

MINISTERUL EDUCAȚIEI ȘI CERCETĂRII ȘTIINȚIFICE



UNIVERSITATEA TEHNICĂ

DIN CLUJ-NAPOCA

**FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL CALCULATOARE**

Proiect final

Proiectare software

-Agenție de turism-

Autor: Buda Andreea Rodica

Grupa: 30231

Cuprins

1.Problema de rezolvat.....	- 4 -
2.Faza de analiză – Diagrama cazurilor de utilizare.....	- 5 -
2.1.Enunțul problemei	- 5 -
2.2.Diagrama cazurilor de utilizare	- 5 -
3.Faza de proiectare – Diagramele de clase	- 7 -
3.1.Diagrama entitate-relație	- 7 -
3.2.Diagrama completă de clase.....	- 10 -
3.2.1.Diagrama de clase-service	- 10 -
3.2.2.Diagrama de clase-client.....	- 11 -
3.3.Șabloane	- 11 -
Proxy Pattern:.....	- 11 -
Abstract Factory Method:	- 12 -
Factory Method:.....	- 12 -
Singleton Pattern:	- 12 -
3.4.Pachetul Model.....	- 13 -
Implementările Specifice pentru Spring Boot și React:	- 14 -
3.5.Subpachetul Repository.....	- 15 -
3.6.Pachetul Controller	- 16 -
3.7. Pachetul Service.....	- 18 -
4.Faza de implementare – Aplicația.....	- 19 -
4.1.Instrumente utilizate	- 19 -
4.2. Aplicația	- 20 -

1.Problema de rezolvat

Obiectivul acestei teme constă în implementarea unei aplicații client/server folosind minim 5 șabloane de proiectare (inclusiv unul creațional, unul comportamental și unul structural) și o arhitectură orientată pe servicii (SOA) pentru comunicarea între aplicația server și aplicația client. Pentru persistența informației, se va utiliza o bază de date relațională (SQL Server, MySQL, etc.).

Cerințele sunt:

- Transformarea aplicației implementate la tema anterioară într-o aplicație client/server.
- Utilizarea minim a 5 șabloane de proiectare, inclusiv unul creațional, unul comportamental și unul structural.
- Implementarea unei arhitecturi orientate pe servicii (SOA) pentru comunicarea între aplicația server și aplicația client.

➤ Faza de Analiză:

În această fază, se va realiza o diagramă a cazurilor de utilizare și diagramele de activități pentru fiecare caz de utilizare. Numărul diagramelor de activități trebuie să fie egal cu numărul de cazuri de utilizare din diagrama cazurilor de utilizare.

➤ Faza de Proiectare:

În această fază se vor realiza:

Două diagrame de clase corespunzătoare aplicației server și aplicației client, respectând principiile DDD și folosind o arhitectură orientată pe servicii (SOA) și minim 5 șabloane de proiectare.

Diagrama entitate-relație corespunzătoare bazei de date.

Diagramă de secvență corespunzătoare tuturor cazurilor de utilizare. Numărul diagramelor de secvență trebuie să fie cel puțin egal cu numărul de cazuri de utilizare din diagrama cazurilor de utilizare.

➤ Faza de Implementare:

În această etapă se va scrie codul pentru îndeplinirea tuturor funcționalităților precizate de diagrama cazurilor de utilizare, utilizând proiectarea dată de diagramele de clase și diagramele de secvență și unul dintre următoarele limbaje de programare: C#, C++, Java, Python.

➤ Finalizarea Proiectului:

Finalizarea temei va consta în predarea unui director care va cuprinde:

- Un fișier cu diagramele UML realizate.
- Baza de date.
- Aplicația software.
- Documentația (minim 20 pagini):
- Numele studentului, grupa.
- Enunțul problemei.
- Instrumentele utilizate.
- Justificarea limbajului de programare ales.
- Descrierea diagramelor UML (inclusiv figuri cu diagramele UML realizate).

2.Faza de analiză – Diagrama cazurilor de utilizare

2.1.Enunțul problemei

-Problema 14-

Dezvoltați o aplicație client/server care poate fi utilizată de către o agenție turistică. Aplicația client va avea 3 tipuri de utilizatori: client, angajat și administrator.

Utilizatorii de tip client pot efectua următoarele operații fără autentificare:

- ❖ Vizualizarea listei pachetelor oferite de agenție, sortată după destinație și perioadă (inclusiv redarea unor imagini cu pachetele turistice, între 1 și 3 imagini pentru fiecare pachet).
- ❖ Filtrarea listei pachetelor după destinație, perioadă și preț.

Utilizatorii de tip angajat pot efectua următoarele operații după autentificare:

- ❖ Toate operațiile permise utilizatorilor de tip client.
- ❖ Operații CRUD (Create, Read, Update, Delete) pentru persistența pachetelor oferite de agenție.
- ❖ Rezervarea unui pachet de către un client.
- ❖ Operații CRUD pentru informațiile legate de utilizatorii de tip client.
- ❖ Vizualizarea listei pachetelor rezervate într-o perioadă specificată;
- ❖ Salvarea listelor cu informații despre pachetele oferite de agenție în mai multe formate: CSV, JSON, XML, DOC.
- ❖ Vizualizarea unor statistici legate de pachetele turistice utilizând grafice (structură radială, structură inelară, de tip coloană etc.).

Utilizatorii de tip administrator pot efectua următoarele operații după autentificare:

- ❖ Operații CRUD pentru informațiile legate de utilizatorii care necesită autentificare.
- ❖ Vizualizarea listei utilizatorilor care necesită autentificare și filtrarea acestora după tipul utilizatorilor.
- ❖ Notificarea fiecărui utilizator care necesită autentificare prin cel puțin 2 variante (email, SMS, WhatsApp, Skype, etc.) la orice modificare a informațiilor de autentificare aferente acelui utilizator.
- ❖ Interfața grafică a aplicației va fi disponibilă în cel puțin trei limbi de circulație internațională.

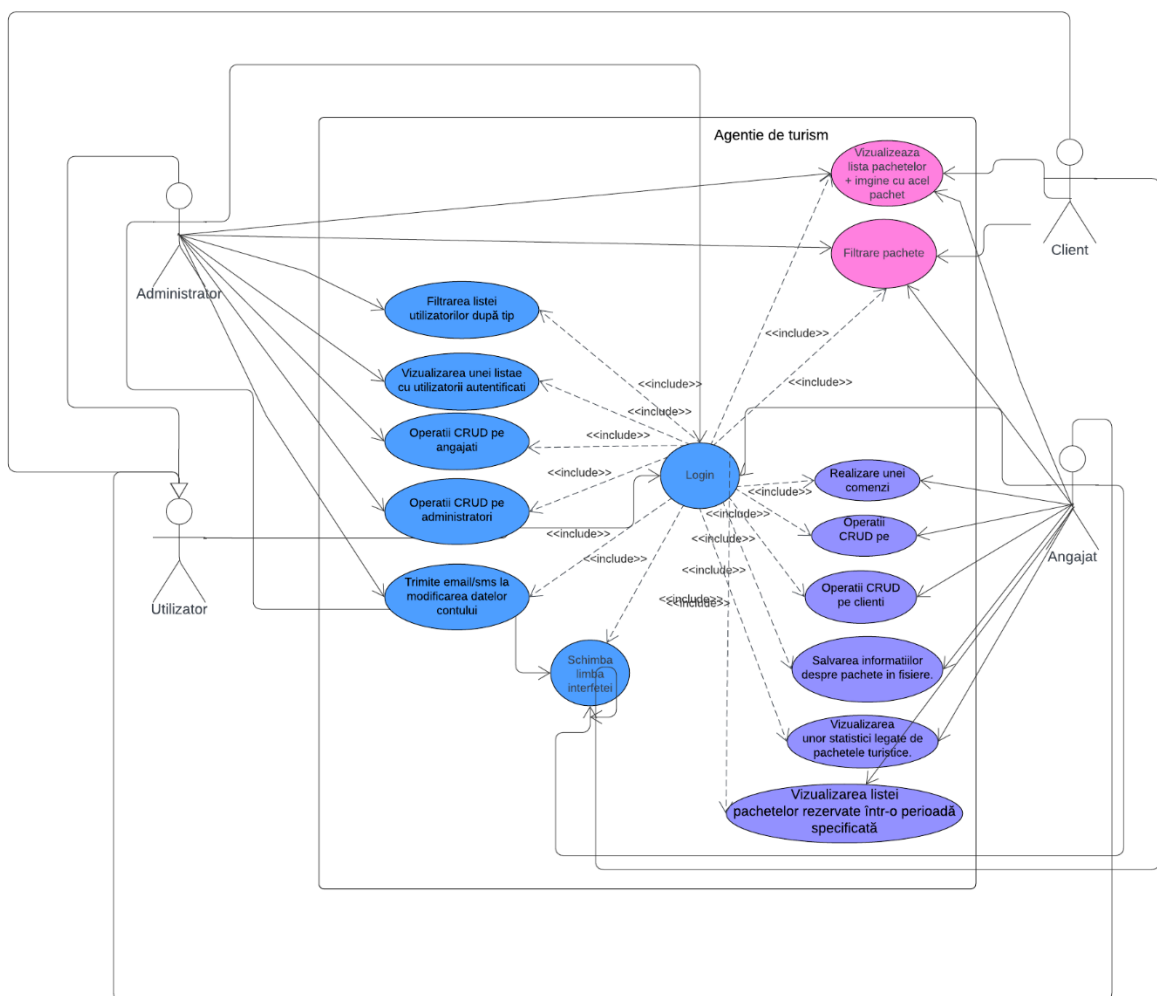
2.2.Diagrama cazurilor de utilizare

Diagrama cazurilor de utilizare pentru aplicația agenției turistice este o reprezentare grafică a interacțiunilor dintre diferiții actori și sistemul software. Această diagramă prezintă acțiunile pe care fiecare actor le poate întreprinde în cadrul aplicației, evidențiind și restricțiile de autentificare.

Din diagrama cazurilor de utilizare reiese că există trei categorii principale de utilizatori în cadrul aplicației agenției turistice: CLIENT, ANGAJAT și ADMINISTRATOR, acestea fiind divizate din actorul principal, UTILIZATOR. Fiecare categorie de utilizatori are acces la anumite funcționalități, care sunt reprezentate ca și cazuri de utilizare distincte.

- ANGAJAT: Utilizator al aplicației care are privilegii extinse și poate accesa funcționalități suplimentare după autentificare.
- ADMINISTRATOR: Utilizator cu privilegii maxime în sistem, capabil să gestioneze toate aspectele aplicației după autentificare.
- CLIENT: Utilizator al aplicației care poate accesa anumite funcționalități fără a fi nevoie de autentificare.

Use case-urile fundamentale sunt "Vizualizarea listei pachetelor + imagini cu acestea" și "Filtrare pachete", care sunt disponibile pentru toți utilizatorii, indiferent de rolul lor în sistem. Use case-urile suplimentare, cum ar fi "Operatii CRUD pe informațiile legate de clienți/angajați/administratori", sunt disponibile doar pentru ANGAJAT și ADMINISTRATOR după autentificare. Utilizatorul ADMINISTRATOR are acces și la "Vizualizarea listei utilizatorilor autentificați", "Trimiterea unui mesaj/email de notificare la fiecare modificare a datelor de logare" și "Filtrarea utilizatorilor după tipul sau", iar utilizatorul ANGAJAT are acces și la "Realizarea unei rezervări", "Salvarea informațiilor despre pachetele turistice în fișiere cu diferite formate", "Vizualizarea pachetelor rezervate" și "Vizualizarea unor statistici despre pachete" care sunt cazuri de utilizare particulare utilizatorilor.



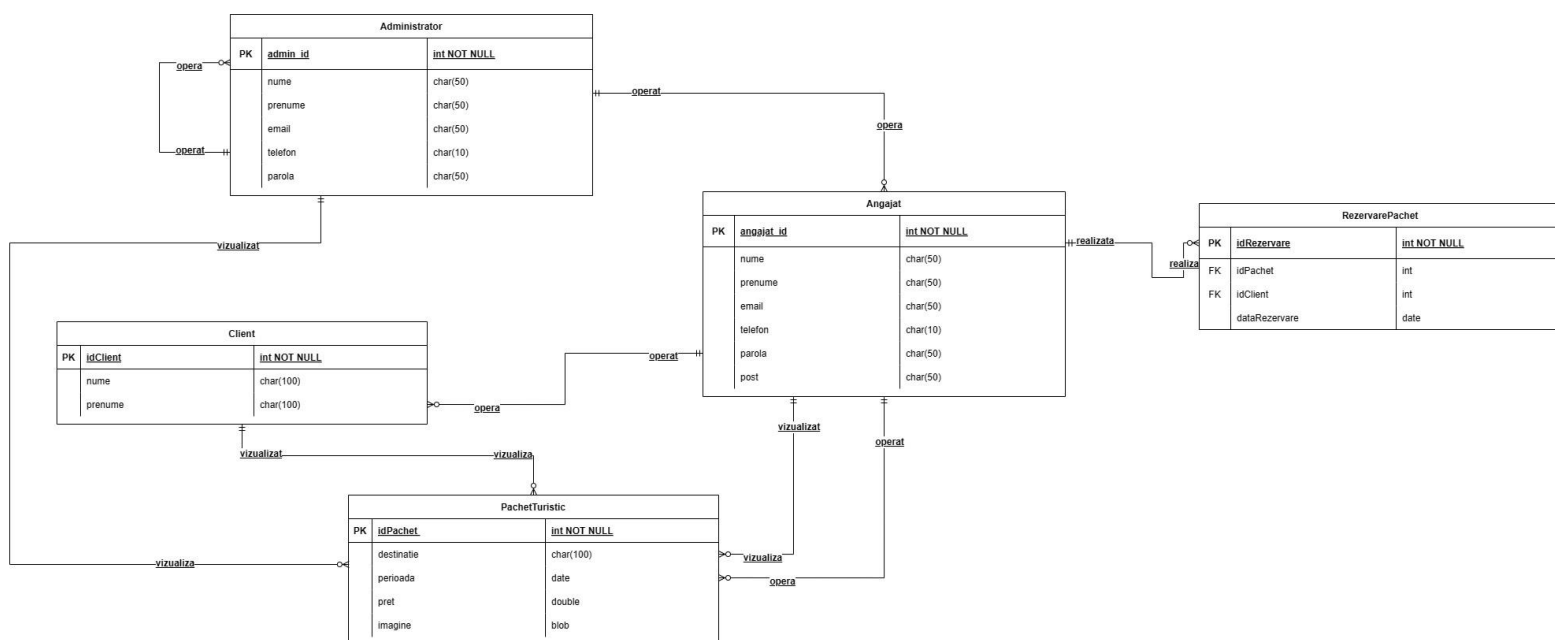
3.Faza de proiectare – Diagramele de clase

3.1.Diagrama entitate-relație

Aplicația utilizează MySQL ca sistem de gestiune a bazelor de date și se conectează la acesta folosind Spring Boot (jpa), cu driverul com.mysql.cj.jdbc.Driver. Această configurație este esențială pentru funcționarea corectă a aplicației și pentru asigurarea unei comunicări eficiente între aceasta și baza de date MySQL.

Baza de date asociată acestei aplicații este proiectată pentru a stoca informațiile referitoare la angajați, administratori, pachete turistice, clienți și rezervări. Aceasta constă dintr-o serie de tabele care sunt interconectate pentru a permite o gestionare eficientă a datelor.

- Tabelul Angajat conține informații despre angajații agenției de turism.
- Tabelul Administrator stochează detalii despre administratorii sistemului.
- Tabelul PachetTuristic este utilizat pentru a reprezenta pachetele turistice disponibile.
- Tabelul Client conține date despre clienți.
- Tabelul RezervarePachet conține date despre rezervările realizate și date despre Clientul și Pachetul pentru care s-a făcut rezervarea.



În cadrul aplicației, relațiile dintre entități sunt esențiale pentru funcționarea și gestionarea eficientă a datelor. Un administrator poate opera unul sau mai mulți alți administratori, iar fiecare administrator este operat de un singur alt administrator. De asemenea, un administrator poate opera unul sau mai mulți angajați, iar fiecare angajat este operat de un singur administrator.

Administratorii au acces la vizualizarea și gestionarea pachetelor turistice, unde fiecare pachet turistic este vizualizat de un singur administrator. Pe de altă parte, angajații pot vizualiza și opera pachete turistice, iar fiecare pachet turistic este vizualizat și operat de un singur angajat.

În relația cu clienții, aceștia pot vizualiza pachetele turistice, iar fiecare pachet este vizualizat de unul sau mai mulți clienți. În plus, angajații pot interacționa cu clienții, operându-le conturile și asigurându-se că aceștia beneficiază de serviciile oferite în mod eficient și organizat.

Rezervarea unui pachet turistic este realizată de către un singur angajat, iar un angajat poate să facă una sau mai multe rezervări, în funcție de cerințele și solicitările clienților.

- Pentru a crea o tabelă Client în baza de date utilizând Spring Boot și JPA (Java Persistence API), trebuie configurată entitatea JPA Client, setat repository-ul pentru interacțiunea cu baza de date și configurată conexiunea la baza de date în fișierul de configurare. Crearea tabelului PachetTuristic:
- Pentru a crea o tabelă PachetTuristic în baza de date utilizând Spring Boot și JPA (Java Persistence API), trebuie configurată entitatea JPA PachetTuristic, setat repository-ul pentru interacțiunea cu baza de date și configurată conexiunea la baza de date în fișierul de configurare.
- Pentru a crea o tabelă RezervarePachet în baza de date utilizând Spring Boot și JPA (Java Persistence API), trebuie configurată entitatea JPA RezervarePachet, setat repository-ul pentru interacțiunea cu baza de date și configurată conexiunea la baza de date în fișierul de configurare.
- Pentru a crea o tabelă Angajat în baza de date utilizând Spring Boot și JPA (Java Persistence API), trebuie configurată entitatea JPA Angajat, setat repository-ul pentru interacțiunea cu baza de date și configurată conexiunea la baza de date în fișierul de configurare.
- Pentru a crea o tabelă Administrator în baza de date utilizând Spring Boot și JPA (Java Persistence API), trebuie configurată entitatea JPA Administrator, setat repository-ul pentru interacțiunea cu baza de date și configurată conexiunea la baza de date în fișierul de configurare.
- Pentru a crea o tabelă Utilizator în baza de date utilizând Spring Boot și JPA (Java Persistence API), trebuie configurată entitatea JPA Utilizator ca entitate abstractă de bază și apoi configurate entitățile derivate cum ar fi Administrator, Angajat, Client care vor extinde această clasă. De asemenea, trebuie configurat repository-ul pentru interacțiunea cu baza de date și conexiunea la baza de date în fișierul de configurare.

1. Configurarea entității JPA Client

Mai întâi, trebuie definită clasa Client ca entitate JPA, care extinde o clasă de bază Utilizator. Entitatea Client va fi mapată la o tabelă în baza de date.

2. Configurarea repository-ului pentru entitatea Client

Apoi, trebuie creat repository-ul ClientRepository pentru a permite efectuarea operațiunilor CRUD (Create, Read, Update, Delete) pe entitatea Client.

3. Configurarea entității JPA PachetTuristic

Mai întâi, trebuie definită clasa PachetTuristic ca entitate JPA. Această entitate va fi mapată la o tabelă în baza de date.

4. Configurarea repository-ului pentru entitatea PachetTuristic

Apoi, trebuie creat repository-ul PachetTuristicRepository pentru a permite efectuarea operațiunilor CRUD (Create, Read, Update, Delete) pe entitatea PachetTuristic.

5. Configurarea entității JPA RezervarePachet

Mai întâi, trebuie definită clasa RezervarePachet ca entitate JPA. Această entitate va fi mapată la o tabelă în baza de date.

6. Configurarea repository-ului pentru entitatea RezervarePachet

Apoi, trebuie creat repository-ul RezervarePachetRepository pentru a permite efectuarea operațiunilor CRUD (Create, Read, Update, Delete) pe entitatea RezervarePachet.

7. Configurarea entității JPA Angajat

Mai întâi, trebuie definită clasa Angajat ca entitate JPA. Această entitate va fi mapată la o tabelă în baza de date.

8. Configurarea repository-ului pentru entitatea Angajat

Apoi, trebuie creat repository-ul AngajatRepository pentru a permite efectuarea operațiunilor CRUD (Create, Read, Update, Delete) pe entitatea Angajat.

9. Configurarea entității JPA Administrator

Mai întâi, trebuie definită clasa Administrator ca entitate JPA. Această entitate va fi mapată la o tabelă în baza de date.

10. Configurarea repository-ului pentru entitatea Administrator

Apoi, trebuie creat repository-ul AdministratorRepository pentru a permite efectuarea operațiunilor CRUD (Create, Read, Update, Delete) pe entitatea Administrator.

11. Configurarea entității JPA Utilizator

Definirea clasei Utilizator ca o entitate abstractă JPA. Această entitate va fi baza pentru alte entități.

12. Extinderea entității Utilizator pentru Administrator

Definirea entității Administrator care extinde Utilizator.

- Configurarea conexiunii la baza de date

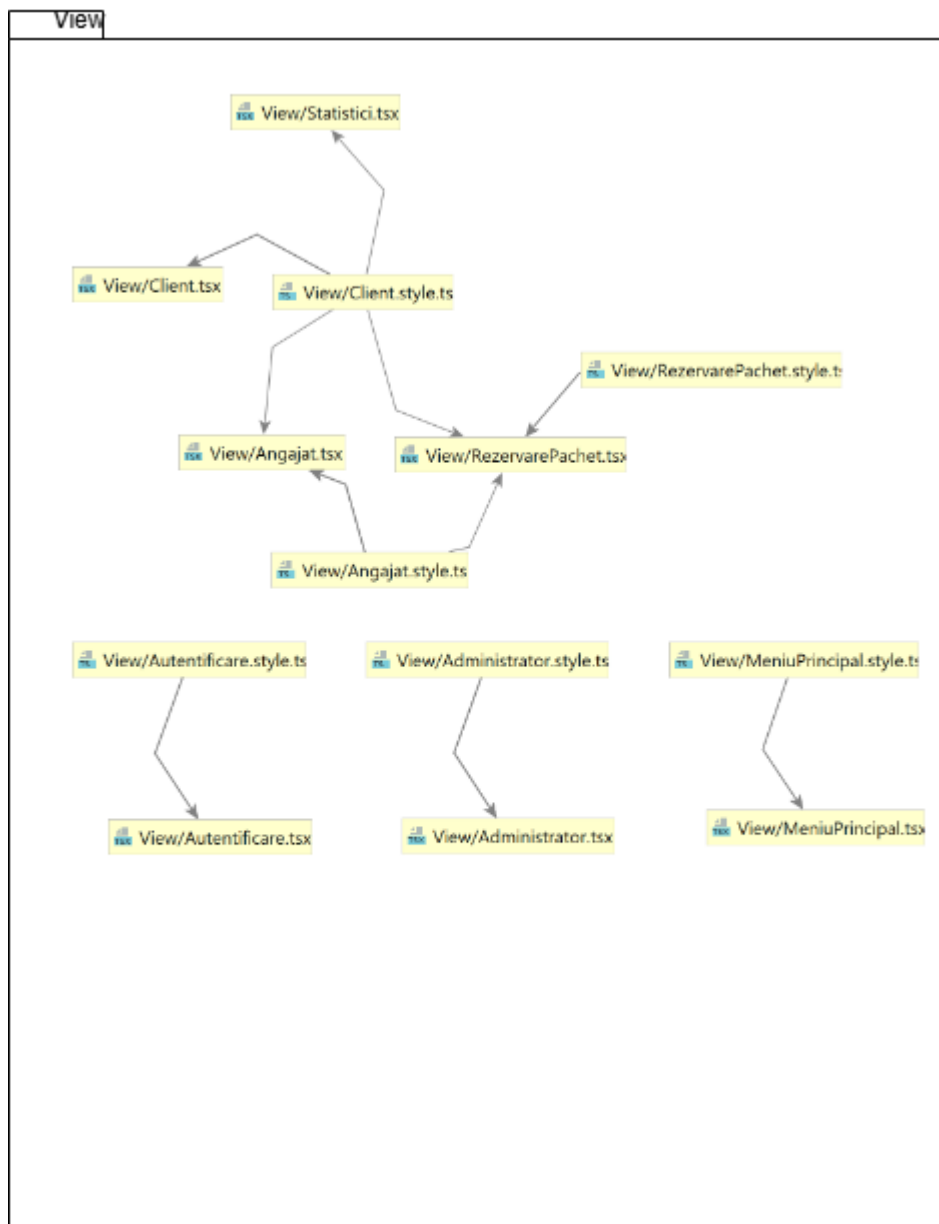
Detaliile de configurare a bazei de date trebuie adăugate în fișierul application.properties pentru a configura conexiunea la baza de date MySQL.

- Rularea aplicației Spring Boot

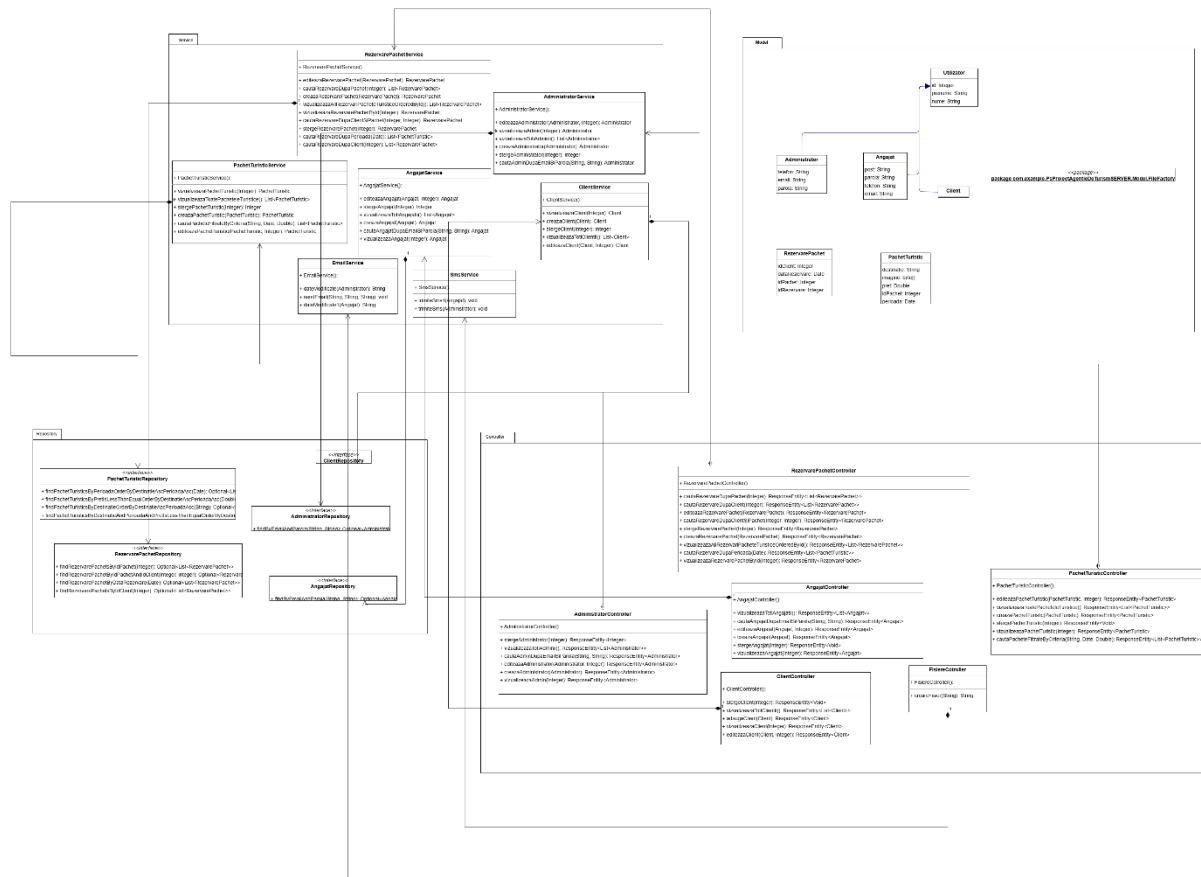
Când aplicația Spring Boot este pornită, Hibernate creează automat tabela clients în baza de date MySQL, conform definiției entității Client. Dacă tabela există deja, Hibernate o actualizează conform configurației.

3.2.Diagrama completă de clase

3.2.1.Diagrama de clase-service



3.2.2. Diagrama de clase-client



3.3. Șabloane

Proxy Pattern:

Descriere: În proiectul meu, folosesc Proxy Pattern pentru a gestiona accesul la baza de date. În loc să apelez direct funcțiile care interacționează cu baza de date din controller, folosesc un strat intermediar de service. Acest service, la rândul său, apelează funcțiile corespunzătoare din repository.

Cum îl folosesc în proiect:

1. Am definit clasele de repository pentru a accesa datele din baza de date.
2. Pentru a izola accesul direct la repository-uri din controller, am creat clase de service care încapsulează logica de acces la date.
3. În controller, apelez funcțiile din service, care la rândul lor accesează funcțiile corespunzătoare din repository-uri.
4. Astfel, obțin un nivel suplimentar de control și gestionare a accesului la date, prin intermediul service-urilor.

Abstract Factory Method:

Descriere: Pentru a gestiona crearea obiectelor de tip fișier în funcție de formatul dorit (cum ar fi DOC, CSS, XML etc.), folosesc Abstract Factory Method. Această abordare îmi permite să creez obiecte de tip fișier care respectă o anumită interfață, în funcție de cerințele specifice.

Cum îl folosesc în proiect:

1. Am definit o interfață comună pentru toate clasele de fișiere (de exemplu, FileWriter).
2. Pentru fiecare format de fișier (DOC, CSS etc.), am creat clase care implementează această interfață (de exemplu, DocWriter, CssWriter).
3. În funcție de cerințele proiectului sau de preferințele utilizatorului, instantiez obiectul corespunzător utilizând Abstract Factory Method, astfel încât să obțin un obiect de tipul dorit care poate scrie într-un fișier cu formatul corespunzător.

Factory Method:

Descriere: Folosesc Factory Method pentru a crea obiecte de tipul unei interfețe comune, dar cu implementări diferite. Această abordare îmi permite să izolez procesul de creare a obiectelor, permițând flexibilitate în alegerea tipului exact de obiect de care am nevoie.

Cum îl folosesc în proiect:

1. Am definit o interfață comună pentru clasele care trebuie create (de exemplu, o interfață pentru procesare de date).
2. Pentru fiecare tip de procesare sau de operație specifică, am creat clase care implementează această interfață (de exemplu, DataProcessor, ReportGenerator).
3. Folosesc Factory Method pentru a crea obiecte de tipul interfeței în funcție de contextul sau de cerințele aplicației mele.

Singleton Pattern:

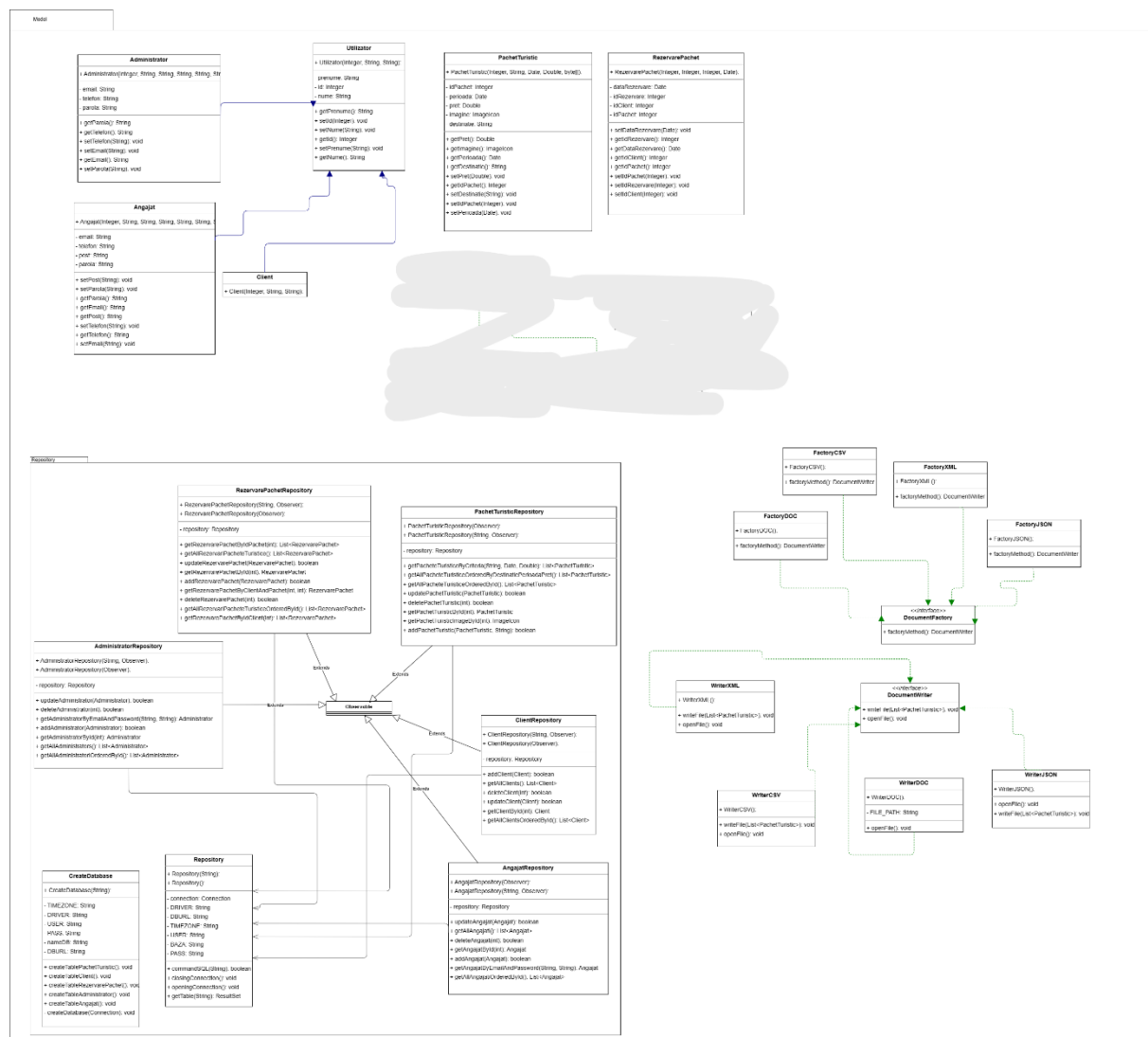
Descriere: În proiectul meu, folosesc Singleton Pattern pentru a asigura că există o singură instanță a unui obiect în întreaga aplicație. Această abordare este utilă în situații în care trebuie să ne asigurăm că există o singură instanță a unei clase și că această instanță este accesibilă global.

Cum îl folosesc în proiect:

1. Folosesc Singleton Pattern în Spring Boot, unde Spring creează un singur obiect și îl injectează în toate clasele care îl necesită.
2. Astfel, pot avea o singură instanță a unui serviciu sau a unei componente care trebuie să fie global accesibilă și gestionată de Spring Container.

Acestea sunt modurile în care folosesc diferitele șabloane de proiectare în proiectul meu pentru a organiza și gestiona codul într-un mod eficient și modular.

3.4. Pachetul Model



În cadrul pachetului model al aplicației noastre de agentie de turism, Spring Boot și React joacă un rol crucial în gestionarea eficientă a entităților și rolurilor utilizatorilor în sistem. Acestea sunt clasele principale și relațiile lor, împreună cu implementările specifice pentru gestionarea interfeței grafice și exportul datelor în diferite formate.

Clasele de Model:

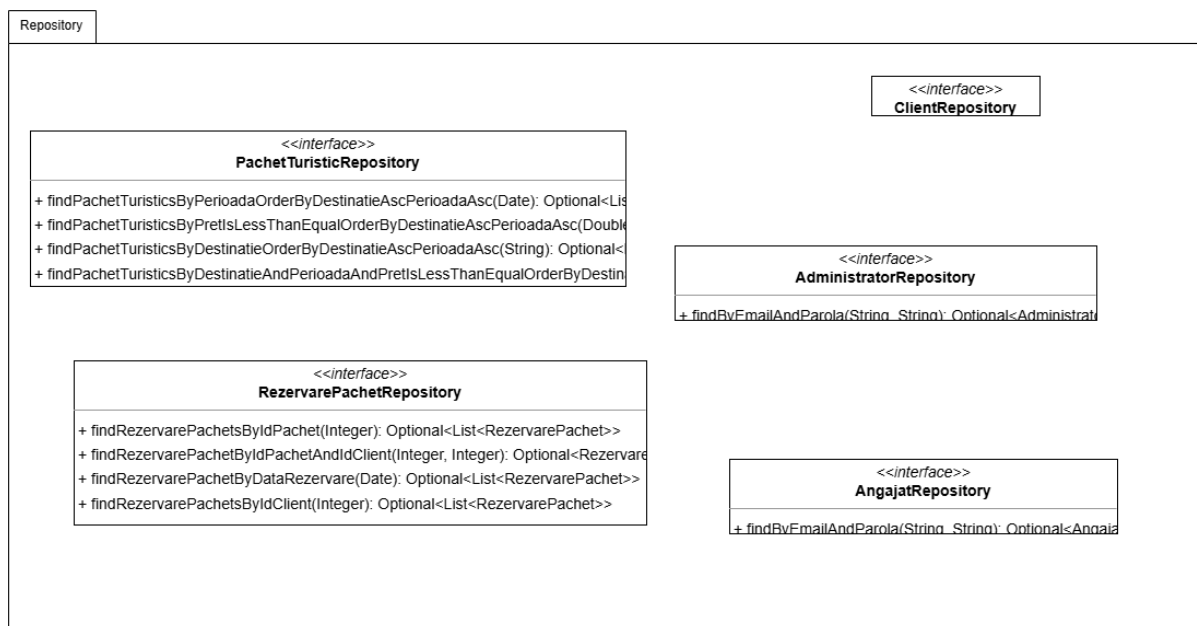
- **Utilizatorul:** Această clasă abstractă furnizează un schelet comun pentru toți utilizatorii din sistem. Include atributele de bază precum ID-ul, numele și prenumele utilizatorului. Această clasă servește ca o bază solidă pentru clasele Client, Angajat și Administrator, permițând o abordare modulară și extensibilă pentru gestionarea diferitelor tipuri de utilizatori.
- **PachetTuristic:** Modelează informațiile asociate unui pachet turistic disponibil în aplicație. Include detalii precum ID-ul pachetului, destinația, perioada și preț.
- **Clientul:** Subclasă a clasei Utilizator, reprezintă un utilizator al aplicației care are rolul de client. Constructorul său inițializează atributele comune ale utilizatorului.

- **Angajatul:** Subclasă a clasei Utilizator, modelează un utilizator care deține un rol de angajat în aplicație. Adaugă detalii specifice angajatului precum postul ocupat, adresa de email, parola și numărul de telefon.
- **Administratorul:** Subclasă a clasei Utilizator, reprezintă un utilizator cu privilegii administrative în aplicație. Constructorul său inițializează atributele specifice administratorului precum adresa de email, parola și numărul de telefon.
- **RezervarePachet:** Responsabilă pentru modelarea și gestionarea informațiilor referitoare la rezervările efectuate în cadrul aplicației de turism. Include atribute precum ID-ul rezervării, ID-ul pachetului turistic și ID-ul clientului, împreună cu data rezervării.

Implementările Specifice pentru Spring Boot și React:

- Utilizăm Spring Boot pentru a gestiona eficient logica din spatele aplicației, inclusiv persistența datelor și gestionarea utilizatorilor. Acesta oferă o abordare convenabilă și puternică pentru dezvoltarea rapidă a aplicațiilor Java.
- React este folosit pentru a crea o interfață de utilizator modernă și interactivă, facilitând dezvoltarea de componente reutilizabile și gestionarea stării aplicației.
- Fiecare clasă care implementează interfața Language este responsabilă pentru crearea dicționarului de traduceri, în timp ce fiecare clasă care implementează DocumentWriter este responsabilă pentru salvarea listei de pachete turistice într-un format specific.
- Interfețele DocumentWriter și DocumentFactory sunt utilizate pentru a defini structura necesară pentru exportul datelor în diferite formate de fișiere. Acestea permit adăugarea de noi formate de fișiere fără a modifica codul existent, respectând astfel principiul Open-Closed.
- Implementările specifice pentru Spring Boot și React permit flexibilitate și extensibilitate în cadrul aplicației noastre, oferind o bază solidă pentru dezvoltarea și extinderea ulterioară a funcționalităților sistemului.

3.5.Subpachetul Repository

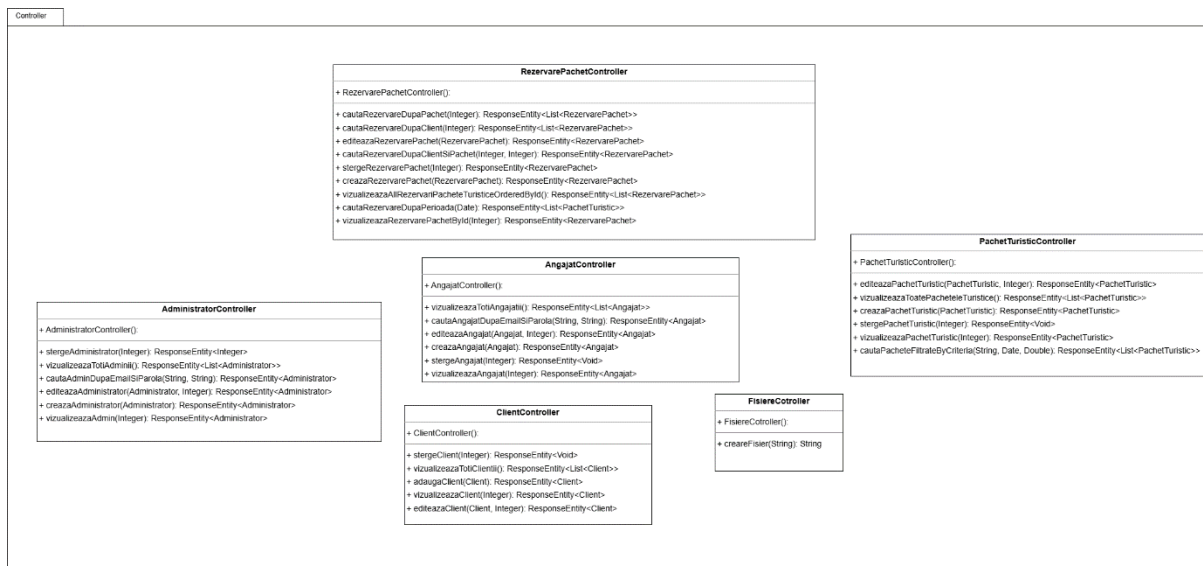


Pachetul Repository este o componentă esențială în cadrul aplicației noastre de agenție de turism, care asigură interacțiunea cu baza de date și manipularea datelor în mod eficient. Acesta conține interfețe care extind JpaRepository, furnizând metode predefinite pentru operațiuni CRUD (create, read, update, delete) pe entitățile noastre.

- **AdministratorRepository:** Această interfață gestionează operațiunile de persistență pentru entitatea Administrator. Ea oferă metode pentru a găsi un administrator după adresa de email și parolă, facilitând autentificarea acestuia în sistem.
- **AngajatRepository:** Similar cu AdministratorRepository, această interfață se ocupă de operațiunile CRUD pentru entitatea Angajat. De asemenea, furnizează metode pentru a găsi un angajat după adresa de email și parolă.
- **ClientRepository:** Interfața pentru manipularea datelor referitoare la clienți. Oferă funcționalități de bază pentru gestionarea entităților Client.
- **PachetTuristicRepository:** Această interfață gestionează persistența datelor pentru entitatea PachetTuristic. Oferă o serie de metode pentru a găsi pachete turistice în funcție de diferite criterii precum destinația, perioada și prețul.
- **RezervarePachetRepository:** Interfața responsabilă pentru gestionarea rezervărilor efectuate în cadrul aplicației. Oferă metode pentru a găsi rezervări în funcție de client, pachet turistic și dată a rezervării.

Prin intermediul acestor interfețe, putem interacționa cu baza de date într-un mod modular și eficient, respectând principiile programării orientate pe obiecte și beneficiind de funcționalitățile oferite de Spring Boot.

3.6. Pachetul Controller



Controllerul din aplicația noastră este o componentă crucială care gestionează fluxul de cereri HTTP primite de la client și coordonează logica de business pentru a procesa aceste cereri. Iată o descriere la persoana a 3-a a fiecărui controller din aplicație și modul în care interacționează cu partea de client, în special cu frontend-ul React, folosind URL-urile definite:

- **AdministratorController:** Acest controller gestionează operațiunile CRUD pentru entitatea Administrator. Prin intermediul URL-urilor precum `/Administrator/creaza`, `/Administrator/editeaza/{id}`, `/Administrator/sterge/{id}`, `/Administrator/vizualizeazaAdmin/{id}`, `/Administrator/vizualizeazaToti`, și `/Administrator/cautaDupaEmailSiParola/{email}/{parola}`, interacționează cu frontend-ul pentru a permite crearea, editarea, ștergerea și vizualizarea administratorilor, precum și autentificarea acestora în sistem.
- **AngajatController:** Acest controller se ocupă de operațiunile CRUD pentru entitatea Angajat. Prin intermediul URL-urilor precum `/Angajat/creaza`, `/Angajat/editeaza/{id}`, `/Angajat/sterge/{id}`, `/Angajat/vizualizeazaAngajat/{id}`, `/Angajat/vizualizeazaToti`, și `/Angajat/cautaDupaEmailSiParola/{email}/{parola}`, facilitează interacțiunea cu frontend-ul pentru gestionarea angajaților și autentificarea acestora.
- **ClientController:** Acest controller gestionează operațiunile CRUD pentru entitatea Client. Prin intermediul URL-urilor precum `/Client/creaza`, `/Client/editeaza/{id}`, `/Client/sterge/{id}`, `/Client/vizualizeazaClient/{id}`, și `/Client/vizualizeazaToti`, permite interacțiunea cu frontend-ul pentru gestionarea clienților și vizualizarea acestora.
- **PachetTuristicController:** Acest controller gestionează operațiunile CRUD pentru entitatea PachetTuristic. Prin intermediul URL-urilor precum `/PachetTuristic/creaza`, `/PachetTuristic/editeaza/{idPachet}`, `/PachetTuristic/sterge/{idPachet}`, `/PachetTuristic/vizualizeazaPachetTuristic/{idPachet}`, `/PachetTuristic/vizualizeazaToate`, și `/PachetTuristic/filtreazaDupaCriteriu/{destinatie}/{perioada}/{pret}`, permite gestionarea pachetelor turistice și aplicarea de filtre pentru căutarea acestora.
- **RezervarePachetController:** Acest controller gestionează operațiunile CRUD pentru entitatea RezervarePachet. Prin intermediul URL-urilor precum `/RezervarePachet/creaza`, `/RezervarePachet/editeaza`, `/RezervarePachet/sterge/{id}`, `/RezervarePachet/vizualizeaza/{id}`, `/RezervarePachet/vizualizeazaToate`, `/RezervarePachet/cautaDupaClientSiPachet/{idClient}/{idPachet}`, `/RezervarePachet/cautaDupaClient/{idClient}`,

/RezervarePachet/cautaDupaPerioada/{perioada}, și
/RezervarePachet/cautaDupaPachet/{idPachet}, facilitează gestionarea rezervărilor pachetelor turistice.

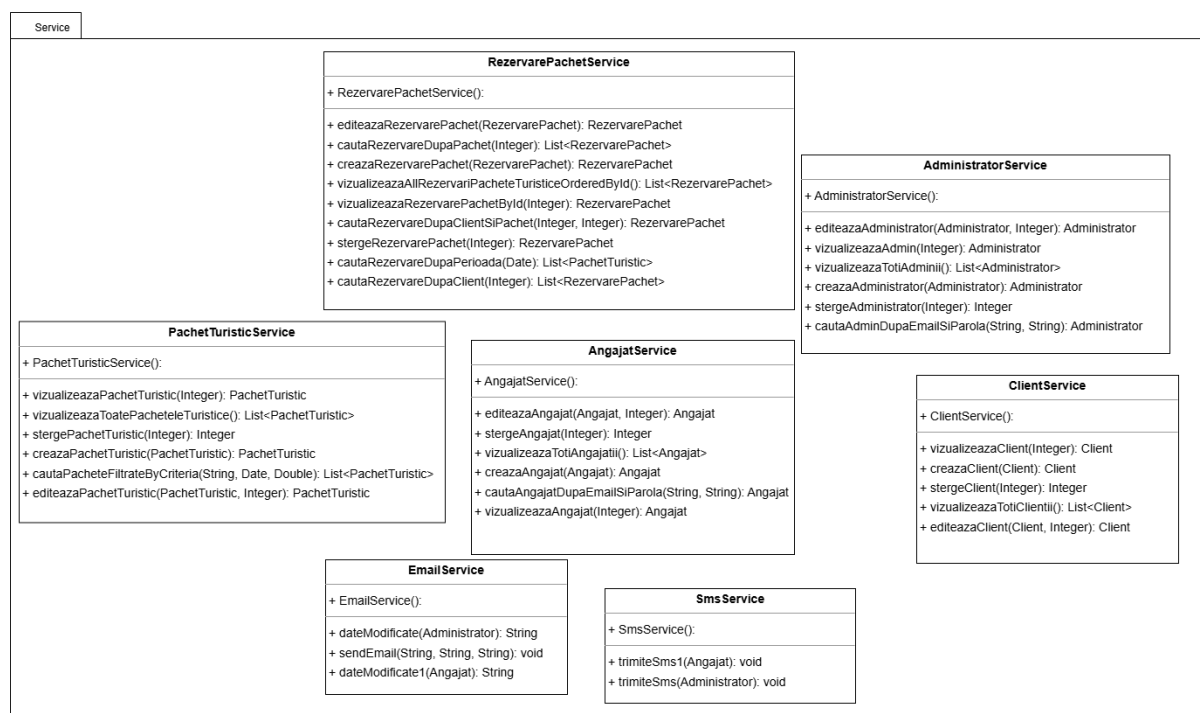
Pentru a interacționa cu aceste endpoint-uri din frontend-ul React, trebuie să efectuăm cereri HTTP către aceste URL-uri din codul frontend-ului. De exemplu, pentru a crea un nou administrator, frontend-ul va trimite o cerere POST către URL-ul /Administrator/creaza cu datele noului administrator în corpul cererii. Același principiu se aplică și pentru celelalte operațiuni CRUD și entități. Este important ca frontend-ul să respecte aceste URL-uri și să trimită cereri HTTP adecvate pentru a interacționa corect cu backend-ul.

1. Metode HTTP:

- **GET:** Utilizat pentru a solicita date de la server. De exemplu, pentru a afișa o listă de entități sau pentru a obține detalii despre o anumită entitate.
 - **POST:** Utilizat pentru a trimite date către server pentru a crea o nouă entitate sau a executa o acțiune care modifică starea serverului.
 - **PUT:** Utilizat pentru a actualiza o entitate existentă pe server.
 - **DELETE:** Utilizat pentru a șterge o entitate existentă pe server.
2. **Serializarea datelor:** Datele trimise și primite între frontend și backend trebuie să fie serializate într-un format convenabil, cum ar fi JSON sau XML. Aceasta înseamnă că obiectele Java trebuie convertite într-un format de text, astfel încât să poată fi transmise prin rețea și apoi convertite înapoi în obiecte Java.
3. **Tratarea răspunsurilor HTTP:** Odată ce serverul primește o cerere HTTP, va procesa cererea și va returna un răspuns HTTP către client. Este important ca frontend-ul să gestioneze aceste răspunsuri corespunzător pentru a lua măsurile adecvate în funcție de rezultatul cererii (de exemplu, afișarea unui mesaj de succes sau de eroare).
4. **Gestionarea erorilor:** În timpul interacțiunii între frontend și backend, pot apărea diverse erori, cum ar fi erori de rețea, erori de validare a datelor sau erori interne ale serverului. Frontend-ul ar trebui să fie pregătit să gestioneze aceste erori într-un mod elegant și să furnizeze utilizatorului informațiile necesare pentru a înțelege și rezolva problema.
5. **Autentificare și autorizare:** Dacă aplicația dvs. necesită autentificare și autorizare, veți avea nevoie de mecanisme adecvate pentru a permite utilizatorilor să se conecteze, să își gestioneze sesiunile și să acceseze doar resursele la care au drepturi.

Interacțiunea dintre frontend-ul React și backend-ul Spring Boot se bazează pe comunicarea prin cereri și răspunsuri HTTP. Folosind URL-urile definite în controlerele Spring Boot, puteți defini interacțiunea între cele două părți ale aplicației și puteți crea o experiență de utilizare fluentă și eficientă pentru utilizatorii aplicației.

3.7. Pachetul Service



În cadrul aplicației noastre de agenție de turism, am pus la punct un set solid de servicii care se ocupă de tot ce este legat de gestionarea utilizatorilor, pachetelor turistice și rezervărilor. Iată cum arată acest set:

- **Serviciul pentru Administratori:** Acesta este colțul nostru de control pentru toate operațiunile legate de administratori. De la adăugarea unui administrator nou și până la ștergerea sau actualizarea datelor unui administrator existent, acest serviciu este responsabil pentru a face toate aceste manevre fără probleme. În plus, ne oferă o modalitate de a căuta un administrator specific după email și parolă pentru a asigura un proces de autentificare eficient.
- **Serviciul pentru Angajați:** Similar cu serviciul pentru administratori, acesta se ocupă de toate aspectele referitoare la angajații noștri. Crearea, editarea, ștergerea și vizualizarea angajaților sunt sarcinile principale ale acestui serviciu. Și, desigur, nu uităm de autentificarea angajaților, oferind o modalitate de a căuta un angajat după email și parolă.
- **Serviciul pentru Clienți:** Aici gestionăm toate detaliile referitoare la clienții noștri. Crearea unui nou client, actualizarea informațiilor sau ștergerea unui cont sunt sarcinile pe care acest serviciu le îndeplinește.
- **Serviciul de Email:** Pentru a menține comunicarea eficientă cu utilizatorii noștri, am pus la punct un serviciu care ne permite să trimitem email-uri în diverse situații, cum ar fi atunci când se modifică datele contului unui administrator sau angajat.
- **Serviciul pentru Pachete Turistice:** Acesta este motorul nostru pentru gestionarea ofertelor noastre de călătorie. De la crearea și editarea pachetelor turistice până la căutarea lor în funcție de destinație, perioadă sau preț, acest serviciu este esențial pentru afacerea noastră.
- **Serviciul pentru Rezervări de Pachete Turistice:** Aici ne ocupăm de toate aspectele legate de rezervările făcute de clienții noștri. De la înregistrarea unei noi rezervări și

până la ștergerea sau actualizarea unei rezervări existente, acest serviciu se asigură că totul merge strună.

- **Serviciul SMS:** Pentru a asigura că nu pierdem niciun detaliu important, am inclus și un serviciu care ne permite să trimitem notificări prin SMS. Astfel, atunci când se modifică datele de autentificare ale unui administrator sau angajat, aceștia vor primi automat un mesaj pe telefonul lor mobil.

Cu acest set complet de servicii, suntem pregătiți să oferim o experiență de utilizare impecabilă clienților noștri și să gestionăm eficient întreaga noastră activitate în domeniul turismului.

4. Faza de implementare – Aplicația

4.1. Instrumente utilizate

Java Spring Boot: Am optat pentru Java Spring Boot pentru dezvoltarea backend-ului aplicației. Spring Boot oferă un cadru puternic și flexibil pentru construirea aplicațiilor Java web, cu facilități de gestionare a dependențelor, configurare simplificată și integrare ușoară cu baze de date.

Hibernate ORM: Pentru interacțiunea cu baza de date MySQL, am folosit Hibernate ORM (Object-Relational Mapping). Acest lucru îmi permite să lucrez cu obiecte Java în loc de instrucțiuni SQL directe, facilitând gestionarea datelor și evitând complexitatea manuală a operării cu baza de date.

Thymeleaf: Pentru dezvoltarea interfeței web, am folosit Thymeleaf, un motor de șablon pentru aplicații web Java. Thymeleaf integrează ușor cu Spring Boot și oferă o sintaxă ușor de înțeles pentru crearea paginilor web dinamice.

Bootstrap: Pentru stilizarea și structurarea interfeței web, am folosit Bootstrap, un cadru front-end cu o serie de componente și șabloane predefinite care fac dezvoltarea interfeței mai rapidă și mai simplă.

Swagger: Pentru documentarea API-ului backend, am integrat Swagger, care generează automat o documentație interactivă a API-ului nostru. Acest lucru facilitează colaborarea și comunicarea între echipele de dezvoltare și oferă utilizatorilor o modalitate simplă de a înțelege și de a interacționa cu API-ul nostru.

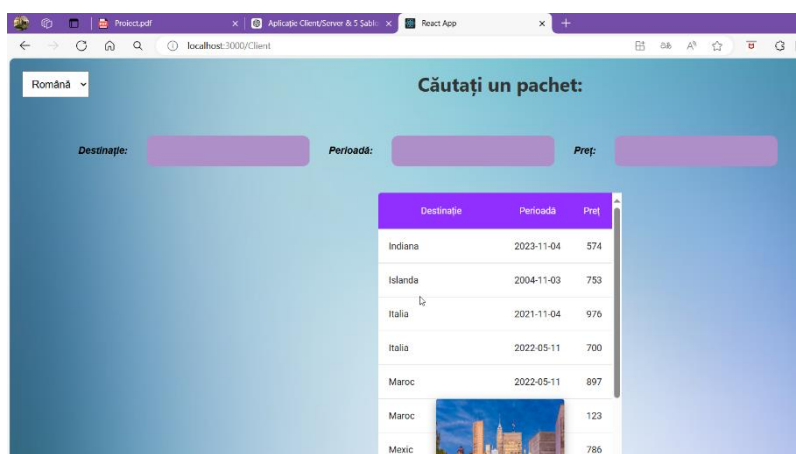
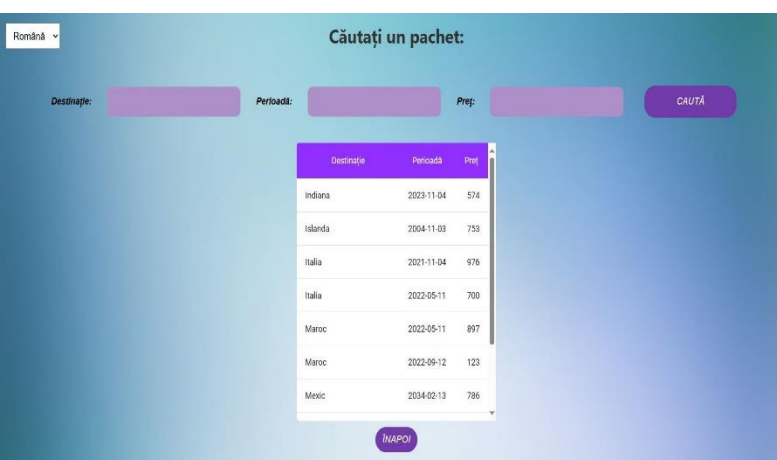
JFreeChart: Pentru generarea și afișarea graficelor și statisticilor, am integrat pachetul Maven JFreeChart. Acesta oferă o gamă largă de funcționalități pentru crearea diferitelor tipuri de grafice și diagrame, ușor de integrat în aplicația noastră Java.

4.2. Aplicația

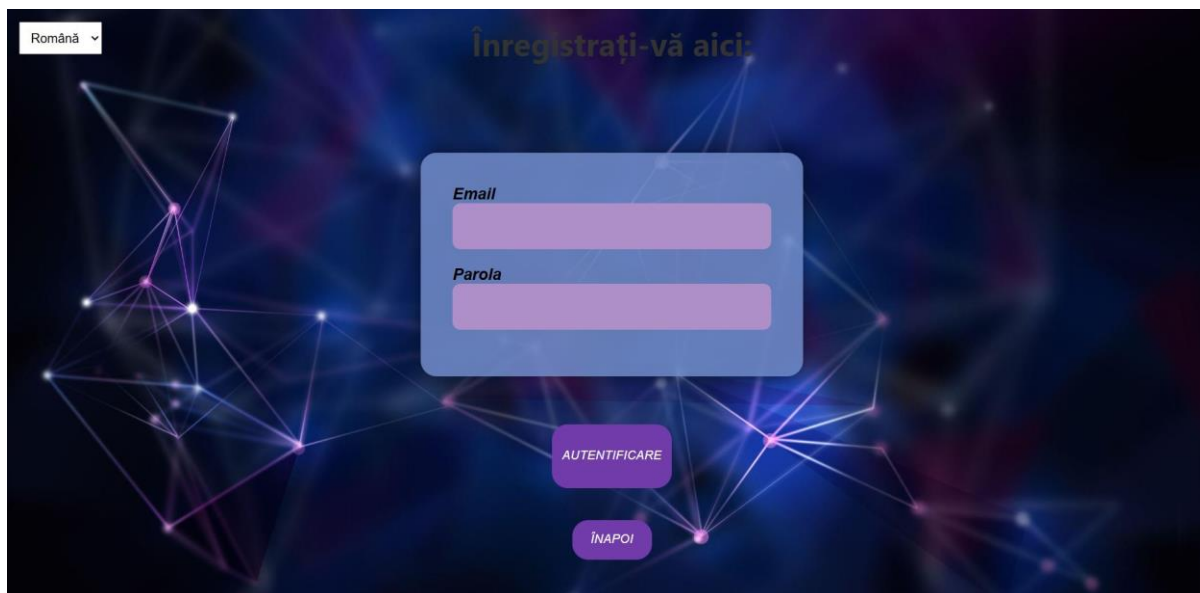
În momentul în care utilizatorul inițiază aplicația, acesta va fi întâmpinat de interfața grafică principală. Aici, va avea două opțiuni distincte pentru a-și începe experiența. În plus față de tema anterioară pe fiecare interfață am adăugat un combobox de unde se poate schimba limba aplicației.



A doua opțiune îi va permite să continue fără a fi necesară autentificarea. Prin intermediul acestei opțiuni, utilizatorul va avea acces direct la vizualizarea pachetelor turistice disponibile și va putea să le filtreze după criterii specifice, cum ar fi destinația, perioada sau prețul. La apăsarea butonului “Caută” se vor filtra pachetele după criteriul specificat în field-ul completat, iar dacă se apasă pe o linie a tabelului se va afișa imagine specifică acelui pachet turistic.



Cea de-a doua opțiune va redirecționa utilizatorul către o pagină de autentificare. Aici, acesta va trebui să introducă adresa de email și parola asociată contului său. După autentificare, aplicația va identifica tipul de utilizator - fie administrator, fie angajat - și îl va direcționa către o interfață corespunzătoare.



Română ▾

Înregistrați-vă aici:


Email

Parola

AUTENTIFICARE

ÎNAPOI

Dacă utilizatorul este identificat ca angajat, acesta va avea acces la funcționalitățile specifice angajatului, cum ar fi vizualizarea, editarea, gestionarea pachetelor și a clienților și de asemenea în plus, se pot salva informațiile despre pachetele turistice în fișiere de diferite formate.



Română ▾

Bine ai venit!

ÎNAPOI

VIZUALIZARE PACHETE

MODIFICARE PACHETE

MODIFICARE CLIENȚI

REALIZEAZA O COMANDA

ADAUGĂ UN PACHET

ȘTERGE UN PACHET

EDITEAZĂ UN PACHET

VIZUALIZARE

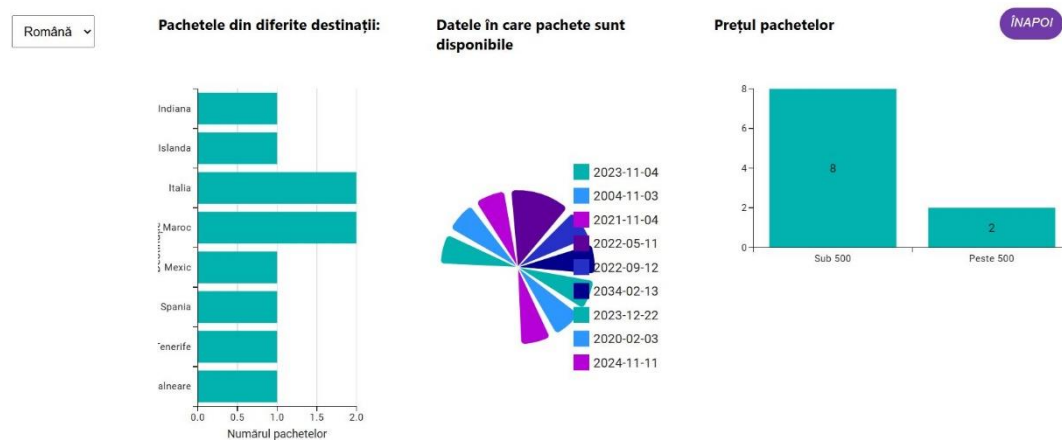
Destinație	Perioadă	Preț
Indiana	2023-11-04	574
Islanda	2004-11-03	753
Italia	2021-11-04	976

~Alegeți un format~ ▾

SALVARE FIȘIER

VIZUALIZARE STATISTICI

La această aplicație am adăgat o funcționalitate unde la apăsarea unui buton se afișează anumite statistici despre pachetele turistice.



De asemenea un angajat poate crea o rezervare pentru un client, facandu-i legatura cu un anumit pachet turistic astfel realizandu-se o rezervare.

Spaniolă ▾

Hacer una reserva:

[REGRESAR](#) [RESERVAR](#) [BUSCAR](#)

Elija el paquete deseado:

--ID-- ▾ destino: precio: periodo:

Elija el cliente deseado:

--ID-- ▾ nombre: apellido:

Ingrese el periodo para los paquetes reservados:

Ingrese el periodo:

ID de reserva	ID de pachet	ID de cliente	Fecha de reserva
1	1	1	2024-02-12
2	2	1	2022-11-04

ID de paquete Destino del paquete

Dacă utilizatorul este identificat ca administrator, acesta va avea acces la funcționalitățile specifice administratorului, cum ar fi adăugarea, editarea sau ștergerea angajaților și a administratorilor dar și vizualizarea și filtrarea utilizatorilor autentificați.

The left screenshot displays the administrator dashboard. At the top, it says "Bine ai venit!" (Welcome!) with a "ÎNAPOI" (Back) button. Below this are several buttons: "VIZUALIZARE PACHETE TURISTICE", "MODIFICARE DATE ANGAJAȚI", "MODIFICARE DATE ADMINI", "VIZUALIZARE UTILIZATORI", "ADAUGĂ UN ADMINISTRATOR", "ȘTERGE UN ADMINISTRATOR", "EDITEAZĂ UN ADMINISTRATOR", and "VIZUALIZARE UN ADMINISTRATOR". A dropdown menu is visible below these buttons. The main content area features a table with the following data:

Nume	prenume	email	telefon	parola	post
Buta	Ana	butana@gmail.com	0787990000	butana1234	Asistent
Popa	Lucia	popalucia	074569912	popalucia	Manager
Buta	Maria	andreeuta.buta@yahoo.com	0787993376	marialucia	Asistent
Buta	Andreea	butuandreea@gmail.com	075432233	butuandreea	
Filip	Maria	filipmaria@yahoo.es	0743211122	filipmaria	

The right screenshot shows the user registration form. It also says "Bine ai venit!" (Welcome!) with a "ÎNAPOI" (Back) button. Below this are buttons: "VIZUALIZARE PACHETE TURISTICE", "MODIFICARE DATE ANGAJAȚI", "MODIFICARE DATE ADMINI", "VIZUALIZARE UTILIZATORI", "ADAUGĂ UN ANGAJAT", "ȘTERGE UN ANGAJAT", "EDITEAZĂ UN ANGAJAT", and "VIZUALIZARE ANGAJAT". The form fields include "Nume", "Prenume", "Email", "Telefon", "Parola", and "Post", followed by an "EXECUTĂ" (Execute) button.

În ambele cazuri, interfețele grafice oferă o experiență intuitivă și facilitează utilizatorilor accesul la informațiile și funcționalitățile dorite, contribuind astfel la o experiență plăcută și eficientă în utilizarea aplicației.