Travel Points

Analysis and Design Document

Student:Bode Andreea-Nicoleta

**Group:30238**

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| <26/03/2023> | <1.0> |  | Bode Andreea-Nicoleta |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

I. Project Specification 4

II. Elaboration – Iteration 1.1 4

1. Domain Model 4

2. Architectural Design 4

2.1 Conceptual Architecture 4

2.2 Package Design 4

2.3 Component and Deployment Diagrams 4

III. Elaboration – Iteration 1.2 4

1. Design Model 4

1.1 Dynamic Behavior 4

1.2 Class Design 4

2. Data Model 4

3. Unit Testing 4

IV. Elaboration – Iteration 2 4

1. Architectural Design Refinement 4

2. Design Model Refinement 4

V. Construction and Transition 5

1. System Testing 5

2. Future improvements 5

VI. Bibliography 5

# Project Specification

Dezvoltarea unei aplicații web pentru turism, numită Travel, care să permită utilizatorilor să găsească informații despre diferite obiective turistice și locații, să adauge.

Utilizatori: Aplicația va avea două tipuri de utilizatori: administrator și turist.

Funcționalități pentru administrator:

Adăugarea de obiective turistice noi în platformă, inclusiv locația, descrierea, prețul de intrare și oferte speciale.

Modificarea și ștergerea obiectivelor turistice existente în platformă.

Vizualizarea de informații grafice privind preturile si cate obiective exista intr-o anumita locatie..

Funcționalități pentru turist:

Căutarea de obiective turistice în funcție de locație și categorie.

Vizualizarea de informații despre obiectivele turistice și locațiile selectate.

Adăugarea de comentarii pentru diferite obiective turistice.

Trimiterea de e-mail-uri către administrator pentru a cere informații sau pentru a cere modificarea/updatarea detaliilor corespunzătoare unei locații sau obiectiv turistic.

Marcarea obiectivelor turistice ca fiind de interes.

Autentificare și securitate:

Utilizatorii vor trebui să se înregistreze și să se autentifice folosind o adresă de e-mail validă și o parolă sigură.

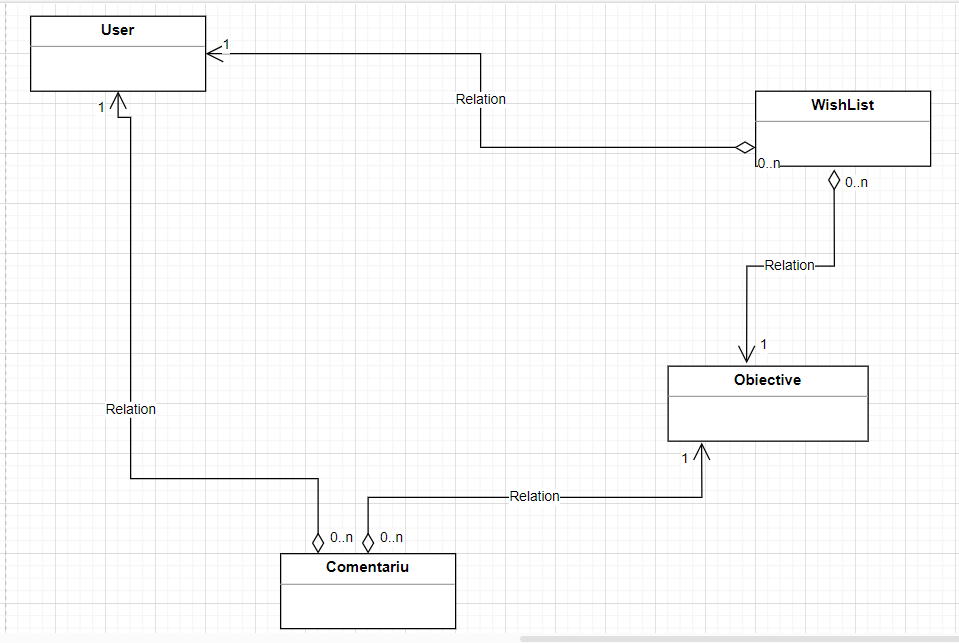
Parolele utilizatorilor vor fi stocate în mod securizat în baza de date.

Administratorul va avea acces la o interfață de autentificare separată, care va fi protejată cu o parolă sigură.

# Elaboration – Iteration 1.1

# Domain Model

* Modelul de domeniu definește diferitele entități și relațiile lor în cadrul sistemului. Pentru aplicatia Travel avem urmatorul Domain Model:
* User: Reprezintă un utilizator al aplicației, cu atribute precum nume, email și parolă.
* Location: Reprezintă o locație fizică care poate fi vizitată, cu atribute precum nume, adresă și coordonate.
* Obiective: Reprezintă o atracție turistică specifică care poate fi vizitată la o locație, cu atribute precum nume, descriere, preț și ore de funcționare.
* Categorie: Reprezintă o categorie la care o atracție turistică aparține, cum ar fi situri istorice, muzee sau obiective naturale.
* Review: Reprezintă o recenzie lăsată de un utilizator pentru o atracție turistică, cu atribute precum o evaluare și un comentariu.
* Notificare: Reprezintă o notificare trimisă unui utilizator atunci când o nouă ofertă este adăugată la o atracție turistică din lista lor de interese.
* WishList: Reprezintă o atracție turistică pe care un utilizator a adăugat-o la lista lor de interese.
* Aceste entități sunt legate între ele în următoarele moduri:
* Un utilizator poate adăuga multe atracții turistice la lista lor de interese, iar fiecare atracție turistică poate fi adăugată la lista de interese a multor utilizatori. Această relație este reprezentată printr-o relație de mulți la mulți între Utilizator și Atracție turistică prin intermediul entității Interes.
* O atracție turistică aparține unei singure categorii, iar o categorie poate avea multe atracții turistice. Această relație este reprezentată printr-o relație de unul la mulți între Categorie și Atracție turistică.
* O atracție turistică poate avea multe recenzii, iar fiecare recenzie aparține unei atracții turistice și unui utilizator. Această relație este reprezentată printr-o relație de mulți la unu între Atracție turistică și Recenzie și între Utilizator și Comentariu.



# Architectural Design

## Conceptual Architecture

Arhitectura conceptuală a sistemului este o arhitectură client-server, care este un stil arhitectural larg utilizat în dezvoltarea software-ului modern. Acest stil de arhitectură se caracterizează prin divizarea responsabilităților între client, care este responsabil pentru interfața și interacțiunea cu utilizatorul, și server, care furnizează procesarea, stocarea și gestionarea datelor.

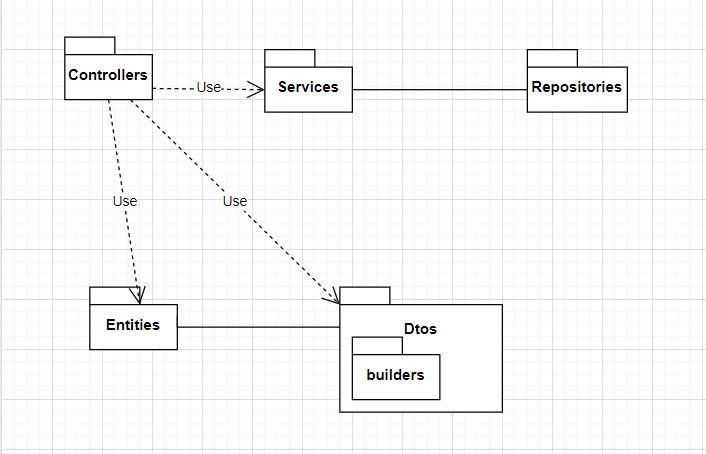
Arhitectura client-server este potrivită pentru aplicația Travel deoarece oferă o soluție scalabilă, flexibilă și ușor de întreținut. Prin separarea interfeței de utilizator de procesarea datelor, arhitectura client-server permite gestionarea mai ușoară a componentelor sistemului și o scalabilitate mai bună. De exemplu, dacă numărul de utilizatori crește, serverul poate fi actualizat pentru a face față încărcăturii crescute fără a afecta clientul.

În această arhitectură, clientul și serverul comunică între ei folosind un protocol standardizat, cum ar fi HTTP, care permite interoperabilitatea între diferite platforme și tehnologii. Serverul furnizează un set de servicii web la care poate accesa clientul, permițând o gamă largă de dispozitive să utilizeze aplicația.

Pentru a îmbunătăți în continuare scalabilitatea și flexibilitatea sistemului, arhitectura va utiliza un model de microservicii. Acest model implică împărțirea aplicației într-un set de servicii mici și independente, fiecare cu o funcționalitate specifică. Fiecare microserviciu poate fi dezvoltat, implementat și scalat independent, ceea ce face mai ușor întreținerea și actualizarea sistemului. De exemplu, dacă un serviciu specific trebuie actualizat, doar acel serviciu trebuie modificat și implementat din nou, în loc să actualizăm întregul sistem.

În concluzie, arhitectura client-server cu modelul de microservicii este o alegere potrivită pentru aplicația Travel datorită scalabilității, flexibilității și ușurinței de întreținere. Acest stil de arhitectură permite aplicației să gestioneze un număr mare de utilizatori, este adaptabil la diferite dispozitive și oferă o metodă eficientă de întreținere și actualizare a sistemului.

## Package Design



## Component and Deployment Diagrams

O imagine care conține diagramă

Descriere generată automat

O imagine care conține diagramă

Descriere generată automat

O imagine care conține diagramă

Descriere generată automatO imagine care conține diagramă

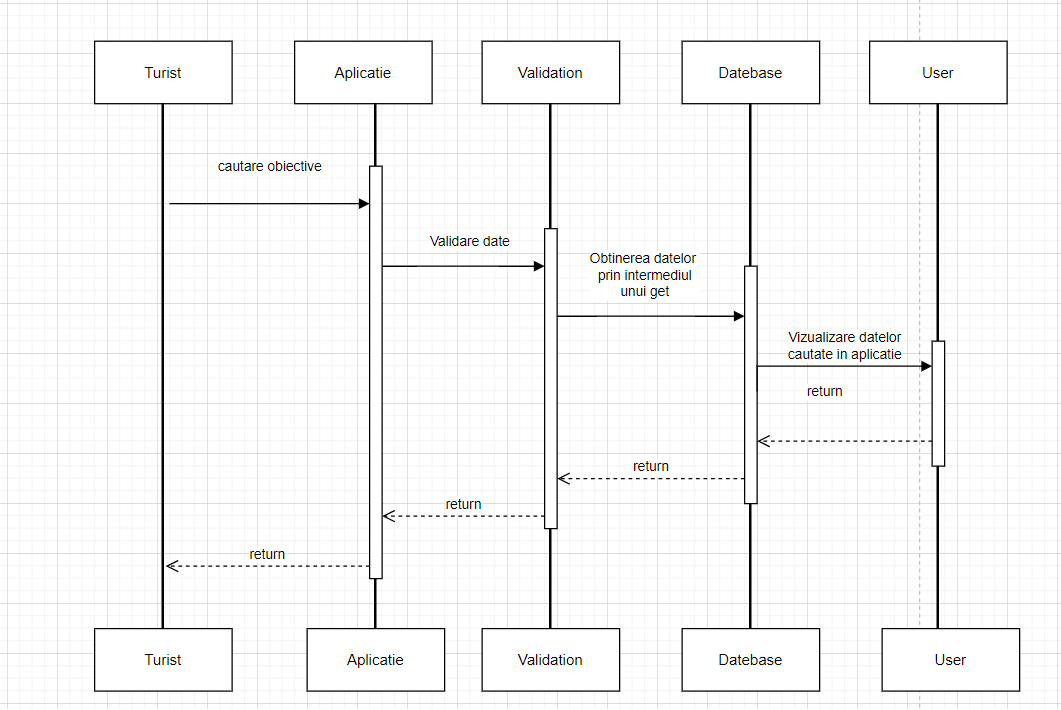
Descriere generată automat

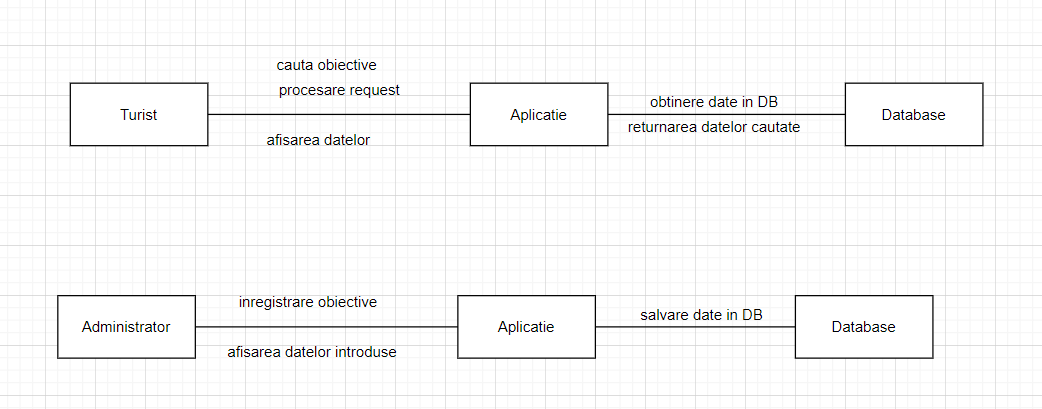
# Elaboration – Iteration 1.2

# Design Model

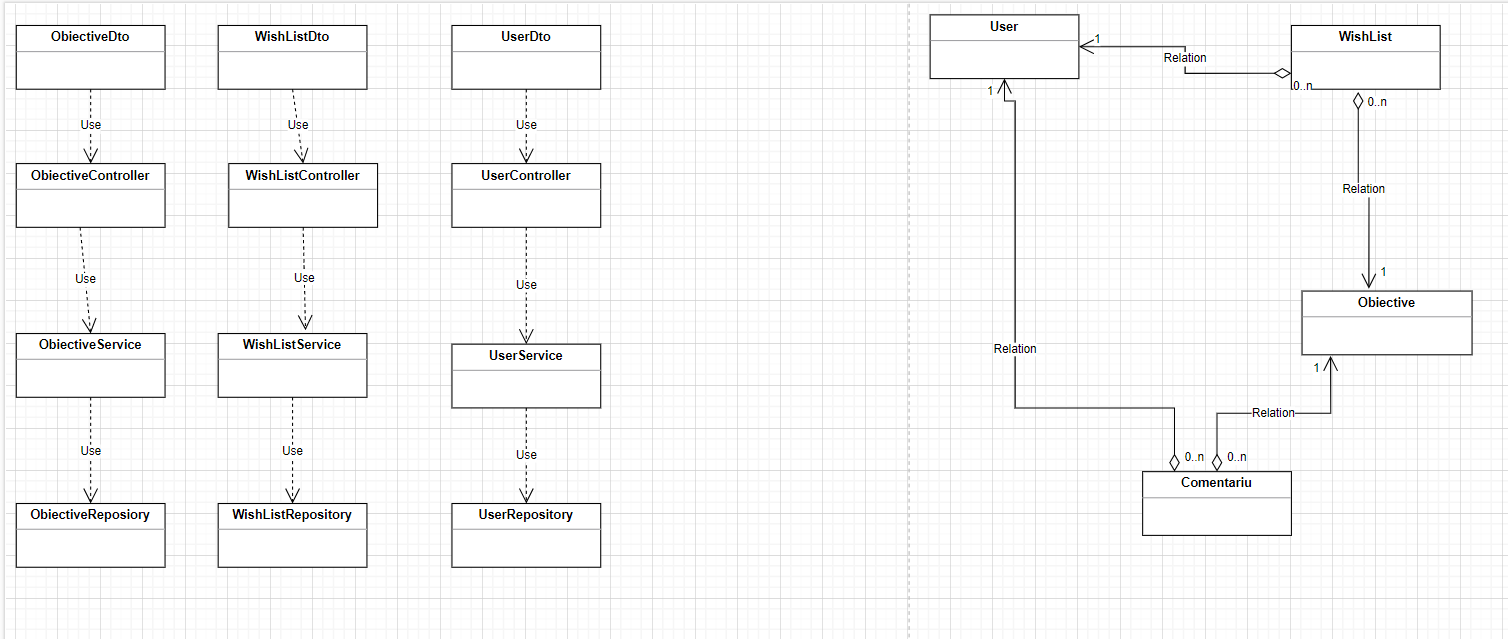
## Dynamic Behavior

O imagine care conține diagramă

Descriere generată automat  




## Class Design



Repository: Acest model este folosit pentru a abstractiza accesul la date și pentru a face schimbul de date între obiecte și baza de date mai ușor de gestionat și mai flexibil. Repository-ul gestionează toate operațiile de citire și scriere pe baza de date și este folosit pentru a izola logica de afaceri de detalii specifice bazei de date.

Observer: Acest model este util pentru a gestiona notificările între diferitele componente ale aplicației. În acest caz, ar fi util să implementăm acest model pentru a notifica utilizatorii logați în momentul în care se adaugă diferite oferte corespunzătoare obiectivelor lor de interes.

Factory: Acest model este util pentru a gestiona crearea de obiecte într-un mod mai eficient și mai ușor de întreținut. Ar fi util să implementăm acest model pentru a gestiona crearea de obiecte precum obiective turistice, comentarii, recenzii

Singleton: Acest model este util pentru a crea o singură instanță a obiectului de baza de date și pentru a oferi acces global la această instanță în cadrul întregii aplicații.

# Data Model

**User**

* ID (integer, primary key)
* Name (string)
* Email (string, unique)
* Password (string)
* Type (string) - either "admin" or "tourist"

**Obiective**

* ID (integer, primary key)
* Name (string)
* Description (string)
* Category ID (integer, foreign key referencing Category)
* Location ID (integer, foreign key referencing Location)
* Entrance Fee (float)
* Image (string)

**WishList**

* ID (integer, primary key)
* Tourist ID (integer, foreign key referencing User)
* Attraction ID (integer, foreign key referencing Attraction)

**Comentariu**

* ID (integer, primary key)
* Tourist ID (integer, foreign key referencing User)
* Attraction ID (integer, foreign key referencing Attraction)
* Rating (integer, 1-5)
* Comment (string)

Acest model de date include entități pentru User, Locație, Categorie,Obiectiv, WishList, Ofertă și Review. Entitatea User conține informații despre utilizatorii sistemului, în timp ce entitatea Vizita oferă context pentru atracții. Entitatea Oferta reprezintă entitatea principală de date a sistemului, conținând informații despre fiecare atracție, inclusiv locația sa, categorie, tarif de intrare și imagine. Entitatea WishList reprezintă relația dintre un turist și o anumită atracție pe care au marcat-o ca interesantă, în timp ce entitatea Ofertă reprezintă o promoție sau o reducere specială pentru o anumită atracție. În cele din urmă, entitatea Review permite turiștilor să lase feedback și evaluări pentru o atracție pe care au vizitat-o.

# Unit Testing

Testarea manuală este un proces de verificare a faptului că o aplicație funcționează conform așteptărilor prin utilizarea sa și evaluarea caracteristicilor, funcționalității și utilizabilității sale. Aceasta implică un tester care trece manual prin un set de cazuri de testare și efectuează acțiuni specifice pentru a testa sistemul.

Pentru aplicația Travel, pot fi utilizate următoarele metode de testare manuală:

Testare funcțională: Aceasta implică testarea funcționalității sistemului prin verificarea dacă fiecare caracteristică și funcție funcționează conform așteptărilor. Cazurile de testare pentru testarea funcțională pot include:

* Înregistrarea unui nou cont de utilizator și verificarea dacă contul este creat cu succes.
* Căutarea unei atracții specifice și verificarea dacă rezultatele căutării sunt precise.
* Adăugarea unei atracții la lista de interese și asigurarea că este adăugată corect.
* Lăsarea unei recenzii pentru o atracție și verificarea dacă recenzia este salvată și afișată corect.

Testare interfață utilizator: Aceasta implică testarea interfeței utilizator a aplicației prin verificarea dacă este ușor de navigat și de utilizat. Cazurile de testare pentru testarea interfeței utilizator pot include:

* Verificarea dacă aspectul aplicației este consistent pe diferite pagini.
* Verificarea că dimensiunea fontului și contrastul culorilor sunt adecvate pentru o citire ușoară.
* Testarea reactivității aplicației prin accesarea acesteia de pe diferite dispozitive cu dimensiuni diferite ale ecranului.

Testare utilizabilitate: Aceasta implică testarea utilizabilității generale a sistemului prin evaluarea cât de ușor poate fi utilizat pentru publicul său țintă. Cazurile de testare pentru testarea utilizabilității pot include:

* Solicitarea unui grup de utilizatori să efectueze sarcini specifice și evaluarea feedback-ului lor cu privire la cât de ușor sau dificil a fost să completeze sarcinile.
* Testarea accesibilității aplicației pentru utilizatorii cu dizabilități prin utilizarea tehnologiei asistive.

Testare performanță: Aceasta implică testarea performanței sistemului prin verificarea dacă poate gestiona sarcinile utilizatorilor și efectua sarcini eficient. Cazurile de testare pentru testarea performanței pot include:

* Simularea unui număr mare de utilizatori care accesează aplicația simultan și verificarea dacă timpul de răspuns este în limitele acceptabile.
* Testarea stabilității sistemului prin rularea acestuia pentru o perioadă prelungită și verificarea apariției eventualelor scurgeri de memorie sau a blocărilor.
* Testarea de securitate: Aceasta implică testarea securității sistemului prin identificarea eventualelor vulnerabilități și asigurarea protejării datelor utilizatorilor. Scenariile de testare pentru testarea de securitate pot include:
* Testarea aplicației pentru vulnerabilitățile de securitate comune, cum ar fi injecția SQL și scripting-ul de site-uri încrucișate.
* Asigurarea că datele utilizatorilor sunt criptate și securizate în timpul transmiterii și stocării.
* Testarea aplicației pentru conformitatea cu standardele de securitate, cum ar fi Regulamentul general privind protecția datelor (GDPR).
* În ansamblu, testarea manuală este o parte crucială a asigurării funcționării corecte a aplicației de călătorie și a îndeplinirii nevoilor utilizatorilor săi.

# Elaboration – Iteration 2

# Architectural Design Refinement

Ca design pattern-uri am folosit:

Factory Pattern: A fost folosit pentru a crea obiecte de tip Chart în funcție de numărul de obiective dintr-o anumită locație. Această abordare ajută la crearea și instanțierea corectă a obiectelor Chart într-un mod flexibil și ușor de extins în viitor.

Decorator Pattern: A fost aplicat pentru a colora obiectivele turistice în funcție de prețul lor. Astfel, este responsabil de decorarea graficului cu verde și roșu, în funcție de prețul specificat. Acest pattern ajută la adăugarea de comportamente suplimentare la obiecte existente, într-un mod modular și extensibil.

Observer Pattern: A fost utilizat pentru a notifica utilizatorii atunci când un nou obiectiv turistic este adăugat. Interfața Observer definește metoda update() care va fi implementată de obiectele User pentru a primi notificări. Clasa Subject (în acest caz, Location) gestionează lista de observatori (User-i) și trimite notificări atunci când un obiectiv nou este adăugat. Acest pattern asigură o comunicare slab cuplată între subiect și observatori și permite notificarea automată a utilizatorilor atunci când se întâmplă un eveniment relevant.

O imagine care conține diagramă, linie, Plan, Desen tehnic

Descriere generată automat

O imagine care conține text, captură de ecran, diagramă, număr

Descriere generată automat

O imagine care conține diagramă

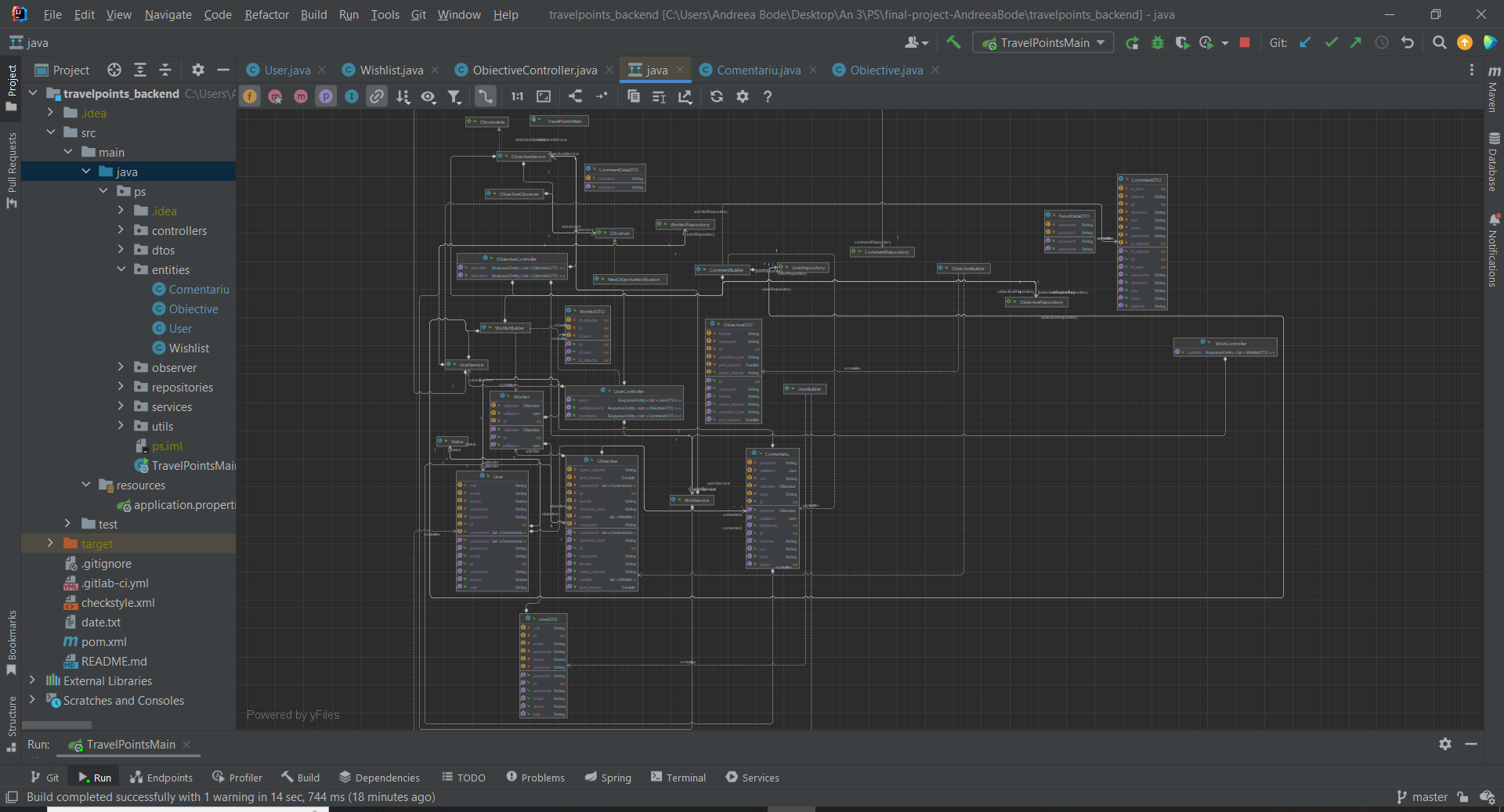
Descriere generată automat

O imagine care conține diagramă

Descriere generată automat

# Design Model Refinement

Am modificat diagrama de clase prin adaugarea pachetului observer, in cadrul acestui pachet sunt definite clasele si interfetele care implementeaza Observer Pattern. Am folosit acest design pattern pentru a notifica utilizatorii cand un nou obiectiv este adaugat in aplicatie. Mai jos este prezentata diagrama finala de clase.



# Construction and Transition

# System Testing

Testarea integrării este un aspect important al asigurării faptului că diferitele componente ale sistemului funcționează împreună corect. În cazul aplicației Travel, testarea de integrare poate fi efectuată pentru a verifica interacțiunile dintre back-end-ul Java (implementat folosind Spring și PostreSQL) și front-end-ul React.

Iată câteva scenarii de caz de testare pentru aplicația Travel:

Înregistrare utilizator:

* Scenariul 1: verificați dacă un nou utilizator se poate înregistra cu succes, furnizând e-mail și parolă valide.
* Scenariul 2: Verificați dacă înregistrarea eșuează dacă e-mailul este deja înregistrat.

Logare utilizator:

* Scenariul 1: verificați dacă un utilizator înregistrat se poate conecta cu succes cu acreditări valide.
* Scenariul 2: Verificați dacă autentificarea eșuează dacă e-mailul sau parola sunt incorecte

Resetare parola:

* Scenariul 1: verificați dacă un utilizator își poate reseta parola furnizând o adresă de e-mail validă.
* Scenariul 2: Verificați dacă resetarea parolei eșuează dacă adresa de e-mail nu este înregistrată.

Gestionarea locației de către administrator:

* Scenariul 1: verificați dacă un administrator poate adăuga o nouă locație cu toate detaliile necesare.
* Scenariul 2: verificați dacă un administrator poate modifica detaliile unei locații existente.
* Scenariul 3: Verificați dacă un administrator poate șterge o locație de pe platformă.

Căutați și vizualizați locații după locatie si categorie:

* Scenariul 1: verificați dacă un turist poate căuta locații pe baza filtrelor de locație și categorie.
* Scenariul 2: Verificați dacă un turist poate vizualiza informații detaliate despre o locație selectată.

Adaugă comentarii de către turiști:

* Scenariul 1: verificați dacă un turist poate adăuga un comentariu la o anumită locație sau atracție.

Contactați administratorul pentru întrebări sau actualizări:

* Scenariul 1: Verificați că un turist poate trimite un e-mail administratorului pentru a solicita mai multe informații despre o locație sau pentru a solicita actualizări.

Marcați locațiile ca preferate în funcție de turist:

* Scenariul 1: verificați dacă un turist poate marca anumite locații ca favorite pentru referințe viitoare.

Aceste scenarii de caz de testare acoperă principalele funcționalități și interacțiuni dintre administrator și rolurile de turist. Prin executarea acestor teste de integrare, ne putem asigura că diferitele componente ale sistemului funcționează împreună conform așteptărilor și gestionează corect interacțiunile utilizatorului.

# Future improvements

În ceea ce privește viitoarele îmbunătățiri ale sistemului Travel aceste funcționalități ar putea fi adăugate:

Adăugarea de Oferte: Se poate implementa posibilitatea ca administratorul să adauge oferte speciale pentru anumite obiective turistice. Utilizatorii ar trebui să fie notificați automat atunci când o nouă ofertă devine disponibilă pentru un obiectiv de pe lista lor de interese. Astfel, această funcționalitate ar oferi utilizatorilor acces la oferte exclusive și i-ar încuraja să viziteze mai multe locații.

Vizualizarea altor graficelor: Se poate dezvolta o funcționalitate care permite utilizatorilor să vizualizeze grafice și statistici relevante despre obiectivele turistice și locațiile de interes. De exemplu, un grafic ar putea arăta câte persoane au adăugat un anumit obiectiv la wishlist-ul lor, ceea ce ar putea oferi informații despre popularitatea și interesul pentru acel obiectiv.

Marcarea Obiectivelor ca vizitate: Utilizatorii pot avea opțiunea de a marca obiectivele turistice pe care le-au vizitat, astfel încât să poată ține evidența locurilor pe care le-au explorat. Aceasta ar putea fi o funcționalitate utilă pentru a vedea progresul și pentru a împărtăși experiențele cu alți utilizatori.

Recomandări personalizate: Sistemul ar putea fi îmbunătățit prin adăugarea de algoritmi de recomandare care să ofere sugestii personalizate utilizatorilor în funcție de interesele lor, locația și preferințele anterioare. Aceasta ar îmbunătăți experiența utilizatorilor prin furnizarea de sugestii relevante și ar încuraja explorarea de noi obiective turistice.

Integrarea cu servicii de hărți și navigare: Adăugarea unei funcționalități de hărți și navigare în aplicație ar permite utilizatorilor să obțină indicații și informații despre rutele și distanțele către obiectivele turistice. Aceasta ar ușura planificarea călătoriilor și ar îmbunătăți experiența de explorare.

Social Sharing: O funcționalitate de partajare socială ar permite utilizatorilor să împărtășească fotografii, recenzii și experiențele lor cu obiectivele turistice direct din aplicație, pe diverse platforme de social media. Aceasta ar încuraja interacțiunea între utilizatori și ar spori vizibilitatea aplicației.

Aceste îmbunătățiri ar adăuga valoare experienței utilizatorilor, ar stimula implicarea și interacțiunea între utilizatori și ar face aplicația Travel mai completă și mai atrăgătoare pentru turiști și administratori deopotrivă.

# Bibliography

* <https://www.baeldung.com/java-observer-pattern>
* <https://www.tutorialspoint.com/design_pattern/decorator_pattern.htm>
* <https://www.tutorialspoint.com/design_pattern/factory_pattern.htm>
* <https://react.dev/>
* <https://www.w3schools.com/java/>
* <https://www.javaguides.net/2020/07/spring-boot-react-js-crud-example-tutorial-spring-boot-backend-part1.html>
* https://www.javatpoint.com/spring-tutorial