 28 și 29

9. Modificarea unui cont prin furnizarea câmpurilor „Parolă” și „Confirmare Parolă” distincte

Rezultat: eșec



Figura 30

5.3. Rezultate experimentale

5.3.1. Date de test

Într-un scenariu simplu de utilizare a aplicației de față, presupunem că următoarele mărfuri sunt introduse în sistem fără a fi alocate:

NR. MARFĂ	LOCAȚIE PLECARE	LOCAȚIE SOSIRE	INTERVAL LIVRARE (ZILE)	GREUTATE (KG)	VOLUM (m ³)
1	Piatra Neamț	Bacău	1	2.200	13
2	Piatra Neamț	Suceava	1	12.000	69
3	Botoșani	Piatra Neamț	2	9.000	52
4	Botoșani	Bacău	2	400	2
5	Piatra Neamț	Suceava	1	5.000	29
6	Iași	Botoșani	2	9.000	52

De asemenea, sunt introduse următoarele camioane:

NR. CAMION	LOCAȚIE INITIALĂ	GREUTATE MAXIMĂ ADMISĂ(KG)	VOLUM MAXIM ADMIS(m ³)	PREȚ/KM (EURO)	COST/KM (EURO)
1	Vaslui	24.000	140	3,78	1,20
2	Bacău	1.500	10	0,99	0,30
3	Botoșani	3.500	20	4,22	1,20

Fără a rula algoritmul, putem observa faptul că în sistem se găsesc de două ori mai multe mărfuri decât camioane. Algoritmul este construit în aşa fel încât să genereze oferte până reușește să rezerve toate camioanele sau toate mărfurile. Este de așteptat, deci, ca anumite mărfurisă rămână neallocate.

După rularea algoritmului, obținem următoarele rezultate:

NR. CAMION	NR. MARFĂ	COST DE TRANSPORT/ CLIET(€)	PROFIT OBȚINUT/ TRANSPORTATOR(€)
1	6	373,10	183,70
2	4	157,44	62,02
3	1	263,71	77,09

Pentru fiecare camion liber, se urmărește maximizarea profitului. Camionul 1 prezintă cea mai mare capacitate de transport, prin urmare, orice cursă poate fi fezabilă din punct de vedere al gabaritului. Cu toate acestea, distanța pe care acesta o parcurge în gol este cea mai scurtă până la camionul 6, prin urmare, se minimizează cheltuielile. Camionul 2 are capacitatea de a transporta doar marfa 4, din cauza restricțiilor de gabarit existente. Pentru camionul 3, Doar marfa 1 și marfa 4 pot fi transportate. Cu atât mai mult, marfa 4 implică cheltuieli suplimentare foarte reduse pentru transportator, deoarece locația camionului și locația de preluare a mărfii coincid. Cu toate acestea, din moment ce camionul 2 a preluat deja marfa 4, camionul 3 rămâne cu o singură comandă pe care o poate onora.

5.3.2. Testarea algoritmului evolutiv

Pentru testarea amplă a rezultatelor algoritmului evolutiv, se consideră trei scenarii de test: un scenariu restrâns, în care trebuie generate oferte pentru trei camioane și trei mărfuri, un scenariu mediu, în care trebuie generate oferte pentru șapte camioane și șapte mărfuri, și un scenariu mai complex, în care trebuie generate oferte pentru paisprezece camioane și paisprezece mărfuri.

- Scenariul 1

Acest scenariu utilizează datele de test de mai jos:

NR. MARFĂ	LOCAȚIE PLECARE	LOCAȚIE SOSIRE	INTERVAL LIVRARE (ZILE)	GREUTATE (KG)	VOLUM (m³)
1	Piatra Neamț	Bacău	1	2.200	13
2	Piatra Neamț	Suceava	1	12.000	69
3	Botoșani	Piatra Neamț	2	9.000	52
NR. CAMION	LOCAȚIE INITIALĂ	GREUTATE MAXIMĂ ADMISĂ(KG)	VOLUM MAXIM ADMIS(m³)	PRET/KM (EURO)	COST/KM (EURO)
1	Vaslui	24.000	140	3,78	1,20
2	Bacău	1.500	10	0,99	0,30
3	Botoșani	3.500	20	4,22	1,20

O primă rulare a acestui caz durează 407 milisecunde și găsește rezultatele:

- Camion 1 => Marfa 3 (100% dintre cazuri)
- Camion 3 => Marfa 1 (100% dintre cazuri)

După alte nouă rulări, timpul mediu de execuție al algoritmului este de 290 milisecunde, însă rezultatele execuției rămân aceleași. Camionul 2 nu poate transporta nicio marfă, camionul 3 nu poate transporta decât marfa 1, iar camionul 1 poate transporta orice marfă. Cum marfa 1 este preluată de camionul 3, camionul 1 are de ales între marfa cu numărul 2 și 3. Marfa 3 este mai profitabilă, deoarece costul de deplasare până la locația initială este mai scăzut.

În cadrul acestui scenariu, pentru timpii de rulare ai algoritmului:

$$Media = \frac{\sum_{i=1}^n x_i}{n} = 297.4 \text{ milisecunde},$$

$$Deviatia_standard = \sqrt{\frac{\sum_{i=1}^n (x_i - Media)^2}{n}} = 39.59 \text{ milisecunde},$$

unde $x = [407, 242, 297, 291, 296, 290, 295, 286, 280, 290]$ (milisecunde)

Profitul mediu al unui transportator este de 65,0 euro, iar costul mediu de transport este de 361,4 euro.

- Scenariul 2

Acest scenariu utilizează datele de test de mai jos:

NR. MARFĂ	LOCAȚIE PLECARE	LOCAȚIE SOSIRE	INTERVAL LIVRARE (ZILE)	GREUTATE (KG)	VOLUM (m³)
1	Piatra Neamț	Bacău	1	2.200	13
2	Piatra Neamț	Suceava	1	12.000	69
3	Botoșani	Piatra Neamț	2	9.000	52
4	Iași	Bacău	2	2.500	18
5	Bacău	Roman	1	15.000	79
6	Roman	Piatra Neamț	2	7.900	40
7	Botoșani	Piatra Neamț	1	4.500	63

NR. CAMION	LOCAȚIE INIȚIALĂ	GREUTATE MAXIMĂ ADMISĂ(KG)	VOLUM MAXIM ADMIS(m³)	PREȚ/KM (EURO)	COST/KM (EURO)
1	Vaslui	24.000	140	3,78	1,20
2	Bacău	1.500	10	0,99	0,30
3	Botoșani	3.500	20	4,22	1,20
4	Iași	22.000	140	4,00	1,60
5	Bacău	1.500	40	0,79	0,50
6	Suceava	3.500	60	3,72	1,10
7	Piatra Neamț	12.100	70	4,92	1,35

Rezultatele găsite sunt:

- Camion 1 => Marfa 7
- Camion 3 => Marfa 1
- Camion 4 => Marfa 3
- Camion 6 => Marfa 4
- Camion 7 => Marfa 6

În cadrul acestui scenariu, pentru timpii de rulare ai algoritmului:

$$\text{Media} = \frac{\sum_{i=1}^n x_i}{n} = 297,4 \text{ milisecunde},$$

$$\text{Deviația standard} = \sqrt{\frac{\sum_{i=1}^n (x_i - \text{Media})^2}{n}} = 362,6 \text{ milisecunde},$$

unde $x = [407, 242, 297, 291, 296, 290, 295, 286, 280, 290]$ (milisecunde).

Profitul mediu al unui transportator este de 60,66 euro, iar costul mediu de transport este de 361,4 euro.

- Scenariul 3

În cadrul acestui scenariu, în completarea mărfurilor și camioanelor menționate în scenariul 2 se mai adaugă următoarele:

NR. MARFĂ	LOCATIE PLECARE	LOCATIE SOSIRE	INTERVAL LIVRARE (ZILE)	GREUTATE (KG)	VOLUM (m ³)
8	Piatra Neamț	Bacău	1	1.400	41
9	Roman	Suceava	1	12.000	75
10	Botoșani	Roman	2	6.000	34
11	Iași	Suceava	2	5.200	63
12	Bacău	Botoșani	1	8.000	14
13	Roman	Piatra Neamț	2	6.000	45
14	Botoșani	Suceava	1	3.000	12

NR. CAMION	LOCATIE INITIALĂ	GREUTATE MAXIMĂ ADMISĂ(KG)	VOLUM MAXIM ADMIS(m ³)	PRET/KM (EURO)	COST/KM (EURO)
8	Vaslui	24.000	140	3,78	1,70
9	Bacău	1.500	10	0,99	0,34
10	Botoșani	3.500	20	4,22	1,60
11	Iași	22.000	140	3,78	1,70
12	Bacău	1.500	40	0,99	0,40
13	Suceava	3.500	60	4,22	1,35
14	Piatra Neamț	12.100	70	4,22	1,40

Rezultatele sunt:

- *Camion 1 => Marfa 5 (10% din cazuri), Marfa 11 (30% din cazuri), Marfa 7 (30% din cazuri), Marfa 12 (10% din cazuri), Marfa 3 (10% din cazuri), Marfa 2 (10% din cazuri)*
- *Camion 2 => nicio alocare*
- *Camion 3 => Marfa 14 (50% din cazuri)*
- *Camion 4 => Marfa 7 (40% din cazuri), Marfa 9 (60% din cazuri)*
- *Camion 5 => nicio alocare*
- *Camion 6 => Marfa 4 (20% din cazuri), Marfa 8 (80% din cazuri)*
- *Camion 7 => Marfa 7 (10% din cazuri), Marfa 10 (90% din cazuri)*
- *Camion 8 => Marfa 9 (30% din cazuri), Marfa 11 (30% din cazuri), Marfa 12 (40% din cazuri)*
- *Camion 9 => nicio alocare*

- *Camion 10* => Marfa 1 (10% din cazuri), Marfa 14 (10% din cazuri)
- *Camion 11* => Marfa 11 (40% din cazuri), Marfa 12 (60% din cazuri)
- *Camion 12* => Marfa 4 (80% din cazuri)
- *Camion 13* => Marfa 1 (60% din cazuri), Marfa 14 (10% din cazuri)
- *Camion 14* => Marfa 1 (40% din cazuri), Marfa 14 (20% din cazuri), Marfa 7 (20% din cazuri)

Se observă că, în cadrul acestui scenariu, alocările se pot schimba de la o rulare la alta. Există cazuri în care alocările sunt relativ identice de la o rulare la alta (de exemplu, camioanele 4, 7 și 12), dar există și cazuri în care alocările diferă semnificativ de la o rulare la alta.

În cadrul acestui scenariu, pentru timpii de rulare ai algoritmului:

$$\text{Media} = \frac{\sum_{i=1}^n x_i}{n} = 828.375 \text{ milisecunde},$$

$$\text{Deviația standard} = \sqrt{\frac{\sum_{i=1}^n (x_i - \text{Media})^2}{n}} = 167.03 \text{ milisecunde},$$

unde $x = [953, 634, 1109, 688, 717, 1033, 733, 760]$ (milisecunde).

În decursul celor 10 rulări, s-au obținut următoarele valori medii ale profitului și costului de transport:

$$\text{profit_mediu} = [270.83, 137.92, 164.56, 133.30, 276.26, 291.00, 280.34, 195.32, 260.47, 276.26] \text{ (euro)}$$

$$\text{cost_mediu} = [257.23, 339.61, 357.04, 376.28, 264.65, 270.21, 352.64, 286.73, 361.53, 264.65] \text{ (euro)}$$

Deci, putem calcula media și deviația standard pentru aceste valori:

$$\text{Media_profit} = \frac{\sum_{i=1}^n x_i}{n} = 237.57 \text{ euro},$$

$$\text{Deviația standard_profit} = \sqrt{\frac{\sum_{i=1}^n (x_i - \text{Media_profit})^2}{n}} = 61.96 \text{ euro, unde } x = \text{profit_mediu}$$

$$\text{Media_cost_transport} = \frac{\sum_{i=1}^n x_i}{n} = 309.34 \text{ euro},$$

$$\text{Deviația standard_cost_transport} = \sqrt{\frac{\sum_{i=1}^n (x_i - \text{Media_cost_transport})^2}{n}} = 44.96 \text{ euro, unde } x = \text{cost_mediu}$$

5.4. Resurse necesare

Atunci când nicio operație nu se execută, utilizarea sistemului este scăzută:

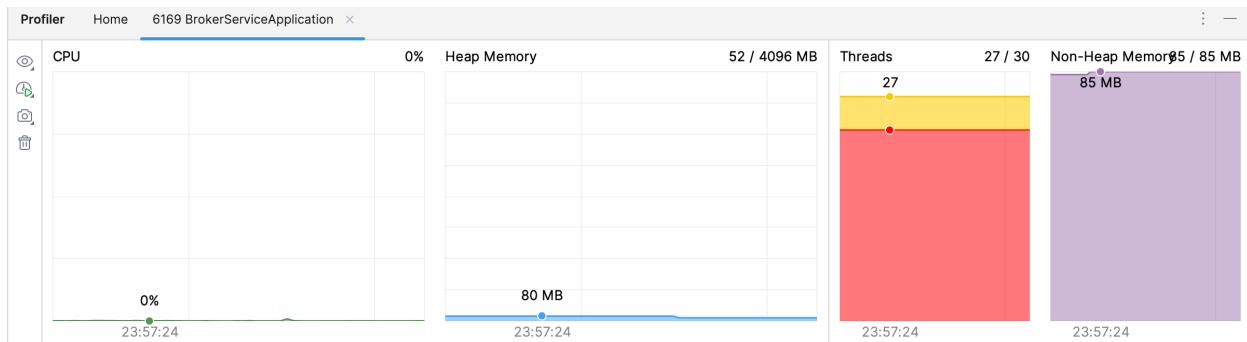


Figura 31. Încărcarea sistemului când nu se execută nicio operațiune.

Pentru testarea programului, s-au creat metode pentru a putea adăuga camioane sau mărfuri fără a declanșa algoritmul de generare a ofertelor. Secvența de cod ce măsoară timpul de execuție a algoritmului este:

```
@GetMapping("/pairtime")
public void PairTime() throws IOException {
    var start = System.currentTimeMillis();
    brokerService.pairCargosTrucks();
    var stop = System.currentTimeMillis();
    System.out.println("Durata rulare algoritm: "+(stop-start));
}
```

Pentru găsirea ofertelor pentru cele 6 mărfuri și 3 camioane descrise în capitolul [5.3.1](#), algoritmul propriu-zis se execută timp de 230-300 de milisecunde. Este important de menționat că, în momentul generării unei oferte, se trimit e-mail-uri către clienți. Dacă se optează pentru trimiterea de e-mail-uri, durata de execuție a algoritmului crește până la 9 secunde.

Utilizarea sistemului în timpul execuție algoritmului se poate vedea în figura următoare:

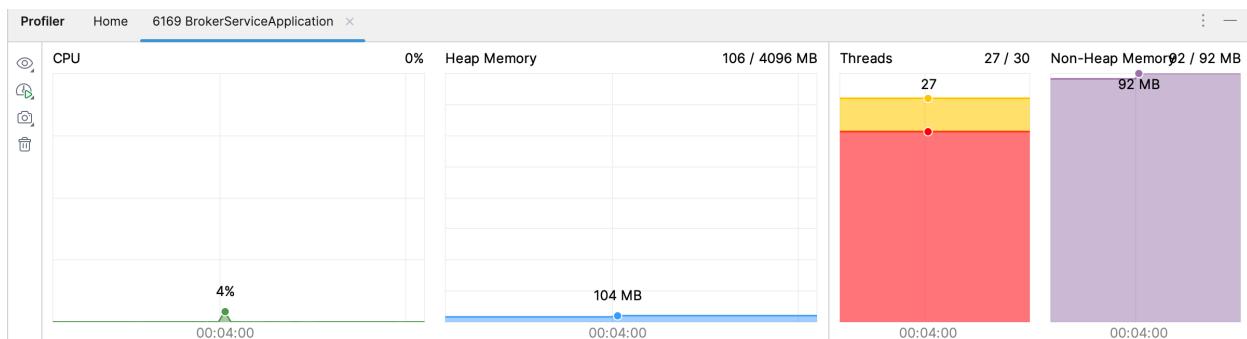


Figura 32. Încărcarea sistemului când algoritmul rulează cu un număr redus de camioane și de mărfuri.

Se observă că încărcarea sistemului este aproape neglijabilă.

S-a simulaat adăugarea în sistem a unui număr mare de camioane și de mărfuri. Adăugarea a 200 de mărfuri și 150 de camioane simultan a durat aproximativ 9 minute. Acest timp ridicat de execuție poate avea două cauze. În primul rând, instrucțiunile de adăugare în baza de date sunt adăugat în mod secvențial (un total de 350 operații de inserare). Înlocuirea acestora cu inserarea multiplă a tuturor datelor (doar două operații de inserare) nu a redus timpul de execuție în mod semnificativ. Acest lucru se datorează apelurilor API-ului Nominatim, utilizat pentru a găsi coordonatele geografice ale punctelor de plecare sau de sosire a mărfurilor. În scopuri demonstrative, accesul către aceste coordonate se putea optimiza în mod semnificativ, prin găsirea tuturor coordonatelor la începutul algoritmului de adăugare de date în baza de date și folosirea acestora ulterior. În acest fel, timpul de generare a ofertelor scade la 11 secunde însă, în realitate, aplicația poate fi influențată de numărul de operații de adăugare a mărfurilor și camioanelor.

Algoritmul evolutiv utilizat pentru generarea automată a ofertelor se execută în aproximativ 52 de secunde pentru cele 200 de mărfuri și 150 de camioane înrolate în sistem și reușește să genereze 148 de oferte. Prin urmare, un număr mare de entități nu înrăutățește în mod sesizabil performanța sistemului.

Utilizarea sistemului în timpul execuției algoritmului se poate observa mai jos:

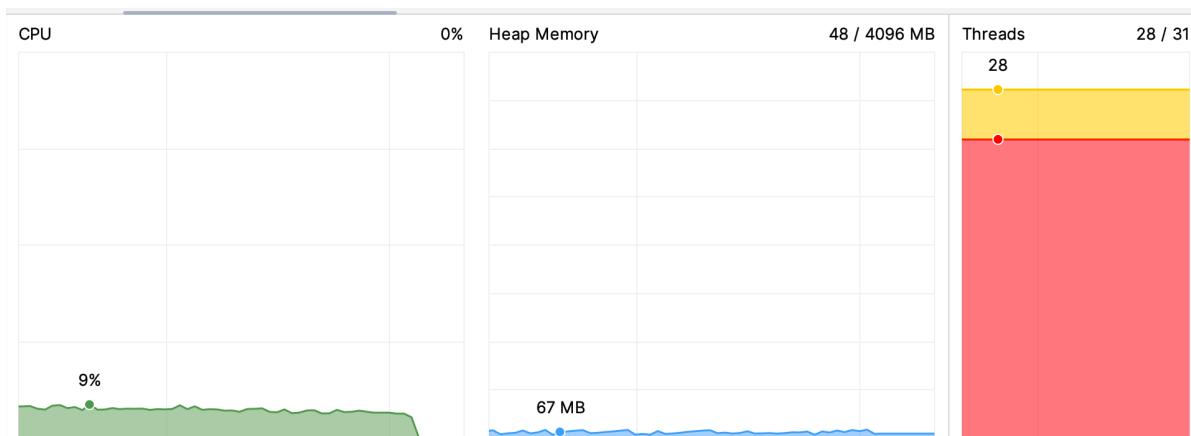


Figura 33. Încărcarea sistemului când se generează oferte pentru un număr mare de camioane și de mărfuri.

Se observă faptul că gradul de utilizare a procesorului crește cu numai 5 procente comparativ cu scenariul simplu ce presupune existența a doar 6 mărfuri și 3 camioane. Așadar, algoritmul implementat poate fi considerat eficient din punct de vedere computațional.

Din perspectiva utilizării rețelei, se observă cum procesul mariadb a trimis un număr mare de megaocteți prin rețea, semnificând extragerea din baza de date a camioanelor, a mărfurilor libere și a relațiilor deja existente dintre aceste două tipuri de entități. De asemenea, se observă faptul că un proces java a primit un număr mare de megaocteți, reprezentând răspunsurile primite din partea API-ului Nominatim.

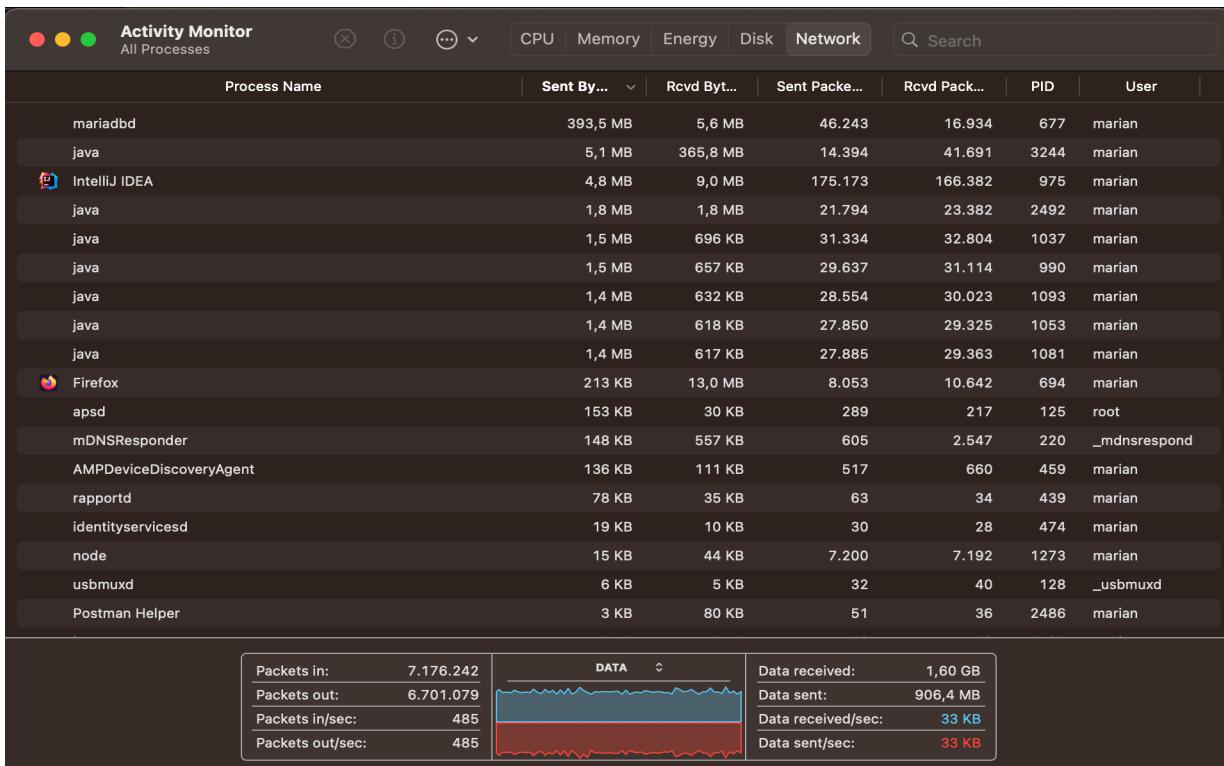


Figura 34. Utilizarea rețelei.

5.5. Securitatea sistemului

Paradigma client-server presupune existența unor parole, pe baza cărora utilizatorii își creează conturi și se autentifică în aplicație. Aceste parole sunt transmise prin rețea între cele două componente și sunt stocate într-o bază de date, unde sunt adăugate, interogate sau modificate. Prin urmare, se impun găsirea unor soluții pentru a păstra confidențialitatea acestora, încrucișând descoperirea acestora de către persoane neautorizate/străine poate duce la compromiterea utilizatorilor atât la nivel de aplicație, cât și pe alte pagini web/conturi, încrucișând majoritatea utilizatorilor aleg să folosească aceeași adresă de e-mail alături de aceeași parolă pe diverse pagini web.

În momentul creării unui cont, un utilizator furnizează o parolă pe care acesta o furnizează în clar. Aceasta este transmisă la server, alături de celelalte date despre client, pentru a fi stocate în baza de date. Un atacator ar putea intercepta cererea către server, descoperind parola utilizatorului. Pentru a preveni acest aspect, s-a utilizat o funcție de hashing a parolei.

Hashing-ul reprezintă generarea unui rezumat/digest al textului în clar furnizat. Acest procedeu este diferit față de criptare, în sensul că nu utilizează chei de criptare sau de decriptare [23]. În cadrul aplicației de față, s-a utilizat biblioteca *bcryptjs*, disponibilă atât pentru SpringBoot, cât și pentru ReactJS. Astfel, este posibilă realizarea hashing-ului unei parole pe baza instrucțiunii următoare:

```
const hashedPwd = await bcrypt.hash(parola, 10);
```

, unde 10 reprezintă numărul de runde de tip Salt aplicate. Salting-ul reprezintă adăugarea unor date aleatorii asupra intrării unui algoritm de hashing pentru a genera mereu un hash unic.

Această metodă prezintă următoarele avantaje:

1. Este un procedeu „cu sens unic”: nu se poate obține textul în clar odată ce asupra acestuia s-a aplicat o funcție de tip hash;
2. Găsirea unui sir de caractere care să producă același hash este dificilă din punct de vedere computațional. Algoritmul *bcryptjs* este scris în JavaScript și este conceput de așa natură încât să ruleze mai greu decât varianta *bcrypt*, ceea ce face găsirea forțată (brute forcing) al unei parole să nu fie fezabilă.
3. Un text în clar are mai multe reprezentări posibile sub formă hash: Datorită salting-ului, același text în clar va genera hash-uri diferite de fiecare dată.
4. Verificarea egalității dintre o parolă și o parolă sub formă hash se face utilizând o funcție de comparare. Prin urmare, prin compararea parolei în clar, la momentul autentificării cu un hash al aceleiași parole stocate în baza de date se permite autentificarea utilizatorului.

Din lista anterioară, punctul 4 ridică o nouă problemă: hashing-ul unei parole produce mereu rezultate diferite, prin urmare sunt distințe între ele. De asemenea, funcția de comparare nu permite găsirea egalității dintre două hash-uri ale aceleiași parole. Prin urmare, în momentul autentificării unui utilizator, trebuie furnizată parola în clar către server, ceea ce ridică din nou problema confidențialității acesteia.

Pentru a preveni această potențială breșă de securitate, s-au utilizat tehnologia gRPC. Acest framework open-source de la Google folosește fișiere de tip Protobuf pentru a serializa și deserializa datelor în vederea păstrării confidențialității.

Fișierul de tip Protobuf utilizat în cadrul aplicației este:

```
syntax = "proto3";
option java_package = "com.freightbroker.customer_service.grpc";

service AuthenticationService{
    rpc authenticate (AuthenticationRequest) returns (AuthenticationResponse){}
    rpc validate (ValidateRequest) returns (ValidateResponse){}
}

message AuthenticationRequest{
    string email = 1;
    string password = 2;
    string user_type = 3;
}
message AuthenticationResponse{
    string token = 1;
}

message ValidateRequest{
    string valid = 1;
}
```

```
message ValidateResponse{  
    string valid = 1;  
}
```

Prin intermediul acestui fișier, se definește corpul unei cereri (AuthenticationRequest) și metodele ce se vor implementa (authenticate și validate).

La nivel de client, se creează un buffer de tip AuthenticationRequest, care este encodat și trimis către server pentru autentificare. Un exemplu de buffer este:

```
Uint8Array(38):[10,16,109,105,114,99,101,97,64,103, ... ]
```

Acest sir de numere întregi reprezintă date serializate și poate fi interpretat la nivel de server astfel încât se pot obține parola, email-ul și tipul de utilizator în vederea autentificării. În acest mod, parola clientului nu este niciodată transmisă în clar între diversele componente ale aplicației.

Concluziile lucrării

Realizări

În cadrul lucrării de față, s-a propus și realizat o platformă web pentru transportul de mărfuri care realizează oferte în mod automat și care urmează arhitectura RESTful ca mod de proiectare. Aceasta permite utilizatorilor, comercianți și transportatorii, să își eficientizeze activitatea prin găsirea unor oferte viabile pentru ambele părți, cu scopul de a economisi timp și resurse energetice. Platforma web implementată vine sub forma unei soluții tehnice complete: aplicație client, server web, ce implementează endpoint-uri pe care clientul le accesează prin intermediul interfeței grafice și o bază de date în care se salvează informațiile.

Interfața client urmează conceptul de Single Page Application și este construită pentru a putea fi utilizată atât de utilizatori avansați, cât și de persoane care nu dețin cunoștiințe sau abilități în utilizarea unui dispozitiv de calcul. Prin interfața grafică și modul de interacțiune (UI/UX) implementate, navigarea în cadrul programului este intuitivă.

Aceasta comunică, utilizând cu un server web realizat utilizând framework-ul SpringBoot. Serverul este construit din mai multe microservicii care realizează diverse scopuri în cadrul aplicației: gestionarea utilizatorilor, gestionarea plășilor, gestionarea notificărilor, gestionarea rutelor din cadrul aplicației și, nu în ultimul rând, gestionarea activității efective de transport. Cel din urmă microserviciu mai sus-menționat este, totodată, și cel mai important dintre ele. Scopul acestuia este de a permite adăugarea sau ștergerea de camioane și de mărfuri, comunică cu un API extern pentru a găsi coordonatele locațiilor introduse de utilizatori, generează oferte în mod automat, cărora clienții le pot da curs sau le pot respinge.

Pentru a realiza generarea automată a ofertelor, s-a utilizat un algoritm evolutiv. Acesta permite transformarea unor cerințe reale sub forma unor cromozomi ce sunt utilizați pentru realizarea unei simulări euristice. Astfel, se pot căuta și analiza cele mai profitabile curse pentru transportatorii. Dintr-o perspectivă tehnică, acest algoritm a fost implementat deoarece realizează un compromis bun între viteza și rezultatele obținute în urma rulării acestuia. În cazul în care există un număr ridicat de mărfuri și de camioane ce trebuie împerecheate simultan, nu este fezabilă simularea tuturor transporturilor posibile pentru fiecare camion în parte. Atât comercianții, cât și transportatorii au posibilitatea de a accepta sau respinge aceste oferte.

În momentul în care o ofertă este acceptată bilateral, comerciantul are posibilitatea de a efectua plata către transportator utilizând Paypal Sandbox, o soluție de tip demonstrativ oferită de către compania Paypal. Astfel, se poate simula o plată securizată între cele două părți.

Confidențialitatea datelor presupune stocarea și transmiterea de parole prin rețea în mod securizat, prin aplicarea unui algoritm de hashing asupra accesotra. În acest mod, datele clientului se află în siguranță, integritatea acestuia fiind dificil de compromis.

Direcții viitoare de dezvoltare

Proiectul de față poate reprezenta cu succes un produs comercial care să fie pus în slujba clienților. Aplicației i se pot adăuga diverse caracteristici care să îmbunătățească satisfacția utilizatorilor, precum adăugarea unor fotografii, adăugarea unor filtre pentru oferte,

îmbunătățirea rezultatelor oferite de către algoritmul de generare a ofertelor. Cu toate acestea, stadiul actual al proiectului reprezintă o piatră de temelie pentru toate aceste capabilități, sistemul fiind construit pentru a permite o extindere facilă a acestuia.

Lecții învățate pe parcursul elaborării proiectului de diplomă

În cadrul acestui proiect a fost necesară înțelegerea nevoii de a proiecta o astfel de aplicație: definirea obiectelor ce o compun, a logicii de business a aplicației și a endpoint-urilor ce o expun, precum și aprofundarea modului de comunicare între client și servicii utilizând cereri de tip HTTP 1.1. Realizarea unei aplicații de tip client, utilizând framework-ul ReactJS, a presupus aprofundarea cunoștințelor din domeniul dezvoltării web (eng. web development): sintaxa HTML, sintaxa CSS (Cascading Style Sheets), dar și limbajul de programare JavaScript.

La nivel de server, am învățat cum se poate realiza autentificarea în mod securizat a unui utilizator, cum se pot folosi diverse framework-uri care îmbunătățesc calitatea codului (de exemplu Lombok, un framework care reduce codul suplimentar - eng. boilerplate), care realizează comunicarea cu alte componente ale unui astfel de proiect (de exemplu Hibernate, care reprezintă un ORM ce facilitează comunicarea cu baza de date MariaDB). Am aprofundat, de asemenea, cum se poate utiliza un API extern pentru a adăuga funcționalități unei aplicații proprii. Nu în ultimul rând, am învățat aplicația practică a unui algoritm din categoria inteligenței artificiale. Implementarea cu succes a unui algoritm ce are capacitatea de a prelua informații reale, stocate sub formă de obiecte, și de a le prelucra pe parcursul mai multor iterări astfel încât să producă rezultate relevante într-un timp rezonabil ce vor fi utilizate de către clienți reprezentă o realizare personală și, alături de celelalte realizări menționate anterior, reprezintă o diversificare și o dezvoltare a bagajului de cunoștințe acumulat în cei 4 ani de studiu.

Bibliografie

[1] Amazon AWS, "What is SOA (Service-Oriented Architecture)?". [Online]

Disponibil la: <https://aws.amazon.com/what-is/service-oriented-architecture/>

[2] Amazon AWS, "What are microservices" [Online]

Disponibil la: <https://aws.amazon.com/microservices/>

[3] Smartbear, "SOAP vs REST: What's the Difference?", 2023 [Online]

Disponibil la: <https://smartbear.com/blog/soap-vs-rest-whats-the-difference/>

[4] Postman, "2023 state of the API Report" [Online]

Disponibil la: <https://www.postman.com/state-of-api/api-technologies/>

[5] Redwerk, "ASP.NET Core Advantages and Disadvantages ", 2021[Online]

Disponibil la: <https://redwerk.com/blog/asp-net-core-pros-and-cons/>

[6] Altexsoft, "The Good and the Bad of Node.js Web App Programming", 2022 [Online]

Disponibil la: <https://www.altexsoft.com/blog/the-good-and-the-bad-of-node-js-web-app-development/>

[7] DEV, "FastAPI - The Good, the bad and the ugly.", 2021 [Online]

Disponibil la: <https://dev.to/fuadrafid/fastapi-the-good-the-bad-and-the-ugly-20ob>

[8] N. Schooner, Boot.dev, " The Pros and Cons of Django for Backend Development", 2023 [Online],

Disponibil la: <https://blog.boot.dev/backend/django-for-backend/>

[9] Bamboo Agile, " Pros and Cons of Using Spring Boot", 2021, [Online]

Disponibil la: <https://bambooagile.eu/insights/pros-and-cons-of-using-spring-boot>

[10] DEV, "Angular: pros and cons", 2021 [Online]

Disponibil la: <https://dev.to/siddharthshyniben/angular-pros-and-cons-m91>

[11] Altexsoft, "The Good and the Bad of Vue.js Framework Programming", 2022 [Online],
Disponibil la: <https://www.altexsoft.com/blog/pros-and-cons-of-vue-js/>

[12] Knowledgehut, " What are the Pros and Cons of React?", 2024 [Online]

Disponibil la: <https://www.knowledgehut.com/blog/web-development/pros-and-cons-of-react>

[13] IBM, "What is a relational database?", [Online]

Disponibil la: <https://www.ibm.com/topics/relational-databases>

- [14] MariaDB, "MariaDB in brief", [Online], disponibil la: <https://mariadb.org/en/#entry-header>
- [15] Geeksforgeeks, "MongoDB Advantages & Disadvantages", 2023 [Online], disponibil la: <https://www.geeksforgeeks.org/mongodb-advantages-disadvantages/>
- [16] Microsoft, "Non-relational data and NoSQL", [Online]
Disponibil la: <https://learn.microsoft.com/en-us/azure/architecture/data-guide/big-data/non-relational-data>
- [17] F. Leon, C. Bădică, *The Linear Programming Approach în An Optimization Web Service for a Freight Brokering System*, 2017, p. 329
- [18] F. Leon, *Optimizarea de tip roi de particule în Curs de Inteligență Artificială*, 2024, pp.21-30
- [19] F. Leon, C. Bădică, *The Heuristic Method în An Optimization Web Service for a Freight Brokering System*, 2017, p. 329
- [20] Apple, "BSD Overview", 2013 [Online], disponibil la
https://developer.apple.com/library/archive/documentation/Darwin/Conceptual/KernelProgramming/BSD/BSD.html#/apple_ref/doc/uid/TP30000905-CH214-TPXREF101
- [21] Apple, "About the Rosetta Translation Environment", [Online], disponibil la
<https://developer.apple.com/documentation/apple-silicon/about-the-rosetta-translation-environment>
- [22] S. Kettle, ESRI, "Distance on a sphere: The Haversine Formula" [Online]
Disponibil la <https://community.esri.com/t5/coordinate-reference-systems-blog/distance-on-a-sphere-the-haversine-formula/ba-p/902128>
- [23] A. Tudorache, *Hash Functions în Cryptography and Data Security*, pp. 3-5

Anexe

Anexa 1 - Implementarea algoritmului evolutiv pentru generarea automată a ofertelor

```

001: class Chromosome {
002:     List<Integer> gene;
003:
004:     public Chromosome() {
005:         this.gene = new ArrayList<>();
006:     }
007:
008:     public Chromosome(int numberOfCargos, List<Long> truckIDs) {
009:         this.gene = new ArrayList<>();
010:         for (int i = 0; i < numberOfCargos; i++) {
011:             var index = new Random().nextInt(truckIDs.size());
012:             gene.add(Math.toIntExact(truckIDs.get(index)));
013:         }
014:     }
015:
016:     public Chromosome(Chromosome chromosome) {
017:         this.gene = new ArrayList<>(chromosome.gene);
018:     }
019: }
020:
021: public class CargoAndDeliveryPrice {
022:     CargoDTO cargo;
023:     TruckDTO truck;
024:     double price;
025:
026:     public CargoAndDeliveryPrice(CargoDTO cargo, TruckDTO truck, double price) {
027:         this.cargo = cargo;
028:         this.truck = truck;
029:         this.price = price;
030:     }
031:
032: }
033:
034: public class TruckAndProfit {
035:     TruckDTO truck;
036:     double profit;
037:     CargoDTO cargo;
038:
039:     public TruckAndProfit(TruckDTO truck, double profit, CargoDTO cargo) {
040:         this.truck = truck;
041:         this.profit = profit;
042:         this.cargo = cargo;
043:     }
044: }
045:
046: public class ProfitAndCargo {
047:     private double profit;
048:     private CargoDTO cargo;
049:
050:     public ProfitAndCargo(double profit, CargoDTO cargo) {
051:         this.profit = profit;
052:         this.cargo = cargo;
053:     }
054:
055:     public double getProfit() {
056:         return profit;
057:     }
058:
```

```

059:     public CargoDTO getCargo() {
060:         return cargo;
061:     }
062: }
063:
064: @Override
065:     public void pairCargosTrucks(int sendMail) throws IOException {
066:         int retryCount = 0;
067:         var freeCargos = cargoService.getAllFree();
068:         var freeTrucks = truckService.getAllFree();
069:         while (freeTrucks.size() != 0 && freeCargos.size() != 0 && retryCount <
5) {
070:             //System.out.println("retryCount = " + retryCount);
071:             //System.out.println("avTrucksCount = " +
truckService.availableTrucksCount());
072:             //System.out.println("avCargosCount = " +
cargoService.availableCargoCount());
073:             Random random = new Random();
074:             List<Long> truckIDs = new ArrayList<>();
075:             List<CargoAndDeliveryPrice> cargoAndDeliveryPrice = new
ArrayList<>();
076:             List<TruckAndProfit> truckAndProfits = new ArrayList<>();
077:
078:
079:             for (var truck : freeTrucks) {
080:                 truckIDs.add(truck.getId_truck());
081:             }
082:             var chromosomes = new ArrayList<Chromosome>();
083:             for (int i = 0; i < 100; i++) {
084:                 chromosomes.add(new Chromosome(cargoService.getAllFree().size(),
truckIDs));
085:             }
086:             //Inceput algoritm
087:             for (int round = 0; round < 100; round++) {
088:                 var parent1 = chromosomes.get(random.nextInt(0,
chromosomes.size()));
089:                 var parent2 = chromosomes.get(random.nextInt(0,
chromosomes.size()));
090:                 Chromosome parent = new Chromosome(parent1);
091:
092:                 //Incrucisare
093:
094:                 var chance = random.nextInt(0, 100);
095:                 if (chance < 90) {
096:                     var parentAux = new Chromosome();
097:                     var threshold = random.nextInt(0, 100);
098:                     for (int i = 0; i < parent1.gene.size(); i++) {
099:                         var alpha = random.nextInt(0, 100);
100:                         if (alpha < threshold) {
101:                             parentAux.gene.add(parent1.gene.get(i));
102:                         } else {
103:                             parentAux.gene.add(parent2.gene.get(i));
104:                         }
105:                     }
106:                     parent = new Chromosome(parentAux);
107:                 }
108:                 //Mutatie
109:                 if (chance < 5) {
110:                     int reset_size = parent.gene.size() / 3;
111:                     for (int i = 0; i < reset_size; i++) {
112:                         parent.gene.set(random.nextInt(0, parent.gene.size()),
Math.toIntExact(truckIDs.get(random.nextInt(0, truckIDs.size()))));
113:                     }
114:                 }
115:
116:                 //Simulare
117:                 //System.out.println("Gene: " + parent.gene);
118:                 int i = 0;
119:                 for (var cargo : freeCargos) {
120:                     var truck = truckService.findByIdTruck((long)

```

```

(parent.gene.get(i)));
121:         if (truck.isPresent() &&
truck.get().getStatus().trim().equalsIgnoreCase("Liber")
122:             &&
cargo.getStatus().trim().equalsIgnoreCase("NeonoratfÉ")) {
123:                 //Daca depaseste greutatea
124:                     if (cargo.getGreutate() > truck.get().getTonajMaxim() *
0.9) {
125:                         //Da, depaseste
126:                         if (round == 0)
127:                             cargoAndDeliveryPrice.add(new
CargoAndDeliveryPrice(cargo, truck.get(), Double.POSITIVE_INFINITY));
128:                             //System.out.println("\n");
129:                             //System.out.println("Nu este fezabil: tonaj
deposit");
130:                             //System.out.println("Gr. cargo: " +
cargo.getGreutate() + " Gr.camion: " + truck.get().getTonajMaxim());
131:
132:                         } else {//Daca depaseste gabaritul
133:                             if (cargo.getVolum() > truck.get().getVolum()) {
134:                                 //Da, depaseste
135:                                 if (round == 0)
136:                                     cargoAndDeliveryPrice.add(new
CargoAndDeliveryPrice(cargo, truck.get(), Double.POSITIVE_INFINITY));
137:                                     //System.out.println("\n");
138:                                     //System.out.println("Nu este fezabil: gabarit
deposit");
139:                                     //System.out.println("Volum cargo: " +
cargo.getVolum() + " Volum camion: " + truck.get().getVolum());
140:
141:                             } else {
142:                                 //Conditii indeplinite
143:                                 double pretCursa = cargo.getDistanta_cursa() *
truck.get().getPretKM();
144:                                 if (round == 0)
145:                                     cargoAndDeliveryPrice.add(new
CargoAndDeliveryPrice(cargo, truck.get(), pretCursa));
146:                                     else {
147:                                         if (pretCursa <
cargoAndDeliveryPrice.get(i).price) {
148:                                             cargoAndDeliveryPrice.set(i, new
CargoAndDeliveryPrice(cargo, truck.get(), pretCursa));
149:                                         }
150:                                         }
151:                                         double distantaCamionLaPctStart =
calculateDistance(truck.get().getLatitudineLocatieInitiala(),
truck.get().getLongitudineLocatieInitiala(), cargo.getLatitudine_pct_ridicare(),
cargo.getLongitudine_pct_ridicare());
152:                                         //System.out.println("\n");
153:                                         //System.out.println("Fezabil: dist. start camion
- start cursa:\n" +
154:                                         //           "de la " +
truck.get().getLocatieInitiala() + " la " + cargo.getAdresa_ridicare() + " : " +
distanțaCamionLaPctStart);
155:
156:                                         }
157:                                         }
158:                                         }
159:                                         i++;
160:                                         }
161:
162:                                         }
163:                                         //Sfarsit simulare
164:
165:                                         //System.out.println("\n\n\n\nDupa cele 3 runde:\n");
166:                                         /*
167:                                         for (var cargo : cargoAndDeliveryPrice) {
168:
169:                                         System.out.println("ID Cargo: " + cargo.cargo.getId_cargo());
170:                                         System.out.println("ID Camion: " + cargo.truck.getId_truck());
171:                                         System.out.println("Pret: " + cargo.price);

```

```

172:
173:         } */
174:
175:         //Calculare profit camion
176:         for (var cargo : cargoAndDeliveryPrice) {
177:             if (cargo.price != Double.POSITIVE_INFINITY) {
178:                 var truckOptional =
truckService.findByIdTruck(cargo.truck.getId_truck());
179:                 if (truckOptional.isPresent()) {
180:                     var truck = truckOptional.get();
181:                     double distantaCamionLaPctStart =
calculateDistance(truck.getLatitudeLocatieInitiala(),
truck.getLongitudeLocatieInitiala(), cargo.cargo.getLatitude_pct_ridicare(),
cargo.cargo.getLongitude_pct_ridicare());
182:                     double profit = truck.getPreKM() *
cargo.cargo.getDistanta_cursa() -
183:                         truck.getCostTransport() *
(cargo.cargo.getDistanta_cursa() + distantaCamionLaPctStart);
184:
185:                     //System.out.println("Profit camion ID:" +
truck.getId_truck() + " - " + profit);
186:
187:                     truckAndProfits.add(new TruckAndProfit(truck, profit,
cargo.cargo));
188:                 }
189:             }
190:         }
191:
192:         Map<Long, ProfitAndCargo> maxProfits = new HashMap<>();
193:         for (var truckAndProfit : truckAndProfits) {
194:             if (maxProfits.containsKey(truckAndProfit.truck.getId_truck())) {
195:                 ProfitAndCargo current =
maxProfits.get(truckAndProfit.truck.getId_truck());
196:                 if (truckAndProfit.profit > current.getProfit()) {
197:                     maxProfits.put(truckAndProfit.truck.getId_truck(), new
ProfitAndCargo(truckAndProfit.profit, truckAndProfit.cargo));
198:                 }
199:             } else {
200:                 maxProfits.put(truckAndProfit.truck.getId_truck(), new
ProfitAndCargo(truckAndProfit.profit, truckAndProfit.cargo));
201:             }
202:         }
203:
204:
205:         //Afisare profit camion
206:         //System.out.println("\n\n\n\nProfituri pentru camioane:\n");
207:         //for (var entry : maxProfits.entrySet()) {
208:             //System.out.println("ID Camion: " + entry.getKey() + ", Profit: " +
entry.getValue());
209:         //}
210:
211:         //Rezervare camion
212:         var firstTruckOptional = maxProfits.entrySet().stream().findFirst();
213:         if (firstTruckOptional.isPresent()) {
214:             Long truckId = firstTruckOptional.get().getKey();
215:             CargoDTO cargo = maxProfits.get(truckId).getCargo();
216:             truckService.reserveTruckByIdIfPossible(truckId);
217:             cargoService.reserveCargoByIdIfPossible(cargo.getId_cargo());
218:             var cost = cargo.getDistanta_cursa() *
truckService.findByIdTruck(truckId).getPreKM();
219:             cargoTruckAssignmentService.save(new CargoTruckAssignmentDTO(new
CargoTruckAssignmentID(cargo.getId_cargo(), truckId),
firstTruckOptional.get().getValue().profit, cost));
220:             //notificam cele doua parti!
221:             if (sendMail == 1) {
222:
sendNotification(customerService.findById(cargo.getId_comerçant()).get().getEmail(),
"Ofertă nouă!", "Veți avea primit o ofertă nouă! Vizitați pagina noastră pentru a o vizualiza!");
223:

```

```

sendNotification(providerService.findById((truckService.findByIdTruck(truckId)).get() .
getIdTransportator().get().getEmail(), "Ofertă nouă!", "Vești bune! Așa că primite o
ofertă nouă! Vizitați pagina noastră pentru a o vizualiza!");
224:         }
225:         //System.out.println("Camion ID: " + truckId + " cu profit
"+firstTruckOptional.get().getValue().profit+" ia Cargo ID: " + cargo.getId_cargo()+" cu costul "+cost );
226:         } else {
227:             retryCount++;
228:             System.out.println("n-am gasit nimic");
229:         }
230:         freeCargos = cargoService.getAllFree();
231:         freeTrucks = truckService.getAllFree();
232:     }
233:     System.out.println("Am terminat pair-ul");
234: }
235:
236: @Override
237: public void releaseAll() {
238:     truckService.releaseTrucksIfPossible();
239:     cargoService.releaseCargosIfPossible();
240:     cargoTruckAssignmentService.deleteAll();
241: }
242:
243: @Override
244: public Object getOfferForCustomer(long idCargo, long idCustomer) {
245:     var cargo = cargoService.findByIdCargo(idCargo);
246:     if (cargo.isPresent()) {
247:         if (cargo.get().getId_comeriant() != idCustomer)
248:             return null;
249:     } else {
250:         var cargosAndTrucks =
cargoTruckAssignmentService.getAllByIdCargo(idCargo);
251:         var cargoAndTruck =
cargoTruckAssignmentService.getAllByIdCargo(idCargo).stream().findFirst().get();
252:         var truck =
truckService.findByIdTruck(cargoAndTruck.getId().getId_truck());
253:         if (truck.isPresent()) {
254:             var provider =
providerService.findById(truck.get().getIdTransportator());
255:             if (provider.isPresent())
256:                 //mai e de tratat cazul in care a fost deja acceptata
cursa de catre transportator
257:                 return new ProviderAndCostDTO(provider.get(),
cargoAndTruck.getCost(), cargoAndTruck.getAccepted_by_client(),
cargoAndTruck.getAccepted_by_provider());
258:         }
259:     }
260:     return null;
261:
262: }
263: return null;
264: }

```

Anexa 2 - Metode pentru găsirea coordonatelor unei locații și calcularea distanței dintre două locații pe baza coordonatelor acestora

```
01: @Override
02:     public double[] getCoordinates(String location) throws IOException {
03:         String urlString =
"https://nominatim.openstreetmap.org/search?format=json&q=" + location.replace(" ", "%20");
04:         URL url = new URL(urlString);
05:         HttpURLConnection conn = (HttpURLConnection) url.openConnection();
06:         conn.setRequestMethod("GET");
07:
08:         Scanner scanner = new Scanner(conn.getInputStream());
09:         StringBuilder response = new StringBuilder();
10:         while (scanner.hasNext()) {
11:             response.append(scanner.nextLine());
12:         }
13:         scanner.close();
14:
15:         JSONArray jsonArray = new JSONArray(response.toString());
16:         JSONObject jsonObject = jsonArray.getJSONObject(0);
17:         double latitude = Double.parseDouble(jsonObject.getString("lat"));
18:         double longitude = Double.parseDouble(jsonObject.getString("lon"));
19:
20:         return new double[]{latitude, longitude};
21:     }
22:
23:     @Override
24:     public double calculateDistance(double lat1, double lon1, double lat2, double
lon2) {
25:         double latDistance = Math.toRadians(lat2 - lat1);
26:         double lonDistance = Math.toRadians(lon2 - lon1);
27:         double a = Math.sin(latDistance / 2) * Math.sin(latDistance / 2)
+ Math.cos(Math.toRadians(lat1)) * Math.cos(Math.toRadians(lat2))
* Math.sin(lonDistance / 2) * Math.sin(lonDistance / 2);
28:         double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
29:         return 6371 * c;
30:     }
31:
32: }
```

Anexa 3 - Acceptarea sau respingerea unei oferte

```

01: @Override
02: public Object acceptOfferForProvider(long idTruck, long idProvider) throws
IOException {
03:     var truck = truckService.findByIdTruck(idTruck);
04:     if (truck.isPresent()) {
05:         if (truck.get().getIdTransportator() != idProvider) {
06:             return null;
07:         } else {
08:             var offer =
cargoTruckAssignmentService.getAllByIdTruck(idTruck).stream().findFirst();
09:             if (offer.isPresent()) {
10:                 if (offer.get().getAccepted_by_provider() != 1) {
11:                     offer.get().setAccepted_by_provider(1);
12:
sendNotification(customerService.findById(cargoService.findByIdCargo(offer.get().getId
() .getId_cargo()).get() .getId_comerclant()).get() .getEmail(), "Ofertă acceptată",
"Vești bune! Vă anunțăm că oferta dvs. a fost acceptată de către transportator!");
13:                     return "OK";
14:                 }
15:             } else return null;
16:         }
17:         return null;
18:
19:     }
20:     return null;
21: }
22:
23: @Override
24: public Object rejectOfferForProvider(long idTruck, long idProvider) throws
IOException {
25:     var truck = truckService.findByIdTruck(idTruck);
26:     if (truck.isPresent()) {
27:         if (truck.get().getIdTransportator() != idProvider) {
28:             return null;
29:         } else {
30:             var offer =
cargoTruckAssignmentService.getAllByIdTruck(idTruck).stream().findFirst();
31:             if (offer.isPresent()) {
32:                 var cargo =
cargoService.findByIdCargo(offer.get().getId() .getId_cargo());
33:                 if (cargo.isPresent()) {
34:                     cargo.get().setStatus("NeonoratfÉ");
35:                     cargoTruckAssignmentService.deleteByIdTruck(idTruck);
36:                     truck.get().setStatus("Liber");
37:
sendNotification(customerService.findById(cargo.get() .getId_comerclant()).get() .getEmail(),
"Ofertă refuzată", "Vești proaste! Vă anunțăm că oferta dvs. a fost refuzată de
către transportator!");
38:                     return "OK";
39:                 }
40:             } else return null;
41:         }
42:     }
43:     return null;
44: }
45:
46: }
47: return null;
48: }

```

Anexa 4 - Trimiterea unui e-mail

```
01: @Autowired
02: private JavaMailSender javaMailSender;
03:
04: @Override
05: public void sendMail(String recipient, String subject, String content)
{
06:     try {
07:         SimpleMailMessage mailMessage = new SimpleMailMessage();
08:         mailMessage.setFrom("marian4505@gmail.com");
09:         mailMessage.setTo(recipient);
10:         mailMessage.setText(content);
11:         mailMessage.setSubject(subject);
12:         javaMailSender.send(mailMessage);
13:     } catch (Exception e) {
14:         System.out.println(e);
15:     }
16: }
```

Anexa 5 - Funcționarea microserviciului de gateway

```
01: //din Controller
02: @GetMapping("/idm/getmycargo")
03: public ResponseEntity<?> getMyCargo(@RequestHeader("Authorization") String token)
throws IOException {
04:     int idCustomer = gatewayService.getIdCustomer(token);
05:     if (idCustomer != -1) {
06:         String myCargoList = gatewayService.getMyCargoRequest(idCustomer);
07:         if (!myCargoList.equals("Error")) {
08:             return ResponseEntity.status(HttpStatus.OK).body(myCargoList);
09:         } else {
10:             return ResponseEntity.status(HttpStatus.NO_CONTENT).body("Lista este
goală");
11:         }
12:
13:     } else {
14:         return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("Token
invalid");
15:     }
16: }
17: //din Service
18: public String getMyCargoRequest(long idCustomer) throws IOException {
19:     URL url = new
URL("http://"+getIp()+":8082/api/freightbroker/brokerservice/getmycargo?id_customer="
+ idCustomer);
20:     HttpURLConnection con = (HttpURLConnection) url.openConnection();
21:     con.setRequestMethod("GET");
22:     if(con.getResponseCode() == 200) {
23:         BufferedReader in = new BufferedReader(new
InputStreamReader(con.getInputStream()));
24:         StringBuilder response = new StringBuilder();
25:         String inputLine;
26:         while ((inputLine = in.readLine()) != null) {
27:             response.append(inputLine);
28:         }
29:         in.close();
30:         return response.toString();
31:     }else{
32:         return "Error";
33:     }
34: }
```

Anexa 6 - Autentificarea în aplicație: componenta server

```

001: @PostMapping(value = "/authenticate", consumes = "application/x-protobuf",
produces = "application/json")
002: public ResponseEntity<?> authenticate(@RequestBody byte[] data) throws
InvalidProtocolBufferException {
003:
004:     Identity.AuthenticationRequest authRequest =
005:         Identity.AuthenticationRequest.parseFrom(data);
006:
007:     final String[] token = new String[1];
008:
009:     io.grpc.stub.StreamObserver<Identity.AuthenticationResponse> responseObserver
= new StreamObserver<>() {
010:         String auxToken;
011:
012:         @Override
013:             public void onNext(Identity.AuthenticationResponse
authenticationResponse) {
014:                 auxToken = authenticationResponse.getToken();
015:             }
016:
017:             @Override
018:                 public void onError(Throwable throwable) {
019:                     System.out.println("Eroare la autentificare: " +
throwable.getMessage());
020:
021:                 }
022:
023:                 @Override
024:                     public void onCompleted() {
025:                         token[0] = auxToken;
026:                     }
027:                 };
028:
029:
030:     authenticationService.authenticate(authRequest, responseObserver);
031:     if (token[0] != null && !token[0].equals("invalid-token")) {
032:         return ResponseEntity.ok(token[0]);
033:     } else {
034:         return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("Token
invalid");
035:     }
036: }
037:
038:
039: @Override
040:     public void authenticate(Identity.AuthenticationRequest request,
041:
io.grpc.stub.StreamObserver<Identity.AuthenticationResponse> responseObserver) {
042:
043:     BCryptPasswordEncoder encoder = new BCryptPasswordEncoder();
044:     if (request.getUserType().trim().equalsIgnoreCase("Comerçant")) {
045:         Optional<CustomerDTO> data =
customerService.findByEmail(request.getEmail());
046:         if (data.isPresent() && encoder.matches(request.getPassword(),
data.get().getParola())) {
047:             String jws = Jwts.builder()
048:                 .claim("id_customer", data.get().getId_customer())
049:                 .claim("email", data.get().getEmail())
050:                 .claim("user_type", "comerçant")
051:                 .setIssuedAt(Date.from(Instant.now()))
052:                 .setExpiration(Date.from(Instant.now().plusSeconds(3600 *
3)))
053:                 .signWith(
SignatureAlgorithm.HS256,
054:                     Base64.getDecoder().decode("Yn2kjbddFAWtnPJ2AF1L8WXmohJMCvigQggaEypa5E="))

```

```

055:                     )
056:                     .compact();
057:             Claims claims = validateToken(jws);
058:
059:             if (claims != null) {
060:                 Identity.AuthenticationResponse result =
Identity.AuthenticationResponse.newBuilder().setToken(jws).build();
061:                 responseObserver.onNext(result);
062:                 responseObserver.onCompleted();
063:             } else {
064:
responseObserver.onError(Status.UNAUTHENTICATED.withDescription("Token validation
failed").asRuntimeException());
065:             }
066:         } else {
067:
responseObserver.onError(Status.UNAUTHENTICATED.withDescription("Token validation
failed").asRuntimeException());
068:         }
069:
070:     } else if
(request.getUserType().trim().equalsIgnoreCase("Transportator")) {
071:         Optional<ProviderDTO> data =
providerService.findByEmail(request.getEmail());
072:         if (data.isPresent() && encoder.matches(request.getPassword(),
data.get().getParola())) {
073:             String jws = Jwts.builder()
                    .claim("id_customer", data.get().getId_customer())
                    .claim("email", data.get().getEmail())
                    .claim("user_type", "transportator")
                    .setIssuedAt(Date.from(Instant.now()))
                    .setExpiration(Date.from(Instant.now()).plusSeconds(3600 *
3)))
                    .signWith(
                        SignatureAlgorithm.HS256,
Base64.getDecoder().decode("Yn2kjabddFAWtnPJ2AF1L8WXmohJMCvigQggaEypa5E=")
081:                     )
082:                     .compact();
083:
Claims claims = validateToken(jws);
084:
085:
086:             if (claims != null) {
087:                 Identity.AuthenticationResponse result =
Identity.AuthenticationResponse.newBuilder().setToken(jws).build();
088:                 responseObserver.onNext(result);
089:                 responseObserver.onCompleted();
090:             } else {
091:
responseObserver.onError(Status.UNAUTHENTICATED.withDescription("Token validation
failed").asRuntimeException());
092:             }
093:
094:         } else {
095:
responseObserver.onError(Status.UNAUTHENTICATED.withDescription("Token validation
failed").asRuntimeException());
096:         }
097:     }
098: }
099: private Claims validateToken(String token) {
100:     try {
101:         return Jwts.parser()
.setSigningKey(Base64.getDecoder().decode("Yn2kjabddFAWtnPJ2AF1L8WXmohJMCvigQggaEypa5E
="))
103:             .parseClaimsJws(token)
104:             .getBody();
105:     } catch (JwtException e) {
106:         return null; } }
```

Anexa 7 - Efectuarea plășilor²

```

01: @PostMapping("/pay")
02:     public ResponseEntity<String> payment(@RequestParam(name="cost") Double cost,
03: @RequestParam(name="email") String email,@RequestParam(name="id_cargo") Long id_cargo)
04:     {
05:         try {
06:             Payment payment = service.createPayment(cost, "EUR", "Platf  prin
platforma Freightbroker",email,id_cargo);
07:             for(Links link:payment.getLinks()) {
08:                 if(link.getRel().equals("approval_url")) {
09:                     System.out.println(link.getHref());
10:                     return ResponseEntity.ok(link.getHref());
11:                 }
12:             } catch (Exception e) {
13:                 System.out.println(e);
14:             }
15:             return null;
16:         }
17:
18:         @GetMapping("/success")
19:         public ResponseEntity<Void> successPay(@RequestParam("paymentId") String
paymentId, @RequestParam("PayerID") String payerId, @RequestParam("payeeEmail") String
payeeEmail, @RequestParam("total") Double total,@RequestParam("id_cargo") Long
id_cargo) {
20:             try {
21:                 var result = service.executePayment(paymentId, payerId,payeeEmail,
total);
22:                 if (result){
23:                     return
ResponseEntity.status(HttpStatus.FOUND).location(URI.create("http://localhost:3000/pay
mentok")).build();
24:                 }
25:                 else
26:                     return
ResponseEntity.status(HttpStatus.FOUND).location(URI.create("http://localhost:3000/pay
menterror")).build();
27:
28:             } catch (Exception e) {
29:                 System.out.println(e);
30:                 return
ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).location(URI.create("http://lo
calhost:3000/payment-error")).build();
31:             }
32:         }
33:
34:         @GetMapping("/cancel")
35:         public ResponseEntity<String> cancelPay(@RequestParam(name = "token") String
token) {
36:             return ResponseEntity.ok("Plata anulata");
37:         }
38:
39: @Autowired
40: PaymentRepository paymentRepository;
41:
42: private static final String INTENT = "SALE";
43: private static final String METHOD = "paypal";
44:
45: @Autowired
46: private APIContext apiContext;
47:
48: public Payment createPayment(Double total, String currency, String description,
String payeeEmail, Long id_cargo) throws PayPalRESTException {

```

² Cod sursă orginal preluat de la <https://github.com/thepeachkhoukha/paypaldemo> și modificat ulterior.

```
49:     Double auxTotal = total;
50:     Amount amount = new Amount();
51:     amount.setCurrency(currency);
52:     total = new BigDecimal(total).setScale(2, RoundingMode.HALF_UP).doubleValue();
53:     amount.setTotal(String.format("%.2f", total));
54:     Transaction transaction = new Transaction();
55:
56:     transaction.setDescription(description);
57:     transaction.setAmount(amount);
58:     Payee payee = new Payee();
59:     payee.setEmail(payeeEmail);
60:     transaction.setPayee(payee);
61:     List<Transaction> transactions = new ArrayList<>();
62:     transactions.add(transaction);
63:
64:     Payer payer = new Payer();
65:     payer.setPaymentMethod(METHOD);
66:
67:     Payment payment = new Payment();
68:     payment.setIntent(INTENT);
69:     payment.setPayer(payer);
70:     payment.setTransactions(transactions);
71:
72:     RedirectUrls redirectUrls = new RedirectUrls();
73:     redirectUrls.setCancelUrl("http://localhost:8085/api/freightbroker/cancel");
74:
75:     redirectUrls.setReturnUrl("http://localhost:8085/api/freightbroker/success?payeeEmail=
" + payeeEmail + "&total=" + auxTotal+"&id_cargo="+id_cargo);
76:     payment.setRedirectUrls(redirectUrls);
77:
78:     return payment.create(apiContext);
79:
80:
81: public boolean executePayment(String paymentId, String payerId, String payeeEmail,
Double total) throws PayPalRESTException {
82:     try {
83:         var result = paymentRepository.findById(new PaymentIdDTO(payerId,
payeeEmail, total));
84:         if (result.isEmpty()) {
85:             Payment payment = new Payment();
86:             payment.setId(paymentId);
87:             PaymentExecution paymentExecute = new PaymentExecution();
88:             paymentExecute.setPayerId(payerId);
89:             payment.execute(apiContext, paymentExecute);
90:             paymentRepository.save(new PaymentDTO(new PaymentIdDTO(payerId,
payeeEmail, total)));
91:             return true;
92:         } else return false;
93:     } catch (Exception e) {
94:         return false;
95:     }
96: }
```

Anexa 8 - Crearea unui cont nou din aplicația web client

```

001: const handleSubmit = async (e) => {
002:
003:     var response = undefined;
004:     e.preventDefault();
005:
006:     if (userType === "Comerçiant") {
007:         if (parola === confirmParola && parola !== '' && confirmParola !== '') {
008:
009:             if
010:                 (!email.toLowerCase().match(/^(([^\<()[]\.\,\;\:\s@"]+(\.[^\<()[]\.\,\;\:\s@"]+)*|\.(^+))@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\])|(([a-zA-Z\-\-0-9]+\.)+[a-zA-Z]{2,}))$)) {
011:                     alert('Adresa de e-mail este invalidă. Încercați din nou!');
012:                     setEmail('');
013:                 } else if (!/[0-9]*$/.test(telefon)) {
014:                     alert('Numărul de telefon nu este valid. Încercați din nou!');
015:                     setTelefon('');
016:                 }
017:             const hashedPwd = await bcrypt.hash(parola, 10);
018:             const user = {
019:                 nume,
020:                 prenume,
021:                 email,
022:                 telefon,
023:                 parola: hashedPwd
024:             };
025:             try {
026:                 response = await fetch('http://' + ipAddress +
027:                                         ':8080/api/freightbroker/createCustomer', {
028:                             method: 'POST',
029:                             headers: {
030:                                 'Content-Type': 'application/json',
031:                             },
032:                             body: JSON.stringify(user),
033:                         });
034:
035:             if (response.ok) {
036:                 await response.json();
037:                 alert('Utilizator adăugat cu succes!');
038:                 navigate('/');
039:             } else {
040:                 alert('Utilizatorul nu a fost creat! Incercati din nou');
041:             }
042:         } catch (error) {
043:             console.error(error);
044:         }
045:     }
046:
047:     } else {
048:         alert('Câmpurile "Parolă" și "Confirmare parolă" nu coincid. Încercați din nou!');
049:         setParola('');
050:         setConfirmParola('');
051:     }
052: } else {
053:     if (parola === confirmParola) {
054:         if
055:             (!email.toLowerCase().match(/^(([^\<()[]\.\,\;\:\s@"]+(\.[^\<()[]\.\,\;\:\s@"]+)*|\.(^+))@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\])|(([a-zA-Z\-\-0-9]+\.)+[a-zA-Z]{2,}))$)) {
056:                 alert('Adresa de e-mail este invalidă. Încercați din nou!');
057:             }
058:     }
059: }

```

```
056:         setEmail('');
057:     } else if (!/^[0-9]*$/.test(telefon)) {
058:         alert('Numărul de telefon nu este valid. Încercați din nou!');
059:         setTelefon('');
060:         return;
061:     } else {
062:         const hashedPwd = await bcrypt.hash(parola, 10);
063:         const user = {
064:             nume,
065:             prenume,
066:             email,
067:             telefon,
068:             adresa: adresaTransportator,
069:             parola: hashedPwd
070:         };
071:
072:         try {
073:             response = await fetch('http://' + ipAddress +
074: ':8080/api/freightbroker/createProvider', {
075:                 method: 'POST',
076:                 headers: {
077:                     'Content-Type': 'application/json',
078:                 },
079:                 body: JSON.stringify(user),
080:             });
081:
082:             if (response.ok) {
083:                 await response.json();
084:                 alert('Utilizator adaugat cu succes!');
085:                 navigate('/');
086:
087:             } else {
088:                 alert('Utilizatorul nu a fost creat! Incercati din nou');
089:             }
090:             catch (error) {
091:                 console.error(error);
092:             }
093:         } else {
094:             alert('Campurile "Parola" si "Confirmare parola" nu coincid. Incercati din nou!');
095:             setParola('');
096:             setConfirmParola('');
097:         }
098:     }
099:
100: }
```