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

LUCRARE DE DIPLOMĂ

Coordonator științific:
Ș.l. dr. ing. Silviu-Dumitru Pavăl

Absolvent:
Carla-Gabriela Hușman

Iași, 2024

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

—Proiectarea și implementarea unei aplicații web pentru construirea itinerariilor turistice personalizate—

LUCRARE DE DIPLOMĂ

Coordonator științific:
Ș.l. dr. ing. Silviu-Dumitru Pavăl

Absolvent:
Carla-Gabriela Hușman

Iași, 2024

DECLARAȚIE DE ASUMARE A AUTENTICITĂȚII PROIECTULUI DE DIPLOMĂ

Subsemnatul Huşman Carla-Gabriela,

legitimat cu CI seria XS nr. 121862, CNP 6010426375224

autorul lucrării Proiectarea și implementarea unei aplicații web pentru construirea itinerariilor turistice personalizate

elaborată în vederea susținerii examenului de finalizare a studiilor de licență, programul de studii Tehnologia informației organizat de către Facultatea de Automatică şi Calculatoare din cadrul Universității Tehnice „Gheorghe Asachi" din Iaşi, sesiunea iulie 2024 a anului universitar 2023-2024, luând în considerare conținutul Art. 34 din Codul de etică universitară al Universității Tehnice „Gheorghe Asachi" din Iaşi (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) şi a convențiilor internaționale privind drepturile de autor.

Data

01.07.2024

Semnătura

Cuprins

Introducere **1**

1 Fundamentarea teoretică și documentarea bibliografică **3**
1.1 Arhitectura aplicației . . . 4
1.2 Referințe la teme/subiecte similare . . . 5
1.2.1 TripAdivisor . . . 5
1.2.2 iPlan.ai . . . 5
1.2.3 Studiu științific . . . 5
1.3 Serviciile folosite în cadrul soluției . . . 6
1.3.1 Web Scraping . . . 6
1.3.2 Open AI . . . 7
1.3.2.1 ChatGPT . . . 7
1.3.2.2 DALL-E-3 . . . 8
1.3.3 Google . . . 8
1.3.3.1 Custom Search . . . 8
1.3.3.2 Place Search API . . . 8
1.3.3.3 Place Detail API . . . 8
1.3.3.4 Maps JavaScript API . . . 9
1.4 Baza de date a orașelor . . . 9
1.5 Tehnologii folosite . . . 9
1.5.1 Kotlin . . . 9
1.5.2 Python . . . 10
1.5.3 Angular . . . 10
1.5.4 Docker . . . 10

2 Proiectarea aplicației **13**
2.1 Modulul Controller . . . 14
2.2 Modulul Gateways . . . 14
2.3 Modulul Servicii . . . 15
2.4 Modele . . . 16
2.5 Managementul erorilor . . . 16
2.6 Componenta UI . . . 16
2.6.1 Memoria cache . . . 18

3 Implementarea aplicației **21**
3.1 Modulul Controller . . . 21
3.2 Modulul Gateways . . . 22
3.3 Modulul Servicii . . . 23
3.3.1 Serviciul extern Custom Search de la Google . . . 23
3.3.2 Serviciul extern WebScraping . . . 24
3.3.3 Serviciile externe ChatGPT și Dall-e de la OpenAI . . . 24

3.3.4 Serviciul extern Places de la Google . . . 26
3.3.5 Serviciul de filtrare a orașelor . . . 27
3.4 Modele . . . 28
3.5 Interfața cu utilizatorul . . . 28
3.6 Docker . . . 28
3.7 Dificultăți întâmpinate și modalități de rezolvare . . . 29

4 Testarea aplicației și rezultate experimentale **31**
4.1 Testare Custom Search . . . 33
4.2 Testare WebScraping . . . 34
4.3 Testare ChatGPT . . . 34
4.4 Testare Google Place . . . 36
4.5 Securitatea și protecția datelor . . . 36

Concluzii **37**

Bibliografie **39**

Anexe **41**
1 Controllerul aplicației . . . 41
2 Implementarea serviciului VacationPlanner pentru planificarea personalizată . . . 46
3 Implementarea serviciului ChatGPT . . . 49
4 Implementarea serviciului pentru managementul excepțiilor . . . 51
5 Fișierul pom.xml folosit pentru instalarea dependențelor proiectului cu Maven . . . 53

—Proiectarea și implementarea unei aplicații web pentru construirea itinerariilor turistice personalizate—

Carla-Gabriela Hușman

Rezumat

În contextul actual al tehnologiei informației, planificarea călătoriilor devine mult mai ușoară de realizat. Soluția proiectată în cadrul acestei lucrări își propune să ofere utilizatorilor itinerarii turistice personalizate, bazate pe informații simple precum locația curentă și preferințele sale. În timpul cercetării, au fost identificate diferențe între proiectul propus și alte aplicații similare, cum ar fi TripAdvisor, iPlan.ai, cât și un studiu pe același subiect, dar care folosește un algoritm de căutare locală iterată. Contribuția personală a constat în crearea unui sistem inovativ de generare a itinerariilor axat pe arhitectura monolit, folosind diverse funcționalități ale serviciilor oferite de companiile Google, OpenAI, WebScraping.

Procesul de planificare constă în preluarea din interfață a unor informații care sunt esențiale în descrierea unei vacanțe, oferind posibilitatea de completare și a unor opțiuni suplimentare pentru o planificare cât mai detaliată, și partajarea acestora către server care va aplica algoritmul implementat. Se utilizează diverse API-uri pentru a genera soluția. Acestea includ căutarea pe Google, extragerea conținutului prin WebScraping, integrarea modelului GPT-3.5 pentru sugestia punctelor de interes și a modelului DALL-E-3 pentru generarea imaginilor sugestive. Pentru fiecare punct de interes, se obțin detalii suplimentare folosind Google Places, iar în final se va livra utilizatorului itinerariul cu toate detaliile prelucrate din răspunsurile fiecărui apel către serviciile externe și se va afișa în interfață. Cu fiecare încercare de generare se observă libertatea răspunsurilor, adică pentru același input, călătoria poate fi diferită.

Aplicația poate fi rulată ca proces local sau într-un container Docker. Testările au fost realizate cu ajutorul variantei locale pentru verificarea comportamentului dorit, iar cea din urmă a fost implementată pentru o direcție viitoare de implementare.

În ceea ce privește securitatea datelor, aplicația utilizează o memorie cache pentru a proteja informațiile utilizatorilor și pentru a le oferi controlul absolut asupra datelor. Această abordare asigură că informațiile sunt stocate local, eliminând riscul de expunere a datelor sensibile și oferind o experiență securizată și controlată.

Introducere

În ultimii zeci de ani, importanța domeniului tehnologiei informației a crescut semnificativ, datorită schimbărilor care s-au petrecut în viețile oamenilor, cerând soluții performante, sigure, rapide și simple la problemele lor de zi cu zi. O problemă a acestor oameni, legată de industria turismului, a fost optimizarea timpului de planificare a unei călătorii, găsirea compactă a informațiilor și experiențe cât mai încântătoare. Astfel, transformările care au avut loc în toți acești ani au influențat drastic modul în care călătorim și ne bucurăm de destinațiile turistice din țara noastră sau din alte țări străine. Toate acestea sunt posibile doar având acces la internet, un smartphone sau un laptop.

Așadar, proiectul de față înglobează două mari domenii și anume turismul și tehnologia informației necesare dezvoltării unei aplicații web pentru planificarea și personalizarea itinerariilor turistice.

Aplicațiile web - programe software care rulează pe servere web fiind accesibile prin intermediul unui browser - au fost dezvoltate pentru a oferi accesibilitate ușoară de pe orice dispozitiv conectat la internet, fără a fi necesară instalarea unui software suplimentar.

Într-un secol dezvoltat din punct de vedere al tehnologiei, în care informațiile sunt ușor de obținut, existența unei aplicații de turism este foarte utilă pentru a satisface nevoile unei noi generații de călători. Aceasta va rezolva o nevoie de a avea acces la informații concise într-un singur loc, nemaifiind obligați să caute pe mai multe site-uri ceea ce îi atrag. În plus, timpul de căutare și planificare este diminuat.

Lumea în care trăim este plină de locuri deosebite, încărcate de istorie, cultură și tradiții, însă multe dintre acestea rămân ascunse publicului larg. Am ales această temă deoarece doresc să creez o aplicație care să servească drept punct de inspirație pentru turiștii începători, ajutându-i să descopere aceste comori ascunse și să creeze noi experiențe.

Această lucrare descrie în detaliu toți pașii care au contribuit la finalizarea soluției. În capitolul 1 sunt explicate conceptele folosite, limbajele utilizate și se prezintă la nivel general folosirea serviciilor externe.

Sistemul implementat se bazează pe arhitectura unui monolit datorită dimensiunii reduse a aplicației, ușurinței de dezvoltare și existența acesteia ca un tot unitar, în urma analizei asupra diferențelor dintre arhitectura pe servicii și a monolitului exemplificată în [1].

Tehnologiile folosite constau în utilizarea limbajului de programare Kotlin, pentru implementarea back-end-ului, datorită sintaxei și a diferitelor avantaje, și Angular pentru construirea interfeței cu utilizatorul. Aceasta din urmă permite folosirea unui structuri modulare, unde fiecare folder și fișier au rolul lor unic de îndeplinit. Pentru containerizarea aplicației s-a folosit Docker.

Înainte de a se trece la partea practică au fost studiate subiectele asemănătoare existente deja pe piață și s-a încercat îmbunătățirea lor, aducerea unui plus de originalitate și construirea unei idei de contur pentru a ști cum trebuie să fie virat utilizatorilor un astfel de produs software. Printre aplicațiile similare se enumeră TripAdvisor, care este recunoscut pentru multitudinea de ghiduri turistice prezente pe site și a interacțiunii dintre utilizatori, și iPlan.ai care se folosește de planificarea zilnică a unei vacanței.

În capitolul 2 se pune accentul pe modul în care a fost proiectată aplicația, fiind prezentate modulele generale ale acesteia și cum interacționează ele. Pentru a fi înțeles algoritmul de planificare personalizată este oferită o diagramă logică care ilustrează toți pașii necesari înțelegerii. Alte diagrame care se regăsesc în acest capitol sunt pentru baza de date și componentele la nivel de back-end și front-end.

Sistemul implementat este structurat în controllere, modele, servicii și gateway-uri, unde front-end-ul și back-end-ul comunică printr-o cerere HTTP. De la o componentă la alta sunt trans-

ferate date cu rol de intrare, urmând ca mai apoi să fie prelucrate după comportamentul dorit, iar ieșirea înglobează toate operațiile și filtrele aplicate pe informațiile introduce în interfață. Astfel au fost folosite câteva servicii externe pentru a beneficia de funcționalitățile acestora, fără a fi nevoie de implementat individual prin tehnici de învățare automată.

Așadar, controllerul utilizat este o componentă care gestionează fluxul de date între utilizator și aplicație, interacționând cu modelele pentru a prelua sau manipula datele și returnează răspunsurile. Modelele sunt responsabile pentru gestionarea comunicării dintre controller și serviciu. Gateway-urile facilitează comunicarea între sistemele actual și serviciile externe, fiind intermediar între cele două. Pentru fiecare conectivitate cu un serviciu extern, este nevoie de o cheie secretă care poate fi creată odată cu crearea unui cont pe site-urile oficiale și de câțiva parametri de configurare prezentați și explicați în capitolul 3 secțiunea 3.2. Serviciile implementate sunt componente care conțin logica și funcționalitățile. Sunt folosite pentru a împărți aplicația în părți mai mici, facilitând reutilizarea codului și testarea.

Dacă în capitolul 2 au fost menționate rolurile componentelor aplicației, în capitolul 3 este prezentat și explicat modul de implementare al fiecăreia. În primă fază se colectează informații din interfață care vor constitui baza creării unui prompt de căutare pentru o cerere către Custom Search de la Google. În urma apelului, se vor obține câteva linkuri, din care se vor selecta doar două, și cu ajutorul serviciului de la WebScraping se va extrage sursa fiecărei pagini pentru a obține orașele cu ajutorul programului de recunoaștere și generare de text, ChatGPT. Acum că avem potențialele orașe țintă, se selectează aleatoriu unul ca destinație și pentru acesta se va genera o descriere, puncte cheie și o listă de puncte de interes cu ChatGPT. Pentru fiecare punct de interes în parte, cu suportul serviciilor Google Places, se vor dobândi informații veridice despre acestea, precum adresă, locație, număr de contact, etc.

Interfața aplicației este concepută pentru a fi simplă, sugestivă și prietenoasă, oferind indicații clare despre utilizarea acesteia. Pentru ca să fie asigurată portabilitatea proiectului, s-a folosit containerizarea soluției cu Docker.

În capitolul 4 au fost menționate testele prin care a fost trecută aplicația pentru a oferi o soluție cât mai aproape de ideal. Până la momentul în care produsul software să fie testat ca o unitate întreagă, s-au realizat teste unitare pentru fiecare serviciu și gateway implementat pentru a asigura buna funcționare și comunicare cu API-urile externe care contribuie la construirea itinerariului.

Problemele care pot apărea pe parcurs țin de serviciile externe care nu sunt mereu disponibile ori din cauza unei probleme la nivel de sistem al companiei, ori din cauza lipsei conectivității la internet. În cazul oricărei probleme, în interfață, utilizatorul este rugat să repete procesul ori să revină peste câteva momente.

Din punct de vedere al securității, s-a optat pentru implementarea unei memorii cache la nivel de browser pentru a stoca detaliile utilizatorului și itinerariile salvate. Această abordare are ca scop protejarea datelor utilizatorului în fața potențialelor amenințări de securitate, oferindu-i totodată controlul asupra informațiilor stocate și posibilitatea de a șterge aceste date oricând.

Capitolul 1. Fundamentarea teoretică și documentarea bibliografică

În ultimele decenii, tehnologia informației a revoluționat modul în care călătorim și experimentăm destinațiile turistice din întreaga lume. Inovațiile tehnologice au transformat dramatic industria turismului, deschizând noi posibilități și prezentând indivizilor oportunități neașteptate. Unul dintre cele mai evidente impacturi ale tehnologiei informației asupra acestei industrii este simplificarea și accelerarea procesului de planificare. Cu accesul ușor la internet și la o varietate de aplicații și platforme online, călătorii pot acum căuta, compara și rezerva transport, cazare și activități în doar câteva clicuri. După cum se menționează în [2], sinergia care există între aceste două domenii va continua să schimbe modul în care călătorim și selectăm informațiile.

Prin urmare, acest proiect se încadrează în domeniul tehnologiei informației, în sectorul turismului și aparține subdomeniului dezvoltării de aplicații web pentru planificarea și personalizarea itinerariilor turistice.

Una dintre problemele care pot apărea în acest proces este aceea că există atât de multe informații, destinații și atracții încât devine dificilă planificarea unei călătorii, ceea ce duce la lipsa de hotărâre. O altă dificultate este securitatea datelor care este abordată în secțiunea 4.5 și planificarea călătoriilor în destinații care pot pune viața unei persoane în pericol, explicată mai pe larg cum s-a evitat această problemă în secțiunea 1.4.

Scopul proiectului prezent este de a dezvolta o aplicație software care să ofere utilizatorilor un itinerariu turistic personalizat pentru destinația dorită. Aplicația propusă oferă un punct de pornire și inspirație pentru călătorie, permițând introducerea informațiilor necesare pentru crearea unei vacanțe. Acestea includ, dar nu se limitează la gen, vârstă, buget, mijloc de transport, anotimp, însoțitori și destinație. Cu toate că utilizatorii sunt încurajați să completeze toate aceste câmpuri pentru o experiență cât mai precisă, doar punctul de plecare și preferințele sunt obligatorii.

Am ales această temă deoarece lumea în care trăim are o multitudine de peisaje diverse și locuri deosebite care merită descoperite. Fiind foarte dificil în a găsi o primă destinație în cazul în care individul e la început, o aplicație web care centralizează dorințele acestuia, este un beneficiu adus tuturor părților implicate. Importanța site-urilor web a crescut în ultimii ani, datorită digitalizării, ceea ce oferă o accesibilitate crescută asupra informațiile la îndemână oricând și oriunde, cu limita de a fi conectat la o rețea de internet.

Îmbinând beneficiile unui site web și experiențele pe care le oferă industria turismului, au fost alcătuite obiectivele care se doresc a fi atinse la finalizarea acestui proiect:

- să poată ajuta turiștii începători să exploreze comorile ascunse ale acestei lumi;
- planificarea să necesite cât mai puțin timp;
- utilizatorul să nu fie nevoit să caute informații legate de adresă, program, număr de telefon etc. pe internet, toate aceste informații fiind într-un singur loc;
- individul să-și creeze o idee despre un anumit punct de interes doar dintr-o descriere plină de tradiții, cultură și istorie;
- călătorul să nu fie trimis în destinații care pot atenta la viața acestuia;
- informațiile personale pe care dorește să le împărtășească utilizatorul în aplicație să fie lăsate la latitudinea acestuia, nefiind constrâns să completeze toate câmpurile;
- securitatea datelor acestuia, făcându-l pe individ să aibă control deplin asupra datelor sale.

Interfața aplicației este concepută pentru a fi simplă, sugestivă și prietenoasă, oferind indicații clare despre utilizarea și scopul acesteia. Se dorește ca experiența utilizatorului cu aplicația să fie plăcută. Aceasta va permite crearea unui cont, dar acesta va fi valabil doar la nivel de browser, odată cu schimbarea dispozitivului sau a motorului de căutare, toate datele salvate nu vor mai putea fi accesate.

Înainte de implementarea și proiectarea aplicației au fost studiate aplicațiile care există deja pe piață și s-au căutat să se aducă îmbunătățiri, punând accentul pe interacțiunea utilizatorului cu aplicația, dar și pe livrarea unor răspunsuri cât mai concise. Răspunsuri care ar putea fi obținute prin căutări succesive pe internet, dar care ar necesita destul de mult timp. Ca urmare a analizei serviciilor puse la dispoziție de diferite companii, au fost alese doar cele ale căror funcționalități coincideau cu viziunea proiectului de față prin furnizarea informațiilor diversificate. Fiecare au fost studiate individual și testate manual pentru a înțelege modul lor de funcționare. De exemplu, în cadrul serviciilor pentru generare de text, prin teste multiple, au fost obținute prompt-uri care să ofere o ieșire cât mai fezabilă prin înțelegerea modului lor de funcționare.

1.1. Arhitectura aplicației

Aplicația de față este de dimensiune mică, unde toate funcționalitățile sunt construite într-o unitate unică, indivizibilă, într-un singur proces. Interfața cu utilizatorul și logica care se află în back-end sunt implementate împreună și strâns legate. Prin analiza unei comparații dintre arhitectura bazată pe servicii și arhitectura monolit [1], s-a optat pentru cea din urmă datorită simplității, dezvoltării ușoare, depanării și instalării. Întreaga aplicație este ambalată într-un singur pachet de instalare, cum ar fi într-un container Docker, iar eventualele erori care pot apărea în timpul rulării se găsesc destul de repede. În cazul în care se dorește să se facă o actualizare, aceasta se poate realiza într-un singur pas prin copierea noului fișier care conține aplicația.

Front-end-ul comunică direct cu back-end-ul și trimite o cerere pentru a obține ieșirea funcționalității implementate printr-o cerere HTTP (***H**yper**t**ext **T**ransfer **P**rotocol*). Acest mod de comunicare induce cuplarea strânsă între cele două entități astfel încât o modificare în front-end, afectează back-end-ul și invers.

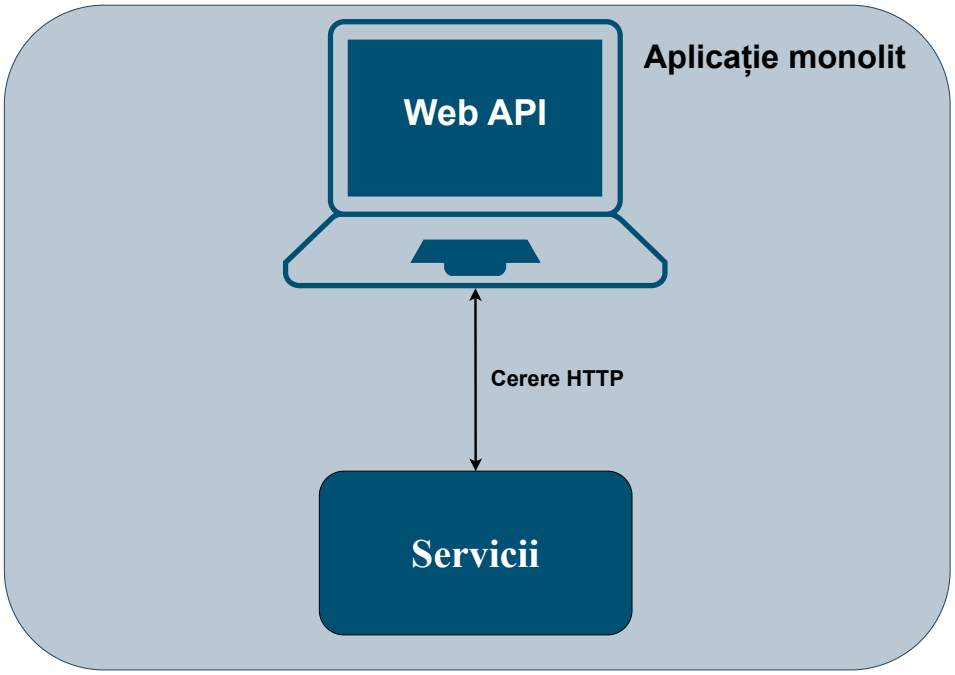

Figura 1.1. Model de interacțiune între front-end și back-end conform arhitecturii monolit

1.2. Referințe la teme/subiecte similare

Pe parcursul procesului de cercetare al pieței, au fost identificate mai multe aplicații similare cu tema proiectului propus, printre care TripAdvisor și iPlan.ai, dar și multe documente științifice pe acest subiect de personalizare. Platformele menționate anterior oferă utilizatorilor posibilitatea de a căuta și alege destinațiile, dar există diferențe semnificative față de proiectul prezent.

1.2.1. TripAdivisor

O aplicație similară cu tema proiectului este TripAdvisor [1], care oferă tururi deja gata implementate și prezentate pe site, rămânând în sarcina utilizatorului efortul de a căuta locațiile care îl interesează. Diferența cheie între proiectul de față și această platformă constă în modul în care generăm și prezentăm itinerariile personalizate. Deși TripAdvisor livrează clienților o multitudine de informații despre hoteluri, restaurante și atracții turistice, găsirea călătoriei dorite necesită un efort în plus. Aplicația propusă dorește să rezolve această problemă prin furnizarea utilizatorului călătoria perfectă cu doar câteva click-uri. Deși TripAdvisor dispune și de opțiunea de a-ți planifica vacanța cu ajutorul inteligenței artificiale, acesta nu este mereu disponibilă, generând disconfort pentru utilizatori în situații neprevăzute.

1.2.2. iPlan.ai

Această platforma[2] adoptă o strategie de planificare zilnică, abordare benefică pentru cei care doresc o programare detaliată. Înainte de a obține produsul final, platforma constrânge utilizatorul să aleagă ce îi atrage atenția dintr-o listă de puncte de interes. Informațiile pentru o singură atracție turistică sunt o scurtă descriere, programul și un link care redirecționează către Google Maps pentru a vedea poziționarea pe hartă. Proiectul de față oferă o experiență adaptată nevoilor specifice utilizatorului prin eliminarea pasului de alegere dintr-o listă extinsă și menținerea hărții pe site. Astfel nu este limitat la un număr fix de puncte de interes într-o singură zi, deoarece poate dori să petreacă mai mult timp într-o anumită zonă și mai puțin în alta. Prin intermediul punctelor de interes sugerate, individul are flexibilitatea de a-și crea propriul program conform preferințelor sale, având posibilitatea de a explora și de a se bucura de călătoria sa în modul dorit.

1.2.3. Studiu științific

În lucrarea științifică „A Fast Algorithm for Personalized Travel Planning Recommendation” [3] este documentată problema care apare la proiectarea itinerariilor turistice personalizate: preferințele utilizatorilor și factorii care influențează acestea, cum ar fi timpul, programul punctelor de interes și distanța dintre ele. Autorii propun un algoritm de căutare locală iterată (Iterated Local Search), o metaeuristică ce îmbunătățește iterativ o soluție, explorând vecinii soluției actuale și ajustând pentru a îmbunătăți calitatea rutei.

În cazul de față, nu s-a optat pentru folosirea algoritmilor de optimizare sau a inteligenței artificiale, ci s-a optat pentru proiectarea și dezvoltarea unui sistem inovativ care integrează o serie de servicii, oferite de companii precum Google (1.3.3), WebScraping (1.3.1) și OpenAI (1.3.2), pentru a sugera noi oportunități de călătorie personalizate pentru fiecare utilizator în parte în funcție de preferințele specifice acestuia.

Sistemul se folosește de serviciile oferite de către OpenAI, pentru a rezolva diferitele sarcini de generare de conținut, în vederea descrierii sugestive a punctelor de interes care pot alcătui diverse itinerarii. Conform [4], „LLM-urile[3] nu sunt capabile să genereze planuri executabile atunci când sunt utilizate în moduri autonome”, dar acestea pot ajuta la procesul de planifi-

[1] https://www.tripadvisor.com/
[2] https://iplan.ai/
[3] Large language models - modele de învățare profundă care sunt antrenate pe seturi mari de date

care. Din această cauză sunt necesare și folosite și alte servicii pentru a perfecționa acest proces. Așadar, prin testele manuale efectuate înainte de implementare, s-a căutat cel mai util prompt, prin îmbunătățirea continuă a acestuia, cu ajutorul inteligenței umane, cât și a celei artificiale.

1.3. Serviciile folosite în cadrul soluției

Pe piață există multe companii care și-au creat propriile servicii care pot fi folosite contra cost sau gratis de către oricine care deține o cheie unică pentru serviciul respectiv. Comunicarea dintre proiectul de față și respectivele funcționalități se realizează conform Figura 1.2. API (***A**pplication **P**rogramming **I**nterface*) este un intermediar între cele două entități și le permite să comunice. Totodată acesta extrage date din entitatea A, cea care inițiază cererea, și le transmite către entitatea B, cea care primește cererea. Astfel, pentru dezvoltarea soluției au fost nevoie de o serie de astfel de servicii pentru a putea obține informațiile necesare care sunt explicate în cele ce urmează.

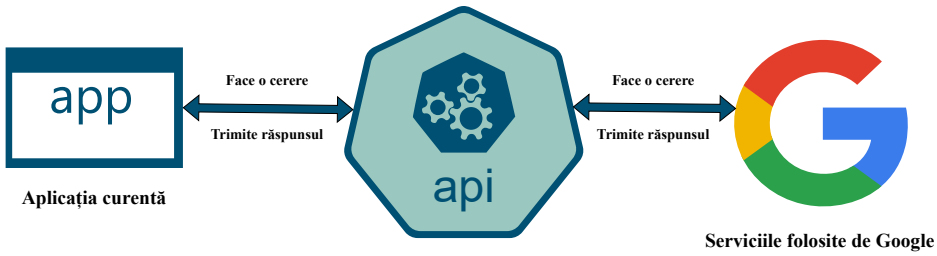

Figura 1.2. Model de comunicare între un serviciu și proiectul curent prin API

1.3.1. Web Scraping

WebScrapingApi [5] este un serviciu care oferă posibilitatea de a extrage informații de pe site-urile web prin descărcarea paginii pentru obținerea codului HTML (***H**yper**T**ext **M**arkup **L**anguage*). Un exemplu de cerere trimisă printr-o comandă cURL (***c**lient **URL***) în terminal se pezintă astfel:

```
$ curl "https://api.webscrapingapi.com/v1?api_key=<cheia>&url=<url_site>& \
render_js=<valoare>&proxy_type=<valoare>"
```

Cererea poate conține mai mulți parametri, în funcție de ceea ce se dorește a se obține și pot fi consultați în documentația serviciului [5]. În acest exemplu și în acest proiect vor fi folosiți doar parametri specificați după cum urmează:

- *api_key* - parametru obligatoriu și este reprezentat de un șir de caractere care compun cheia unică a utilizatorului cu care poate accesa funcționalitățile serviciului;

- *url* - parametru obligariu și specifică URL-ul (***U**niform **R**esource **L**ocator*)[4] paginii web țintă, din care se dorește extragerea informațiilor. Este important de precizat faptul că șirul de caractere care alcătuiește URL-ul trebuie să fie sub formă encodată;

- *render_js*[5]- parametru opțional și este o valoarea întreagă care poate fi *0* sau *1*. Exprimă dorința utilizatorului dacă să activeze codul JavaScript sau nu;

- *proxy_type*[6] - opțional și poate lua două valori: *datacenter* sau *residential*. Diferența majoră între cele două constă în faptul că atunci când se accesează site-ul, acesta poate să blocheze adresa IP (***I**nternet **P**rotocol address*) în cazul *residential*.

1.3.2. Open AI

OpenAI este o companie care dorește să ofere diferite servicii indivizilor pentru a se putea folosi de beneficiile pe care le poate oferi inteligența artificială.

1.3.2.1. ChatGPT

Pentru generarea de text pe baza unui input, numit prompt, se folosește modelul gpt-3.5-turbo [6] care este accesibil oricărui utilizator, posesor al unei chei create. Exemplul unei cereri pentru generarea de text este ilustrat mai jos, unde există două roluri: cel de sistem, care ajută modelul să înțeleagă rolul lui și pe ce domeniu să se axeze când livrează răspunsurile, și cel de utilizator, care conține ceea ce vrea el să comunice, mesajul propriu-zis.

```
$ curl https://api.openai.com/v1/chat/completions \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer $OPENAI_API_KEY" \
  -d '{
    "model": "gpt-3.5-turbo",
    "messages": [
      {
        "role": "system",
        "content": "Mesaj care ii spune sistemului rolul sau"
      },
      {
        "role": "user",
        "content": "Textul pe care doresti sa il trimiti"
      }
    ]
  }'
```

Acest model are câteva limite care constau în:

- cheia generată este valabilă doar 3 luni, urmând ca mai apoi să se treacă la modelul gpt-4o care este mai inteligent și rapid, dar contra cost;

- permite doar 3 cereri pe minut pentru a nu face abuz și ca să se evite un atac de tipul DoS (***D**enial-**o**f-**S**ervice*)[7] asupra sistemului.

[4] identificator unic utilizat pentru identificarea unică a unei resurse pe internet

[5] https://docs.webscrapingapi.com/webscrapingapi/advanced-api-features/rendering-javascript

[6] https://docs.webscrapingapi.com/webscrapingapi/advanced-api-features/proxies

[7] atac care poate face serviciul inaccesibil din cauza cererilor abuzive rău intenționate

1.3.2.2. DALL-E-3

Pentru funcționalitatea de generare de imagini [7] se folosește modelul DALL-E-3 prin crearea unui prompt care conține detaliile descriptive ale imaginii. Inputul[8] nu trebuie să conțină cuvinte precum *real* deoarece modelul poate avea dificultăți în înțelegerea sensului.

Exemplu de cerere:

```
$ curl https://api.openai.com/v1/images/generations \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer $OPENAI_API_KEY" \
  -d '{
    "model": "dall-e-3",
    "prompt": "Descriere imagine",
    "n": 1,
    "size": "1024x1024"
  }'
```

În exemplul anterior *n* reprezintă numărul de imagini care se doresc a fi generate, iar *size* este dimensiunea imaginii.

Imaginile obținute în răspuns pot fi accesate prin link-ul oferit, dar acesta este valabil doar o oră pentru a nu se aglomera baza de date a furnizorului de servicii.

1.3.3. Google

1.3.3.1. Custom Search

Acest serviciu [8] este util pentru a obține răspunsurile în urma căutării pe Google prin intermediul codului, pentru a prelucra respectivele informații.

Cererea se creează în modul următor:

```
$ curl "https://customsearch.googleapis.com/customsearch/v1?q=<prompt>&key= \
<cheia_ta>&safe=<valoare>"
```

Parametrii folosiți au următoarele semnificații: *q* reprezintă textul de căutare, *key* este cheia și *safe* indică dacă rezultatele să fie cât mai sigure posibil, eliminând conținutul explicit pentru adulți.

1.3.3.2. Place Search API

În urma introducerii unei denumiri care reprezintă un oraș, un restaurant, o destinație sau orice altceva, serviciul [9] va returna o listă de locații care se potrivesc cu textul inputului. Cu cât textul este mai precis, cu atât rezultatele vor fi mai precise.

Un exemplu de cerere este ilustrat astfel:

```
$ curl "https://maps.googleapis.com/maps/api/place/textsearch/json?key= \
<cheia_ta>&query=<denumire_punct_de_interes>"
```

În cererea anterioară *key* este cheia unică pentru serviciu și *query* este șirul de caractere după care se dorește a se face căutarea. De exemplu „Centru Vechi, Brașov, România”.

Răspunsul va fi sub format JSON (***J**ava**S**cript **O**bject **N**otation*) și va conține informații despre adresă, latitudine, longitudine, denumire, id-ul unic prin care se identifică respectiva locație.

1.3.3.3. Place Detail API

Serviciul anterior nu oferă destule informații despre o locație, de aceea este necesar Place Detail [10]. Informațiile obținute în plus față de cele de la subsecțiunea anterioară sunt programul

[8]Tutorial pentru contruirea promptului: https://the-decoder.com/prompt-design-for-dall-e-2-photorealism-emulating-reality/

de lucru al punctului de interes, referință către o imagine reprezentativă pentru acesta, numărul de telefon, website și rating.

O cerere către acest serviciu arată astfel, unde *key* este cheia unică și *place_id* este id-ul obținut de Place Search API:

```
$ curl "https://maps.googleapis.com/maps/api/place/details/json?key= \
<cheia_ta>&place_id=<valoare_id>"
```

1.3.3.4. Maps JavaScript API

Acest serviciu [11] oferit de Google este folosit pentru a construi aplicații web dinamice și interactive, integrând pe pagină harta ca utilizatorul să vizioneze în timp real destinațiile. Cu ajutorul acestui API, s-au adăugat repere numerotate ca să se identifice locațiile pe hartă.

1.4. Baza de date a orașelor

În urma căutării pe google pentru o posibilitate de a obține toate orașele din lume, am găsit un fișier text [12] pe un cont de GitHub, care conține informațiile dorite. Acest fișier constituie baza de date a aplicației care a fost obținută prin prelucrarea și manipularea informațiilor din fișierul inițial.

Pentru a oferi turiștilor o experiență sigură, au fost excluse orașele și țările care ar fi putut pune viața indivizilor în pericol, informații care au fost obținute prin consultarea portalurilor de știri. Totodată, s-au eliminat destinațiile care au un număr de locuitori mai mic decât 1000, dar și denumirile de orașe care conțineau cuvinte precum *sat*, *comună*, cât și echivalentul acestora în celelalte limbi străine, cum ar fi *Okres*, *Kreis*, etc. Astfel dintr-un total inițial de 150.573 de înregistrări, s-au obținut aproximativ 111.800 de orașe, la care s-au adăugat denumirile continentelor și țărilor din care fac parte.

1.5. Tehnologii folosite

Mai jos sunt enumerate toate tehnologiile folosite în implementarea proiectului. Sunt evidențiate și motivele alegerii acestora în defavoarea altor tehnologii existente pe piață. Detalierea fiecărei se limitează doar la cele folosite în acest proiect.

1.5.1. Kotlin

Logica aplicației este implementată în Kotlin în favoarea altor limbaje de programare datorită sintaxei sale concise și sigure. Este folosit împreună cu framework-ul[9] Spring Boot, care oferă o integrare cu Java, facilitând dezvoltarea rapidă și permite eliminarea configurațiilor care să se repete în mai multe locuri fără nicio modificare. Totodată, permite injectarea dependențelor prin adnotarea claselor: *@Service*, *@Controller*, *@CotrollerAdvice*, etc. În plus este folosit Maven, un sistem de build și administrare a proiectelor, datorită beneficiilor aduse în gestionarea dependențelor care sunt specificate într-un fișier XML (***E**xtensible **M**arkup **L**anguage*), iar instalarea se realizează automat de către Maven. Fișierul este disponibil în Anexa 5.

Tabelul 1.1. De ce Kotlin și nu alt limbaj?

	Kotlin	***Java***
Sintaxa	Este mult mai concisă, iar o anumită operație din Java poate fi făcută în câteva rânduri cu Kotlin	Este mai complicată, necesită mai mult efort de scriere a codului

[9]Framework-ul este o structură predefinită de cod pe care se poate construi software

Continuarea tabelului 1.1

	Kotlin	***Java***
Siguranța null	Implicit variabilele au valorile diferite de null până în momentul atribuirii explicite. Erorile pot apărea la compilare, nu în timpul execuției	Din cauza că permite variabilelor să aibă valoarea null, în timpul execuției pot apărea excepții
Compatibilitate	Este compatibil cu Java. În cazul în care se dorește o anumită librărie din Java, aceasta se poate obține imediat	-
Compilator	Reduce erorile din timpul compilării, nu în timpul executării, prin efectuarea unei multitudini de verificări	Nu este la fel de performant ca cel din Kotlin
Clase de date	Pentru a păstra date, în definirea clasei se inserează cuvântul *data*, iar apoi compilatorul va genera automat constructorul și funcțiile de obținere și setare pentru câmpuri	Într-o clasă trebuie definite manual constructorul, funcțiile de obținere și setare pentru câmpuri

1.5.2. Python

Acest limbaj a fost folosit pentru crearea unui script care a ajutat la obținerea bazei de date a orașelor lumii prin prelucrarea fișierului inițial. S-a optat pentru Python datorită simplității și eficienței în gestionarea operațiilor de citire, scriere, procesare a datelor și al lucrului cu vectori.

1.5.3. Angular

Angular este un framework care gestionează cerințele complexe ale aplicațiilor web și este axat pe o arhitectură bazată pe componente, care folosește TypeScript. A fost ales datorită modului în care fișierele sunt împărțite în funcție de rolul lor, ceea ce a facilitat implementarea. Este utilizat Angular Material, care a ajutat la designul aplicației și la construirea paginilor web atractive.

1.5.4. Docker

Această platformă de dezvoltare, livrare și rulare a aplicațiilor oferă un mediu izolat indiferent de configurația sistemului gazdă. Atât front-end-ul, cât și back-end-ul, cu toate dependențele lor, au fost împachetate într-o singură unitate denumită container. Acesta asigură că aplicația rulează la fel pe orice platformă.

Pentru a putea realiza cele menționate mai sus au fost create fișiere Dockerfile pentru fiecare, utile în definirea mediului și a dependențelor necesare rulării atât a front-end-ului, cât și a back-end-ului.

Exemplu de fișier Dockerfile:

```
1  FROM <imagine>
2  WORKDIR <folder_de_lucru>
3  COPY <target> <folder_de_lucru>
4  EXPOSE <port>
5  RUN <comanda>
6  CMD <comanda>
```

Instrucțiunile de mai sus au următorul rol:

- *FROM* specifică imaginea de bază pe care se construiește noua imagine Docker;
- *WORKDIR* setează directorul de lucru pentru următoarele instrucțiuni;
- *COPY* copiază fișiere sau directoare din proiectul local în imaginea Docker;
- *EXPOSE* specifică portul pe care containerul îl va expune la rulare;
- *RUN* execută comenzi în timpul construirii imaginii;
- *CMD* specifică comanda care va fi rulată când containerul este pornit.

Pentru ca cele două componente să poată rula împreună și comunica, este nevoie de un fișier Docker Compose în care se specifică configurațiile necesare ca serviciile să funcționeze împreună. Exemplu de fișier Docker Compose:

```
1  version: '<versiune>':
2  services:
3    backend:
4      build:
5        context: <folder_backend>
6        dockerfile: Dockerfile
7      ports:
8        - "<port_container>:<port_gazdă>"
9    frontend:
10     build:
11       context: <folder_frontend>
12       dockerfile: Dockerfile
13     ports:
14       - "<port_container>:<port_gazdă>"
```

Capitolul 2. Proiectarea aplicației

Propun o abordare clasică pentru construirea unui API prin folosirea arhitecturii software RESTful (***RE**presentational **S**tate **T**ransfer*) API utilă creării serviciilor web. Astfel, sistemul este compus dintr-un client care trimite o cerere pentru o resursă și un server care are resursa respectivă. Controllerele REST se ocupă de gestionarea cererilor HTTP și de maparea acestora la metodele corespunzătoare din servicii, care conțin logica de business, iar modelele reprezintă datele utilizate de aplicație.

Front-end-ul și back-end-ul comunică printr-o cerere HTTP cu metoda POST[10], endpoint-ul fiind *http://localhost:8080/api/v1/relation-trip/planner*. În corpul cererii este trimis un JSON care conține informațiile introduse de utilizator, necesare pentru procesul de personalizare.

Aplicația este structurată în mai multe componente interconectate conform Figura 2.1, unde se poate observa prezența componentei de UI (***U**ser **I**nterface - Interfața cu utilizatorul*), care înglobează funcționalitățile front-end-ului și componenta server a aplicației care constituie serverul alcătuit din controllere, servicii, gateways, modele și un manager de excepții, rolul fiecăruia fiind detaliat în cele ce urmează.

S-a optat pentru această structură fragmentată deoarece s-a dorit ca fiecare responsabilitate unică pe care trebuie să o îndeplinească aplicația să nu se amestece cu celelalte. Avantajele acestei abordări constau în organizarea și întreținerea codului, grad mare de reutilizare, lizibilitatea îmbunătățită și înțelegerea codului.

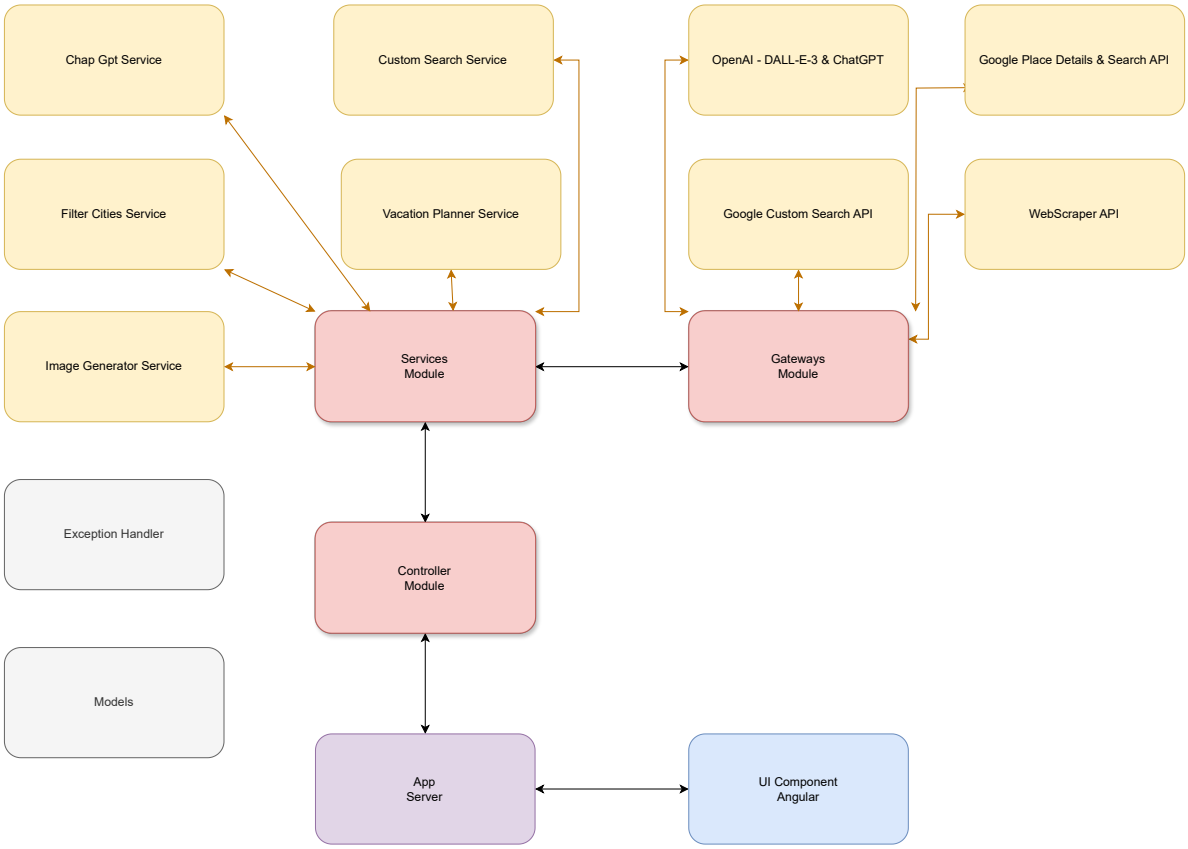

Figura 2.1. Diagrama de componente a sistemului

[10]trimite date către server, permite crearea sau actualizarea unei resurse, modificând astfel starea acestora

Pentru a asigura portabilitatea și consistența mediului de rulare, aplicația a fost containerizată folosind Docker, ceea ce a implicat împachetarea împreună a tuturor componentelor cu toate dependențele într-un singur container.

Proiectul de față nu este constrâns de cerințe hardware specifice și poate fi rulat pe un PC, laptop sau smartphone. Pentru dezvoltare, s-a folosit un laptop unde s-a instalat mediile de dezvoltare integrate IntelliJ IDEA pentru partea de back-end și WebStorm pentru front-end. Au fost alese deoarece oferă o gamă largă de funcționalități, inclusiv suport pentru diverse limbaje de programare, instrumente de depanare avansate și integrare cu sisteme de control al versiunilor (Git).

2.1. Modulul Controller

Acest modul are rolul de a realiza comunicarea cu front-end-ul și de a returna un plan de vacanță personalizat bazat pe parametrii de intrare furnizați. Acest controller dispune de o funcționalitate adițională care constă în obținerea unor orașe din baza de date în funcție de un filtru.

Pe lângă aceste atribuții, sunt implementate și alte operații care apelează serviciile secundare cu rolul de a verifica că modul în care se comportă etapele intermediare ale personalizării au comportamentul dorit.

2.2. Modulul Gateways

Rolul modulului Gateway este în gestionarea comunicării cu diverse servicii externe, cum ar fi cele de la Google, WebScraping și OpenAI. Acestea necesită folosirea unei chei private pentru a reuși obținerea informațiilor, de aceea s-a întocmit un fișier de configurare *application.yml*, care înglobează toate configurările necesare fiecăruia.

Pentru serviciile de la WebScraping și Google s-a putut trimite o cerere HTTP directă către acestea, în schimb, pentru integrarea cu modelele de la OpenAI, a fost nevoie de construirea unei clase de configurare personalizate conform Listing 2.1.

```
1   class GatewayConfiguration(
2       private val apiKey: String,
3       private val numberOfConnectionsPool: Int,
4       private val numberOfConnectionsPoolPerRoute: Int,
5       private val socketTimeout: Int,
6       private val connectionTimeout: Int,
7       private val connectionRequestTimeout: Int,
8   ) {
9       fun restTemplate(): RestTemplate {
10          val connectionManager =
             ↪  PoolingHttpClientConnectionManagerBuilder
11              .create()
12              .setMaxConnTotal(numberOfConnectionsPool)
13              .setMaxConnPerRoute(numberOfConnectionsPoolPerRoute)
14              .setDefaultConnectionConfig(
15                  ConnectionConfig.custom()
16                  .setConnectTimeout(Timeout.ofSeconds(
17                  connectionTimeout.toLong()))
18                  .setSocketTimeout(Timeout.ofSeconds(
19                  socketTimeout.toLong()))
20                  .build()
21              )
22              .build()
23
```

```
24          val requestConfig = RequestConfig.custom()
25              .setConnectionRequestTimeout(Timeout.ofSeconds(
26              connectionRequestTimeout.toLong()))
27              .build()
28
29          val httpClient = HttpClientBuilder.create()
30              .setConnectionManager(connectionManager)
31              .setDefaultRequestConfig(requestConfig)
32              .build()
33
34          val clientHttpRequestFactory =
             ↪ HttpComponentsClientHttpRequestFactory(httpClient)
35
36          val restTemplate = RestTemplate(clientHttpRequestFactory)
37          restTemplate.interceptors.add(ClientHttpRequestInterceptor {
             ↪ request: HttpRequest, body: ByteArray?, execution:
             ↪ ClientHttpRequestExecution ->
38              request.headers.add("Authorization", "Bearer $apiKey")
39              execution.execute(request, body!!)
40          })
41
42          return restTemplate
43      }
44  }
```

Listing 2.1. clasa *Metoda de configurare pentru OpenAI*

2.3. Modulul Servicii

Gateway-urile descrise anterior sunt utilizate pentru a furniza servicii externe, iar acestea sunt folosite în serviciile create în cadrul acestui proiect. Au fost create servicii dedicate unui anumit gateway doar pentru acele cazuri care necesitau o logică mai avansată:

- *Serviciul Custom Search* - are rolul de a genera un prompt de căutare pe Google în funcție de niște parametri introduși de utilizator în interfață. Acest prompt va fi folosit pentru solicitarea răspunsurilor pe motorul de căutare cu ajutorul gateway-ului specializat pe comunicarea cu serviciul Google Custom Search;

- *Serviciul ChatGPT* - acest serviciu implementează funcționalitățile oferite de modelul ChatGPT de la OpenAI și are sarcina de a construi prompturi specializate fiecărui utilizator pentru a obține detaliile călătoriei sale. Un alt rol este acela de a extrage orașele dintr-un text;

- *Serviciul Image Generator* - responsabilitatea acestuia este de a genera o imagine cu ajutorul modelului DALE-E-3 de la OpenAI pe baza denumirii turului;

- *Serviciul Filter Cities* - Serviciul FilterCities interacționează cu baza de date a orașelor (fișier text) și efectuează o căutare în funcție de ceea ce introduce utilizatorul în câmpul pentru destinație. Orașele sunt filtrate în funcție de textul dat, diacriticele, literele mari, cât și accentele din diferite limbi străine, sunt luate în considerare și interpretate ca litere din alfabetul latin. În fișierul text sunt 111866 de înregistrări, iar în urma unei filtrări pot rezulta un număr foarte mare de linii. De aceea se va returna utilizatorului doar primele 10 rezultate ale căutării pentru a nu încetini viteza de lucru a browserului;

- *Serviciul Vacation Planner* - în Figura 2.2 este descris algoritmul de personalizare care este implementat în cadrul acestui serviciu. Pe baza unui model de intrare care înglobează informații introduse din interfață, se va obține un model de ieșire compus din răspunsurile serviciilor anterioare.

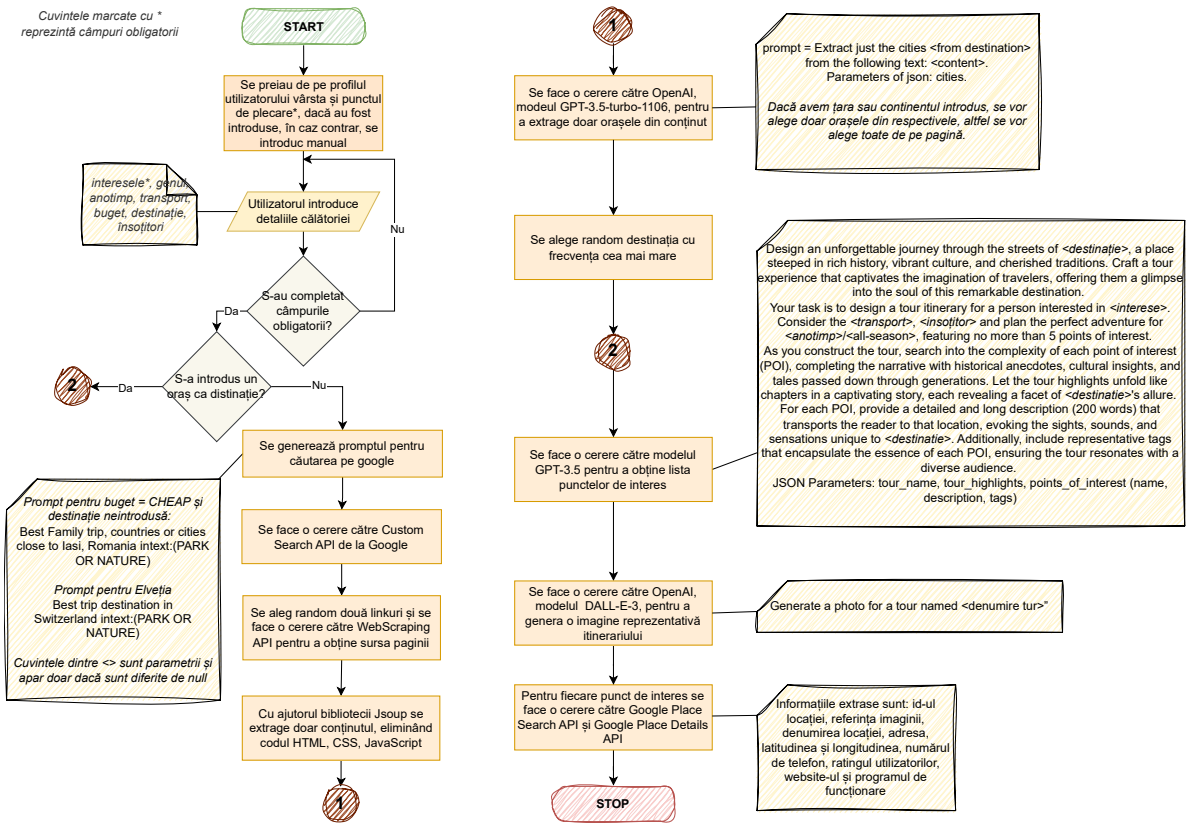

Figura 2.2. Diagrama logică pentru procesul de personalizare

2.4. Modele

Modelele utilizate sunt obiecte de transfer de date (*DTO - **D**ata **T**ransfer **O**bject*) care au rolul de a încapsula datele și de a le trimite de la controller la serviciu. Tot în acest modul sunt construite și câteva clase enum pentru a stoca constante necesare implementării.

2.5. Managementul erorilor

Prin utilizarea adnotării *@ControllerAdvice*, specifică framework-ului Spring, s-a creat o clasă specializată pe gestionarea excepțiilor și a erorilor la nivel de aplicație web, a cărei implementare este vizibila în Anexa 4. Rolul principal al unui astfel de fișier de management este să intercepteze și să proceseze excepțiile aruncate de controlerul aplicației, oferind astfel o manieră elegantă de a răspunde la erori. Au fost definite metode care gestionează diferite tipuri de excepții, cât și mesaje de eroare personalizate, fiind împărțite pe gradul lor de severitate: eroare sau warning.

2.6. Componenta UI

Componenta de UI din cadrul proiectului se folosește de framework-ul Angular și biblioteca Angular Material pentru dezvoltarea unei interfețe de utilizator receptive și interactive. Componentele reprezintă elementele de bază ale interfeței, unde fiecare este responsabilă de o anumită funcționalitate și sunt definite printr-o clasă TypeScript, un template HTML și un fișier de stiluri

CSS (***C**ascading **S**tyle **S**heets*). Biblioteca menționată anterior oferă componente predefinite care au fost modificate conform comportamentului și aspectului dorit.

Serviciile au rolul de a gestiona logica de afaceri și de a furniza date către componente, interacționând cu API-urile back-end-ului sau implementând o logică necesară la nivelul frot-end-ului. Modelele sunt utilizate pentru a defini și descrie forma obiectelor de date manipulate de componente și servicii. Interacțiunea cu utilizatorul este gestionată prin binding-ul[11] datelor în ambele sensuri și a evenimentelor în Angular.

Această arhitectura în care componentele sunt separate, conectate între ele, dar nu dependente unele de altele, permite organizarea codului în module distincte, fiecare având o responsabilitate unică. Totodată permite și încărcarea lazy-loading[12], îmbunătățind performanța aplicației. Toate acestea ducând la o aplicație ușor de întreținut.

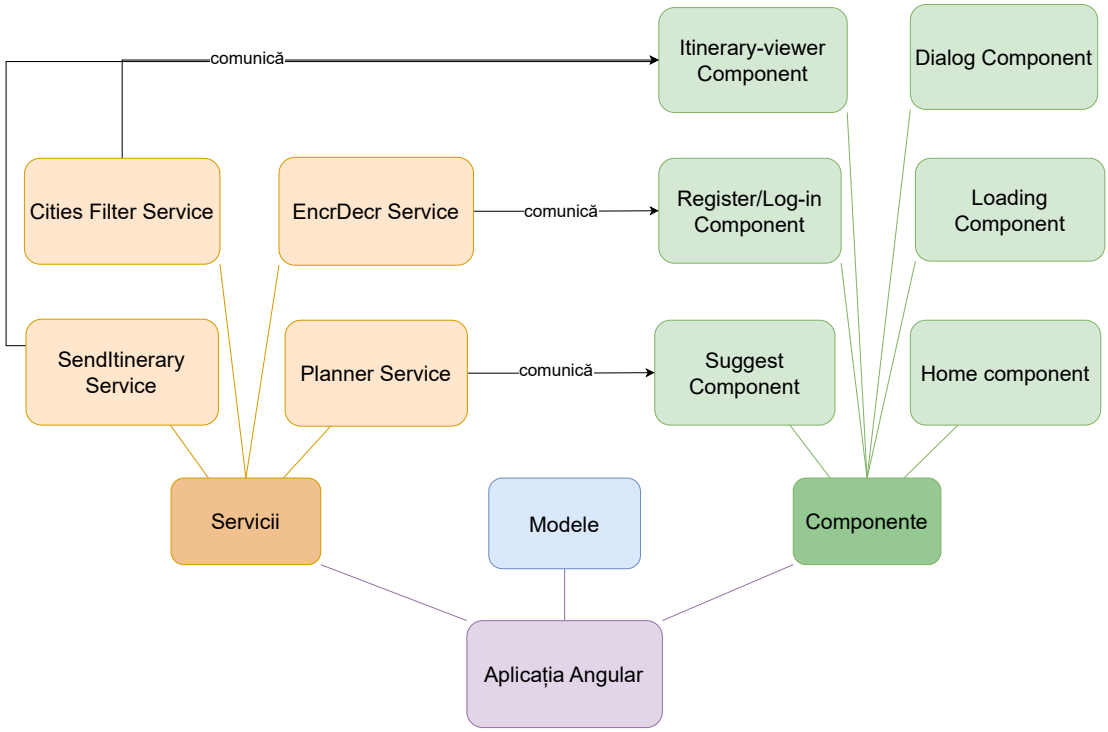

Figura 2.3. Structura componentei de UI

Rolul fiecărei structuri din Figura 2.3:

- *Componenta Itinerary_viewer* - prezintă vizual utilizatorului rezultatul final al procesului de planificare;
- *Componenta Dialog* - componentă reutilizabilă care constituie un model pentru afișarea mesajelor într-o căsuță de dialog;
- *Componenta Loading* - procesul de planificare durează în jur de 1-2 minute și prin această componentă se afișează fereastra de așteptare;

[11] coordonarea părții vizuale a componentei cu logica componentei, adică variabilele din componentă pot fi folosite direct în template

[12] încărcare lenește - tehnică în care se încarcă doar modulele sau componentele care sunt necesare pentru reprezentarea vizuală actuală, decât să se încarce întreaga aplicație în avans

- *Componenta Register/Log-in* - conține ambele funcționalități prin apăsarea unui buton și înglobează design-ul acestora;
- *Componenta Home* - pagina principală a aplicației și are rolul de a afișa informațiile introduse la crearea contului, lista de vacanțe create și un ghid de utilizare al aplicației;
- *Componenta Suggest* - are rolul de a permite introducerea informațiilor despre vacanță;
- *Serviciul Cities Filer* - filtrează orașele din baza de date conform unui text introdus de utilizatorul în interfață, în câmpul pentru destinație;
- *Serviciul EncrDecr Service* - criptează și decriptează parolele utilizatorilor. S-au folosit pentru a nu introduce în memoria cache parolele în clar;
- *Serviciul SendItinerary* - comunică cu componenta de vizualizare a itinerariului, iar acest serviciu transmite respectivele informații;
- *Serviciul Planner* - are rolul de a comunica cu controllerul dedicat procesului de specializare.

2.6.1. Memoria cache

Baza de date a aplicației constă în folosirea Dexie în Angular, o soluție de stocare locală bazată pe IndexedDB care permite aplicațiilor web să stocheze date la nivelul browserului. Prin această modalitate s-a implementat o memorie cache pentru stocarea utilizatorilor și a itinerariilor, organizată conform Figura 2.4. Se observă prezența unei tabele în plus, *Poi*, care este ilustrată doar pentru a se observa ceea ce este stocat în atributul *pois* din tabela *Itineraries*.

Figura 2.4. Structura memoriei cache

Această soluție oferă câteva avantaje importante în ceea ce privește securitatea și controlul datelor utilizatorilor. Prin stocarea datelor local, fiecare individ își menține informațiile pe dispozitivul său propriu, fără a fi necesară trimiterea acestora către un server extern. Acest lucru elimină riscul ca datele sensibile să fie expuse în timpul transmiterii către și de la server, reducând vulnerabilitatea la atacurile cibernetice și protejând confidențialitatea datelor personale. Toate acestea

oferă utilizatorilor control complet asupra informațiilor lor, fără a-și face griji cu privire la posibilitatea accesului neautorizat la date. Chiar dacă pe un browser se pot crea mai multe conturi, se presupune că persoanele care folosesc același dispozitiv fac parte din aceeași familie.

Capitolul 3. Implementarea aplicației

3.1. Modulul Controller

În Listing 3.1 este listată implementarea controlerului aplicației, în cadrul căruia sunt evidențiate cele două metode principale care comunică cu aplicația Angular. Prima metodă POST este folosită pentru a obține vacanța personalizată prin apelarea serviciului de planificare, iar celelalte două metode GET[13], sunt utilizate pentru a obține primele 10 orașe din baza de date a acestora în urma filtrării și de a verifica dacă un anumit oraș introdus de utilizator în interfață există. Celelalte metode existente în acest fișier au fost folosite doar pentru a observa că pașii intermediari ai planificării au comportamentul dorit. Codul complet este disponibil în Anexa 1.

```
1   @RestController
2   @RequestMapping("/api/v1/relation-trip")
3   @Tag(
4       name = "Vacation Planner Controller", description = "This
        ↪  provides all operations for managing vacation planner"
5   )
6   @CrossOrigin(origins = ["http://localhost:4200"])
7   class VacationPlannerController(
8       private val vacationPlannerService: VacationPlannerService,
9       private val chatGptService: ChatGptService,
10      private val placesApiGateway: PlacesApiGateway,
11      private val searchGateway: SearchApiGateway,
12      private val webScrapingGateway: WebScrapingApiGateway,
13      private val citiesService: FilterCitiesService,
14  ) {
15          @Operation(
16          summary = "Vacation Planner",
17          description = "This operation returns a personalized vacation
            ↪  plan based on the provided input parameters. "
18      )
19      @PostMapping("/planner")
20      fun vacationPlanner(@RequestBody vacationPlannerInput:
        ↪  VacationPlannerInput): VacationPlannerOutput {
21          return vacationPlannerService
22              .vacationPlanner(vacationPlannerInput)
23      }
24
25      @Operation(
26          summary = "City Exists",
27          description = "Checks if a city exists in the database. The
            ↪  operation returns true if the city exists, otherwise
            ↪  false."
28      )
29      @GetMapping("/city-exists/{city}")
30      fun cityExists(@PathVariable city: String): Boolean {
31          return citiesService.existsCity(city)
32      }
33
```

[13]cere date de la un server, obține informații din resursele web fără a modifica starea acestora

```
34      @Operation(
35          summary = "Filter Cities",
36          description = "Filters cities based on the provided text. The
             ↪ operation returns a list of cities that match the text."
37      )
38      @GetMapping("/filter-cities/{text}")
39      fun filterCities(@PathVariable text: String): List<String> {
40          return citiesService.filterCities(text)
41      }
42  }
```

Listing 3.1. Metodele principale din controllerul Vacation Planner

3.2. Modulul Gateways

Cum a fost menționat în secțiunea 2.2, acest modul este specializat în gestionarea comunicării cu diverse servicii externe, dar și în configurarea acestora. Prin urmare, un gateway construit pentru un serviciu extern are anumiți parametri de care are nevoie pentru a se realiza conexiunea. Valorile acestora sunt injectați în variabilele Kotlin cu ajutorul adnorătii *@Value* dintr-un fișier de configurare extern denumit *application.yml*, vizibil în Listing 3.2.

În acest fișier, pentru serviciul ChatGPT se poate observa prezența unor parametri de timeout și gestionare a conexiunilor. Setările de timeout, cum ar fi *socket-timeout*, *connection-timeout* și *connection-request-timeout*, determină cât timp clientul HTTP va aștepta pentru primirea răspunsului în urma cererii, 120 de minute în acest caz, prevenind întârzierile care pot apărea. În plus, configurarea pool-ului de conexiuni prin parametrii *number-of-connections-pool* și *number-of-connections-pool-per-route* permite controlul numărului maxim de conexiuni simultane, atât la nivel global, cât și pe fiecare rută individuală, îmbunătățind astfel utilizarea resurselor și evitând suprasolicitarea serverului.

Rolul celorlalți parametri folosiți în acest fișier a fost explicat în secțiunea 1.3.

```
1   integration:
2     gateway:
3       chatgpt:
4         timeouts:
5           rest:
6             socket-timeout: 120
7             connection-timeout: 120
8             connection-request-timeout: 120
9         number-of-connections-pool: 20
10        number-of-connections-pool-per-route: 20
11        model: gpt-3.5-turbo-1106
12        base-uri: https://api.openai.com/v1/chat/completions
13        api-key: <API_KEY>
14      dalle:
15        model: dall-e-3
16        base-uri: https://api.openai.com/v1/images/generations
17        api-key: <API_KEY>
18      maps:
19        places:
20          endpoint-text-search:
             ↪ https://maps.googleapis.com/maps/api/place/textsearch/json
21          endpoint-details:
             ↪ https://maps.googleapis.com/maps/api/place/details/json
```

```
22          api-key: <API_KEY>
23      google-search:
24        endpoint: https://www.googleapis.com/customsearch/v1?
25        api-key: <API_KEY>
26        cx: a022e7c8cfbdd4596
27      web-scraping:
28        endpoint: https://api.webscrapingapi.com/v1?
29        api-key: <API_KEY>
30        render-js: 0
31        proxy-type: datacenter
```

Listing 3.2. Fișierul de configurare *application.yml*

3.3. Modulul Servicii

În cele ce urmează va fi prezentat modul de implementare al procesului de planificare care integrează mai multe servicii externe, explicate fiecare în ordinea utilizării. Serviciul cel mai important este disponibil la Anexa 2, metodele de mai jos fiind componente care ajută la implementarea acestuia.

3.3.1. Serviciul extern Custom Search de la Google

Prima etapă este cea de construire a promptului pentru căutarea pe Google pentru a găsi diferite destinații. Bazându-se pe informații precum bugetul de călătorie, destinația și interesele utilizatorului, aplicația decide ce termeni de căutare să includă în prompt. Dacă utilizatorul nu specifică o destinație, se generează un prompt pentru a căuta cele mai bune opțiuni de călătorie din întreaga lume. În cazul în care este specificată o destinație, promptul este ajustat în funcție de aceasta. Dacă se alege o vacanță la un preț scăzut, se caută în jurul punctului de plecare. Pentru a obține rezultate relevante și personalizate, se folosesc operatori de căutare Google, cum ar fi *intext*, care sunt utilizați pentru a specifica anumite condiții și restricții în căutare. Exemple de prompturi: *„Best trip destination in Switzerland intext:(PARK OR NATURE)”*, *„Best Family trip, countries or cities close to Iasi, Romania intext:(MUSEUM OR NATURE)”*.

```
1   override fun generatePromptCustomSearch(vacationPlannerInput:
     ↪  VacationPlannerInput): String {
2           val budget = vacationPlannerInput.budget?.let {
3               when (it) {
4                   Budget.CHEAP, Budget.AFFORDABLE -> "trip, countries
                     ↪  or cities close to
                     ↪  ${vacationPlannerInput.startingPoint} "
5                   else -> "$it trip destination in the world "
6               }
7           } ?: "destination in the whole world "
8           // if destination is empty, we will search for the best trip
             ↪  in the world
9           if (vacationPlannerInput.destination == "") {
10              return "Best " + (vacationPlannerInput.attendant?.let {
                 ↪  "$it " } ?: "") + budget +
11                      (vacationPlannerInput.season?.let { ", $it " } ?:
                         ↪  "") +
12                      "intext:(${vacationPlannerInput.interests
13                      .joinToString(" OR ")})"
14          }
```

```
15          // if destination is a continent or a country
16          else if
             ↪  (vacationPlannerInput.destination.matches(Regex("^(Asia|
17          Africa|North America|South America|Antarctica|Europe
18          |Australia)\$"))
19              vacationPlannerInput.destination.matches(
20              Regex("^[a-zA-Z\\s]+\$"))
21          ) {
22              return "Best " + (vacationPlannerInput.attendant?.let {
                 ↪  "$it " } ?: "") +
23                      (vacationPlannerInput.budget?.let { "$it " } ?:
                         ↪  "") +
24                      "trip destination in
                         ↪  ${vacationPlannerInput.destination} " +
25                      (vacationPlannerInput.season?.let { ", $it " } ?:
                         ↪  "") +
26                      "intext:(${vacationPlannerInput.interests
27                      .joinToString(" OR ")})"
28          }
29          // if destination is a city
30          else if (vacationPlannerInput.destination.matches(
31          Regex("^[a-zA-ZÀ-ÿĀ-žǍ-ǜ''.`\\s]+,
32          \\s?[a-zA-ZÀ-ÿĀ-žǍ-ǜ''.`\\s]+\$"))) {
33              return ""
34          }
35
36          throw
             ↪  HttpClientErrorException(HttpStatus.UNPROCESSABLE_ENTITY,
             ↪  "Format of the destination is not valid.")
37      }
```

Listing 3.3. Metoda pentru construirea promptului necesar căutării pe Google

După generarea promptului, urmează etapa de efectuare a căutării propriu-zise. Aceasta constă în trimiterea unei cereri către API-ul Custom Search cu promptul creat anterior pentru a obține rezultatele căutării sub format JSON. Răspunsul va conține 10 rezultate pe fiecare pagină (afișare paginată a răspunsului), iar pentru simplitate se vor folosi doar cele de pe prima pagină.

În cazul în care, destinația introdusă este deja un oraș, căutarea pe Google este omisă.

3.3.2. Serviciul extern WebScraping

Din lista obținută anterior în urma căutării pe Google se selectează doar 2 linkuri și se trimite o cerere către WebScraping API pentru a extrage conținutul paginii web corespunzătoare. Odată primit, este utilizată biblioteca Jsoup pentru a elimina codul HTML, CSS și JavaScript al paginii web. În cazul apariției unei erori, cum ar fi o eroare care indică că site-ul nu funcționează, se alege alt link din lista disponibilă. Această acțiune este repetată până când se obțin informațiile de la exact 2 linkuri sau până când lista este parcursă integral.

3.3.3. Serviciile externe ChatGPT și Dall-e de la OpenAI

Modelul GPT-3.5-turbo-1106, de la OpenAI, a fost antrenat până în septembrie 2021 și, conform menționilor din [13], limitările acestui LLM sunt evidențiate în planificare, din cauza dependenței de informații care nu sunt în timp real și a cunoștințelor incomplete. De aceea este folosită inițial o căutare pe Google, unde informațiile sunt de actualitate. Itinerariile generate direct

de LLM ar putea fi lipsite de detalii și ar putea include puncte de interes inexistente, drept urmare, utilizatorul este încurajat să fie cât mai precis în procesul de completare a câmpurilor.

Aplicația este constrânsă de către model să efectueze doar trei cereri pe minut, conform politicilor de utilizare stabilite pentru a asigura o distribuție echitabilă a resurselor și pentru a menține performanța și disponibilitatea serviciului. Din aceste motive, cererile au fost împărțite conform specificațiilor următoare.

Determinarea destinației călătoriei constă în efectuarea a 2 cereri pentru conținuturile obținute anterior și solicitarea modelului să extragă doar orașele prezente în text sub formă de obiect JSON, utilizând parametrul *cities*. Fiecare locație identificată este înregistrată și contorizată. Apoi, alegem destinația călătoriei pe baza acestor informații: fie selectăm orașul cu cea mai mare frecvență de apariție, fie alegem aleatoriu unul dintre orașele cu frecvență egală maximă.

Ultima cerere este pentru a obține lista cu punctele de interes pentru destinația aleasă. Se obține de la modelul GPT-3.5 un răspuns sub forma unui obiect JSON, care conține detalii despre itinerariu, precum numele și descrierea acestuia, punctele de interes și aspectele evidențiate.

Pentru a obține astfel de detalii descriptive, a fost necesară construirea unui prompt. Prin repetate încercări manuale, promptul vizibil în Listing 3.4 a fost îmbunătățit pentru a dobândi o descriere cât mai consistentă în istorie, cultură și tradiții. Codul complet al acestui serviciu poate fi găsit în Anexa 3.

```
1   val prompt =
2       "Design an unforgettable journey through the streets of $city, a
         ↪ place steeped in rich " +
3       "history, vibrant culture, and cherished traditions. Craft a tour
         ↪ experience that captivates the " +
4       "imagination of travelers, offering a glimpse into the soul of
         ↪ this remarkable destination.\n\n" +
5       "Your task is to design a tour itinerary for a person with
         ↪ interests in ${
6           interests.joinToString(
7               " and "
8           )
9       }${if (otherInterests != null) ", $otherInterests" else ""}.
         ↪ Consider the ${
10          transport?.let { "$it, " } ?: ""
11      }${
12          attendant?.let { "$it, " } ?: ""
13      } and plan the perfect adventure for ${
14          season?.let { "$it " } ?: "all-season "
15      }, featuring no more than 5 points of interest.\n\n" +
16      "As you construct the tour, search into the complexity of each
         ↪ point of interest (POI), completing the " +
17      "narrative with historical anecdotes, cultural insights, and
         ↪ tales passed down through generations. " +
18      "Let the tour highlights unfold like chapters in a captivating
         ↪ story, each revealing a facet of $city's allure.\n\n" +
19      "For each POI, provide a detailed and long description (200
         ↪ words) that transports the reader to that location, " +
20      "evoking the sights, sounds, and sensations unique to $city.
         ↪ Additionally, include representative " +
21      "tags that encapsulate the essence of each POI, ensuring that
22      the tour resonates with a diverse audience.\n\n" +
23      "JSON Parameters:\n" +
24      "- tour_name\n" +
```

```
25        "- tour_highlights\n" +
26        "- points_of_interest (name, description, tags)"
```

Listing 3.4. Promptul pentru obținerea detaliilor itinerariului

Am integrat modelul DALL-E-3 de la OpenAI pentru a genera imagini sugestive pentru itinerariile turistice. API-ul returnează un link de imagine, care este valid pentru o perioadă limitată de timp, aproximativ o oră. Pentru a asigura o experiență stabilă și continuă pentru utilizatori, am implementat o metodă de redimensionare și conversie a imaginilor obținute în format base64, astfel încât imaginile afișate direct în aplicație să persiste și să nu depindă de linkurile temporare furnizate de API.

```
1   @Service
2   class ImageGeneratorService(
3       private val dalleGateway: IOpenAiGateway
4   ) : IImageGeneratorService {
5       override fun generateImage(itineraryName: String): String {
6           val prompt = "Generate a photo for a tour named
            ↪ $itineraryName"
7           val imageUrl = dalleGateway.generateImage(prompt)
8
9           // Reading the image from the URL
10          val inputImage: BufferedImage = ImageIO.read(URL(imageUrl))
11
12          // Resizing the image
13          val outputImage: Image = inputImage.getScaledInstance(400,
            ↪ 300, Image.SCALE_SMOOTH)
14          val resizedImage = BufferedImage(400, 300,
            ↪ BufferedImage.TYPE_INT_RGB)
15          val graphics2D = resizedImage.createGraphics()
16          graphics2D.drawImage(outputImage, 0, 0, null)
17          graphics2D.dispose()
18
19          // Converting the image to base64
20          val outputStream = ByteArrayOutputStream()
21          ImageIO.write(resizedImage, "jpg", outputStream)
22          val bytes = outputStream.toByteArray()
23          val base64Image = Base64.getEncoder().encodeToString(bytes)
24
25          return "$base64Image"
26      }
27  }
```

Listing 3.5. Serviciul ImageGenerator

3.3.4. Serviciul extern Places de la Google

Pentru fiecare punct de interes, se inițiază o solicitare către API-ul Google Places pentru a obține detaliile locației respective. API-ul Place Search permite căutarea informațiilor despre locuri folosind denumirea unui punct de interes. Inputul constă într-un șir de caractere compus din denumirea punctului de interes și destinația acestuia, iar rezultatul returnat conține informații sumare. Printre acestea se enumără id-ul unic al locului, referința imaginii, denumirea locației, adresa, latitudinea și longitudinea, necesare pentru reprezentarea pe hartă la nivel de front-end.

Odată obținut id-ul unic, se trimite o cerere către API-ul Place Details. Această cerere furnizează informații mult mai detaliate despre punctul de interes, inclusiv numărul de telefon, ratingul utilizatorilor, website-ul și programul de funcționare, contribuind astfel la completarea detaliilor necesare prezentării în fața utilizatorului.

3.3.5. Serviciul de filtrare a orașelor

Acest serviciu are rolul de a realiza operații asupra bazei de date a orașelor cu scopul de a le filtra. A fost necesară crearea unui obiect *companion*, pentru ca variabila *cities* să fie asociată cu clasa. În comparație cu celelalte limbaje de programare, abordarea este similară cu variabilele statice. Aceasta este importantă deoarece atunci când serverul pornește, se face încărcarea în variabilă a tuturor înregistrărilor din fișierul text, prin metoda *uploadCities*, operația fiind realizată o singură dată.

Metoda *existsCity* verifică dacă o anumită destinație există în baza de date, iar *filterCities* filtrează înregistrările pe baza unui input și returnează primele 10 potriviri. Ultima metodă are rolul de a elimina diacriticele din fiecare denumire pentru a se putea verifica egalitatea dintre două șiruri de caractere scrise cu litere latine.

```
1   @Service
2   class FilterCitiesService : IFilterCitiesService {
3       companion object {
4           var cities = mutableListOf<String>()
5       }
6
7       override fun filterCities(text: String): List<String> {
8           return cities.filter {
9               it.removeDiacritics().contains(text.removeDiacritics(),
10              ignoreCase = true) }.shuffled().take(10)
11      }
12
13      override fun existsCity(city: String): Boolean {
14          val normalizedCity = city.trim().removeDiacritics()
15              .lowercase(Locale.getDefault())
16          return cities.any { it.trim().removeDiacritics()
17              .lowercase(Locale.getDefault()) == normalizedCity }
18      }
19
20      override fun uploadCities() {
21          val inputFile: InputStream? = javaClass.classLoader
22              .getResourceAsStream("city_country.txt")
23          val inputString = inputFile?.bufferedReader().use {
24              it?.readText() ?: "" }
25          cities = inputString.split("\n").toMutableList()
26      }
27
28      private fun String.removeDiacritics(): String {
29          return Normalizer.normalize(this, Normalizer.Form.NFD)
30              .replace("\\p{InCombiningDiacriticalMarks}+".toRegex(),
31              "")
32      }
33  }
```

Listing 3.6. Serviciul de filtrare

3.4. Modele

Modelele au fost create pentru a ușura comunicarea obiectelor de date între componente. În acest proiect au fost necesare 3 modele: *Poi*, *VacationPlannerInput*, *VacationPlannerOutput* și diferite clase enum[14] pentru a avea niște valori predefinite pentru transport, anotimp, interese, gen, însoțitori, buget.

3.5. Interfața cu utilizatorul

Utilizatorul accesează pagina dedicată introducerii detaliilor călătoriei necesare pentru realizarea itinerariului, unde poate să completeze o serie de câmpuri adiționale, precum vârsta, genul, anotimpul, transportul, bugetul și însoțitorii pentru o planificare cât mai detaliată. Nu este obligatorie completarea acestor câmpuri, iar lipsa lor nu va afecta funcționarea aplicației.

Punctul de plecare este o informație obligatorie, care poate fi introdusă manual și este utilizată pentru a evita generarea unui itinerariu care să includă aceeași destinație. În cazul introducerii aceleiași destinații ca și punct de plecare, informația despre locația de plecare nu va fi luată în considerare.

Destinația călătoriei este un aspect suplimentar în procesul de planificare pentru a oferi indivizilor flexibilitate și posibilitate de explorare a unor opțiuni variate, chiar și atunci când nu sunt decișii încă de un loc specific. Pentru introducerea acesteia, utilizatorul trebuie să respecte un format particular: „denumire continent”, „denumire țară”, „denumire oraș, denumire țară”. Este important de menționat că este permisă doar introducerea destinațiilor care sunt stocate în baza noastră de date.

3.6. Docker

```
1  FROM openjdk:18
2  WORKDIR /app
3  COPY ./target/Proiect-0.0.1-SNAPSHOT.jar /app
4  EXPOSE 8080
5  CMD ["java", "-jar", "Proiect-0.0.1-SNAPSHOT.jar"]
```

Listing 3.7. Dockerfile pentru back-end

```
1  FROM node:alpine
2  WORKDIR /app
3  COPY . /app
4  RUN npm install -g @angular/cli
5  RUN npm install
6  CMD ["ng", "serve", "--host", "0.0.0.0"]
```

Listing 3.8. Dockerfile pentru front-end

```
1  version: '3'
2  services:
3    backend:
4      build:
5        context: ./Project
6        dockerfile: Dockerfile
7      ports:
```

[14]tip de date care permite ca o variabilă să fie un set de constante predefinite

```
8       - "8080:8080"
9     frontend:
10      build:
11        context: ./Angular-Project
12        dockerfile: Dockerfile
13      ports:
14        - "4200:4200"
```

Listing 3.9. DockerCompose.yml

3.7. Dificultăți întâmpinate și modalități de rezolvare

O problemă întâmpinată a fost găsirea unei fotografii care să fie sugestivă pentru itinerariul creat. În cele din urmă am decis să utilizez modelului DALL-E pentru generarea imaginilor, însă după prima încercare am avut experiența neplăcută de a afla că acestea sunt temporare și nu persistă în timp. Am încercat descărcarea lor și stocarea acestora local, dar nu am ajuns la un rezultat final deoarece în interfață ele nu erau vizibile. În final, am folosit o metodă de creare a unui link care să persiste în timp prin codificarea imaginii cu base64. Din cauză că imaginea avea o dimensiune prea mare, în urma codificării, doar o pătrime din imagine se încărca, de aceea a fost redimensionată.

Crearea unei interfețe interactive și vizual plăcută, de impact, a fost o altă provocare pentru îmbunătățirea experienței utilizatorului cu aplicația. Prin multiple schițe și încercări, s-au testat diferite versiuni de design, în cele din urmă optând pentru o afișare sub formă de bilet de avion, care a îmbunătățit vizual interfața, aducându-i un plus de originalitate.

În timpul integrării cu ChatGPT, au apărut probleme legate de răspunsurile în format JSON, ceea ce a dus la dificultăți în recunoașterea tuturor parametrilor necesari de către aplicație, deoarece răspunsul lua diferite forme de la o cerere la alta. Pentru a rezolva această problemă, am făcut multiple teste și ajustări asupra promptului utilizat. După numeroase încercări, s-a construit unul specific care generează de fiecare dată un răspuns structurat după dorința proprie.

Capitolul 4. Testarea aplicației și rezultate experimentale

Pentru a demonstra soluția, am dezvoltat un API, care realizează cele explicate în capitolul 3 și este apelat în front-end. Propun testarea creării unei vacanțe pe timpul verii cu prietenii în orice destinație, cu plecare din București pentru o persoană interesată de artă, natură și drumeții. Un astfel de input pentru API este ilustrat în Listing 4.1. Acest JSON este creat la nivel de front-end prin introducerea acestor informații în pagina pentru sugestie și este trimis către back-end în forma menționată anterior. Se poate observa că destinația nu a fost completată, ceea ce va determina generarea unei locații din orice colț al lumii.

```
{
    "destination": "",
    "startingPoint": "Bucharest, Romania",
    "age": 25,
    "gender": null,
    "attendant": "FRIENDS",
    "season": "SUMMER",
    "transport": null,
    "budget": null,
    "interests": ["ART", "NATURE", "HIKING"],
    "otherInterests": null
}
```

Listing 4.1. Obiectul de intrare pentru procesul de planificare

Un răspuns este prezentat în Listing 4.2, unde se pot observa toate caracteristicile care descriu vacanța integrate într-un singur răspuns. Acesta este un exemplu, deoarece, la fiecare cerere, procesul poate să ofere destinații diferite, ceea ce implică, alte caracteristici descriptive, așa cum se poate observa în Figura 4.1, unde pentru același input s-a obținut o altă locație. Este prezentată afișarea vacanței la nivel de utilizator. În partea stângă este afișat itinerariul cu fiecare punct de interes și în partea dreaptă este prezentă mapa cu fiecare punct însemnat pentru a putea fi vizibil și pentru a putea naviga.

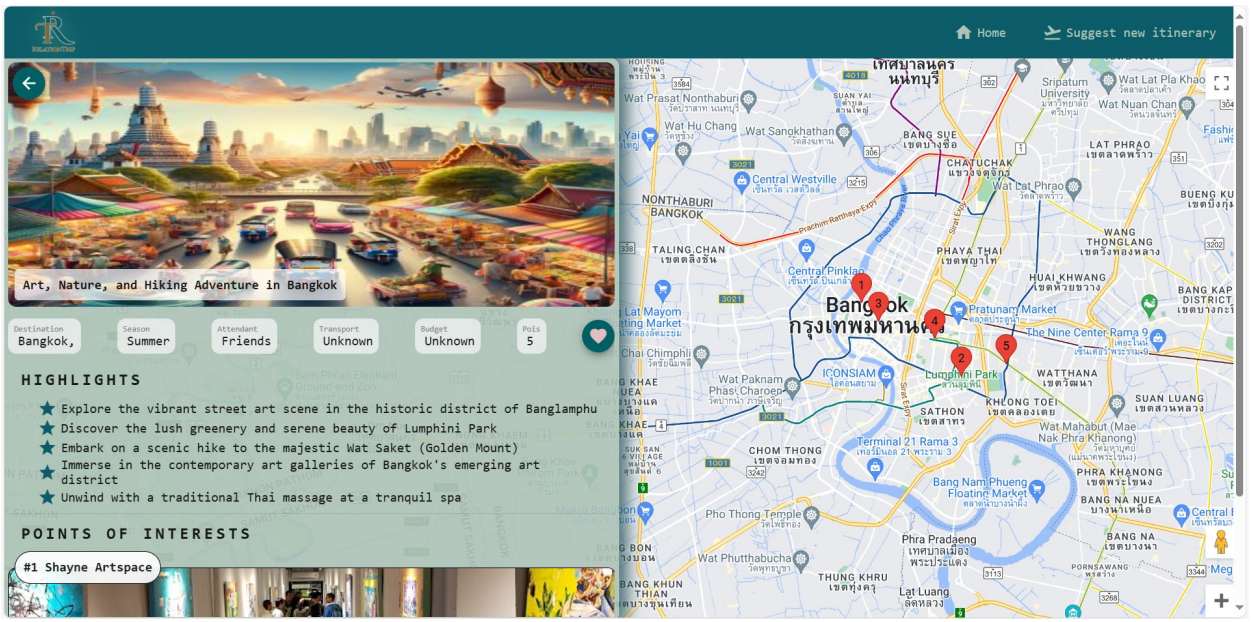

Figura 4.1. Imagine care ilustrează rezultatul final

```
{
    "photo": "...",
    "name": "Art, Nature, and Hiking Adventure in Istanbul",
    "destination": "Istanbul, ",
    "startingPoint": "Bucharest, Romania",
    "season": "SUMMER",
    "attendant": "FRIENDS",
    "transport": null,
    "budget": null,
    "poisNumber": 5,
    "highlights": [
        "Explore the vibrant art scene of Istanbul",
        "Discover the natural beauty of the city",
        "Embark on a scenic hiking adventure"
    ],
    "pois": [
        {
            "name": "Istanbul Museum of Modern Art",
            "photo":
             ↪  "https://maps.googleapis.com/maps/api/place/photo?
            maxwidth=600&photo_reference=<cod_referinta>",
            "description": "Istanbul Modern is a contemporary art
             ↪  museum located on the shores of the Bosphorus. The
             ↪  museum showcases a diverse collection of Turkish and
             ↪  international modern art, providing a platform for
             ↪  emerging artists and established talents. As you
             ↪  wander through the galleries, you'll encounter
             ↪  thought-provoking installations, avant-garde
             ↪  sculptures, and captivating paintings that reflect
             ↪  the dynamic spirit of Istanbul's art scene. The
             ↪  museum's waterfront terrace offers breathtaking views
             ↪  of the Bosphorus, creating a serene space to
             ↪  contemplate the intersection of art and nature.
             ↪  Istanbul Modern is a testament to the city's evolving
             ↪  artistic landscape, inviting visitors to engage with
             ↪  creativity in a modern context.",
            "longitude": 28.9828383,
            "latitude": 41.02591959999999,
            "tags": [
                "art",
                "contemporary",
                "museum",
                "Bosphorus"
            ],
            "stars": 4.4,
            "address": "Kılıçali Paşa, Tophane İskele Cd. No:1/1,
             ↪  34433 Beyoğlu/İstanbul, Türkiye",
            "phone": "+90 212 334 73 00",
            "website": "https://www.istanbulmodern.org/tr",
            "schedule": [
                "Monday: Closed",
                "Tuesday: 10:00 AM - 6:00 PM",
```

```
                    "Wednesday: 10:00 AM - 6:00 PM",
                    "Thursday: 10:00 AM - 6:00 PM",
                    "Friday: 10:00 AM - 8:00 PM",
                    "Saturday: 10:00 AM - 6:00 PM",
                    "Sunday: Closed"
                ]
            },
            ...
        ]
    }
```

Listing 4.2. Obiectul de ieșire pentru procesul de planificare

În procesul de planificare pot apărea adesea erori din cauza indisponibilității unui serviciu extern sau pierderea conexiunii la rețea. În astfel de cazuri rezultatul nu poate fi dus până la sfârșit, iar generarea se va opri, utilizatorul fiind informat să încerce din nou prin mesajul de eroare prezentat în Figura 4.2.

O altă cauză a erorii poate fi cauzată de răspunsurile diferite ale serviciilor, iar când se încearcă accesarea unui parametru și nu există, generarea se oprește. Aceste cazuri au fost frecvente în cazul generării de text cu ChatGPT, dar prin îmbunătățirea promptului șansele apariției unei astfel de erori au fost diminuate.

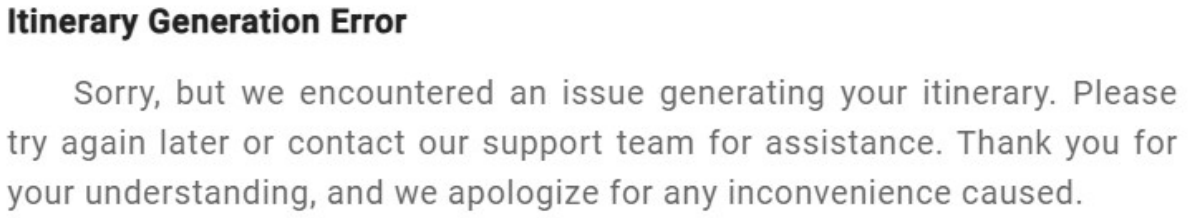

Figura 4.2. Mesaj de eroare pe care îl poate vedea utilizatorul

Pentru a se asigura comportamentul dorit al aplicației, s-au realizat teste intermediare pentru a testa funcționalitatea fiecărei unități care ajută la construirea itinerariului. Așadar, în controller este creată câte o metodă pentru fiecare operație cu serviciile externe pentru a testa integrarea cu aplicația, comportamentul și modul în care livrează răspunsurile.

4.1. Testare Custom Search

Cererea trimisă către API-ul pentru căutarea pe Google arată conform Listing 4.3, iar răspunsul conține o listă de linkuri din care se vor selecta doar 2.

```
Cerere:
    http://localhost:8080/api/v1/relation-trip/custom-search?
    prompt=Best Cheap Romania travel destinations Autumn
    intext:(art OR museum OR history)

Răspuns:
[
    {
        "title": "whats the cheapest country in europe to visit.
         ↪  While also being a ...",
        "link": "https://www.reddit.com/r/Shoestring/comments/
```

```
        1661fue/whats_the_cheapest_country_in_europe_to_visit/",
        "cachedId": ""
    },
    {
        "title": "The Perfect Romania Itinerary: 2 Week Road Trip!",
        "link": "https://www.wheregoesrose.com/road-trip-
        romania-itinerary/",
        "cachedId": ""
    },
    ...
]
```

Listing 4.3. Cererea și răspunsul pentru testarea integrării cu Custom Search API

4.2. Testare WebScraping

Cererea trimisă către API-ul de web scraping este un apel HTTP GET, specificând URL-ul paginii care trebuie analizată. Răspunsul de la server conține conținutul web.

```
Cerere:
    http://localhost:8080/api/v1/relation-trip/web-scraping?
    url=https://www.cntraveler.com/galleries/2015-11-27/the-
    50-most-beautiful-places-in-the-world

Răspuns:
    Skip to main content Newsletter Sign In Search Search
     ↪ Inspiration Destinations Places to Stay News & Advice Hot
     ↪ List Shopping Cruise Women Who Travel Video All products and
     ↪ listings featured on Condé Nast Traveler are independently
     ↪ selected by our editors. If you purchase something through
     ↪ our links, we may earn an affiliate commission. Islands &
     ↪ Beaches The 51 Most Beautiful Places in the World From the
     ↪ Serengeti to the Grand Canyon.
```

Listing 4.4. Cererea și răspunsul pentru testarea integrării cu WebScraping API

4.3. Testare ChatGPT

Pentru a extrage orașele dintr-o pagină web, s-a testat prin încercarea extragerii acestora dintr-un text predefinit în back-end, acest apel fiind necesar doar pentru a putea realiza extragerea. Ca răspuns se va primi o listă cu orașele din textul specificat, alături de țara căror aparțin.

```
Cerere:
    http://localhost:8080/api/v1/relation-trip/chatgpt-cities
Răspuns:
[
    "Maramures, Romania",
    "Sighisoara, Romania",
    "Magura, Romania",
    "Brasov, Romania",
    "Sinaia, Romania",
    "Busteni, Romania",
    "Predeal, Romania",
```

```
    "Poiana Brasov, Romania"
]
```

Listing 4.5. Cererea și răspunsul pentru testarea integrării cu ChatGPT pentru obținerea orașelor dintr-un text

```
Cerere:
    http://localhost:8080/api/v1/relation-trip/chatgpt-poi?
    destination=Cluj Napoca,Romania&gender=FEMALE&attendant=COUPLE&
    season=SPRING&interests=ART,MUSEUM,NATURE

Răspuns:
{
    "tour_name": "Enchanting Cluj Napoca: Art, Museums, and Nature",
    "highlights": [
        "Discover the vibrant art scene of Cluj Napoca",
        "Explore the rich history and culture through museums",
        "Immerse in the natural beauty of the surrounding
         ↪ landscapes"
    ],
    "points_of_interest": [
        {
            "name": "The Painted Monasteries of Moldova",
            "description": "Embark on a journey to the breathtaking
             ↪ Painted Monasteries of Moldova, a UNESCO World
             ↪ Heritage site. Admire the intricate frescoes that
             ↪ adorn the exterior walls, depicting religious scenes
             ↪ and historical events. The vibrant colors and
             ↪ detailed craftsmanship of these centuries-old
             ↪ masterpieces offer a glimpse into the artistic
             ↪ heritage of Romania. As you wander through the
             ↪ peaceful courtyards and ornate chapels, feel the
             ↪ spiritual significance and cultural resonance of
             ↪ these sacred landmarks. The serene atmosphere and
             ↪ stunning surroundings make this a truly
             ↪ unforgettable experience.",
            "tags": [
                "art",
                "history",
                "spirituality"
            ]
        },
        ...
    ]
}
```

Listing 4.6. Cererea și răspunsul pentru testarea integrării cu ChatGPT pentru obținerea descrierea itinerariului

Cum e ilustrat în Listing 4.6, pentru anumiți parametri trimiși în cadrul cererii și conform promptului implementat, s-a obținut descrierea vacanței.

4.4. Testare Google Place

Mai jos este prezentat cum arată un răspuns intermediar în urmă unei cereri către Google Place pentru a obține informațiile despre un punct de interes. Se poate observa că pe lângă denumirea locației, se atașează și orașul din care face parte pentru identificarea precisă a unei locații. Acesta este răspunsul final prin combinarea răspunsurilor de la Places Text Search și Details.

```
Cerere:
    http://localhost:8080/api/v1/relation-trip/places?
    destination=Palatul Culturii, Iasi, Romania

Răspuns:
{
    "placeId": "ChIJ5Sbc0537ykARA6AkbssRP3w",
    "photoReference": "https://maps.googleapis.com/maps/api/place/
    photo?maxwidth=600&photo_reference=<codul_referinta>
    5hk9-Hd&key=AIzaSyCI1fWshdzjt6duD1oftwVk8FECV92VoRs",
    "placeName": "Palace of Culture",
    "placeAddress": "Bulevardul Ștefan cel Mare și Sfânt 1, Iași
     ↪ 700028, Romania",
    "latitude": 47.1574351,
    "longitude": 27.586461,
    "rating": 4.8,
    "phone": "+40 232 275 979",
    "website": "https://palatulculturii.ro/",
    "schedule": [
        "Monday: Closed",
        "Tuesday: Closed",
        "Wednesday: 10:00 AM – 5:00 PM",
        "Thursday: 10:00 AM – 5:00 PM",
        "Friday: 10:00 AM – 5:00 PM",
        "Saturday: 10:00 AM – 5:00 PM",
        "Sunday: 10:00 AM – 5:00 PM"
    ]
}
```

Listing 4.7. Cererea și răspunsul pentru testarea integrării cu Places API

4.5. Securitatea și protecția datelor

Din punct de vedere al securității și confidențialității, se adoptă utilizarea unei baze de date la nivel de browser și implementarea unui mecanism de stocare în cache. Această soluție oferă câteva avantaje importante în ceea ce privește securitatea și controlul datelor utilizatorilor. Prin stocarea datelor local, fiecare individ își menține informațiile pe dispozitivul său propriu, fără a fi necesară trimiterea acestora către un server extern. Acest lucru elimină riscul ca datele sensibile să fie expuse în timpul transmiterii către și de la server, reducând vulnerabilitatea la atacurile cibernetice și protejând confidențialitatea datelor personale. Toate acestea oferă utilizatorilor control complet asupra informațiilor lor, fără a-și face griji cu privire la posibilitatea accesului neautorizat la date.

Concluzii

În această lucrare, am abordat o problemă semnificativă din domeniul turismului: planificarea călătoriilor prin intermediul unei aplicații de personalizare. Scopul proiectului a fost dezvoltarea unui sistem inovativ care să ofere călătorilor un itinerariu turistic adaptat preferințelor lor individuale, simplificând astfel procesul de planificare, prin folosirea diverselor servicii externe oferite de către companii precum Google, OpenAI, WebScraping. Acestă funcționalitate este livrată utilizatorilor printr-o aplicație web care este posesoare a unei interfețe prietenoase și plăcute pentru călători.

Unul dintre avantajele acestei lucrări este securitatea și protecția datelor utilizatorilor. Am implementat o soluție la nivel de browser și am creat un mecanism de stocare în cache pentru a asigura confidențialitatea și integritatea informațiilor personale ale utilizatorilor, protejându-i astfel de amenințările expunerii datelor personale terțelor părți. Un alt avantaj este abordarea flexibilă a planificării, care nu se bazează pe o structură strictă a programului zilnic. Această abordare permite indivizilor să exploreze și să aleagă activitățile în funcție de preferințele lor, fără a fi încadrați într-un program predefinit. De asemenea, aplicația oferă informații ample despre punctele de interes, inclusiv istorie, cultură și tradiții.

Toate acestea se traduc în costuri reduse pentru planificare, deoarece aplicația optimizează timpul petrecut pentru o planificare manuală. Gradul mare de libertate în privința opțiunilor disponibile permite utilizatorilor să aleagă dintre multiple variante care pot fi generate și să personalizeze planificarea în funcție de dorințele fiecăruia.

La finalul dezvoltării și implementării aplicației s-a observat ca majoritatea obiectivelor stabilite inițial au fost îndeplinite, acestea fiind evidențiate prin avantajele prezentate anterior.

Direcții viitoare de dezvoltare

Cu toate cele menționate mai sus, precum orice altă aplicație, există și anumite limite, cum ar fi dependența de API-uri externe care poate duce la apariția de erori sau incoerență între informații. Personalizarea itinerariilor poate fi, de asemenea, limitată, iar opțiunile de adaptabilitate pot fi insuficiente pentru anumite nevoi specifice ale utilizatorilor.

Aplicația este disponibilă exclusiv pe laptopuri sau PC-uri. Cu toate că poate fi accesată și de pe smartphone-uri, experiența utilizatorilor poate fi destul de anevoioasă deoarece dispozitivul și browser-ul de pe care a fost accesată aplicația prima oară, nu împarte aceeași memorie cache cu a telefonului. Această restricție poate să limiteze accesul indivizilor în timpul călătoriilor, în special atunci când se află în mișcare sau când doresc să vizualizeze din nou itinerariul. Dar cum orice limitare poate pune bazele unor idei de dezvoltare viitoare, se dorește crearea unei aplicații mobile, pentru a oferi acces imediat și ușor la funcționalitățile aplicației, indiferent de locație sau de momentul zilei. Această extindere ar îmbunătăți experiența utilizatorului și ar face planificarea călătoriilor și mai convenabilă și accesibilă pentru toți utilizatorii de smartphone-uri.

Lecții învățate pe parcursul dezvoltării lucrării de diplomă

În cadrul procesului de dezvoltare unul dintre cele mai importante aspecte a fost necesitatea creării unei diagrame de bussiness înainte de a începe scrisul propriu-zis al codului. Astfel am învățat importanța unei planificări detaliate și unei stabiliri a obiectivelor clare încă de la începutul proiectului. Definirea clară a obiectivelor mi-a oferit o viziune pentru luarea deciziilor și prioritizarea sarcinilor.

Un alt aspect esențial pe care l-am învățat este să investesc timp în testarea riguroasă a fiecărei componente a aplicației. Testele frecvente și feedback-ul continuu m-au ajutat să identific

şi să corectez erorile într-un stadiu care era la început.

Am avut oportunitatea de a învăţa şi de a aplica diverse tehnologii avansate în dezvoltarea web, inclusiv Angular Material şi alte metode complexe. Am implementat componente precum câmpuri de autocompletare, încărcarea imaginilor de profil.

Bibliografie

[1] G. Blinowski, A. Ojdowska, and A. Przybyłek, "Monolithic vs. microservice architecture: A performance and scalability evaluation," *IEEE Access*, vol. 10, 2022, DOI: 10.1109/ACCESS.2022.3152803.

[2] P. J. S. Pierre J Benckendorff, Zheng Xiang, "Introduction to tourism and information technology," in *Tourism Information Technology*, 3rd ed. CABI Tourism Texts, 2019, ch. 1, pp. 1–2, ISBN: 9781786393432, DOI: 10.1080/13032917.2019.1644106.

[3] H. C. L. Aldy Gunawan and K. Lu, "A fast algorithm for personalized travel planning recommendation," in *Proceedings of the 11th International Conference of the Practice and Theory of Automated Timetabling*, 2016, pp. 163–179, ISBN: 9780992998417.

[4] S. Kambhampati, K. Valmeekam, L. Guan, M. Verma, K. Stechly, S. Bhambri, L. P. Saldyt, and A. B. Murthy, "Position: LLMs can't plan, but can help planning in LLM-modulo frameworks," in *Forty-first International Conference on Machine Learning*, 2024, DOI: 10.48550/arXiv.2402.01817.

[5] "WebScrapingAPI Docs," https://docs.webscrapingapi.com/webscrapingapi/getting-started, Ultima accesare: 12.06.2024.

[6] OpenAI, "Chat Completion," https://platform.openai.com/docs/guides/text-generation/chat-completions-api, Ultima accesare: 12.06.2024.

[7] OpenAI, "Images," https://platform.openai.com/docs/guides/images, Ultima accesare: 12.06.2024.

[8] Google, "Custom Search API," https://developers.google.com/custom-search/v1/overview#api_key, Ultima accesare: 12.06.2024.

[9] Google, "Place Search API," https://developers.google.com/maps/documentation/places/web-service/search-text, Ultima accesare: 12.06.2024.

[10] Google, "Place Details API," https://developers.google.com/maps/documentation/places/web-service/details, Ultima accesare: 12.06.2024.

[11] Google, "Maps JavaScript API," https://developers.google.com/maps/documentation/javascript, Ultima accesare: 13.06.2024.

[12] D. Gada, "countries-states-cities-database," https://github.com/dr5hn/countries-states-cities-database/blob/master/countries%2Bcities.json, 2016, Ultima accesare: 28.06.2024.

[13] Y. Tang, Z. Wang, A. Qu, Y. Yan, K. Hou, D. Zhuang, X. Guo, J. Zhao, Z. Zhao, and W. Ma, "Synergizing spatial optimization with large language models for open-domain urban itinerary planning," 2024, DOI: 10.48550/arXiv.2402.07204.

Anexe

Anexa 1. Controllerul aplicației

```
@RestController
@RequestMapping("/api/v1/relation-trip")
@Tag(
    name = "Vacation Planner Controller", description = "This provides all operations for managing vacation planner"
)
@CrossOrigin(origins = ["http://localhost:4200"])
class VacationPlannerController(
    private val vacationPlannerService: VacationPlannerService,
    private val chatGptService: ChatGptService,
    private val placesApiGateway: PlacesApiGateway,
    private val searchGateway: SearchApiGateway,
    private val webScrapingGateway: WebScrapingApiGateway,
    private val citiesService: FilterCitiesService,
) {
    @Operation(
        summary = "Vacation Planner",
        description = "This operation returns a personalized
            vacation plan based on the provided input parameters. "
    )
    @PostMapping("/planner")
    fun vacationPlanner(@RequestBody vacationPlannerInput: VacationPlannerInput): VacationPlannerOutput {
        return vacationPlannerService
            .vacationPlanner(vacationPlannerInput)
    }

    @Operation(
        summary = "City Exists",
        description = "Checks if a city exists in the database. The operation returns true if the city exists, otherwise false."
    )
    @GetMapping("/city-exists/{city}")
    fun cityExists(@PathVariable city: String): Boolean {
        return citiesService.existsCity(city)
    }

    @Operation(
        summary = "Filter Cities",
        description = "Filters cities based on the provided text. The operation returns a list of cities that match the text."
    )
    @GetMapping("/filter-cities/{text}")
    fun filterCities(@PathVariable text: String): List<String> {
        return citiesService.filterCities(text)
    }

```

```
44      @Operation(
45          summary = "Generate a Point of Interest (POI)",
46          description = "Used for testing the ChatGPT service to
             ↪  generate a Point of Interest (POI). " +
47                  "The operation showcases the ability to generate
                     ↪  content related to a specific point of interest."
48      )
49      @GetMapping("/chatgpt-poi")
50      fun chatGpt(
51          @RequestParam(required = true) destination: String,
52          @RequestParam(required = false) gender: Gender?,
53          @RequestParam(required = false) attendant: Attendants?,
54          @RequestParam(required = false) season: Season?,
55          @RequestParam(required = false) transport: Transport?,
56          @RequestParam(required = true) interests: List<Interests>,
57          @RequestParam(required = false) otherInterests: String?
58      ): Itinerary? {
59          return chatGptService.generatePoi(
60              city = destination,
61              gender = gender,
62              attendant = attendant,
63              season = season,
64              transport = transport,
65              interests = interests,
66              otherInterests = otherInterests
67          )
68      }
69
70      @Operation(
71          summary = "Extract Cities from HTML",
72          description = "Used for testing the ChatGPT service to
             ↪  extract cities from HTML content. "
73      )
74      @GetMapping("/chatgpt-cities")
75      fun chatGptCities(): List<String>? {
76          return chatGptService.extractCitiesFromText(
77              "Best Romanian fall destinations - RomaniaTourStore\n
78        To Search, type and hit enter\n
79        Best Romanian fall destinations\n
80              Do you feel the need of a getaway, even for just a
                 ↪  weekend? Why not take a trip to Romania and see the
                 ↪  way the nature is changing colors into the new
                 ↪  season. You can opt for an urban adventure in one of
                 ↪  Romania's cities, learn more about the rural life in
                 ↪  some of Why should you choose Romania as a travel
                 ↪  fall destination? Because it's one of the most
                 ↪  affordable ones in Europe, it offers a lot of variety
                 ↪  in terms of travel destinations and tourist
                 ↪  attractions and it's relatively accessible. Here are
                 ↪  some great Romanian travel destinations for the fall
                 ↪  season:\n
81              Maramures region\n
```

```
82              The Maramures region is a great option for any fall
                 ↪ getaway. The villages scattered on hills that change
                 ↪ color in rust hues are a perfect place to relax. You
                 ↪ can do hiking, visits to the Merry Cemetery in
                 ↪ Sapanta, unique in the world, and to the Barsana and
                 ↪ Ieud monasteries. If you decide to go hiking, don't
                 ↪ forget to pack some warm clothes with you and a rain
                 ↪ coat.\n
83              Sighisoara\n
84              If you're not interested in spending some time in nature,
                 ↪ but rather go on a romantic getaway, you can opt for
                 ↪ a If you take a trip to Sighisoara at the end of
                 ↪ October, when Halloween is just around the corner,
                 ↪ numerous events and parties are organized in various
                 ↪ hot sports of the town to celebrate the day of the
                 ↪ undead. After all, Vlad Tepes' home is also located
                 ↪ in Sighisoara and you can visit it, as it functions
                 ↪ as a thematic restaurant and museum at the same
                 ↪ time.\n
85              Magura village\n
86              Located between Bucegi and Piatra Craiului Mountains, at
                 ↪ an altitude of approximatley 1,000 meters, the
                 ↪ village of Magura offers breathtaking scenery,
                 ↪ especially during the fall season when, nature
                 ↪ overwhelms you with all its colors, giving you a nice
                 ↪ feeling of tranquility.\n
87              What can you do in Magura? Of course, there are a lot of
                 ↪ trails you can cover on foot or by bike. The simplest
                 ↪ and most satisfying one is a trail to Curmatura and
                 ↪ another simple trail, even for beginner is the one to
                 ↪ the Zarnesti Steeps.\n
88              Brasov city\n
89              As we mentioned before, You can't pass through Brasov
                 ↪ without passing through Sfatului Square (Piata
                 ↪ Sfatului). During medieval times, this was the place
                 ↪ where Romanian, Saxon and Hungarian merchants used to
                 ↪ organize fairs. In the building with an observation
                 ↪ tower, also known as Casa Sfatului, there were
                 ↪ arranged chamber for magistrates. In time, Casa
                 ↪ Sfatului was a city hall and now it functions as a
                 ↪ History Museum.\n
90              Of course, near the square is the famous Black Church,
                 ↪ the largest gothic cathedral from southeastern
                 ↪ Europe. The church was built at the end of the
                 ↪ fourteenth century as a Catholic church, but after
                 ↪ the religious reforms established by Johannes
                 ↪ Honterus in Transylvania, it was converted into an
                 ↪ Evangelical church.\n
91              The medieval sites also include two remaining gates,
                 ↪ Ecaterina Gate and Schei Gate, as well as various
                 ↪ medieval towers dedicated to the various guilds.\n
92              Omu Peak in the Bucegi Mountains\n
```

```
93              Reaching the Omu Peak is always interesting, because you
                ↪ have to pass by the Babele and the Sphinx, the
                ↪ mysterious and unique natural rock formations found
                ↪ in the Bucegi Mountains. Some say aliens made them,
                ↪ but that's a story for another time. After that,
                ↪ every hiker going up on the Bucegi will inevitably
                ↪ rest a little or spend the night at the Omu Hut. The
                ↪ trail offers some breathtaking, picture - perfect
                ↪ landscapes. Also, you shouldn't be surprised if you
                ↪ encounter some shepherds with their sheep and dogs
                ↪ just casually lounging in the grass.\n
94              Two castles: Bran and Peles\n
95              Prahova Valley is not only a crowded during winter, when
                ↪ tourists enjoy skiing or practicing other types of
                ↪ winter sports in Sinaia, Busteni, Predeal or Poiana
                ↪ Brasov. This is also a great fall destination if you
                ↪ want to visit some of the most beautiful castles in
                ↪ Europe: Bran and Peles.\n
96              Bran Castle is located near Brasov and it's now
                ↪ practically an amusement park for Dracula's fans.
                ↪ However, if you're not into the whole vampire myth
                ↪ thing, you can still visit the rooms of the castle,
                ↪ admire the unique architectural style and various
                ↪ items dating back to centuries ago.\n
97              You can visit both castles in just one day, by booking
                ↪ our If you're not convinced yet that Romanian country
                ↪ is a great option for fall destinations, be it
                ↪ weekend getaways or longer trips, take a look at our
                ↪ other\n "
98          )
99      }
100
101     @Operation(
102         summary = "Get Place Information",
103         description = "Retrieves information about a specific place
                ↪ of interest (POI) for testing purposes. " +
104                 "The provided destination parameter specifies the
                    ↪ place for which information is requested."
105     )
106     @GetMapping("/places", params = ["destination"])
107     fun places(@RequestParam(required = true) destination: String):
            ↪ PlaceDetails? {
108         return placesApiGateway.searchPlace(destination)
109     }
110
111     @Operation(
112         summary = "Custom Search",
113         description = "Performs a custom search for tourist
                ↪ destinations based on the provided prompt. " +
114                 "The prompt is a search query that may include
                    ↪ keywords related to travel destinations and user
                    ↪ interests"
115     )
```

```
116         @GetMapping("/custom-search", params = ["prompt"])
117         fun customSearch(@RequestParam(required = true) prompt: String):
             ↪ List<SearchDetails> {
118             return searchGateway.search(prompt)
119         }
120
121         @Operation(
122             summary = "Web Scraping with Content Cleanup",
123             description = "Retrieves information by performing web
                 ↪ scraping on the provided URL. " +
124                     "The operation includes cleaning up the content by
                         ↪ removing JavaScript, HTML and CSS code. "
125         )
126         @GetMapping("/web-scraping", params = ["url"])
127         fun webScraping(@RequestParam(required = true) url: String):
             ↪ String? {
128             return webScrapingGateway.getWebScrapingResults(url)
129         }
130     }
```

Anexa 2. Implementarea serviciului VacationPlanner pentru planificarea personalizată

```
@Service
class VacationPlannerService(
    private val chatGptService: ChatGptService,
    private val imageGeneratorService: ImageGeneratorService,
    private val customSearchService: CustomSearchService,
    private val placesApiGateway: PlacesApiGateway,
    private val webScrapingApiGateway: WebScrapingApiGateway
) : IVacationPlannerService {
    override fun vacationPlanner(vacationPlannerInput:
     ↪ VacationPlannerInput): VacationPlannerOutput {
        // search on Google for the given destination
        val searchPrompt = customSearchService
            .generatePromptCustomSearch(vacationPlannerInput)
        val searchResults: List<SearchDetails> =
         ↪ customSearchService.search(searchPrompt)

        // empty list means that we have a city as destination, and
         ↪ we will use it for chatgpt
        val destination: String = if (searchResults.isEmpty()) {
            vacationPlannerInput.destination
        }
        // not empty list means that we have a continent, a country
         ↪ or nothing as destination
        else {
            val cities = mutableMapOf<String, Int>()

            // for 2 random results, call web scraping and get the
             ↪ content
            val randomSearchResultsContent = mutableListOf<String>()
            for (element in searchResults.shuffled()) {
                val content = webScrapingApiGateway
                    .getWebScrapingResults(element.link)
                if (content != "")
                    randomSearchResultsContent.add(content)
                if (randomSearchResultsContent.size == 2)
                    break
            }

            for (content in randomSearchResultsContent) {
                // for every content, call chatgpt and get the cities
                val citiesFromContent =
                    chatGptService.extractCitiesFromText(content,
                     ↪ vacationPlannerInput.destination)

                for (city in citiesFromContent) {
                    cities[city] = cities.getOrDefault(city, 0) + 1
                }
            }

            if (cities.isEmpty()) {
                throw HttpClientErrorException(HttpStatus.NO_CONTENT,
                 ↪ "No cities founded for the given destination.")
```

```
46              }
47
48              // remove from list the same cities as starting point
49              cities.remove(vacationPlannerInput.startingPoint)
50
51              // get the best city from the list of cities
52              // which means the city with the most frequency in the
                 ↪  list
53              // if there are more cities with the same frequency, we
                 ↪  will choose it randomly
54              val maxFrequency = cities.values.max()
55              val mostFrequentCities = cities.filterValues { it ==
                 ↪  maxFrequency }.keys.toList()
56              mostFrequentCities.shuffled().first()
57          }
58
59          // generate pois with chatgpt
60          val chatGptOutput = chatGptService.generatePoi(
61              city = destination,
62              gender = vacationPlannerInput.gender,
63              attendant = vacationPlannerInput.attendant,
64              season = vacationPlannerInput.season,
65              transport = vacationPlannerInput.transport,
66              interests = vacationPlannerInput.interests,
67              otherInterests = vacationPlannerInput.otherInterests
68          ) ?: throw HttpClientErrorException(
69              HttpStatus.NO_CONTENT,
70              "ChatGpt can't generate pois for the given destination:
                 ↪  $destination"
71          )
72
73          // call places api for every poi
74          val listOfPois: MutableList<Poi> =
             ↪  buildListOfPois(chatGptOutput.points_of_interest,
             ↪  destination)
75          if (listOfPois.isEmpty()  destination == "") {
76              throw HttpClientErrorException(HttpStatus.NO_CONTENT,
77                  "No tour founded for the given destination.")
78          }
79
80          // call OpenAi for the photo
81          val photoOfItinerary = imageGeneratorService
82              .generateImage(chatGptOutput.tour_name)
83
84          // return vacation planner output
85          return VacationPlannerOutput(
86              photo = photoOfItinerary,
87              name = chatGptOutput.tour_name,
88              destination = destination,
89              startingPoint = vacationPlannerInput.startingPoint,
90              season = vacationPlannerInput.season,
91              attendant = vacationPlannerInput.attendant,
92              budget = vacationPlannerInput.budget,
```

```
93                transport = vacationPlannerInput.transport,
94                distance = 0.0,
95                poisNumber = listOfPois.size,
96                highlights = chatGptOutput.highlights,
97                pois = listOfPois
98            )
99        }
100
101       private fun buildListOfPois(pointsOfInterest: List<ItineraryPoi>,
          ↪ destination: String): MutableList<Poi> {
102           val pois = mutableListOf<Poi>()
103
104           pointsOfInterest.forEach {
105               val placeDetails = placesApiGateway.searchPlace(it.name +
                  ↪ ", " + destination)
106               if (placeDetails != null) {
107                   pois.add(
108                       Poi(
109                           name = placeDetails.placeName,
110                           photo = placeDetails.photoReference,
111                           description = it.description,
112                           tags = it.tags,
113                           stars = placeDetails.rating,
114                           address = placeDetails.placeAddress,
115                           phone = placeDetails.phone,
116                           website = placeDetails.website,
117                           schedule = placeDetails.schedule,
118                           latitude = placeDetails.latitude,
119                           longitude = placeDetails.longitude
120                       )
121                   )
122               }
123           }
124
125           if (pois.isEmpty()) {
126               throw HttpClientErrorException(HttpStatus.NO_CONTENT,
127                   "No pois founded for the given destination.")
128           }
129
130           return pois
131       }
132   }
```

Anexa 3. Implementarea serviciului ChatGPT

```
@Service
class ChatGptService(
    private val chatGptGateway: IOpenAiGateway
) : IChatGptService {
    override fun generatePoi(
        city: String,
        gender: Gender?,
        attendant: Attendants?,
        season: Season?,
        transport: Transport?,
        interests: List<Interests>,
        otherInterests: String?
    ): Itinerary? {
        val prompt =
            "Design an unforgettable journey through the streets of ${city}, a place steeped in rich history, vibrant culture, and cherished traditions. Craft a tour experience that captivates the imagination of travelers, offering a glimpse into the soul of this remarkable destination.\n\n Your task is to design a tour itinerary for a person with interests in ${
                interests.joinToString(
                    " and "
                )
            }${if (otherInterests != null) ", $otherInterests" else ""}. Consider the ${
                transport?.let { "$it, " } ?: ""
            }${
                attendant?.let { "$it, " } ?: ""
            } and plan the perfect adventure for ${
                season?.let { "$it " } ?: "all-season "
            }, featuring no more than 5 points of interest.\n\n" +
            "As you construct the tour, search into the complexity of each point of interest (POI), completing the narrative with historical anecdotes, cultural insights, and tales passed down through generations. Let the tour highlights unfold like chapters in a captivating story, each revealing a facet of $city's allure.\n\n For each POI, provide a detailed and long description (200 words) that transports the reader to that location, evoking the sights, sounds, and sensations unique to $city. Additionally, include representative tags that encapsulate the essence of each POI, ensuring that the tour resonates with a diverse audience.\n\n" +
            "JSON Parameters:\n" +
            "- tour_name\n" +
            "- tour_highlights\n" +
            "- points_of_interest (name, description, tags)"

        val content = JSONObject(chatGptGateway.runPrompt(prompt))
        val pointsOfInterest = mutableListOf<ItineraryPoi>()
```

```
34
35          for (i in 0 until
             ↪  content.getJSONArray("points_of_interest").length()) {
36              pointsOfInterest.add(ItineraryPoi(
37                  name = content.getJSONArray("points_of_interest")
38                  .getJSONObject(i).getString("name"),
39                  description =
                     ↪  content.getJSONArray("points_of_interest")
40                  .getJSONObject(i).getString("description"),
41                  tags = content.getJSONArray("points_of_interest")
42                  .getJSONObject(i).getJSONArray("tags")
43                      .map { it.toString() }
44              ))
45          }
46
47          return Itinerary(
48              tour_name = content.getString("tour_name"),
49              highlights = if (content.has("tour_highlights")) {
50                  val tourHighlights = content.get("tour_highlights")
51                  if (tourHighlights is JSONArray) {
52                      tourHighlights.map { it.toString() }
53                  } else {
54                      listOf(tourHighlights.toString())
55                  }
56              } else {
57                  listOf()
58              },
59              points_of_interest = pointsOfInterest
60          )
61      }
62
63      override fun extractCitiesFromText(
64          text: String,
65          destination: String
66      ): List<String> {
67          // limit the text to 16385 characters
68          val textWithLimit = if (text.length > 16180) {
             ↪  text.substring(0, 16180) } else  { text }
69          val textForDestination = if (destination != "") " from
             ↪  $destination " else ""
70          val prompt = "Extract all the cities name $textForDestination
             ↪  from the following text:\n$textWithLimit \n" +
71                  "Return them as a list of values in the format
                     ↪  `city_name, country_name`\n" +
72                  "Parameters of json: cities"
73          val content = JSONObject(chatGptGateway.runPrompt(prompt))
74          val cities = mutableListOf<String>()
75          for (i in 0 until content.getJSONArray("cities").length()) {
76              cities.add(content.getJSONArray("cities").getString(i))
77          }
78          return cities
79      }
80  }
```

Anexa 4. Implementarea serviciului pentru managementul excepțiilor

```
@ControllerAdvice
class VacationPlannerExceptionHandler :
    ResponseEntityExceptionHandler() {
    private val log: Logger = LoggerFactory
        .getLogger(VacationPlannerExceptionHandler::class.java)

    @ExceptionHandler(HttpClientErrorException::class)
    fun handleHttpClientErrorException(
        ex: HttpClientErrorException
    ): ResponseEntity<ErrorMessage> {
        val message = ErrorMessage(
            ex.statusCode.value(),
            "Http error: ${ex.message}",
            "",
        )
        when (ex.statusCode) {
            HttpStatus.NOT_FOUND,
            HttpStatus.NO_CONTENT -> log.info(message.message)
            HttpStatus.UNAUTHORIZED -> log.warn(message.message)
            HttpStatus.FORBIDDEN,
            HttpStatus.BAD_REQUEST,
            HttpStatus.UNPROCESSABLE_ENTITY,
            HttpStatus.CONFLICT,
            HttpStatus.INTERNAL_SERVER_ERROR ->
                log.error(message.message)
            else -> log.error(message.message)
        }
        return ResponseEntity<ErrorMessage>(message, ex.statusCode)
    }

    @ExceptionHandler(Exception::class)
    fun handleException(
        ex: Exception
    ): ResponseEntity<ErrorMessage> {
        val message = ErrorMessage(
            HttpStatus.BAD_REQUEST.value(),
            "Error: ${ex.localizedMessage}",
            "",
        )
        log.error(message.message)
        return ResponseEntity<ErrorMessage>(message,
            HttpStatus.BAD_REQUEST)
    }

    override fun handleMethodArgumentNotValid(
        ex: MethodArgumentNotValidException,
        headers: HttpHeaders,
        status: HttpStatusCode,
        request: WebRequest
    ): ResponseEntity<Any>? {
        val message = ErrorMessage(
            status.value(),
```

```
50                  "Validation error: ${ex.message}",
51                  request.getDescription(false),
52              )
53              log.warn(message.message)
54              return super.handleExceptionInternal(ex, message, headers,
                 ↪  status, request)
55          }
56
57          override fun handleExceptionInternal(
58              ex: Exception,
59              body: Any?,
60              headers: HttpHeaders,
61              statusCode: HttpStatusCode,
62              request: WebRequest
63          ): ResponseEntity<Any>? {
64              val message = ErrorMessage(
65                  statusCode.value(),
66                  "Unknown error: ${ex.message}",
67                  request.getDescription(false),
68              )
69              log.error(message.message)
70              return super.handleExceptionInternal(ex, message, headers,
                 ↪  statusCode, request)
71          }
72      }
```

Anexa 5. Fișierul pom.xml folosit pentru instalarea dependențelor proiectului cu Maven

```
1       <?xml version="1.0" encoding="UTF-8"?>
2   <project xmlns="http://maven.apache.org/POM/4.0.0"
     ↪  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
       ↪  https://maven.apache.org/xsd/maven-4.0.0.xsd">
4     <modelVersion>4.0.0</modelVersion>
5     <parent>
6       <groupId>org.springframework.boot</groupId>
7       <artifactId>spring-boot-starter-parent</artifactId>
8       <version>3.1.5</version>
9       <relativePath/> <!-- lookup parent from repository -->
10    </parent>
11    <groupId>com.example</groupId>
12    <artifactId>Proiect</artifactId>
13    <version>0.0.1-SNAPSHOT</version>
14    <name>Proiect</name>
15    <description>Proiect</description>
16    <properties>
17      <java.version>17</java.version>
18      <kotlin.version>1.8.22</kotlin.version>
19    </properties>
20    <dependencies>
21      <dependency>
22        <groupId>org.springframework.boot</groupId>
23        <artifactId>spring-boot-starter-web</artifactId>
24      </dependency>
25      <dependency>
26        <groupId>com.fasterxml.jackson.module</groupId>
27        <artifactId>jackson-module-kotlin</artifactId>
28      </dependency>
29      <dependency>
30        <groupId>org.jetbrains.kotlin</groupId>
31        <artifactId>kotlin-reflect</artifactId>
32      </dependency>
33      <dependency>
34        <groupId>org.jetbrains.kotlin</groupId>
35        <artifactId>kotlin-stdlib</artifactId>
36      </dependency>
37
38      <dependency>
39        <groupId>org.springframework.boot</groupId>
40        <artifactId>spring-boot-devtools</artifactId>
41      </dependency>
42      <dependency>
43        <groupId>org.projectlombok</groupId>
44        <artifactId>lombok</artifactId>
45        <optional>true</optional>
46      </dependency>
47      <dependency>
48        <groupId>org.springframework.boot</groupId>
49        <artifactId>spring-boot-starter-test</artifactId>
50        <scope>test</scope>
```

```
51        </dependency>
52        <dependency>
53          <groupId>io.swagger.core.v3</groupId>
54          <artifactId>swagger-annotations</artifactId>
55          <version>2.2.15</version>
56        </dependency>
57        <dependency>
58          <groupId>jakarta.persistence</groupId>
59          <artifactId>jakarta.persistence-api</artifactId>
60          <version>3.1.0</version>
61        </dependency>
62        <dependency>
63          <groupId>org.apache.httpcomponents.client5</groupId>
64          <artifactId>httpclient5</artifactId>
65          <version>5.2.1</version>
66        </dependency>
67        <dependency>
68          <groupId>org.json</groupId>
69          <artifactId>json</artifactId>
70          <version>20231013</version>
71        </dependency>
72        <dependency>
73          <groupId>com.google.maps</groupId>
74          <artifactId>google-maps-services</artifactId>
75          <version>2.2.0</version>
76        </dependency>
77        <dependency>
78          <groupId>org.jsoup</groupId>
79          <artifactId>jsoup</artifactId>
80          <version>1.16.2</version>
81        </dependency>
82        <dependency>
83          <groupId>org.springframework.boot</groupId>
84          <artifactId>spring-boot-starter-validation</artifactId>
85        </dependency>
86      </dependencies>
87
88      <build>
89        <sourceDirectory>
90            ${project.basedir}/src/main/kotlin
91        </sourceDirectory>
92        <testSourceDirectory>
93                ${project.basedir}/src/test/kotlin
94        </testSourceDirectory>
95        <plugins>
96          <plugin>
97            <groupId>org.springframework.boot</groupId>
98            <artifactId>spring-boot-maven-plugin</artifactId>
99            <configuration>
100             <image>
101               <builder>
102                   paketobuildpacks/builder-jammy-base:latest
103               </builder>
```

```
104             </image>
105             <excludes>
106               <exclude>
107                 <groupId>org.projectlombok</groupId>
108                 <artifactId>lombok</artifactId>
109               </exclude>
110             </excludes>
111           </configuration>
112         </plugin>
113         <plugin>
114           <groupId>org.jetbrains.kotlin</groupId>
115           <artifactId>kotlin-maven-plugin</artifactId>
116           <configuration>
117             <args>
118               <arg>-Xjsr305=strict</arg>
119             </args>
120             <compilerPlugins>
121               <plugin>spring</plugin>
122             </compilerPlugins>
123           </configuration>
124           <dependencies>
125             <dependency>
126               <groupId>org.jetbrains.kotlin</groupId>
127               <artifactId>kotlin-maven-allopen</artifactId>
128               <version>${kotlin.version}</version>
129             </dependency>
130           </dependencies>
131         </plugin>
132       </plugins>
133     </build>
134
135   </project>
```