UNIVERSITATEA DIN PITEȘTI
FACULTATEA DE ȘTIINȚE, EDUCAȚIE FIZICĂ ȘI INFORMATICĂ
PROGRAMUL DE STUDII UNIVERSITARE DE LICENȚĂ: INFORMATICĂ

LUCRARE DE LICENȚĂ

Coordonator științific,
Lector univ. dr. Tudose Cristina

Absolvent,
Voicea Georgiana

Pitești
2023

UNIVERSITATEA DIN PITEȘTI
FACULTATEA DE ȘTIINȚE, EDUCAȚIE FIZICĂ ȘI INFORMATICĂ
PROGRAMUL DE STUDII UNIVERSITARE DE LICENȚĂ: INFORMATICĂ

Platformă web pentru găsirea de parteneri de călătorie

Coordonator științific,
Lector univ. dr. Tudose Cristina

Absolvent,
Voicea Georgiana

Pitești
2023

Cuprins

Introducere

Prin intermediul acestei lucrări de licență intitulată „Platformă web pentru găsirea de parteneri de călătorie” obiectivul meu este de a realiza o aplicație care să promoveze domeniul de turism și dezvoltarea umană prin experiențe inedite ajutând utilizatorii să găsească oameni cu care să împărtășească pasiunea de a călătorii.

Motivul care m-a influențat să aleg această temă, a fost dorința oamenilor de a călătorii, dar și frica acestora de a păși in necunoscut singuri. De asemenea, experiența de a călătorii cu diferite tipologii de persoane mi-a arătat ca compania are o mare influență asupra împlinirii scopului călătoriei. Pentru a avea o experiență cât mai plăcută, partenerii noștri de călătorie trebuie să aibă aceleași scopuri și interese ca si noi, iar în acest punct aplicația creată își va împlinii utilitatea.

Idea proiectului „Travel With Me” este de a crea o aplicație web, unde utilizatori din toată lumea să se cunoască și să programeze excursii. Utilizatorii își vor crea un cont în baza căruia vor furniza date de contact, poze reprezentative și interese pe baza cărora se vor genera compatibilități.

O aplicație web este un program stocat pe un server la distanță și livrat pe internet. Aplicațiile nu trebuie descărcate, deoarece sunt accesate prin rețea. Pentru a funcționa, ele au nevoie de un server web, server de aplicație și o baza de date.

Lucrarea de licență este împărțită în trei capitole reprezentative.

În prima parte este descris domeniul în care aplicația va activa și motivele pentru care este important să fie promovată această industrie, dar și beneficiile aduse atât societății cât și omului în mod personal. De asemenea, vom arăta platforme oficiale, dar și neoficiale care promovează călătoriile și schimbul cultural.

În capitolul al doilea, sunt prezentate tehnologiile folosite pentru funcționalitatea aplicației. Instrumentele esențiale folosite pentru scrierea și gestionarea codului au fost Intellij IDEA, Visual Studio Code, Sourcetree asociat la platforma de GitHub și MySQL WorkBranch. Pentru partea de back-end s-a folosit limbajul Java cu framework-ul Spring Boot si Hibernate, iar pentru front-end s-a folosit React și CSS. Baza de date a fost scrisa in MySQL , fiind creată prin intermediul entităților dar și a unor scripturi liquibase.

În final, se prezinta proiectarea aplicației, care constă în descrierea implementării, secvențe de cod semnificative dar și capturi ale platformei web. De asemenea, este descris și scenariului de utilizare.

Capitolul 1. Descrierea domeniului

1.1. Impactul asupra societății

Călătoriile au devenit o parte care se înglobează cu viața modernă, oferind oportunități de explorare, relaxare și interacțiune culturală. În același timp, călătoriile au un impact semnificativ asupra individului și societății în ansamblu.

Unul dintre principalele aspecte ale impactului călătoriilor asupra omului este dezvoltarea personală. Călătoriile oferă oportunitatea de a învăța și de a experimenta lucruri noi, extinzând orizonturile individuale. Prin expunerea la culturi diferite, moduri de viață și perspective, călătoriile pot contribui la dezvoltarea abilităților de adaptare, deschiderea mentală, înțelegerea culturală și toleranța. Ele pot oferi o experiență profundă de auto descoperire. În timpul călătoriilor, oamenii se confruntă adesea cu noi provocări și situații neașteptate, ceea ce îi ajută să-și descopere și să-și dezvolte resursele personale, cum ar fi reziliența, creativitatea și capacitatea de a lua decizii în timp real. De asemenea, călătoriile pot ajuta la dezvoltarea abilităților sociale și de comunicare, deoarece implică interacțiunea cu oameni din diferite medii culturale și lingvistice.

În ceea ce privește impactul călătoriilor asupra societății, acestea au un rol semnificativ în dezvoltarea economică și a turismului. Călătoriile internaționale și naționale generează venituri pentru destinații prin cheltuielile turistice, creând astfel locuri de muncă și stimulând economia locală. Turismul poate spori dezvoltarea infrastructurii, inclusiv hoteluri, restaurante, atracții turistice, facilități de transport și alte servicii, contribuind la creșterea economică a unei regiuni.

Un alt aspect important al impactului călătoriilor asupra societății este interculturalitatea și schimbul cultural. Călătoriile facilitează întâlnirea și interacțiunea cu persoane din culturi și medii diferite. Această interacțiune poate promova înțelegerea, toleranța și respectul reciproc între oameni. Călătoriile pot contribui la dezvoltarea unei societăți multiculturale și la promovarea diversității culturale.

1.2. Promovarea călătoriilor

Călătoriile au devenit tot mai populare datorită promovării acesteia în diverse domenii. Promovarea acestora are un rol esențial în încurajarea oamenilor să iasă din zona lor de confort și să se angajeze în experiențe memorabile și edificatoare.

O dată cu dezvoltarea tehnologiei, marketingul a profitat și le-a folosit din plin. De aceea, cea mai folosită metoda de promovare este prin intermediul rețelelor sociale și a platformelor online care oferă o multitudine de campanii de marketing autentice și inspiraționale, prezentând destinații și experiențe unice. Pe aceste rețele sociale se găsesc și povești de călătorie, imagini și videoclipuri captivante, care să inspire și să atragă noi călători. O altă metodă de promovare a călătoriilor prin intermediul tehnologiilor este și prin platformele digitale, care facilitează planificarea și rezervarea călătoriilor, oferind informații relevante și convenabile ale unui plan de călătorie.

Călătoriile nu sunt promovate doar de companii pentru a avea un câștig, dar și de domeniul educațional. Câteva exemple de promovări ale călătoriilor prin acest domeniu sunt:

- Programe de schimb studențesc, cum ar fi programul Erasmus+, care facilitează mobilitatea internațională a studenților și promovează învățarea interculturală.
- Organizarea de conferințe, seminarii și ateliere tematice care pun accentul pe importanța călătoriilor și pe beneficiile acestora asupra dezvoltării personale și profesionale.
- Integrarea călătoriilor în programele școlare și universitare, pentru a oferi elevilor și studenților oportunitatea de a cunoaște alte culturi și perspective, cum ar fii programul Fulbright (program ce iți oferă șansa de a studia un semestru în USA)

Un alt domeniu prin care călătoriile sunt promovate este voluntariatul și responsabilitățile sociale. Există numeroase programe de voluntariat internațional care combină călătoriile cu contribuția la comunități locale, oferind oportunități de învățare, dezvoltare personală și schimb cultural, dar și colaborarea cu organizații non-guvernamentale și comunități locale pentru a promova turismul responsabil și sustenabil, respectând mediul înconjurător și valorile culturale.

1.3. Platforme similare

Există mai multe platforme care încurajează și facilitează călătoriile și experiențele culturale, oferind oportunități de schimb cultural, găzduire gratuită sau ieftină și interacțiune cu localnicii.

Câteva dintre platformele cele mai folosite sunt:

- Airbnb: este o platformă online care permite închirierea sau închirierea de locuințe și camere de la particulari. Oferă o alternativă flexibilă la hoteluri și permite călătorilor să se cazeze în case autentice și să interacționeze cu gazdele locale.
- Workaway: este o platformă care facilitează schimbul de muncă voluntară pentru cazare și masă. Călătorii pot găsi anunțuri de la gazde care oferă cazare și masă în schimbul ajutorului la diverse proiecte sau activități, cum ar fi grădinăritul, construcția, învățarea limbilor străine sau asistarea într-un hotel.
- HelpX: este o altă platformă care conectează călătorii cu gazde care oferă cazare și masă în schimbul ajutorului la diverse sarcini și proiecte. Acestea pot include lucrul în grădină, creșterea animalelor, lucrul în ferme sau asistarea într-un mic hotel sau restaurant.
- Couchsurfing este o platformă care facilitează găzduirea gratuită între membrii comunității. Călătorii pot găsi gazde dispuse să ofere un loc de cazare gratuit sau contra unei sume modice, permițându-le să experimenteze cultura și stilul de viață local.

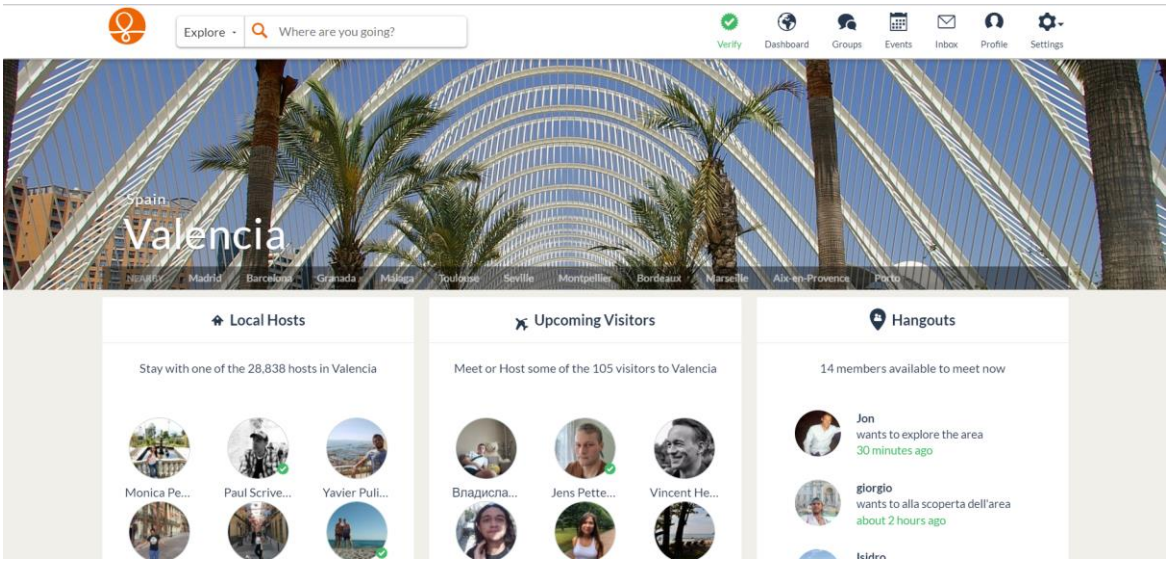

Fig. 1: Pagina de start a aplicației Couchsurfing

Acestea sunt doar câteva dintre platformele care încurajează și facilitează călătoriile și interacțiunea culturală. Fiecare dintre ele oferă oportunități unice de a explora lumea, de a învăța și de a crea legături cu oameni din diferite culturi și medii.

	Gratuită	*Flexibilitate în călătorii*	*Ocazia de ați face prieteni*	*Avantaje economice*	*Interacțiunea cu oameni cu aceleași domenii de interes*
Couchsurfing		X	X	X	
HelpX	X		X	X	
Workaway	X		X	X	
Airbnb	X	X		X	
TravelWithMe	X	X	X	X	X

Majoritatea platformelor din domeniul călătoriilor ajută oamenii să călătorească ieftin sau să găsească cazări. Scopul platformei TravelWithMe este de a conecta oamenii care văd călătoriile cu aceeași ochi și să-i aducă împreună.

Sunt oameni care au destui bani și timp să călătorească dar nu au curajul de a o face singuri iar persoanele din jurul lor au alte viziuni asupra călătoriilor sau chiar nu își doresc. Cu această platformă poți alege unde preferi să dormi, ce alegi să mănânci, în ce buget te încadrezi și ce activități preferi să faci. Astfel, oamenii pot face lucrurile care ii fac fericiți, fără prea multe compromisuri. Acesta este avantajul platformei TravelWithMe pe care alte platforme nu o au.

Capitolul 2. Tehnologii folosite

Tehnologiile de dezvoltare web reprezintă o multitudine de limbaje de programare și instrumente care sunt utilizate pentru a produce platforme web. Deoarece construirea unei aplicații se împarte in două ramuri, tehnologiile preiau aceeași structură, fiind împărțite in tehnologii front-end și tehnologii back-end.

Tehnologiile front-end reprezintă partea clientului. Sunt folosite pentru a dezvolta componentele interactive și pentru a produce elementele pe care utilizatorii le văd și cu care interacționează. Acestea includ culorile, stilurile textului, imaginile, butoanele și meniurile de navigare. Platforma creata a folosit instrumente precum Visual Studio Code pentru scrierea codului și limbaje precum HTML, CSS și biblioteca React din JavaScript.

Tehnologiile back-end reprezintă partea serverului. Sunt folosite pentru a dezvolta bazele tehnice. Acestea stochează, modelează datele și se asigură că aplicația funcționează așa cum ar trebui. De asemenea, utilizând tehnologiile back-end putem rula scripturi, crea procese eficiente, trimite email-uri sau documente atât la cerere cât și prin programare. Câteva instrumente folosite in crearea codului pentru aplicația „Travel With Me ” sunt IntelliJ IDEA, Postman, GitHub, Sourcetree, iar ca și limbaje de programare utilizate sunt Java utilizat cu framework-ul Spring Boot și Hibernate.

Baza de date în care au fost stocate datele a fost gestionată MySql, care utilizează limbajul de programare SQL[1]. Baza de date a fost creată prin backend, iar datele statice au fost introduse prin framework-ul liquibase. Instrumentul folosit pentru gestionarea bazei de date a fost WorkBrench .

[1] Structured Query Language

2.1. Medii de dezvoltare

2.1.1. IntelliJ IDEA

IntelliJ IDEA este un mediu de dezvoltare integrat conceput special pentru dezvoltarea Java și Kotlin. A fost creat de JetBrains și a fost lansat pentru prima dată în 2001. De atunci, a devenit unul dintre cele mai populare IDE-uri Java, cu o bază de utilizatori loială, care include dezvoltatori individuali, startup-uri mici dar și organizații mari. Caracteristica principală a acestui instrument este setul său puternic de instrumente pentru construirea, testarea și depanarea aplicațiilor. Acesta include un editor de cod care oferă caracteristici cum ar fi evidențierea sintaxei, completarea codului și formatarea codului, ceea ce face mai ușor pentru dezvoltatori să scrie cod curat, bine organizat. Editorul include, de asemenea, instrumente pentru refactorizarea codului, permițând dezvoltatorilor să facă cu ușurință modificări ale codului lor, fără a introduce bug-uri.

Un beneficiu semnificativ ale IntelliJ este integrarea sa strânsă cu sisteme precum Git și Maven. Simplifică fluxul de împingere a schimbărilor dar și de rezolvare a conflictelor în Git și oferă comenzi prestabilite ale Maven.

Printre multele caracteristici pe care acesta le are, o caracteristică important de menționat este suportul său pentru testarea și depanarea unităților. IDE include un canal de testare încorporat care permite dezvoltatorilor să ruleze rapid și să depaneze testele unitare , facilitând prinderea și remedierea erorilor la începutul procesului de dezvoltare, dar și un sistem de depanare puternic care permite dezvoltatorilor să treacă prin cod, să inspecteze variabilele și să găsească cauza principală a problemelor.

Pentru a face dezvoltatorii și mai productivi, IntelliJ IDEA include o gamă largă de comenzi rapide de la tastatură care pot fi utilizate pentru a efectua sarcini comune și pentru a naviga prin cod rapid și ușor. Aceste comenzi rapide pot fi personalizate pentru a se potrivi preferințelor individuale ale fiecărui dezvoltator, permițându-le să lucreze mai eficient și mai eficient. Câteva exemple de comenzi des folosite sunt :

- ctrl + shift + F - caută un text.

- ctrl + click - are la declarația variabilei / funcției.
- alt + F7 - listează utilizările variabilei / funcțiilor.
- alt + enter - afișează o listă de sugestii pentru îmbunătățiri ale codului și remedieri rapide.
- ctrl + alt + M - transformă codul selectat într-o funcție.
-

2.1.2. Visual Studio Code

Visual Studio Code (sub denumirea și de VS Code) este un editor de text gratuit, oferit de Microsoft. VS Code este disponibil pentru toate tipurile populare de sisteme de operare: Windows, Linux și macOS.

Editorul suportă o gamă largă de limbaje de programare de la Java, C++ și Python la CSS, Go și Dockerfile. În plus, permite adăugarea și chiar crearea de noi extensii, debug și suport de dezvoltare cloud și web. Vine cu suport încorporat pentru JavaScript, TypeScript și Node.js. Astfel, acest tool este folosit in proiectul prezentat pentru scrierea codului de front-end.

Multe extensii sunt actualizate ori de câte ori este necesar. Suportul oferit variază între diferitele limbaje de programare și extensiile acestora, variind de la evidențierea simplă a sintaxelor și potrivirea parantezelor până la depanare și refactorizare.

Interfața IDE permite o mulțime de interacțiune în comparație cu alți editori de text. Pentru a simplifica experiența utilizatorului, VS Code este împărțit în cinci regiuni principale:

- Bara de activități (fig.2 - A)
- Bara laterală (fig.2 - B)
- Grupuri de editori (fig.2 - C)
- Panoul principal (fig.2 - D)
- Bara de stare (fig.2 - E)

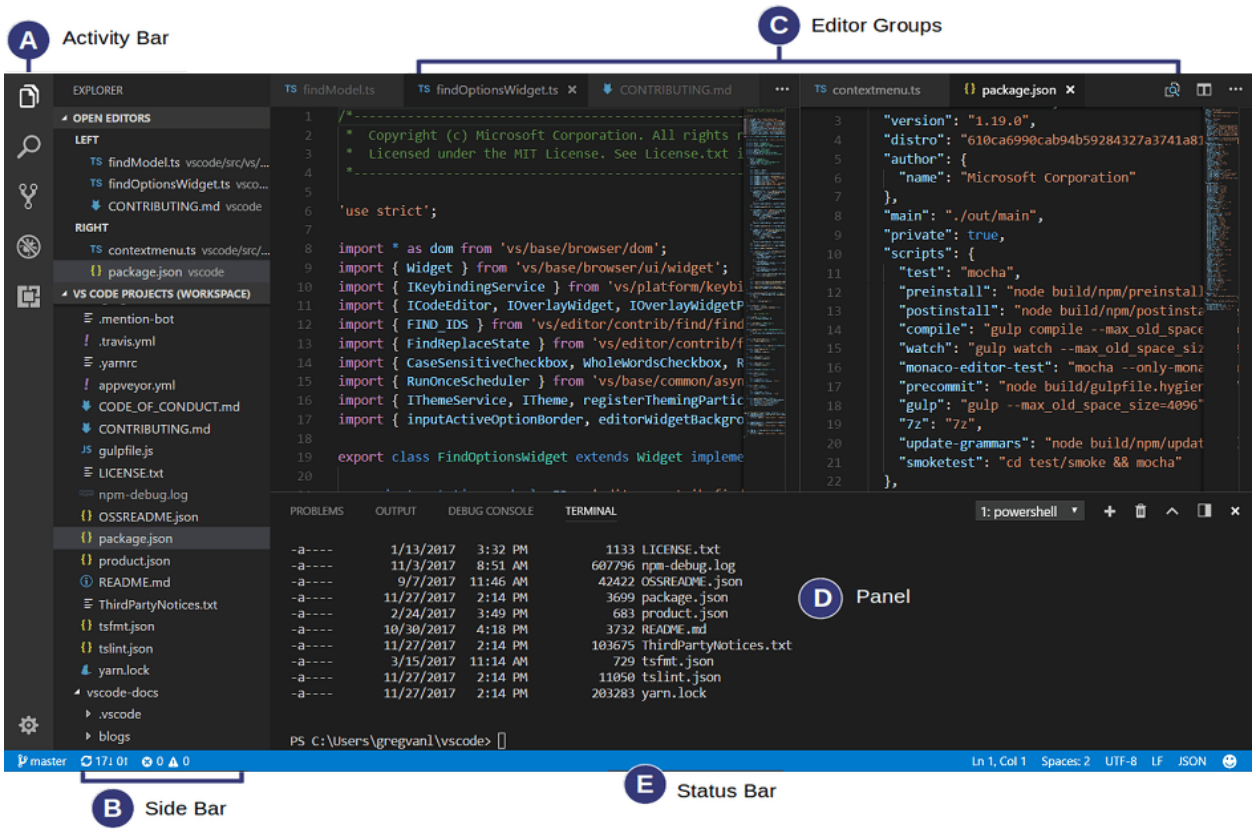

Fig. 2: Aspectul aplicației VS Code [2]

2.1.3.MySQL WorkBench

MySQL Workbench este o aplicație software dezvoltată de MySQL AB, care oferă un mediu integrat pentru dezvoltarea și administrarea bazelor de date MySQL. Este utilizată de dezvoltatori și administratori de baze de date pentru a gestiona și manipula datele într-un mod eficient și ușor de utilizat. Unul dintre aspectele-cheie ale MySQL Workbench este editorul de interogări, care permite utilizatorilor să scrie și să execute interogări SQL într-un mod intuitiv. Acesta oferă funcții de evidențiere a sintaxei, completare automată a codului și verificare a erorilor pentru a ajuta la dezvoltarea interogărilor corecte și eficiente.

Aplicația include, de asemenea, o funcționalitate puternică de modelare vizuală, care permite utilizatorilor să creeze și să modifice scheme de baze de date utilizând diagrame entitate-relație (ERD). Aceasta facilitează proiectarea structurii bazei de date, inclusiv tabele, relații între tabele, chei primare și străine etc.

MySQL Workbench oferă, de asemenea, un set de instrumente pentru administrarea serverelor MySQL și bazelor de date asociate. Utilizatorii pot configura și gestiona conexiuni la servere MySQL, efectua backup-uri și restaurări de date, crea utilizatori și gestiona permisiuni, precum și efectua alte activități de administrare.

Pe lângă dezvoltare și administrare, MySQL Workbench include și facilități pentru optimizarea și depanarea performanței bazelor de date. Utilizatorii pot analiza interogările pentru a identifica posibile probleme de performanță, pot examina planurile de execuție și pot optimiza schema bazei de date pentru a obține performanțe mai bune. În concluzie, MySQL Workbench este un instrument puternic și versatil pentru dezvoltarea și administrarea bazelor de date MySQL. Prin intermediul editorului de interogări, modelării vizuale, funcționalităților de administrare și depanare, oferă dezvoltatorilor și administratorilor un mediu cuprinzător pentru gestionarea eficientă a bazelor de date MySQL.

2.2. Backend

2.1.1. Java

James Gosling a creat limbajul de programare orientat pe obiecte Java, care a fost apoi achiziționat de Oracle. A fost lansat în 1995 și este în prezent unul dintre cele mai populare limbaje de programare folosite. Acesta poate fi folosit pentru a dezvolta aplicații pentru o varietate de medii. Una dintre principalele caracteristici ale Java este că este o platformă independentă. Acest lucru înseamnă că un program scris în Java poate fi executat pe orice sistem de operare (cum ar fi Windows, Mac sau Linux) deoarece codul este compilat în bytecode care poate rula pe orice mașină virtuală Java (JVM), indiferent de arhitectura computerului.

Mașină virtuală java

O mașină virtuală Java (JVM) este un program care oferă mediul de rulare necesar pentru executarea programelor Java. Programele Java nu pot rula fără JVM pentru platforma hardware și OS corespunzătoare.

Când o mașină virtuală preia un program Java pentru execuție, programul nu este furnizat ca cod sursă în limbajul Java. În schimb, sursa limbajului Java trebuie să fi fost convertită (sau

compilată) într-o formă cunoscută sub numele de bytecode Java. Java bytecode trebuie să fie furnizate la JVM într-un format numit fișiere de clasă. Aceste fișiere de clasă au întotdeauna o extensie .class.

JVM are două funcții principale:

- Să permită programelor Java să ruleze pe orice dispozitiv sau sistem de operare (cunoscut sub numele de principiul "scrie o dată, rula oriunde");
- Să gestioneze și să optimizeze memoria programului.

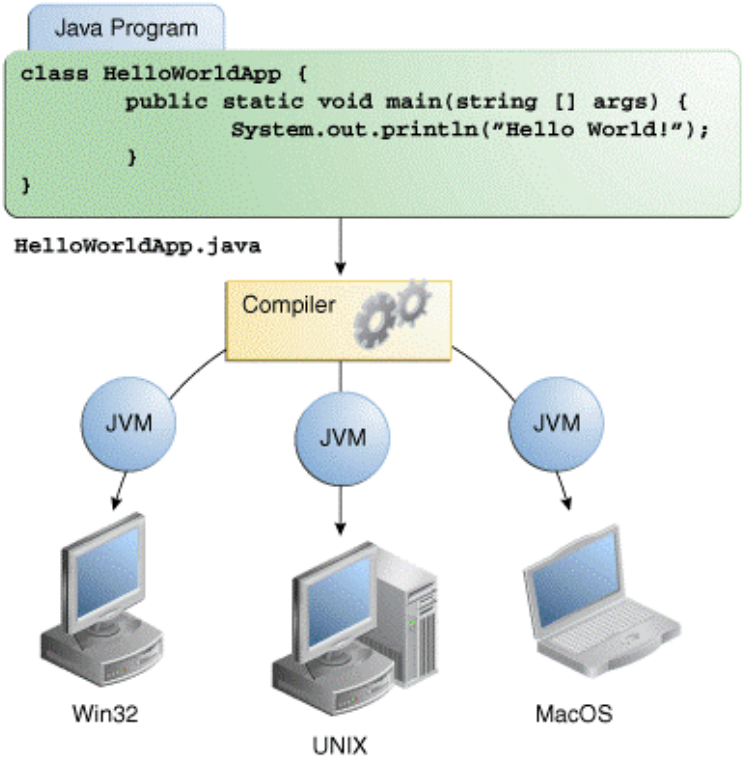

Fig. 3: Mașina Virtuală Java [10]

Structura unui program Java

Un program Java are, în general, următoarea structură:

- *Pachete* - Declararea pachetelor utilizate în cadrul programului. Acestea sunt grupuri de clase care sunt organizate în mod logic pentru a facilita reutilizarea și gestionarea codului.

- *Importuri* - Includerea claselor necesare din alte pachete în cadrul programului. Acestea permit accesul la funcționalitățile oferite de alte clase din alte pachete.
- *Clasa principală* - Clasa care conține metoda principală (main) și care este punctul de intrare în program. Aceasta trebuie să fie publică și să fie numită exact ca și numele fișierului.
- *Variabile* - Declararea variabilelor utilizate în cadrul programului. Acestea trebuie să fie declarate cu tipul lor de date corespunzător.
- *Instrucțiuni* - Instrucțiunile sunt comenzi care sunt executate de program. Acestea pot fi de mai multe tipuri, inclusiv declarații de variabile, instrucțiuni de control a fluxului, expresii și apeluri de metode.

Variabile

O variabilă ne oferă un spațiu de stocare pe care programele noastre îl pot manipula. Acestui spațiu de stocare trebuie să îi atribuim un nume unic. Fiecare variabilă din Java are un tip specific, care determină dimensiunea și aspectul memoriei variabilei; intervalul de valori care pot fi stocate în acea memorie; și setul de operațiuni care pot fi aplicate variabilei.

Un nume de variabilă în Java poate conține numai litere, numere, subliniere (_) sau semnul dolarului ($). Cu toate acestea, primul caracter nu poate fi un număr. Convenția este de a începe întotdeauna numele variabilelor cu a litera, nu „$" sau „_". Numele variabilelor trebuie să fie scurte, dar semnificative, concepute pentru a indica cititor ocazional intenția utilizării sale.

În funcție de locul în care ele sunt declarate, se împart in 3 tipuri:

- *Variabile locale* - sunt declarate în cadrul metodelor, constructorilor sau blocurilor și sunt valabile numai în interiorul acelui context în care au fost definite.
    - modificatorii de acces nu pot fi utilizați pentru variabilele locale.
    - sunt vizibile numai în cadrul metodei declarate, al constructorului sau al blocului.
    - nu există o valoare implicită pentru variabilele locale, astfel încât variabilele locale trebuie declarate și o valoare inițială trebuie atribuită înainte de prima utilizare.

- *Variabilele de instanță* - sunt declarate într-o clasă, dar în afara unei metode, constructor sau orice bloc.
    - modificatorii de acces pot fi utilizați pentru variabilele locale.
    - sunt vizibile pentru toate metodele, constructorii și blocul din clasă.
    - au valori implicite. Pentru numere, valoarea implicită este 0, pentru boolean este falsă, iar pentru referințele obiecte este nulă.
- *Variabile de clasă/statice* - sunt declarate cu cuvântul cheie static într-o clasă, dar în afara unei metode, constructor sau un bloc.
    - ar exista doar o copie a fiecărei variabile de clasă pe clasă, indiferent de câte obiecte sunt create din ea.
    - sunt stocate în memoria statică. Este rar să se utilizeze variabile statice, altele decât cele declarate finale și utilizate fie ca constante publice, fie ca constante private.

Tipuri de date

Pe baza tipurilor de date ale unei variabile, sistemul de operare aloca memorie si decide ce anume poate fi salvat in memoria rezervata. De aici, prin asocierea diferitelor tipuri de date variabilelor, putem salva integer, decimal, sau caractere in aceste variabile.

Sunt doua tipuri de date in Java:

- date primitive
- date de tip referință

Tipurile de date primitive permit stocarea unor valori. Aceste tipuri de date sunt predefinite de limbaj si asociate unui cuvant. Java suportă opt tipuri:

- *short* - este un număr ce cuprinde valori între -32,768 și 32,767. Se utilizează în general pentru stocarea numerelor care nu pot atinge o valoare foarte mare.
    - utilizează 2 octeți de spațiu de stocare
    - valoarea default este 0
- *int* - este un număr întreg ce cuprinde valori între -2.147.483.648 și 2.147.483.647. Int este cel mai des utilizat tip pentru stocarea numerelor.
    - utilizează 4 octeți de spațiu de stocare

    - valoarea default este 0
- *long* - este un număr întreg între 9.223.372.036.854.775.808 și 9.223.372.036.854.775.807. Este utilizat pentru numere foarte mari, care depășesc aria de acoperire a tipului int.
    - utilizează 8 octeți de spațiu de stocare
    - valoarea default este 0L
- *float* - reprezintă numere raționale in virgula mobila. Tipul float are o precizie de pană la 7 cifre după virgulă, tot ce depășește această dimensiune va fii rotunjit.
    - utilizează 4 octeți de spațiu de stocare
    - valoarea default este 0.0f
- *double* - reprezintă numere raționale in virgula mobila. Tipul double este mai precis decât float, având o precizie de până la 15 cifre după virgulă.
    - utilizează 8 octeți de spațiu de stocare
    - valoarea default este 0.0d
- *char* - este folosit pentru a stoca caractere unice.
    - utilizează 2 octeți de spațiu de stocare
    - are o valoare minima de '\u0000'(0) si o valoare maxima '\uffff'(65)
- *boolean* - este un tip special de date care poate conține doar două valori: adevărat (true) și fals (false). Valoarea default pentru boolean este false. Acesta este utilizat în mod obișnuit în declarațiile de flux de control.

Tipurile de date de referință sunt create prin constructorii unei clase. Acestea nu stochează date reale, în schimb, stochează o referință la date. Nu spune compilatorului care este valoarea datelor, ci îi spune compilatorului unde să găsească datele reale. Valoarea implicită a tipurilor de referință este nul (înseamnă că variabila nu stochează nicio adresă).

Modificatori de acces

Precum și în alte limbaje, este posibil sa modificam clase, metode, variabile, dar și altele folosind modificatori. Exista doua tipuri de modificatori:

- *Modificatori de access*:
    - *Public*: vizibil peste tot.
    - *Private*: vizibil doar în interiorul clasei respective.

- *Protected*: vizibil in pachet si in toate subclasele.
- *Default*: este vizibil in întregul pachet.

- *Non-access modificatori:*
  - *Final*: pentru a finaliza implementarea unei clase, metode si variabile.
  - *Abstract*: pentru crearea de clase abstracte si metode.
  - *Static*: pentru crearea metodelor unei clase si a variabilelor.

Clase și obiecte

Programarea orientată pe obiecte (POO) reprezintă o încercare de a face programele să modeleze mai îndeaproape modul în care oamenii gândesc. În centrul programării obiectivate găsim obiecte – entități care au comportamente, care dețin informații și care pot interacționa unul cu celălalt. Programarea constă în proiectarea unui set de obiecte care modelează cumva problema. Obiectele software din program pot reprezenta entități reale sau abstracte din domeniul problemei. Acest lucru ar trebui să facă designul programului mai natural și, prin urmare, mai ușor de obținut corect și mai ușor de înțeles.

Ne putem gândi la un obiect în termeni standard de programare ca la un set de variabile împreună cu un set de instrucțiuni pentru manipularea acestor variabile. Obiectele sunt create din șabloane cunoscute sub numele de clase. O clasă poate fii asociată cu un plan al unei clădiri. Un obiect este clădirea reală pe care o construim pe baza planului.

O clasa poate conține:

- *Câmpuri* – sunt variabile care sunt declarate în interiorul unei clase. Ca oricare alte variabile, acestea sunt folosite pentru a stoca date. Câmpurile unei clase folosesc întotdeauna modificatorul de acces: *private*.
- *Constructori* - sunt blocuri de cod (similar unei metode) care sunt folosite pentru a crea un obiect din șablonul clasei. Acesta are întotdeauna același nume ca și clasa și este utilizat în mod obișnuit pentru a inițializa domeniile clasei.
- Metode - este un bloc de cod care efectuează o anumită sarcină.

Fundamentele POO

Programarea orientată pe obiecte se clădește pe patru mari fundamente:

- Încapsulare
- Abstractizare
- Moștenire
- Polimorfism

Încapsularea este mecanismul care leagă codul și datele pe care le manipulează și le menține în siguranță împotriva interferențelor externe și a utilizării incorecte. În Java, baza încapsulării este clasa.

Există mecanisme pentru ascunderea complexității implementării în cadrul clasei. Fiecare metodă sau variabilă dintr-o clasă poate fi marcată ca fiind privată sau publică. Interfața publică a unei clase reprezintă tot ceea ce utilizatorii externi ai clasei trebuie să știe sau ar putea să știe. Metodele și datele private pot fi accesate numai prin cod care este membru al clasei. Prin urmare, orice alt cod care nu este membru al clasei nu poate accesa o metodă privată sau o variabilă. Deoarece membrii privați ai unei clase pot fi accesați numai de alte părți ale programului prin metodele publice ale clasei, ne putem asigura că nu au loc acțiuni necorespunzătoare.

Abstractizarea este unul dintre conceptele cheie ale limbajelor de programare orientate pe obiecte. Scopul său principal este de a gestiona complexitatea prin ascunderea detaliilor inutile de la utilizator. Acest lucru permite utilizatorului să implementeze o logică mai complexă pe lângă abstractizarea furnizată, fără a înțelege toată complexitatea ascunsă.

De exemplu, oamenii nu se gândesc la o mașină ca la un set de zeci de mii de piese individuale. Ei se gândesc la el ca la un obiect bine definit, cu propriul său comportament unic. Această abstractizare permite oamenilor să folosească o mașină pentru a conduce la locația dorită, fără a-și face griji cu privire la complexitatea pieselor care formează mașina. Acestea pot ignora detaliile despre modul în care funcționează motorul, transmisia și sistemele de frânare. În schimb, ei sunt liberi să utilizeze obiectul ca întreg.

O modalitate puternică de a gestiona abstractizarea este prin utilizarea clasificărilor ierarhice. Acest lucru ne permite să stratificăm sistemele complexe, rupându-le în bucăți mai ușor de gestionat. Abstracțiile ierarhice ale sistemelor complexe pot fi aplicate și programelor de calculator.

Datele dintr-un program tradițional orientat spre proces pot fi transformate prin abstractizare în obiecte componente. O secvență de pași de proces poate deveni o colecție de mesaje între aceste obiecte. Astfel, fiecare dintre aceste obiecte își descrie propriul comportament unic. Putem trata aceste obiecte ca pe niște entități concrete care răspund mesajelor care le spun să facă ceva.

Moștenirea se referă la faptul că o clasă poate moșteni o parte sau toată structura și comportamentul său de la o altă clasă. Clasa care moștenește este considerată o subclasă a clasei de la care moștenește. Dacă clasa B este o subclasă a, spunem, de asemenea, că clasa A este o superclasă a clasei B. O subclasă se poate adăuga la structura și comportamentul pe care le moștenește. De asemenea, poate înlocui sau modifica comportamentul moștenit (deși nu structura moștenită).

Moștenirea este o modalitate puternică de a realiza reutilizarea codului, dar nu este întotdeauna cel mai bun instrument pentru lucrare. Folosit necorespunzător, duce la software fragil. Este sigur să utilizați moștenirea într-un pachet, în cazul în care implementările subclasei și superclasei sunt sub controlul acelorași programatori. De asemenea, este sigur să se utilizeze moștenirea atunci când se extind clasele special concepute și documentate pentru extindere. Cu toate acestea, moștenirea din clase creeate de noi peste granițele pachetelor este periculoasă.

Acesta modalitate se folosește în general pentru a grupa caracteristicele comune a mai multor clase. De exemplu, atat un câine cât și o pisică sunt animale și au caracteristici comune de animale, așa că ambele pot moștenii o clasă animal.

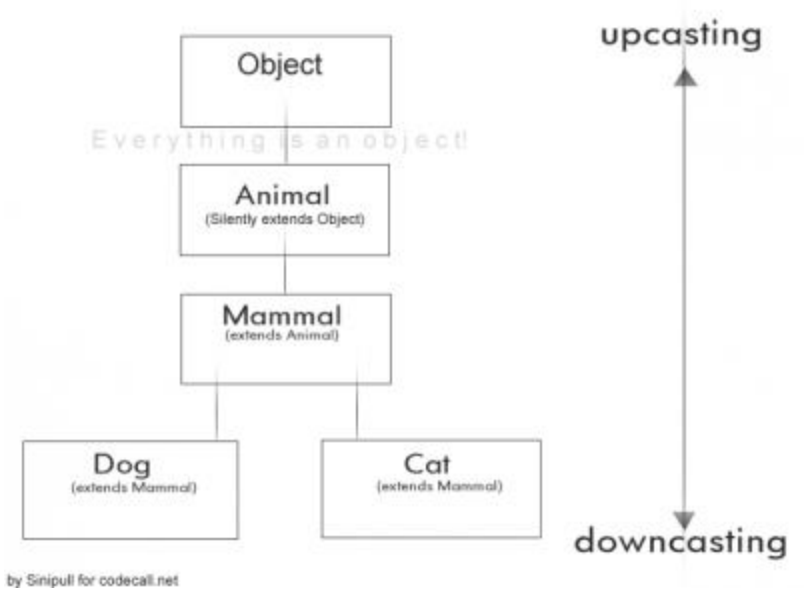

Fig. 4: Schemă a utilizării moștenirii

Atunci când o clasă este derivată dintr-o clasă care este, de asemenea, derivată dintr-o altă clasă, adică o clasă care are mai mult de o clasă părinte, dar la niveluri diferite, un astfel de tip de moștenire se numește *moștenire pe mai multe niveluri*.(fig. 4 : Animal - Mammal - Dog).Când o clasă are mai mult de o clasă copil (subclasă), atunci un astfel de tip de moștenire este cunoscut sub numele de *moștenire ierarhică*. (fig. 4 : Mammal - Dog, Mammal - Cat).

Moștenirea hibridă este o combinație de moștenire multiplă și moștenire pe mai multe niveluri. Deoarece moștenirea multiplă nu este acceptată în Java, deoarece duce la ambiguitate, acest tip de moștenire poate fi obținut numai prin utilizarea interfețelor.

Polimorfismul este o caracteristică care permite o interfață să fie utilizată pentru o clasă generală de acțiuni. Acțiunea specifică este determinată de natura exactă a situației. O metodă este polimorfică dacă acțiunea efectuată de metoda depinde de tipul real al obiectului la care se aplică metoda. Polimorfismul este una dintre principalele caracteristici distinctive ale programei orientate pe obiecte.

Polimorfismul înseamnă a lua mai multe forme, unde „poli” înseamnă mai multe și „morf” înseamnă forme. Este capacitatea unei variabile, a unei funcții sau a unui obiect de a lua mai multe forme. Cu alte cuvinte, polimorfismul ne permite să definim o interfață sau o metodă și să avem mai multe implementări.

De exemplu, simțul mirosului unui câine este polimorfic. Dacă câinele miroase o pisică, va lătra și va alerga după ea. Dacă câinele își miroase mâncarea, va saliva și va alerga la castronul său. Același simț al mirosului este la lucru în ambele situații. Diferența este ceea ce este mirosit, adică tipul de date care sunt operate de nasul câinelui. Luați în considerare o stivă. S-ar putea să avem un program care necesită trei tipuri de stive. O stivă este utilizată pentru valori întregi, una pentru valori cu virgulă mobilă și una pentru caractere. Algoritmul care implementează fiecare stivă este același, chiar dacă datele stocate diferă.

Polimorfismul este de două tipuri:

- *Run time polimorfism* - se referă la un proces în care un apel la o metodă suprascrisă este rezolvat la runtime, mai degrabă decât la compilare. Metoda de suprascriere este un exemplu de polimorfism în runtime.
- *Compile time polimorfism* - se referă la un proces în care apelul la o metodă supraîncărcată este rezolvat la timpul de compilare, mai degrabă decât la timpul de rulare. Supraîncărcarea metodei este un exemplu de compile time.

2.1.2. Spring Boot Framework

Spring Boot este o extensie a framework-ului Spring și se bazează pe el. Spring Boot simplifică și accelerează dezvoltarea aplicațiilor Java utilizând Spring prin furnizarea de configurație implicită, eliminând astfel multe aspecte repetitive și necesitatea unei configurații manuale extinse.

Spring Boot oferă o platformă bună pentru dezvoltatorii Java pentru a dezvolta o aplicație autonomă și de producție care poate fii rulată. Principalele avantaje sunt faptul că este ușor de înțeles și de dezvoltat aplicații ,crește productivitatea și reduce timpul de dezvoltare.

Spring Boot este foarte ales datorită caracteristicilor și beneficiilor pe care le oferă:

- oferă o modalitate flexibilă de a configura bean-urile Java, configurații XML și tranzacții cu baze de date.
- oferă o procesare puternică a loturilor și gestionează endpointurile REST.

- totul este configurat automat; nu sunt necesare configurații manuale.
- oferă o aplicație spring bazată pe adnotari care facilitează gestionarea dependențelor
- include un container de servlet încorporat(Embedded Servlet Container).

Spring Boot configurează automat aplicația pe baza dependențelor pe care le adaugă dezvoltatorul în proiect, utilizând adnotarea @EnableAutoConfiguration. De exemplu, dacă baza de date MySQL se află pe classspath, dar nu ați configurat nicio conexiune la baza de date, atunci Spring Boot configurează automat o bază de date în memorie. Punctul de intrare al aplicației de pornire spring boot este clasa care conține adnotarea @SpringBootApplication și metoda principală. Spring Boot scanează automat toate componentele incluse în proiect utilizând adnotarea @ComponentScan.

Pentru a crea o astfel de aplicație, sistemul trebuie să aibă instalate și configurate minim următoarele versiuni:

- Java 7
- Maven 3.2 (sau Gradle 2.5)

Dependințe

O dependență este pur și simplu un fișier JAR utilizat de o aplicație Java. Pe baza fișierului unui fișier configrat custom, se va descărca și adăuga fișierul JAR în calea noastră Java. Java va putea apoi să găsească și să utilizeze clasele din fișierul JAR. Aceste fișiere sunt librării oferite de Spring Boot sau proiecte Java externe.

Dependențele, împreună cu pluginurile și celelalte configurări ale Spring boot sunt adăugate într-un fișier numit pom.xml (dacă se utilizează maven) sau build.gradle (pentru gradle).

Echipa Spring Boot oferă o listă de dependențe pentru a sprijini versiunea Spring Boot pentru fiecare nouă lansare. Nu este necesar să furnizați o versiune pentru dependențe în fișierul de configurare. Spring Boot configurează automat versiunea de dependențe pe baza versiunii. Atunci când versiunea Spring Boot este schimbată in cadrul proiectului, dependențele se vor actualiza automat.

Dependențele minime pe care un proiect spring boot le are sunt spring-boot-starter și spring-boot-starter-test. Dependințele aplicației pot fii adăugate folosind unul din cele două instrumente compatibile cu Spring Boot :

- Maven

```
<dependencies>
   <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter</artifactId>
   </dependency>

   <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
   </dependency>
</dependencies>
```

- Gradle

```
dependencies {
      implementation("org.springframework.boot:spring-boot-starter")
      testImplementation("org.springframework.boot:spring-boot-
starter-test")
}
```

Fișierul de proprietăți

Fișierele de proprietăți sunt utilizate pentru a păstra toate proprietățile aplcației într-un singur fișier pentru a rula aplicația într-un mediu diferit. În Spring Boot, proprietățile sunt păstrate în fișierul application.properties sub classspath. Fișierul application.properties se află în directorul src/Main/Resources. Codul pentru fișierul application.properties este scris astfel:

```
server.port=9090
spring.application.name=travelWithMe
```

în codul afișat se specifică numele aplicației și portul cu care se pornește.

Spring Boot acceptă, de asemenea, și configurații bazate pe YAML pentru a rula aplicația. În loc de application.properties, putem folosi fișierul application.yml. Acest fișier YAML ar trebui să fie păstrat în același loc. Fișierul Application.yml are următoarea structură:

```
spring:
  application:
    name: demoservice
server:
  port: 9090
```

2.1.3. Hibernate Framework

Hibernate ORM este un framework de mapare obiect-relație, facilitând conversia unui model de domeniu orientat pe obiecte într-o bază de date relațională tradițională. Hibernate rezolvă problemele de nepotrivire a impedanței relaționale obiect prin înlocuirea accesărilor directe ale bazei de date legate de persistență cu funcții de manipulare a obiectelor de nivel înalt.

Hibernate este unul dintre cele mai populare framework-uri ORM din lumea Java. Permite dezvoltatorilor să mapeze structurile obiectelor din clasele Java normale la structura relațională a unei baze de date. Cu ajutorul unui cadru ORM, munca de a stoca date din instanțe de obiecte în memorie într-un magazin de date persistent și de a le încărca înapoi în aceeași structură de obiect devine semnificativ mai ușoară.

În același timp, soluțiile ORM, cum ar fi Hibernate, își propun să facă abstracție de produsul specific utilizat pentru stocarea datelor. Acest lucru permite utilizarea aceluiași cod Java cu diferite produse de baze de date, fără a fi nevoie să scrieți cod care să se ocupe de diferențele subtile dintre produsele acceptate.

Hibernate este, de asemenea, un furnizor JPA, ceea ce înseamnă că implementează Java Persistence API (JPA). JPA este o specificație independentă de furnizor pentru maparea obiectelor Java la tabelele bazelor de date relaționale.

Hibernatul constă din trei componente diferite:

- *Entități* - reprezintă clasele care sunt mapate de Hibernate la tabelele unui sistem de baze de date relaționale (simple obiecte Java).

- *Object-relational metadata* - reprezintă Informațiile despre cum sa mapezi entitățile in baza de date relațională. Ele sunt fie furnizate de adnotări (din Java 1,5), fie de fișiere de configurare bazate pe XML. Informațiile din aceste fișiere sunt utilizate în timpul rulării pentru a efectua maparea la depozitul de date și înapoi la obiectele Java.
- *Hibernate Query Language (HQL)* - specifica interogările trimise la baza de date, care nu trebuie să fie formulate în SQL nativ. Deoarece aceste interogări sunt traduse în timpul rulării în dialectul utilizat în prezent al produsului ales, interogările formulate în HQL sunt independente de dialectul SQL al unui anumit furnizor.

2.3. Frontend

2.2.1. HTML

Documentul HTML este pur și simplu un fișier text care conține informațiile pe care doriți să le publicați și instrucțiunile de marcare corespunzătoare care indică modul în care browser-ul ar trebui să structureze sau să prezinte documentul. Elementele de marcare sunt alcătuite dintr-o etichetă de pornire, cum ar fi < > și, de obicei, deși nu întotdeauna, o etichetă finală, care este indicată printr-o linie din etichetă, cum ar fi < / >. Perechea de etichete ar trebui să cuprindă complet orice conținut care să fie afectat de element, inclusiv text și alte marcaje HTML. Există elemente de marcare, numite elemente goale, care nu conțin conținut, astfel încât nu au nevoie deloc de etichete apropiate sau, în cazul HTML, utilizați o schemă de identificare auto-închisă. De exemplu, pentru a insera o pauză de linie, utilizați o singură etichetă < br >, care reprezintă elementul gol, deoarece nu conține conținut și, prin urmare, nu are o etichetă apropiată corespunzătoare. Structura de baza a acestor documente arată așa:

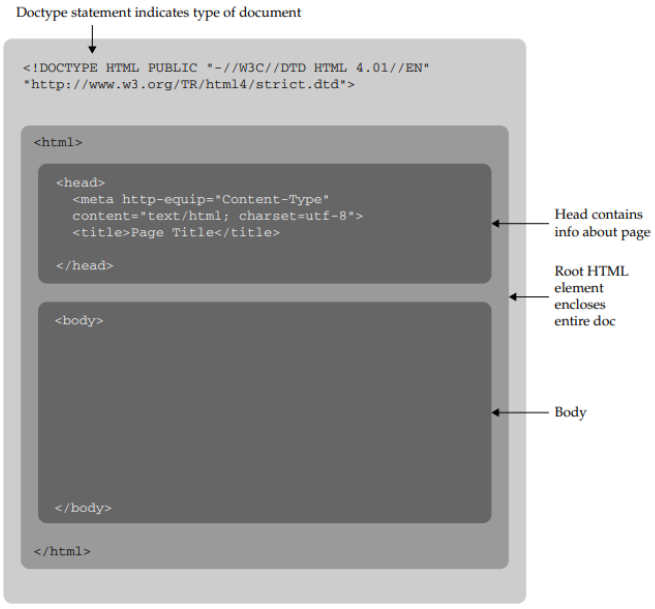

Fig. 5: Structură HTML

Documentele HTML5 pot conține un element antet, care este utilizat pentru a seta secțiunea antet a unui document și, astfel, conține adesea elementele de titlu standard h1 până la h6:

<header>
< h1 > Primul titlu< / h1 >
< h2 > Al doilea titlu </ h2 >
< / header >

În mod similar, un element de subsol este furnizat pentru autorii de documente pentru a defini conținutul subsolului unui document, care conține adesea informații de navigare, legale și de contact:

< footer >
< p >Exemplu de footer < / p >
< / footer >

Conținutul real care trebuie plasat într-un < footer > poate fi inclus în div, p sau alte elemente de bloc.

Există câteva etichete de bază care sunt utilizate în mod frecvent la construirea paginilor web. Acestea includ:

- <html>: Eticheta care definește începutul și sfârșitul unui document de tip HTML.

- <head>: Eticheta care conține informații despre document, precum titlul și legăturile către fișiere externe.
- <body>: Eticheta care conține conținutul vizibil al paginii, cum ar fi textul, imaginile și linkurile.
- <h1>, <h2>, <h3>...: Etichetele pentru titlurile pe diferite nivele (de la cel mai mare la cel mai mic).
- <p>: Eticheta pentru paragrafe de text.
- și <li>: Etichetele pentru liste neordonate.
- <a>: Eticheta pentru linkuri.
- <img>: Eticheta pentru imagini.

HTML oferă și etichete semantice care descriu semnificația conținutului. Acestea ajută la îmbunătățirea accesibilității și la optimizarea motoarelor de căutare. Exemple de etichete semantice includ <header>, <nav>, <main>, <section>, <article> și altele.

Deșii HTML este documentul principal care structurează și prezintă informațiile in browser, acesta este limitat. Pentru un aspect mai plăcut, HTML este folosit în combinație cu CSS.

2.2.2. CSS

Cascading Style Sheets este o tehnologie destul de veche în ceea ce privește Web-ul. Primele idei despre CSS au fost prezentate încă din 1994, iar până în decembrie 1996, specificația CSS1 a fost ratificată. Această versiune timpurie a CSS a fost parțial acceptată în browsere precum Internet Explorer 3 în grade diferite. În timp ce caracteristicile CSS1 erau cu mult superioare celor prezentate HTML cu etichetele și soluțiile sale < font >, absorbția a fost lentă. CSS1 a oferit multe caracteristici pentru a schimba granițele, marjele, fundalurile, culorile și o varietate de caracteristici ale textului, dar capacitatea mult solicitată de a poziționa direct obiectele a lipsit.

Ulterior a fost rulat în CSS2 care a fost lansat în mai 1998. În timp ce CSS2 a introdus multe caracteristici valoroase, inclusiv poziționarea, tipurile media pentru foi de stil, foi de stil fonic și multe altele, nu totul a fost implementat chiar și în cele mai moderne browsere. O revizuire a acestei specificații, CSS 2.1 lansată în 2007, a eliminat o serie de funcții neimplementate și a normalizat specificația la o viziune mai realistă a ceea ce fac de fapt browser-ele.

În timp ce viitorul este clar CSS3 cu multitudinea sa de module pentru abordarea culorilor, constrângerilor dispozitivului, redării limbilor străine, imprimării îmbunătățite și multe altele, este departe de a fi clar când se va lansa viitoarea versiune.

Informațiile despre stil pot fi incluse într-un document HTML folosind una dintre cele trei metode:

- Un fișier extern, fie importat, fie prin conectare.
- incorporare directa a unui stil la nivel de document în elementul principal al documentului.
- setarea unei reguli de stil folosind atributul de stil direct pe un element

Un alt mod de a utiliza stiluri la nivel de document, mai degrabă decât tastarea proprietăților direct într-o etichetă < style > este sa fie importate. O foaie de stil extern este încă referită, dar în acest caz, referința este similară cu o macro-comandă. Sintaxa pentru importarea unei foi de stil este @import, urmată de url-ul cuvântului cheie și adresa URL reală a foii de stil care trebuie inclusă și terminată cu punct și virgulă:

@import url (corerules.css)

Pentru a aplica un stil pe o gamă mai largă de elemente putem folosii selectori. Aceștia permit aplicarea de stiluri specifice unor elemente individuale, grupuri de elemente sau anumite condiții. Câteva exemple de selectori CSS comuni sunt:

- Selector de element:

Exemplu: p selectează toate elementele <p> (paragrafe) din pagină.

- Selector de clasă:

Exemplu: .clasa selectează toate elementele care au atributul class cu valoarea clasa.

- Selector de ID:

Exemplu: #id selectează elementul care are atributul id cu valoarea id.

- Selector de atribut:

Exemplu: [atribut] selectează elementele care au un anumit atribut.

- Selector de atribut cu valoare specifică:

Exemplu: [atribut="valoare"] selectează elementele care au un anumit atribut cu o valoare specifică.

- Selector de copil:

Exemplu: elementul părinte selectează elementul copil care este descendent direct al unui element părinte.

- Selector de urmaș:

Exemplu: elementul părinte selectează elementul urmaș care este direct copil al unui element părinte.

- Selector de pseudo-clasă:

Exemplu: element: pseudo-clasă selectează elementele în funcție de o stare sau un comportament specific.

- Selector de pseudo-element:

Exemplu: element::pseudo-element selectează părți specifice ale unui element.

- Selector de combinare:

Exemplu: selector1, selector2 selectează elementele care se potrivesc cu oricare dintre selecțiile specificate.

2.2.3. React JS

React este o bibliotecă JavaScript care își propune să simplifice dezvoltarea interfețelor vizuale. Dezvoltat la Facebook și lansat în 2013, acesta conduce unele dintre cele mai utilizate aplicații, alimentând Facebook și Instagram printre nenumărate alte aplicații.

Spre deosebire de alte framework-uri, React nu operează direct pe *modelul de obiect al documentului* (DOM) al browser-ului imediat, ci pe un DOM virtual. Adică, în loc să manipuleze documentul într-un browser după modificări ale datelor noastre (care pot fi destul de lente), rezolvă modificările pe un DOM om construit și rulat în întregime în memorie. După ce DOM-ul virtual a fost actualizat, React determină în mod inteligent ce modificări să facă în DOM-ul real al browser-ului. DOM-ul virtual există în întregime în memorie și este o reprezentare a DOM-ului browser-

ului web. Din acest motiv, noi, atunci când scriem o componentă React, nu scriem direct la DOM, ci scriem o componentă virtuală care reacționează se va transforma în DOM.

React în sine are un API foarte mic, și se bazează pe 4 concepte:

- Componente
- JSX
- State
- Props

Componente

Componentele se află în centrul tuturor aplicațiilor React. O componentă este un modul autonom care produce o anumită ieșire. Putem scrie elemente de interfață, cum ar fi un buton sau un câmp de intrare ca o componentă React. O componentă poate conține una sau mai multe componente în interiorul său.

Pentru a scrie aplicații React scriem componente React care corespund diferitelor elemente de interfață. Apoi organizăm aceste componente în interiorul componentelor de nivel superior care definesc structura aplicației noastre.

De exemplu, un formular poate consta din mai multe elemente de interfață, cum ar fi câmpuri de intrare, etichete sau butoane. Fiecare element din interiorul formei poate fi scris ca o componentă React. Apoi am scrie o componentă de nivel superior, componenta care reprezintă formularul în sine. Componenta de formular ar specifica structura formularului și ar include fiecare dintre aceste elemente de interfață în interiorul acestuia.

Componentele funcționale sunt funcții, nu obiecte; prin urmare, ele nu au un domeniu de aplicare. Deoarece sunt funcții simple, le folosim cât mai mult posibil în aplicațiile noastre. Poate veni un punct în care componenta funcțională nu este suficient de robustă și trebuie să ne întoarcem la utilizarea clasei dar, în general, cu cât sunt utilizate mai mult, cu atât mai bine. Componentele

funcționale sunt funcții care preiau proprietăți și returnează un element DOM. Componentele funcționale sunt o modalitate bună de a practica regulile programării funcționale. Acest lucru încurajează simplitatea și face ca codul de bază să fie extrem de vizibil.

Componentele funcționale păstrează arhitectura aplicației simplă, iar echipa React promite unele câștiguri de performanță prin utilizarea lor.

```
import React from "react";
import { Link } from "react-router-dom";
import styles from "../../../css/Presentation.module.css";

const Presentation = () => {
  return (
    <section className={styles.presentation}>
      <section className={styles.introduction}>
        <article className={styles.intro_text}>
          <h1>The world is just one step from here</h1>
          <p>
          Embark on a journey of discovery and adventure with like-minded travel enthusiasts.
          Create unforgettable memories as you explore new destinations and share your passions with new people.
          </p>
        </article>

        <article className={styles.buttons}>
          <button className={styles.btn_first} method="POST">
            <Link to="signup" className={styles.btn_first}>
              Sign up
            </Link>
          </button>
        </article>

      </section>
    </section>
  );
};

export default Presentation;
```

JSX

JSX[2] este o extensie a limbajului JavaScript utilizată în biblioteca React pentru a defini structura și aspectul componentelor UI. JSX oferă o sintaxă similară cu HTML[3], care permite

[2] JavaScript XML
[3] HyperText Markup Language

programatorilor să scrie cod într-un format mai intuitiv și ușor de înțeles. În plus, permite utilizarea expresiilor JavaScript prin intermediul acoladelor {} pentru a include valori dinamice.

JSX permite definirea componentelor reutilizabile în React. O componentă JSX este o funcție sau o clasă care returnează un element JSX sau o ierarhie de elemente JSX. Aceste componente pot fi apoi utilizate în alte componente sau în aplicația principală. De asemenea, se poate utiliza orice expresie JavaScript validă în interiorul acoladelor {}. Aceasta înseamnă că se pot efectua operații, accesa variabile, itera prin bucle și se pot utiliza condiții în JSX.

În JSX, stilurile pot fi definite utilizând atributele HTML standard precum style. Cu toate acestea, stilurile în React sunt de obicei definite utilizând CSS în fișiere separate sau utilizând biblioteci precum Styled Components sau CSS Modules.

Codul JSX nu poate fi executat direct de browser. Prin urmare, este necesară o etapă de transformare a codului JSX în JavaScript pur, folosind un compilator sau un instrument de construire precum Babel. Acest proces este adesea integrat în fluxul de lucru React cu ajutorul instrumentelor de dezvoltare precum Create React App.

State

Fiecare componentă React poate avea propriul său state (stare). State este setul de date gestionat de componentă. În React sau în orice alt cadru / bibliotecă bazat pe componente, toate aplicațiile se bazează și utilizează foarte mult starea componentelor.Starea se gestionează folosind utilitatea useState oferită de React și se importează astfel:

```
import { useState } from "react";
```

Apelând useState ( ), se va returna nouă variabilă de stare, o funcție care poate fi apelată mai târziu. UseState ( ) acceptă valoarea inițială a elementului de stare și returnează un tablou care conține variabila de stare și funcția pe care se apelează pentru a modifica starea.

Exemplu: const [count, setCount] = useState (0)

Valoarea unei variabile de stare nu poate fi modificată direct. Trebuie să apelăm la funcția de modificator. În caz contrar, componenta React nu își va actualiza UI pentru a reflecta

modificările datelor. Apelarea modificatorului este modul în care React știe că starea componentei s-a schimbat.

Props

Termenul "props" este prescurtarea de la "properties" și face referință la datele pe care le pasăm componentelor în React. Props sunt utilizate pentru a transfera și a partaja informații între componente.

În React, componentele sunt considerate funcții pure care iau un set de proprietăți (props) și returnează o reprezentare vizuală bazată pe acestea. Props sunt pasate de la componenta părinte către componenta copil și sunt imutabile, adică nu pot fi modificate în interiorul componentei copil. Pentru a utiliza props într-o componentă React, acestea sunt primite ca parametri în funcția componentei și sunt accesate folosind notația cu punct.

Exemplu de cod care folosește props:

```
function WelcomeMessage({myprop}){
    return <p>{myprop}</p>
}
```

2.3. Baze de date

O bază de date este o modalitate eficientă de stocare și organizare a datelor.Bazele de date au devenit esențiale în lumea digitală de astăzi. Este folosit în diverse domenii, cum ar fi afaceri, cercetare, sănătate și tehnologie. Bazele de date permit stocarea, accesul și gestionarea eficientă a unor cantități mari de date, susținând dezvoltarea de aplicații complexe și analiza datelor.

O bază de date constă din tabele, care sunt structuri în care sunt stocate datele. Un tabel este format din rânduri (înregistrări) și coloane (câmpuri). Coloanele definesc tipurile de date și limitele pe care le poate avea o anumită coloană.

Există două tipuri principale de baze de date: relaționale și non-relaționale.

Baze de date relaționale

Bazele de date relaționale folosesc un model relațional pentru a organiza datele în tabele. Datele sunt organizate în rânduri (înregistrări) și coloane (câmpuri), iar tabelele pot fi legate prin chei primare și externe. Limbajul pe care ele îl folosesc SQL (Structured Query Language) pentru a interoga și a manipula datele. SQL oferă un set standardizat de instrucțiuni pentru gestionarea bazelor de date relaționale.

Datele din bazele de date relaționale trebuie să urmeze tipare predefinite și să fie structurate conform schemelor predefinite. Acest lucru asigură consistența și integritatea datelor. De asemenea, acestea acceptă tranzacții ACID (Atomicity, Consistency, Isolation, Durability) pentru a se asigura că modificările datelor sunt gestionate într-un mod sigur și consecvent.

MySQL, PostgreSQL, Oracle și Microsoft SQL Server sunt exemple de sisteme de gestionare a bazelor de date relaționale.

Baze de date non-relaționale (NoSQL):

Bazele de date non-relaționale au o structură flexibilă care vă permite să stocați și să vă gestionați datele mai liber. Sunt disponibile diferite modele de memorie. De exemplu: documente, diagrame, valori cheie sau coloane late. Ele sunt proiectate să extindă, gestionând cantități mari de date și crescând traficul fără a sacrifica performanța.

Bazele de date non-relaționale nu necesită un cadru rigid sau un model de date predefinit. Acest lucru vă oferă flexibilitate atunci când lucrați cu date structurate diferit. De asemenea, ele facilitează replicarea și distribuirea datelor pe mai multe servere sau noduri, contribuind la redundanța și disponibilitatea datelor.

MongoDB, Cassandra, Redis și Neo4j sunt exemple de sisteme de gestionare a bazelor de date non-relaționale.

2.3.1.MySql

SQL (Structured Query Language) este un limbaj standard pentru gestionarea bazelor de date. El permite creearea, interogarea și manipularea de date din baza de date. Instrucțiunile SQL permit preluarea de informații specifice, actualizarea datelor, inserarea de înregistrări noi și ștergerea înregistrărilor existente.

MySQL este un sistem de gestionare a bazelor de date gratuit și open source. Oferă o gamă largă de funcții, inclusiv stocarea și gestionarea datelor, interogări complexe, securitate avansată și multe altele. MySQL este compatibil cu multe limbaje de programare și poate fi utilizat în multe aplicații și scenarii diferite. MySQL poate fi instalat și configurat pe diverse platforme, cum ar fi Windows, Linux și macOS. Există mai multe modalități de a instala MySQL, dar una dintre cele mai comune este prin pachetul XAMPP, care configurează automat toate componentele necesare pentru dezvoltarea aplicațiilor web.

Se pot creea baze de date și tabele în MySQL folosind instrucțiuni SQL. Acestea pot fi personalizate pentru a satisface nevoile specifice ale aplicației. De asemenea, se pot adăuga constrângeri, cum ar fi cheile primare și cheile externe, pentru a asigura integritatea datelor și relațiile dintre tabele. MySQL oferă o mare varietate de instrucțiuni de manipulare a datelor, toate într-un singur loc.

MySql oferă avantaje precum:

Fiabilitate și stabilitate: MySQL este un sistem de gestionare a bazelor de date foarte fiabil și stabil. Este utilizat pe scară largă în industrie de mulți ani și a fost testat și validat într-o varietate de scenarii. Acesta oferă un nivel ridicat de durabilitate și integritate a datelor.

Performanță și scalabilitate: MySQL este cunoscut pentru performanța sa excelentă și capacitatea de a scala pentru a gestiona cerințele crescânde. Acesta utilizează tehnici de optimizare a interogărilor și gestionează eficient memorie cache pentru a oferi un timp de răspuns rapid și a face față unor volume mari de date.

Suport extins pentru limbaje de programare: MySQL oferă drivere și suport pentru majoritatea limbajelor de programare populare, cum ar fi Java, PHP, Python, C#, și altele. Acest lucru facilitează integrarea cu aplicațiile existente și dezvoltarea de noi aplicații în diverse medii.

Comunitate activă și suport: MySQL are o comunitate mare și activă de utilizatori și dezvoltatori. Există forumuri, grupuri de discuții și alte resurse online unde se pot obține răspunsuri la întrebări, sfaturi și suport. De asemenea, există documentație bogată și tutoriale care pot fi utile în procesul de dezvoltare și administrare a bazelor de date MySQL.

Costuri: MySQL este un sistem de gestionare a bazelor de date gratuit și open-source, ceea ce înseamnă că nu necesită licențe costisitoare. Aceasta poate reduce semnificativ costurile pentru dezvoltarea și implementarea proiectelor.

2.3.2. Liquibase

Liquibase este o unealtă open-source pentru gestionarea schimbărilor structurale în bazele de date. Este conceput pentru a facilita managementul versiunilor și migrarea bazelor de date într-un mod controlat și reproducibil. Liquibase utilizează un format XML, YAML sau JSON pentru a defini și a urmări modificările structurale într-un mod consistent.

Rolul principal al Liquibase este de a asigura coerența și sincronizarea structurii bazei de date în diferite medii și etape de dezvoltare. Prin definirea schimbărilor structurale într-un format declarativ, Liquibase permite dezvoltatorilor să descrie în mod explicit modificările dorite, cum ar fi adăugarea sau ștergerea de tabele, coloane sau indecși, modificarea tipurilor de date, adăugarea constrângerilor, etc.

Pentru a folosi Liquibase, este necesară definirea un fișier de configurare care specifică detaliile de conectare la baza de date și calea către fișierul de schimbări. Fișierul de schimbări conține modificările structurale definite într-un format specific (XML, YAML sau JSON), împreună cu etichetele de versiune asociate cu fiecare schimbare.

Liquibase oferă mai multe modalități de a defini schimbările structurale:

- XML: Formatul XML este cel mai utilizat în Liquibase. Fiecare schimbare este definită ca un element XML într-un fișier. Există diferite tipuri de schimbări disponibile, cum ar fi crearea tabelelor, adăugarea coloanelor, definirea constrângerilor și multe altele.

- JSON: Liquibase oferă și suport pentru formatul JSON. Acesta are o structură similară cu celelalte formate și permite definirea schimbărilor structurale într-un stil JSON.

- YAML: Liquibase acceptă și formatul YAML pentru definirea schimbărilor structurale. Este un format mai concis și ușor de citit. Exemplu de cod YAML folosit de asemenea și în proiectul dat este acesta:

```yaml
databaseChangeLog:
  - changeSet:
      id: 202304241802_changelog#1
      author: Georgiana Voicea
      preConditions:
        - onFail: MARK_RAN
        - and:
            not:
              - columnExists:
                  tableName: avatars
                  columnName: avatar_priority
      changes:
        - addColumn:
            tableName: avatars
            columns:
              - column:
                  name: avatar_priority
                  type: INT
                  afterColumn: avatar_name
                  constraints:
                    nullable: false
```

Un exemplu de utilizare a Liquibase ar fi definirea un set inițial de schimbări structurale pentru a crea baza de date și tabelele necesare. Apoi, pe măsură ce evoluează aplicația și sunt necesare modificări adiționale, se pot adăuga schimbări suplimentare în fișierele de schimbări și

Liquibase va gestiona aplicarea acestora în mod corespunzător în baza de date. Liquibase va detecta automat schimbările noi și va aplica aceste modificări în baza de date.

Liquibase adaugă două tabele speciale în baza de date pentru a-și gestiona schimbările structurale și pentru a păstra o evidență a acestora:

Tabela DATABASECHANGELOG – are rolul de a stoca informații despre schimbările structurale aplicate în baza de date. Ea servește ca un jurnal de audit pentru schimbările efectuate și asigură că fiecare schimbare este aplicată doar o singură dată.

Tabela DATABASECHANGELOGLOCK – are rolul de a asigura integritatea schimbărilor structurale în mediul multi-utilizator. Ea servește ca un mecanism de blocare pentru a preveni conflictul între mai multe execuții simultane ale Liquibase pe aceeași bază de date.

Aceste două tabele sunt esențiale pentru funcționarea corectă a Liquibase și asigură că schimbările structurale sunt aplicate într-un mod controlat și consistent. Ele permit Liquibase să țină evidența schimbărilor aplicate și să prevină aplicarea repetată sau conflictuală a acestora.

În afară de gestionarea schimbărilor structurale în bazele de date, Liquibase oferă și alte funcții utile care permit un control mai precis asupra evoluției schemei bazei de date. Unele dintre cele mai utilizate sunt:

- *Rollback* (revenire): Funcția de rollback în Liquibase permite anularea sau revocarea schimbărilor structurale aplicate anterior într-o bază de date. Acest lucru este util în situații în care o modificare a fost aplicată greșit sau este necesară o revenire la o versiune anterioară a schemei bazei de date. Pentru a efectua un rollback, este necesar să se specifice o etichetă de versiune sau un număr de schimbare la care se dorește revenirea. Liquibase va aplica schimbările de rollback corespunzătoare pentru a aduce schema bazei de date în starea anterioară.

- *Update* (actualizare): Funcția de update în Liquibase permite aplicarea schimbărilor structurale în baza de date. Atunci când se rulează comanda de update, Liquibase verifică schimbările nereplicate și le aplică în baza de date în ordinea specificată în fișierele de schimbări. Comanda de update poate fi rulată pentru a aduce o bază de date la o anumită etichetă de versiune specificată sau la cea mai recentă versiune.

- *Diff*: această funcție permite generarea automată a unui diferențial între două versiuni ale bazei de date. Asta înseamnă că se poate compara schema curentă a bazei de date cu o versiune anterioară și se poate obține un raport detaliat al diferențelor dintre ele. Acest lucru facilitează identificarea și înțelegerea modificărilor făcute în structura bazei de date.

- *Tagging*: Liquibase permite marcarea sau etichetarea versiunilor specifice ale bazei de date utilizând taguri. Această funcție oferă un mecanism simplu și eficient pentru a identifica și reveni la anumite puncte de referință în evoluția schemei bazei de date. Tagurile pot fi utilizate pentru a marca versiuni stabile sau importante și pentru a facilita procesul de rollback sau reutilizare a unor versiuni specifice.

De exemplu, dacă există deja un set de schimbări structurale și s-a rulat deja comanda de update pentru a aplica aceste schimbări, Liquibase va ține evidența schimbărilor care au fost aplicate în tabela DATABASECHANGELOG. Atunci când se rulează comanda de update din nou, Liquibase va verifica această tabelă și va aplica doar schimbările noi care nu au fost aplicate anterior.

Capitolul 3. Proiectarea aplicației

3.1. Scop și obiective

Aplicația pentru găsirea partenerilor de călătorie prin alocarea de interese are ca scop facilitarea conectării persoanelor care doresc să călătorească împreună, având interese comune. Aceasta oferă utilizatorilor o platformă interactivă și ușor de utilizat pentru a găsi parteneri de călătorie potriviți, cu care să împărtășească experiențe și aventuri.

Prin intermediul aplicației, utilizatorii pot crea profiluri personale și pot specifica interesele, destinațiile preferate, activitățile pe care doresc să le desfășoare în timpul călătoriei și alte preferințe relevante.

Prin intermediul aplicației utilizatorii au acces la un forum comun unde pot pune întrebari care să-i ajute în călătoriile lor.

Algoritmul aplicației folosește aceste informații, precum interesele utilizatorilor, pentru a potrivi utilizatorii cu potențiali parteneri de călătorie, bazându-se pe similarități și compatibilitate.

Interacțiunea între utilizatori este ușurată prin intermediul emailurilor folosite pentru a începe conversația. Deoarece orice utilizator are email, acesta este platforma de început, urmând să aleagă împreună modul de comunicare ulterior. Utilizatorii pot iniția conversații și pot discuta detaliile călătoriilor propuse, precum itinerariul, bugetul, preferințele de cazare și orice alte aspecte importante.

Aplicația pentru găsirea partenerilor de călătorie prin alocarea de interese este disponibilă pe platforme web, permițând astfel utilizatorilor să acceseze și să utilizeze aplicația oriunde și oricând. Prin intermediul acestei aplicații, călătorii pasionați pot descoperi parteneri de aventură care împărtășesc aceleași interese și dorințe de explorare, ceea ce duce la experiențe de călătorie mai pline de satisfacții și memorabile.

3.2. Proiectarea aplicației

Proiectarea unei aplicații implică o abordare riguroasă și o planificare atentă pentru a asigura dezvoltarea unei soluții eficiente și funcționale.

Structura bazei de date

Un aspect important al proiectării aplicației este proiectarea bazei de date. Diagrama entitate-relație (DER) din fig.6 a permis vizualizarea entităților principale și relațiile dintre acestea. Am definit tabelele, coloanele, cheile primare și străine pentru a asigura o structură coerentă a bazei de date.

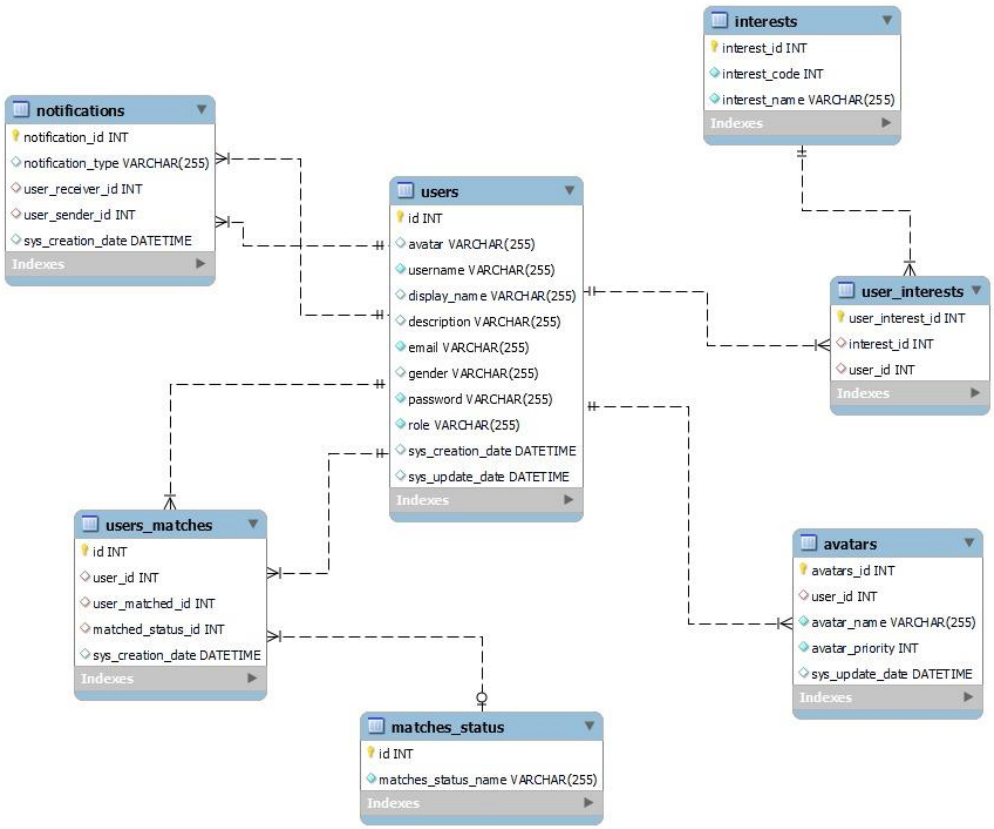

Fig. 6: Diagrama entitate-relație

Prima necesitate vizibilă este de un tabel care stochează informații generale despre utilizator, precum și câteva informații tehnice. Un astfel de tabel a fost creat și numit 'users'. Acesta stochează codul personal al utilizatorului (ID-ul său), numele, numele care va fi folosit pentru autentificare, parola criptata si email utilizatorului, și, de asemenea, calea fizica către fișierul avatar

al utilizatorului de pe server. Tot aici, se scot și informații tehince ce țin despre când a fost creată această înregistrare de utilizator și când ea a fost modificata ultima dată.

Un tabel crucial pentru aplicația TravelWithMe este tabelul numit ‘users_matches’. În acest tabel se păstrează informația despre legăturile între useri. Odată ce un user a acceptat un alt utilizator sau l-a respins, informația despre aceasta va fi introdusa automat în tabelul cu statusul respectiv .

La fel de important este si tabelul ‘user_interests’. În acest tabel se păstrează datele despre interesele unui utilizator. Acest tabel este legat prin Foreign Key la tabelul ‘interests’ în care se păstrează denumirile tuturor interese și codurile lor.

Alt tabel important este tabelul ‘notifications’, în care se reține informația despre toate notificări trimise din aplicația.

Scheme UML

În cadrul acestui proiect, am urmat o metodologie bine definită și am utilizat diagrame UML pentru a documenta și modela proiectarea aplicației.

UML este un limbaj pentru vizualizarea, specificarea, construirea și documentarea artefactelor sistemelor informaționale. Orice limbaj constă dintr-un dicționar și reguli pentru combinarea cuvintelor pentru a face construcții semnificative, astfel că limbajul UML merge pe aceeași structură. Caracteristica sa distinctivă este că vocabularul limbii UML este format din elemente grafice. Fiecare simbol grafic are o semantică specifică, astfel încât un model creat de un dezvoltator poate fi înțeles fără ambiguitate de către altul, precum și de un instrument care interpretează UML-ul. Din aceasta, în special, rezultă că un model de sistem informațional prezentat în UML poate fi tradus automat într-un limbaj de programare orientat pe obiecte, precum Java. Acesta oferă o gamă largă de diagrame care pot fi utilizate pentru a modela aspecte ale unui sistem informațional. Aceste diagrame ajută dezvoltatorii, arhitecții și echipele de proiect să înțeleagă, să colaboreze și să implementeze sistemele într-un mod eficient.

Pentru a reprezenta structura aplicației într-un mod mai abstract, am folosit diagrama de clasă UML (fig.7). Diagrama de clasă în cadrul limbajului UML este una dintre cele mai utilizate și importante diagrame.

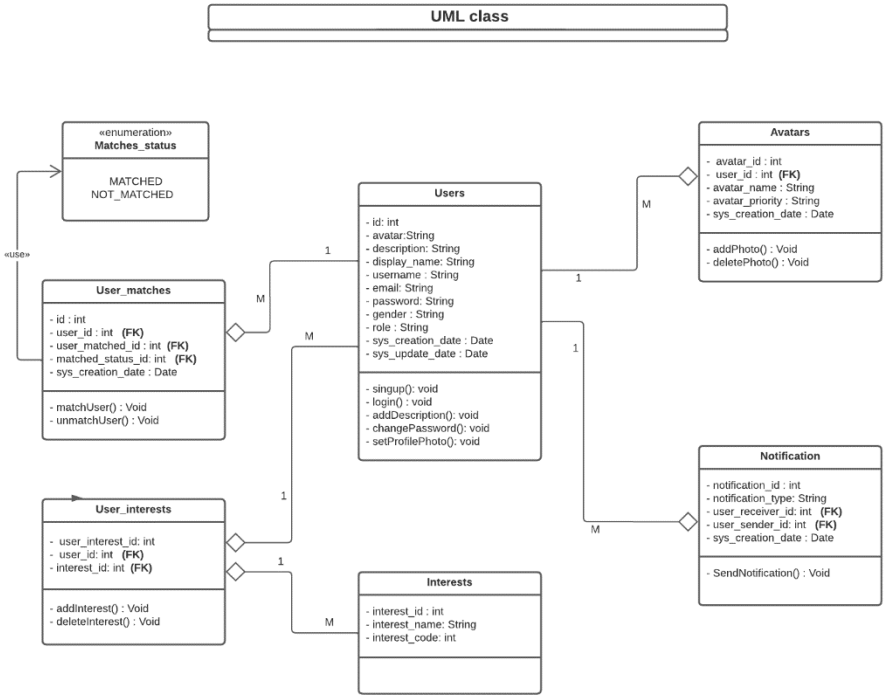

Fig. 7: Diagrama de clasă UML

Această diagramă oferă o imagine de ansamblu asupra claselor aplicației și a relațiilor dintre ele. Am identificat clasele principale, atributele și metodele acestora, oferind astfel o bază solidă pentru implementare. Clasele care reprezintă obiectele sunt 1:1 cu tabelele din baza de date.

După cum sugerează și diagrama, clasele principale ale aplicației sunt : User, User_matches, User_interests, Avatars și Notifications. Aceste clase au de asemenea și metode atât CRUD, cât și metode specifice. Obiectul principal care predomină în aplicație și care are o legătură directă cu toate obiectele principale, este User.

Diagrama de clasa a ajutat la respectarea și urmărirea unui plan concis în implementarea aplicației.

Pentru a ilustra funcționalitatea principală a aplicației, am creat o diagramă de activități (fig.8). Diagrama de activități ajută la identificarea și înțelegerea fluxurilor de lucru principale în cadrul sistemului. Aceasta poate evidenția secvența acțiunilor și deciziilor, precum și regulile de tranziție între stările diferite ale fluxului de lucru.

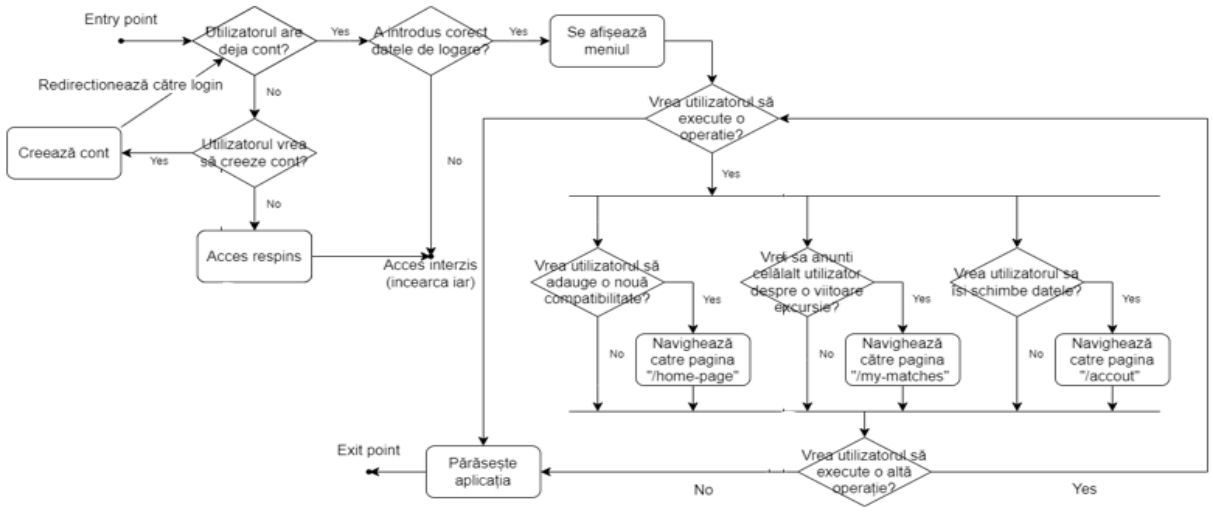

Fig. 8: Diagramă de activități UML

Această diagramă reprezintă pașii cheie în desfășurarea procesului în aplicația TravelWithMe, evidențiind interacțiunea între utilizator și sistem. Am reprezentat acțiunile, elementele de interfață și fluxul de date pentru a evidenția funcționalitatea principală a acesteia. Punctul de pornire începe cu creearea unui cont. Sunt reprezentate apoi 3 dintre cele mai importante acțiuni, care definesc, de altfel, și fluxul de bază al aplicației. Punctul de sfârșit se remarcă prin părăsirea aplicației.

Deoarece diagrama urmează cursul întregii aplicații, ea nu acoperă detaliat procesului de acceptare sau respingere a utilizatorilor pentru a începe o călătorie noua. Pentru asta a fost elaborata alta diagrama Use Case. (fig.9). Diagrama Use Case este o diagramă care descrie interacțiunea între actori (utilizatori) și sistemul într-un anumit scenariu. Aceasta este folosită pentru a identifica și modela funcționalitățile și comportamentul sistemului din perspectiva utilizatorului.

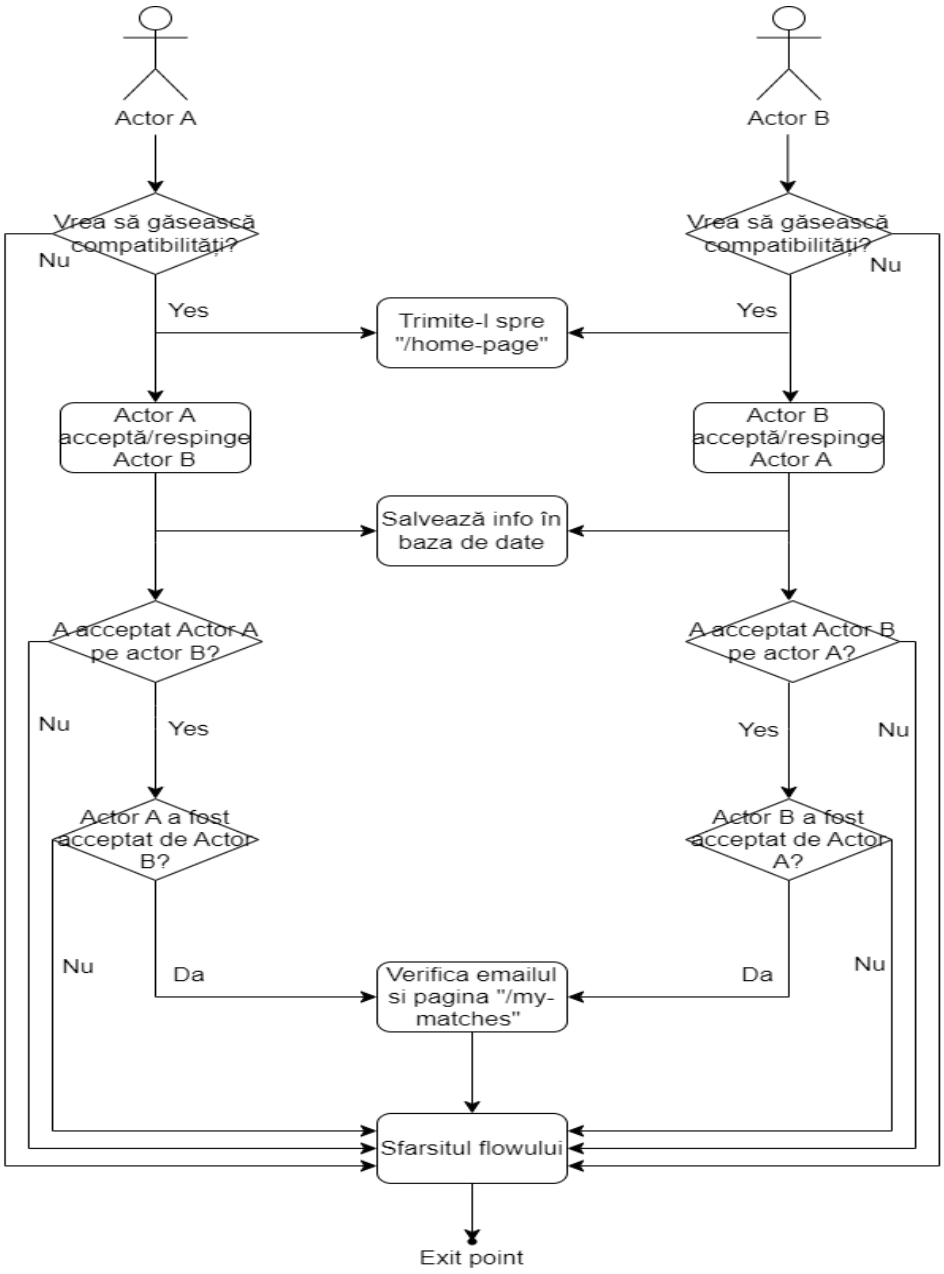

Fig. 9: Diagrama Use Case UML

În diagrama curentă (fig.9) actorii reprezintă doi utilizatori diferiți care repetă aceleași acțiuni. Ceea ce diagrama subliniază este acțiunea de trimitere a emailului și compatibilitate acceptată, care se reproduce doar când aceeași pași sunt urmați de utilizatorii din ambele părți.

Din câte sugerează și diagrama UseCase, fiecare utilizator are aceleași acțiuni posibile. Pentru orice acțiune negativă, posibilitatea de a forma o legătură cu celălalt utilizator se oprește.

Separat, pentru flow-ul de autentificare, a fost elaborata o digrama separată (fig.10) care descrie toate validarile facute pe nivel de front-end a aplicației.

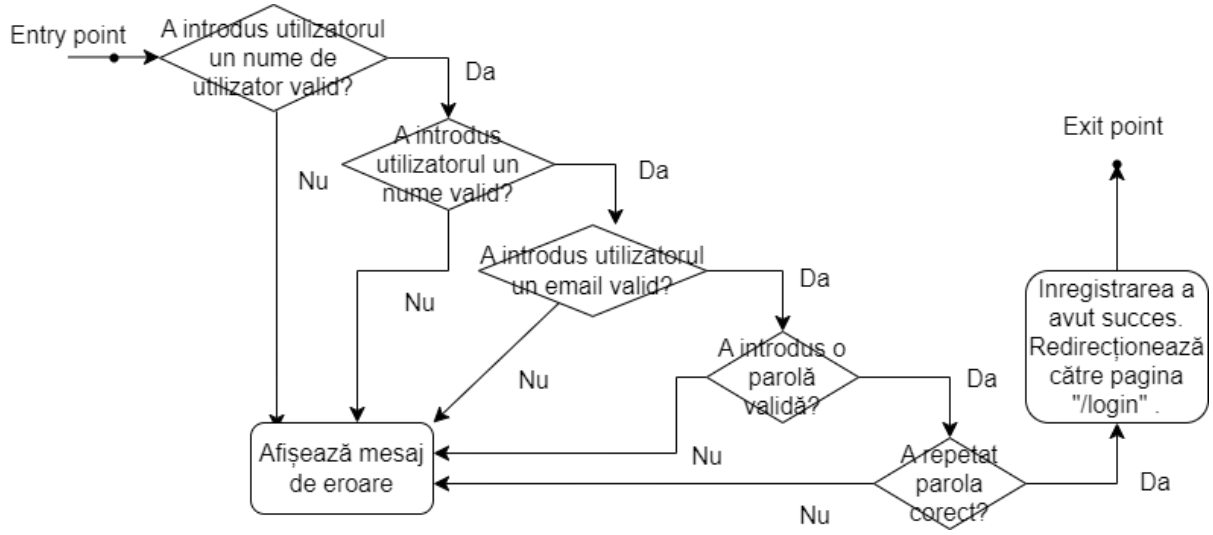

Fig. 10: Diagramă de autentificare

3.3. Implementarea aplicației

Structura proiectului

Proiectul propus este o aplicație web dezvoltată în Java, utilizând framework-ul Spring Boot, pentru partea de backend și React pentru partea de frontend. Arhitectura proiectului este bazată pe un model de monolit, iar modelul arhitectural utilizat este Model-View-Controller .

Aplicația backend este organizată în pachete și module, respectând principiile de împărțire a responsabilităților și coeziune. Aceasta include pachete pentru modele de date, servicii, controlori, repozitoriuri și configurații. Definirea modelelor de date care sunt utilizate în aplicație, reprezintă entitățile de bază și relațiile dintre ele. Aceste modele sunt utilizate pentru interacțiunea cu baza de date și pentru transferul datelor între backend și frontend. Implementarea serviciilor

care gestionează logica și operațiile CRUD [4] pe modelele de date. Aceste servicii sunt responsabile de interacțiunea cu baza de date și de procesarea cererilor primite de la frontend. Definirea controlorilor gestionează rutarea și manipularea cererilor HTTP[5] primite de la frontend. Acești controlori sunt responsabili de validarea datelor de intrare, apelarea serviciilor corespunzătoare și returnarea răspunsurilor către frontend.

Aplicația frontend este organizată în componente, respectând principiile de modularizare și reutilizare. Aceasta include componente pentru vizualizarea datelor, interacțiunea cu utilizatorul și gestionarea stării aplicației. Componentele de vizualizare afișează datele utilizatorului și permit interacțiunea cu acestea. Aceste componente folosesc datele primite de la backend și actualizează starea aplicației în funcție de acțiunile utilizatorului. Componentele de control gestionează evenimentele și interacțiunile utilizatorului. Aceste componente comunica cu backend-ul prin intermediul cererilor HTTP și gestionează răspunsurile primite.

Comunicarea dintre backend și frontend se face prin expunerea de API-uri REST. Aceste API-uri definesc rutele și metodele de acces (GET, POST, PUT, DELETE) pentru a permite manipularea datelor. Datele sunt transferate în format JSON, facilitând interoperabilitatea și comunicarea între cele două componente.

Baza de date

Pentru lucrul cu baza de date am folosit framework-urile ORM: Hibernate și JPA.

Definirea entităților a fost realizată prin crearea unor clase, care reprezintă fiecare tabel din baza de date, și câmpurile acestora care reprezintă coloanele tabelei. Acească implementare este posibilă prin intermediul adnotarilor @Entity și @Column oferite de Hibernate.

[4] create, read, update, delete
[5] Hypertext Transfer Protocol

Tabela în jurul căreia se întâmplă toate acțiunile, este tabela user. Pentru a o crea în baza de date, dar și pentru a o putea folosii în aplicație ca pe un obiect, am folosit acest cod:

```
package travel.entities;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import javax.persistence.*;
import java.sql.Timestamp;

@Data
@Entity
@Table(name = "users")
@NoArgsConstructor
@AllArgsConstructor
@Inheritance(strategy = InheritanceType.JOINED)
public class UserEntity extends BaseEntity  {

   @Column(nullable = false, unique = true)
   private String username;
   @Column(nullable = false, unique = true)
   private String email;
   @Column(name="display_name")
   private String displayName;
   @Column
   private String gender;
   @Column
   private String description;
   @Column(nullable = false)
   private String role;
   @Column(nullable = false)
   private String password;
   @Column
   private String avatar;
   @Column(name = "sys_creation_date", insertable = false, updatable = false)
   private Timestamp creationDate;
   @Column(name = "sys_update_date", insertable = false, updatable = false)
   private Timestamp updateDate;

   @Override
   public Object clone() throws CloneNotSupportedException {
      return super.clone();}
}
```

Crearea celorlalte tabele se face printr-un cod asemănător.

Pentru manipularea datelor din baza de date am folosit JPA Repository, unde interogările utilizate sunt personalizate cât și generate automat prin convenții de nume.

Implementarea funcționalităților

În cadrul acestei lucrări de licență, am dezvoltat o aplicație care oferă utilizatorilor posibilitatea de a interacționa și a realiza anumite acțiuni în conformitate cu interesele și nevoile lor. Aplicația pune accentul pe funcționalități relevante și utile, care contribuie la îmbunătățirea experienței utilizatorilor și la eficientizarea activităților specifice domeniului de aplicare.

Principal Component Analysis

Una dintre principalele funcționalități implementate în aplicație este capacitatea de a ordona utilizatorii în funcție de interesele lor. Această funcționalitate oferă utilizatorilor o perspectivă mai clară asupra altor utilizatori cu interese similare și facilitează procesul de conectare și colaborare. Implementarea acestei ordonări este bazată pe tehnica Principal Component Analysis (PCA). Algoritmul PCA se folosește in multe domenii inclusiv pentru reducerea optima a spațiului de lucru sau ca o preprocesare a datelor de intrare sau in aplicații de rețele neuronale.

Prin această tehnică am grupat interesele care fac parte dintr-o anumită categorie de oameni prin asocierea cu numere apropiate. De exemplu cazările cele mai ieftine, locurile în care poți mânca cele mai ieftine, sunt asociate cu cele mai mici numere, împreună cu voluntariatul și alte date relevante. Numărul asociat crește în funcție de situația financiară și gradul de confort ridicat.

Algoritmul PCA este aplicat pentru a calcula covarianța dintre interesele utilizatorilor și interesele utilizatorului de referință. Covarianța măsoară relația statistică între două variabile și este utilizată pentru a determina similaritatea între interesele utilizatorilor. Sortarea descrescătoare a userilor în funcție de valorile de covarianță ne permite să identificăm utilizatorii care au cele mai mari valori de covarianță, adică utilizatorii cu interese similare. Acesta este reprezentat in limbajul java astfel:

```
Map<Integer, Double> covarianceValues = new HashMap<>();
for (int userId : allUserIds) {
    List<Integer> userInterests =
userInterestsRepository.findByUserEntityId(userId)
            .stream().map(interest ->
interest.getInterests().getInterestCode())
            .collect(Collectors.toList());
    double covariance = calculateCovariance(userInterests, myUserInterests);
```

```
    covarianceValues.put(userId, covariance);

}

List<Integer> sortedDescendantUserIds = new
ArrayList<>(covarianceValues.keySet());
sortedDescendantUserIds.sort((u1, u2) -> {
    double covariance1 = covarianceValues.get(u1);
    double covariance2 = covarianceValues.get(u2);
    return Double.compare(covariance2, covariance1);
});

private static double calculateCovariance(List<Integer> userInterests,
List<Integer> myUserInterests) {

    final double mean1 = userInterests.stream()
            .mapToInt(Integer::intValue)
            .average()
            .orElse(0.0);

    final double mean2 = myUserInterests.stream()
            .mapToInt(Integer::intValue)
            .average()
            .orElse(0.0);

    double covariance = 0.0;
    final int n = Math.min(userInterests.size(), myUserInterests.size());

    for (int i = 0; i < n; i++) {
        double deviation1 = userInterests.get(i) - mean1;
        double deviation2 = myUserInterests.get(i) - mean2;
        covariance += deviation1 * deviation2;
    }

    covariance /= n;
    return covariance;
}
```

Strategy design pattern

Strategy design pattern este unul dintre cele mai utilizate și cunoscute pattern-uri de proiectare în programarea orientată pe obiect. Acesta face parte din categoria pattern-urilor comportamentale și se concentrează pe gestionarea și schimbarea dinamică a algoritmilor în cadrul unei aplicații.

Scopul principal al strategy pattern este de a separa logica algoritmică de clasa care o utilizează, astfel permite înlocuirea sau schimbarea algoritmului fără a afecta clasa principală. Acesta promovează modularitatea, flexibilitatea și reutilizarea codului, ceea ce face ca aplicațiile să fie mai ușor de întreținut și de extins. Utilizarea acesteia se recomandă în situațiile în care există

mai multe variante sau alternative pentru implementarea unei funcționalități și se dorește o flexibilitate crescută pentru a schimba între aceste variante. De asemenea, este util atunci când se dorește separarea logicii de prelucrare în clase independente și când se dorește o structură clară și modulară a codului.

Acest pattern este folosit în aplicație pentru trimiterea emailurilor. Este creată o interfață comună, numită TemplateStrategy de unde algoritmii pentru fiecare email sunt încapsulați în clase diferite, numite clase de strategie. Fiecare clasă de strategie are setată o valoare. Clasa principală, TemplateFactory, primește ca parametru o valoare care este căutată în fiecare clasă de strategie. Prin această metodă se alege algoritmul cerut, în contextul actual, se alege template-ul emailului care trebuie trimis.

Freemarker

Freemarker este un motor de șablon open-source, dezvoltat în limbajul Java, care oferă capacități puternice de generare a conținutului dinamic, precum pagini web sau emailuri. Această tehnologie se bazează pe șabloane (FTL - FreeMarker Template Language), care permit combinarea textului static cu expresii și variabile dinamice.

Prin intermediul șabloanelor, am creat o structură standard a unui email, urmând ca pentru fiecare apelare să se personalizeze cu date relevante oferite prin intermediul codului. Datele personalizate sunt date de variabile și expresii care formează conținutul dinamic.

Pentru a adăuga variabile externe în șablonul FTL, am definit o mapă care are ca și cheie numele variabilei, iar valoarea reprezintă conținutul variabilei. Această mapă trimite toate variabilele și valorile asociate pe care dorim să le utilizăm către șablonul FTL.

De exemplu, putem adăuga variabila "nume" în modelul de date și să-i atribuim valoarea "John". Apoi, în șablonul FTL, putem accesa valoarea variabilei folosind sintaxa *${nume}*. La generarea emailului, Freemarker va înlocui expresia *${nume}* cu valoarea corespunzătoare din modelul de date, rezultând un email personalizat pentru fiecare destinatar în parte.

Unul dintre avantajele majore ale utilizării Freemarker este separarea logicii de prezentare de codul Java. Utilizarea acestuia pentru crearea emailurilor personalizate oferă avantaje precum

flexibilitatea în manipularea conținutului, modularitatea și reutilizarea, precum și o integrare ușoară cu limbajul Java.

Stocarea persistentă a datelor

Stocarea persistentă a datelor este o tehnică bine cunoscută pentru separarea și adăugarea de date în fișiere specifice în memorie. Această metodă face referire la modul în care datele sunt gestionate și stocate într-o formă persistentă într-un mediu de memeorie.

Pentru gestionarea eficientă a pozelor adăugate în memorie de către utilizatori, am creat în spațiul de memorie de pe disc o listă de fișiere generate automat. Fiecare folder conține imagini specifice unui utilizator, iar asocierea se face prin numele care este 1:1 cu numele utilizatorului.

Pentru a transmite imaginile din backend, acestea sunt criptate prin sistemul de codificare base69 și transformate în șiruri de caractere ASCII prin intermediul bibliotecii StandardCharsets. În frontend, imaginea primită pe răspuns se decriptează și afișează în interfață.

Parcursul prin care o imagine este extrasă din stocarea internă și trimisă pe răspuns se poate vedea în metoda de mai jos:

```
public String getEncodedFile(String fileName, String username) {
    final String pathname = getPathMyPhotos(username);
    File directory = new File(pathname);
    try {
        if (directory.exists() && directory.isDirectory()) {
            File[] files = directory.listFiles();
            assert files != null;
            for (File file : files) {
                if (file.getName().equals(fileName)) {
                    byte[] encoded =
Base64.encodeBase64(FileUtils.readFileToByteArray(file));
                    return new String(encoded, StandardCharsets.US_ASCII);
                }
            }
        }
        return null;

    } catch (IOException e) {
        throw new NotFoundException("File not found in directory");
    }
}
```

Securitate

Securitatea unei aplicații este un aspect important pentru protejarea datelor și a utilizatorilor. Unul dintre cele mai importante aspecte ale securității într-o aplicație este gestionarea autentificării și autorizării utilizatorilor. OAuth2 este un protocol deschis și

standardizat care oferă o soluție pentru autentificarea și autorizarea securizată a utilizatorilor în aplicații atât web cât și mobile.

OAuth2 funcționează pe baza unui model de autorizare prin delegare, în care se acordă permisiuni unei aplicații pentru a accesa resursele sale protejate de la un furnizor de servicii. În acest fel, utilizatorii pot acorda acces la informațiile lor fără a dezvălui parolele sau informațiile de autentificare direct către aplicația terță.

Prin OAuth2 generez un token de acces cu care utilizatorul poate obține acces în aplicație. Token-ul se generează automat în momentul în care credențialele de logare sunt valide. Acesta este apoi utilizat de aplicația client pentru a accesa resursele protejate în numele utilizatorului.

Utilizarea tokenilor de acces este folosită în locul parolelor. În loc să fie necesară trimiterea parolelor utilizatorilor între aplicație și server, OAuth2 utilizează token-uri de acces care pot fi revocate sau expirate, oferind un nivel mai mare de securitate.

De asemenea, acest protocol oferă opțiunea de a acorda permisiuni specifice și limitate aplicațiilor. Prin acest intermediu clienții au acces doar la resursele și acțiunile pentru care li se primite. În contextul actual, deoarece aplicația este folosită integral de toți utilizatorii, accesul este permis pentru toate paginile și api-urile existente.

Un aspect important al securității, îl constituie criptarea parolei. Pentru asta, folosesc BCrypt. BCrypt este un algoritm puternic de criptare, considerat un algoritm sigur pentru stocarea parolelor în baze de date. Utilizarea BCrypt îmi asigură o securitate mai sporită a aplicației prin adăugarea unui salt și a unui număr de iterații pentru generarea hash-ului parolei.

Pentru a asigura o securitate mai bună utilizatorilor, un utilizator nu poate utiliza aplicația fără a avea un cont în care să fie logat.

3.4. Scenariul de utilizare

Platforma TravelWithMe ajută oamenii care pleacă singuri în călătorii să găsească alți oameni cu care să împartă această experiență. Deoarece în acest context țara din care provin nu

este un impediment, ba chiar se dorește să fie o diversitate de cultură, limba prioritară a aplicației este limba internațională, engleza.

Această platformă web are ca și pșagină de start o ilustrație semnificativă despre ce reprezintă aplicația (fig.11).

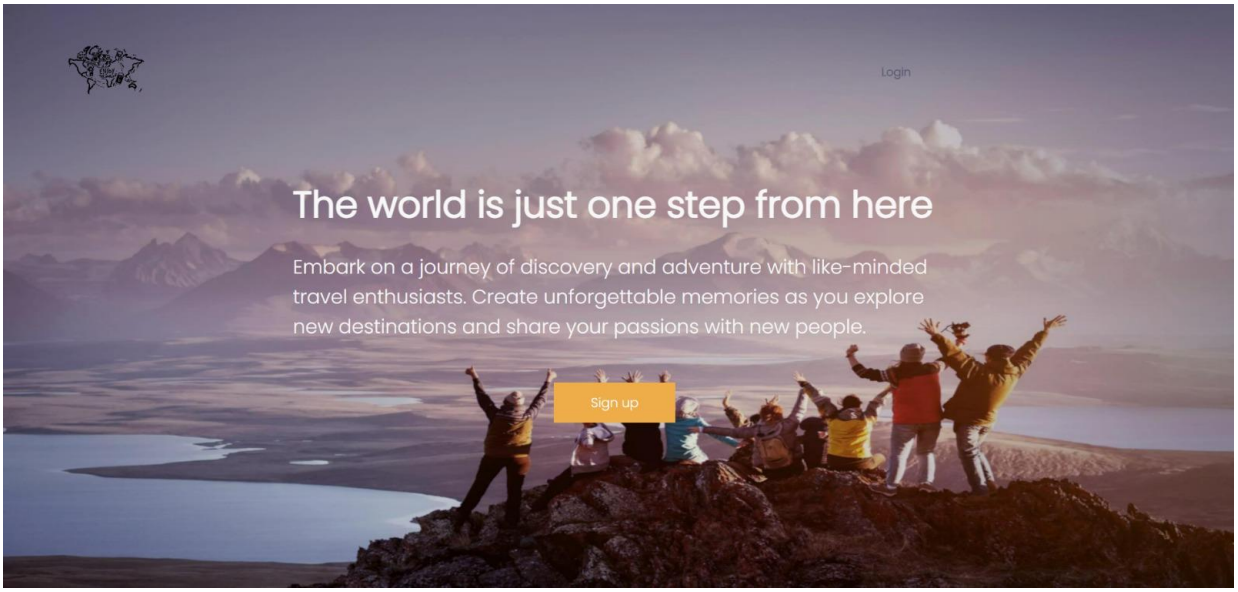

Fig. 11: Pagina de start

Din această pagină de start avem două opțiuni:

- *Înregistrarea în aplicație (Sign Up):* este cel mai vizibil buton din pagină, marcat cu o culoare care iese în evidentă, pentru a ajuta utilizatorii noi să urmeze cu ușurință cursul aplicației. Acest buton redirecționează utilizatorul în pagina de creare cont.
- *Logarea în aplicație (Login):* este situat în partea dreaptă-sus a paginii, vizibil, dar marcat cu o culoare neutră pentru a nu duce utilizatorii noi în eroare cu prea multe ilustrații și lăsându-i focusați pe esențialul paginii. Acest buton redirecționează utilizatorul în pagina de logare în cont.

Înregistrarea și autentificarea

Deoarece singura modalitate prin care un utilizator poate accesa aplicația este să fie conectat, primul pas pe care un nou utilizator este obligat să îl facă este să își creeze un cont nou.

Redirecționat din pagina de start, acesta ajunge în pagina de înregistrare (fig.12) unde trebuie să își adauge datele cu grijă.

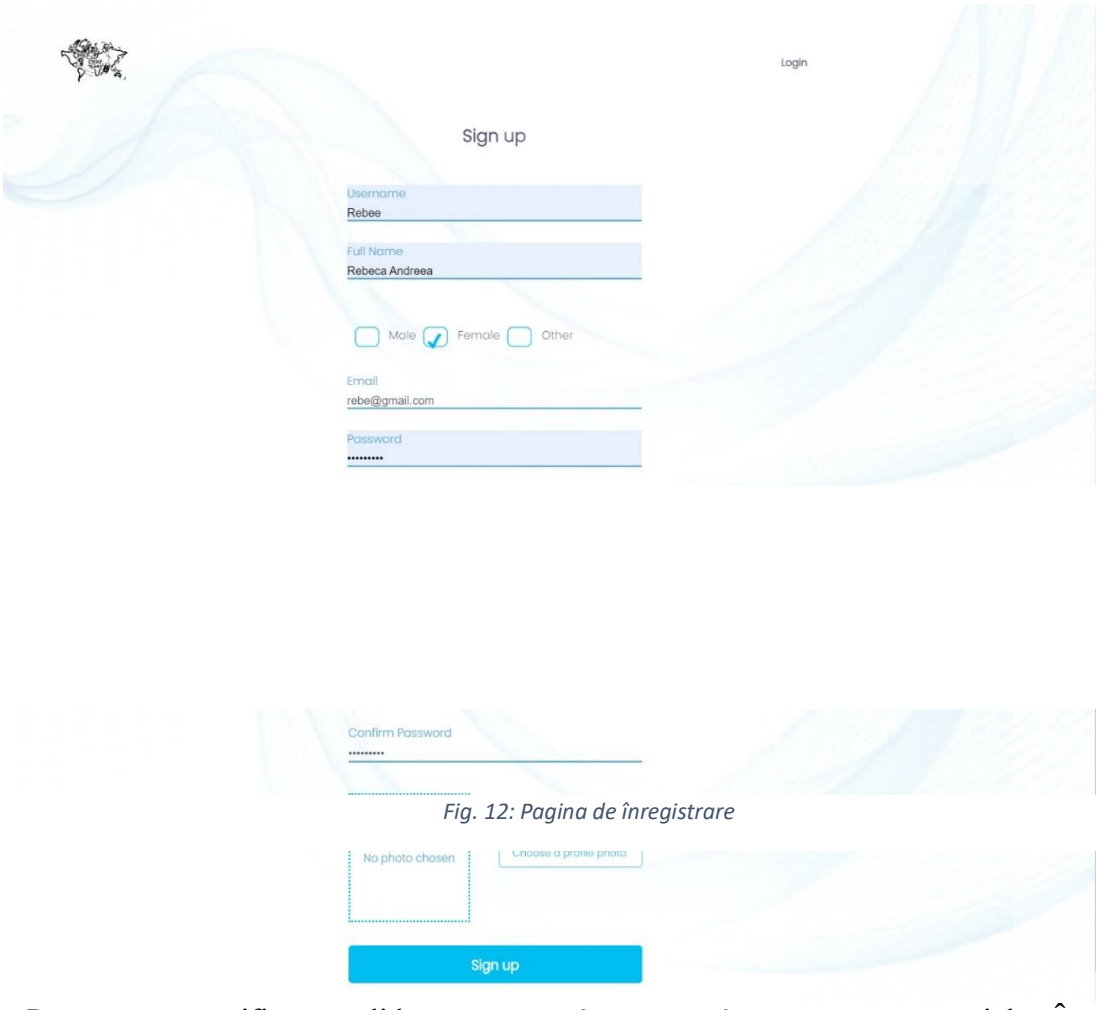

Fig. 12: Pagina de înregistrare

Pentru o autentificare validă, urmăresc în spate câteva aspecte esențiale. În cadrul înregistrării am decis completarea profilului doar cu datele esențiale, astfel, toate elementele au validarea de a fii completate. În caz contrar, se va afișa un mesaj lângă elementul necompletat.

În primul rând, deoarece conectarea se face pe bază de username, iar mesajele primite în aplicație se vor primii prin email, aceste două câmpuri au ca și validare unicitatea lor în cadrul aplicației. Această verificare se face pe partea de back-end cu ajutorul bazei de date care ține minte toate informațiile.

Pentru siguranță, pe câmpul de username avem câteva validări si din partea de front-end. Aceste sunt: completarea obligatorie cu minim cinci caractere și inexistența spațiilor. Pentru verificarea spațiilor s-a folosit o expresie regex:

```
const regEx = new RegExp("^(.*\\s+.*)+$");

   if (!info.username) {
     errors.username = "Required";
   } else if (info.username.length < 5) {
     errors.username = "Minimum 5 char";
   } else if (regEx.test(info.username)) {
     errors.username = "No spaces allowed";
   }
```

Alte validări importante sunt :

- caracterele introduse în câmpul de parolă și confirmare parolă trebuie să fie identice;
- emailul trebuie să aibă sintaxa specifică. Acest lucru este verificat prin regexul :

```
!/^[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,}$/i
```

După ce validările sunt efectuate cu succes, se redirecționează utilizatorul în pagina de logare (fig.13), pentru a intra în aplicație. Dacă utilizatorul are deja cont, poate accesa pagina și prin butonul din stânga-sus.

Cel puțin o dată, fiecare om și-a uitat parola de la un cont. Pentru a evita cazul nefericit prin care un utilizator își uită parola există posibilitatea de a-ți recupera parola prin intermediul emailului. Acest demers se începe prin apăsarea butonului *Forgot you password?* . Butonul te redirecționează pe o pagină unde se cere emailul. Dacă emailul se găsește în baza de date, vei primii un mesaj cu instrucțiunile necesare.

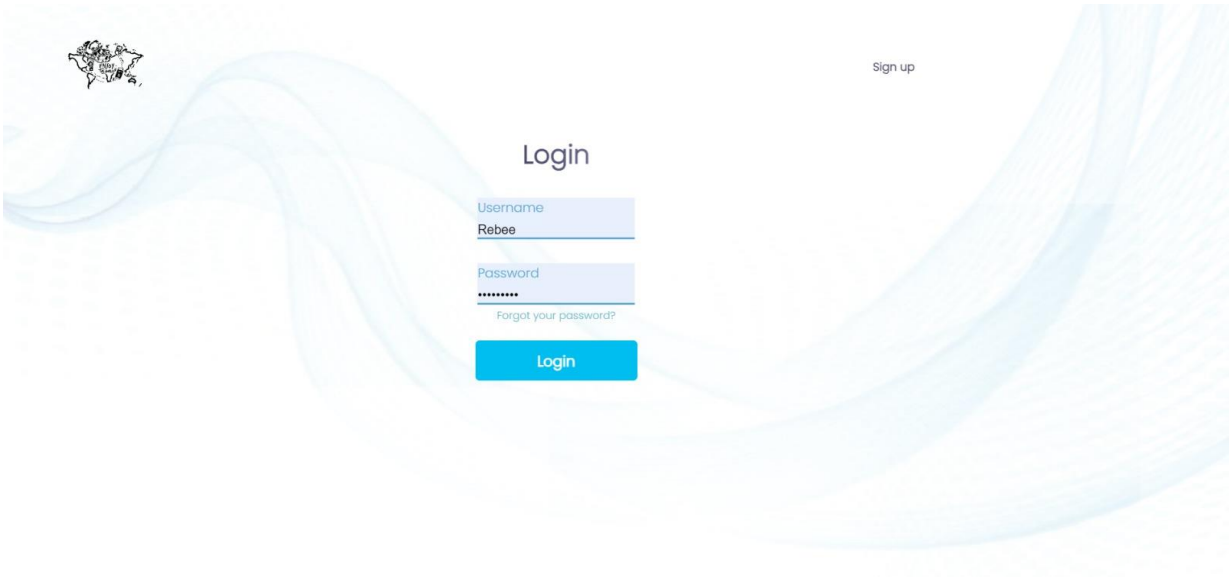

Fig. 13: Pagina de logare

Pe partea de conectare în aplicație se introduc datele specificate în fig.13. Autentificarea se face printr-un API care validează credențialele folosind baza de date pentru a compara rezultatele, apoi se creează un token folosit de utilizator pe parcursul interacționării cu aplicația.

După autentificarea făcută cu succes, se redirecționează utilizatorul în aplicație, unde poate face orice operație vizibilă pe ecran. Prima pagină afișată este pagina Home (fig.14).

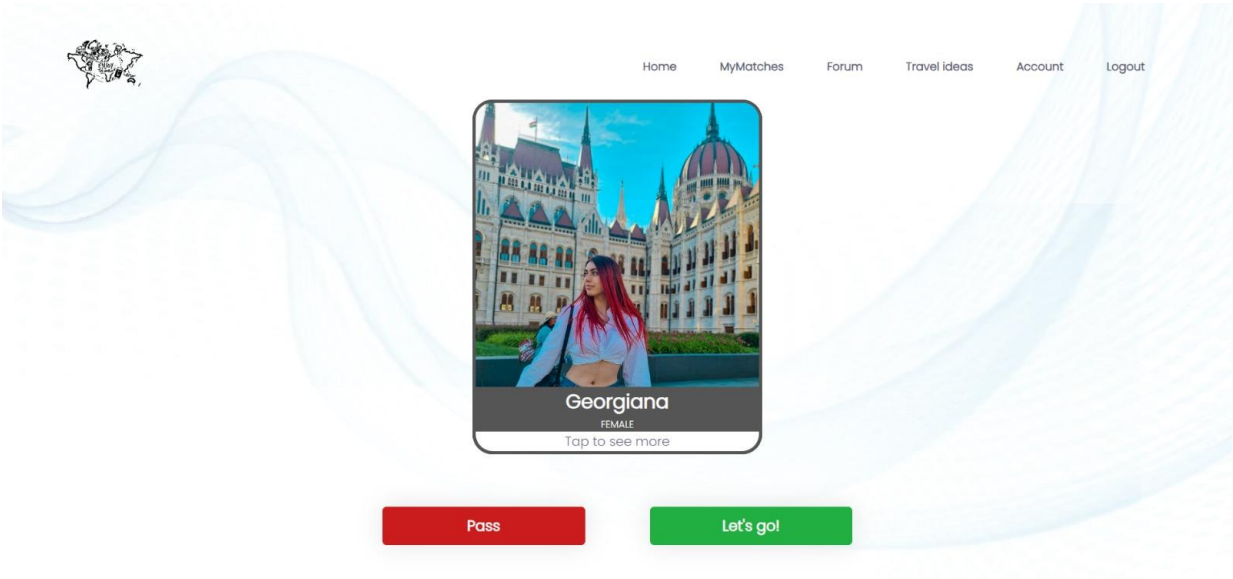

Fig. 14: Pagina principală web

Orice pagină din interiorul aplicației are același header, compus dintr-un meniu cu șase elemente. În funcție de elementul apăsat body-ul/ corpul aplicației se va schimba.

În pagina Home se poate ajunge atat prin elementul Home aflat primul în cadrul meniului, dar și prin apăsarea pe sigla aplicației din partea stângă.

Aplicația are design responsive, astfel aspectul este plăcut atât pentru dispozitiv (cum ar fi telefoane și tablete) și pe dispozitive de tip web (cum ar fi computere desktop sau laptop-uri). În momentul în care rezoluția se micșorează, meniul nelipsit se transformă într-un simbol format din patru pătrate (fig.15). La apăsarea acestuia se afișează aceleași elemente precum în fig.14 , dar sub formă listată.

Celelalte elemente, în funcție de dimensiuni și așezarea în pagină au două opțiuni: își micșorează dimensiunea sau se reorganizează în pagină. Un exemplu sunt butoanele Pass și „Let's go!" Din figurile 14 și 15.

Fig. 15: Pagina principală mobile

După cum spune și numele, pagina Home este pagina principală a aplicației. Ea conține funcționalitatea de bază, făcând posibilă alegerea utilizatorilor cu care vrei să interacționezi prin apăsarea butonului „Let's go!" De asemenea, putem exclude persoanele cu care considerăm că nu avem interese comune prin apăsarea butonului Pass. Pentru afișarea utilizatorilor după ordinea celor mai bune compatibilități s-a folosit algoritmul PCA.

Utilizatorii sunt afișați inițial într-o căsuță care conține poza de profil, numele și genul persoanei. Pentru a putea vedea restul informațiilor folosim butonul „*Tap to see more"* aflat în partea de jos a căsuței (fig.14). Astfel, utilizatorul este redirecționat către o pagină unde poate vizualiza poze, informații de bază despre interesele alese și descrierea utilizatorului ales pentru a facilita o alegre corectă.

Înainte de a începe căutările, trebuie ca fiecare utilizator să înțeleagă importanța de a ști informații despre ceilalți participanți și să ofere la rândul său aceleași informații. Pentru a-și actualiza profilul, atât la prima utilizare, cât și ulterior, se accesează pagina Account (fig. 16), penultima din meniul de navigație.

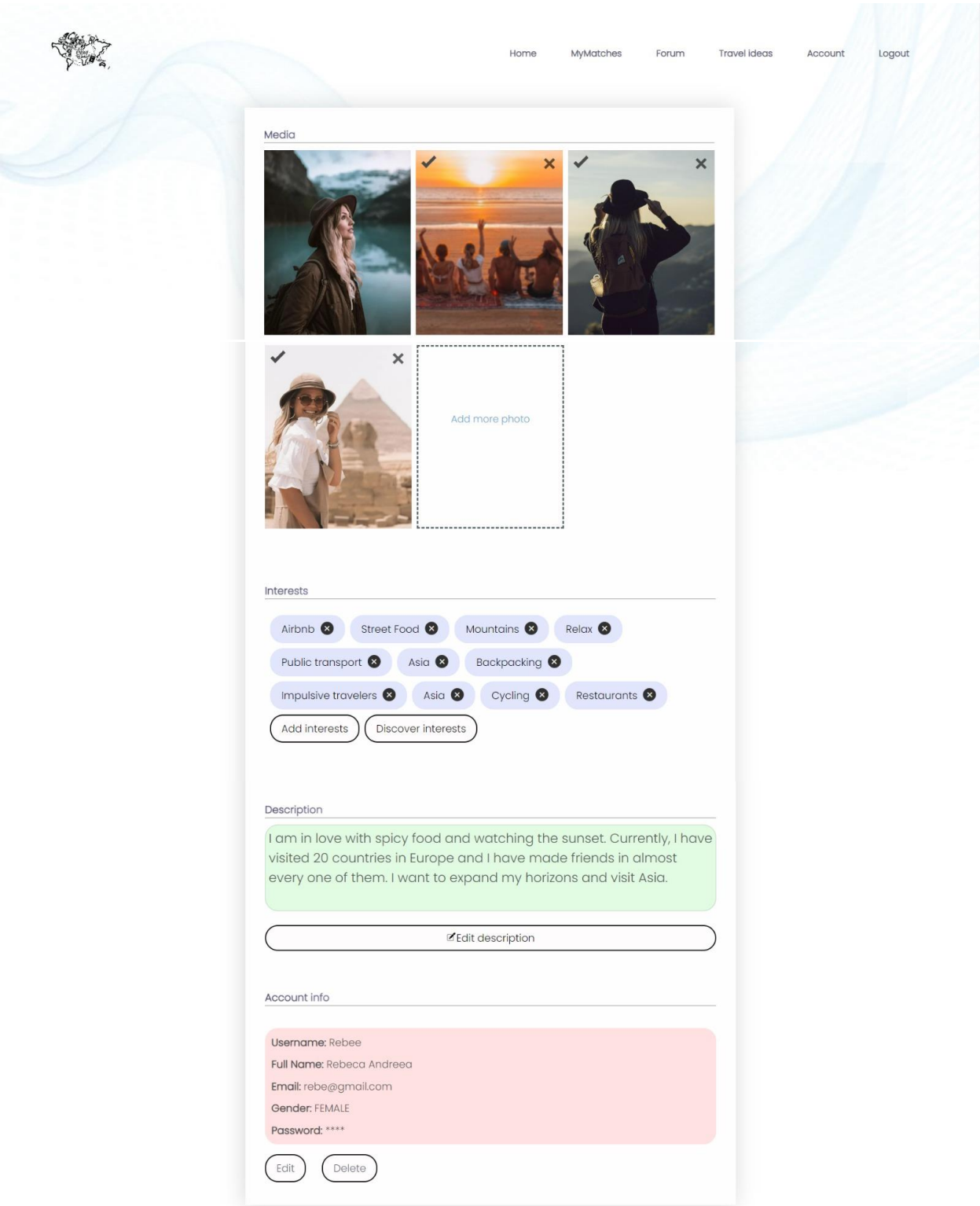

Fig. 16: Pagina „Contul meu"

Privind fig.16 putem observa că pagina Account este împărțită în patru componente semnificative: Media, Interests, Description și Account Info.

Media reprezintă componenta care afișează pozele pe care utilizatorul le are încărcate la moment, dar și posibilitatea de a face acțiuni asupra acestea. Utilizatorul are posibilitatea de a-și adăuga noi poze în profil, de a șterge poze, cu restricția de a avea cel puțin o poză și de a seta o poză ca fiind de profil. Dacă există o singură poză pe profil, ea v-a fii setată automat ca fiind poza de profil a utilizatorului. Pentru a adăuga o poză nouă selectezi căsuța goala cu mesajul *Add more photo*.

Ștergerea unei poze este reprezentată de un x aflat în dreapta-sus a fiecărei poze, iar pentru a selecta o poză de profil se apasă semnul bifat din stânga-sus. Poza de profil se așază prima și nu are aceste două opțiuni.

Interests este o secțiune care se ocupă de organizarea intereselor utilizatorului. Pentru utilizatorii noi, dar și pentru cand ai nevoie de ajutor să îți reorganizezi interesele există opțiunea *Descoperă-ti interesele* (Discover interests). Butonul te redirecționează într-un test cu zece întrebări atât cu o singură varianta de răspuns (butoane de tip RadioButton – fig.18) dar și cu mai multe (butoane de tip CheckBox – fig.17). Fiecare răspuns are în spate un interes care se adaugă sub forma din figura 16. Structura și aspectul întrebărilor este sub forma aceasta:

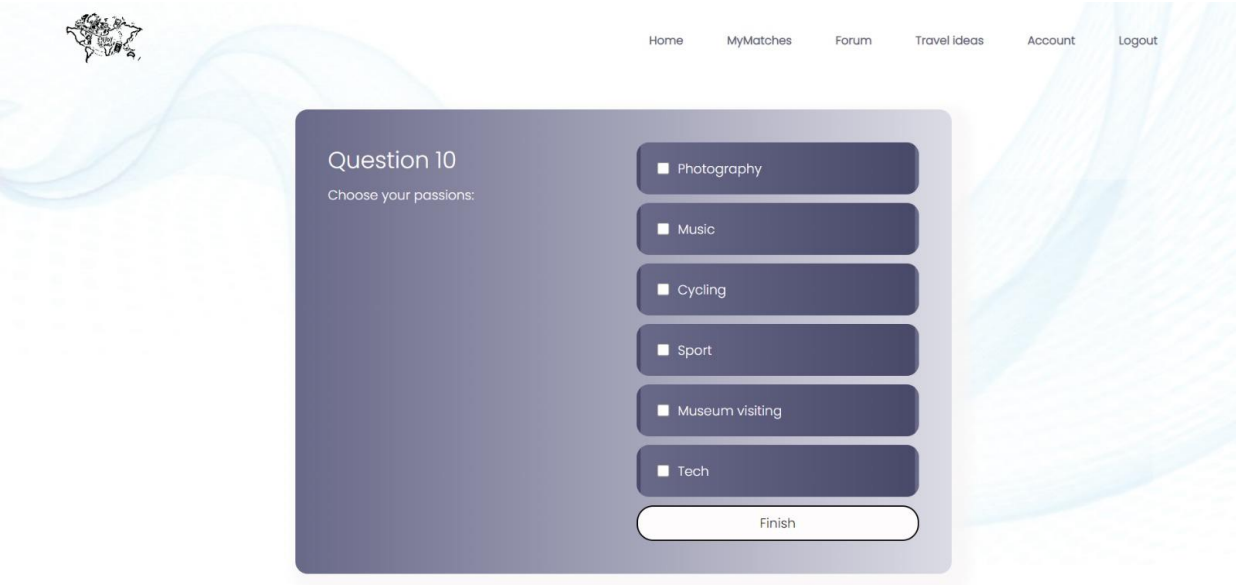

Fig. 17: Întrebare cu mai multe variante de răspuns

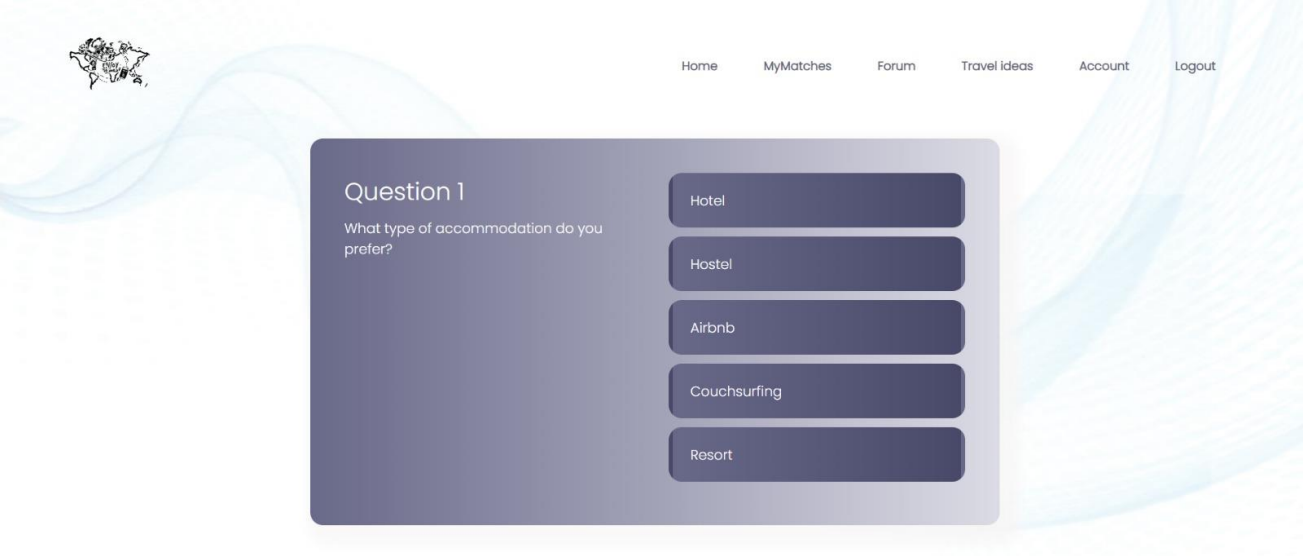

Fig. 18: Întrebare cu o variantă de răspuns

Ulterior, interesele se pot modifica, prin adăugarea de noi interese dar și prin ștergerea lor. Pentru a adăuga interese noi folosim butonul *Adaugă interese* (Add interests), iar pentru a șterge se apasă butonul x de pe interesul pe care dorești să îl scoți.

Description este cea mai simplă componentă a paginii. Ea reprezintă, după cum sugerează și numele, descrierea utilizatorului. Pentru a adăuga, edita sau șterge text folosim butonul de editare descriere. Descrierea salvata va fii textul rămas în textbox-ul afișat.

Account info este ultima componentă deoarece, în general, este cea mai puțin utilizată. Sunt afișate informațiile despre cont împreună cu două butoane care îți oferă posibilitatea de a edita sau șterge contul. Câmpurile editabile ale contului sunt parola și genul.

După ce utilizatorul parcurge pagina Account și își pregătește contul cu date relevante, poate să caute compatibilități. Deoarece nu se dorește deranjarea utilizatorilor cu mesaje nedorite, contactul dintre doi utilizatori nu este permis până când ambii își oferă unul celuilalt un răspuns afirmativ, prin apăsarea butonului verde. Când unul din utilizatorii pe care i-ai selectat, te-au selectat anterior, se va afișa o notificare în partea dreapta-sus care îți oferă această informație. Dacă ești primul care a inițiat această compatibilitate, nu este nevoie să intri în aplicație pentru a vedea

notificările. Orice compatibilitate care apare în momentul în care nu ești logat în aplicație va fii trimisă prin email.

O dată ce ai aceste compatibilități, le poți vizualiza în pagina MyMatches (fig.19).

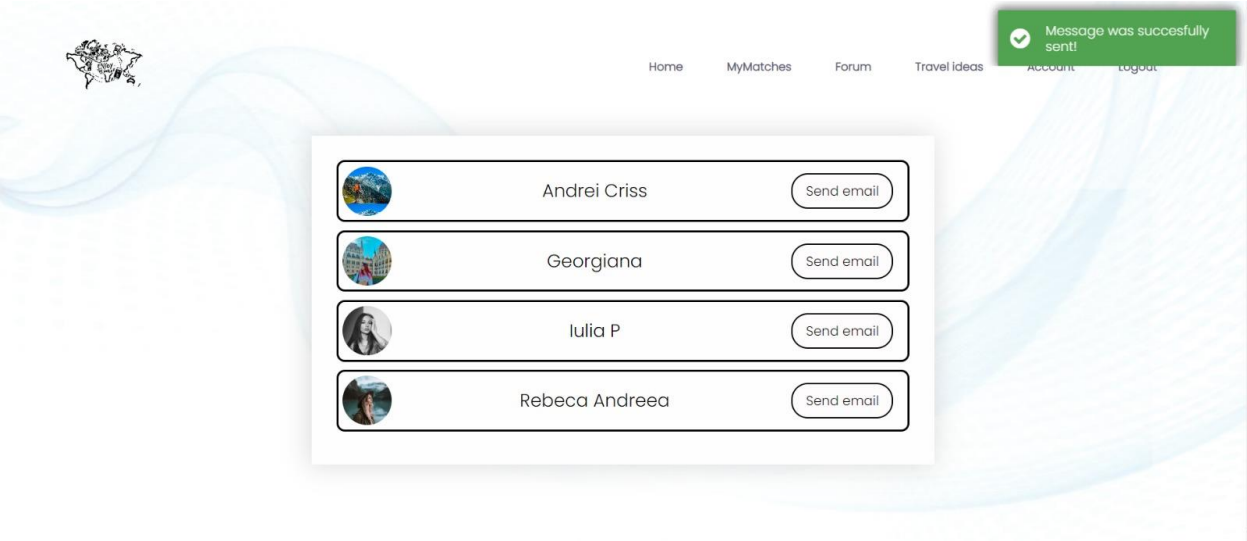

Fig. 19: Pagina „Compatibilitățile mele"

Pagina este alcătuită dintr-o listă unde se află utilizatorii cu care ai creeat o conexiune. Dacă duci mouse-ul pe numele utilizatorului se va afișa un text care specifică faptul că la un click distanță poți vizualiza detaliile utilizatorului. Detaliile au aceeași structură precum în pagina principală, cu diferența că secțiunea cu informațiile de contact ale utilizatorului devine vizibilă.

Pentru a solicita începerea unei conversații se folosește butonul *Trimite email* (Send email) aflat în partea dreaptă a fiecărui utilizator. O dată trimis emailul, se primește o notificare, așa cum este afișat in partea de sus a figurii 19.

Emailurile trimise sunt create și personalizate în back-end cu numele și datele potrivite. Ele au fost construite printr-un template html. Un exemplu de email trimis, este fig.20, unde utilizatorul dorește să contacteze un altul.

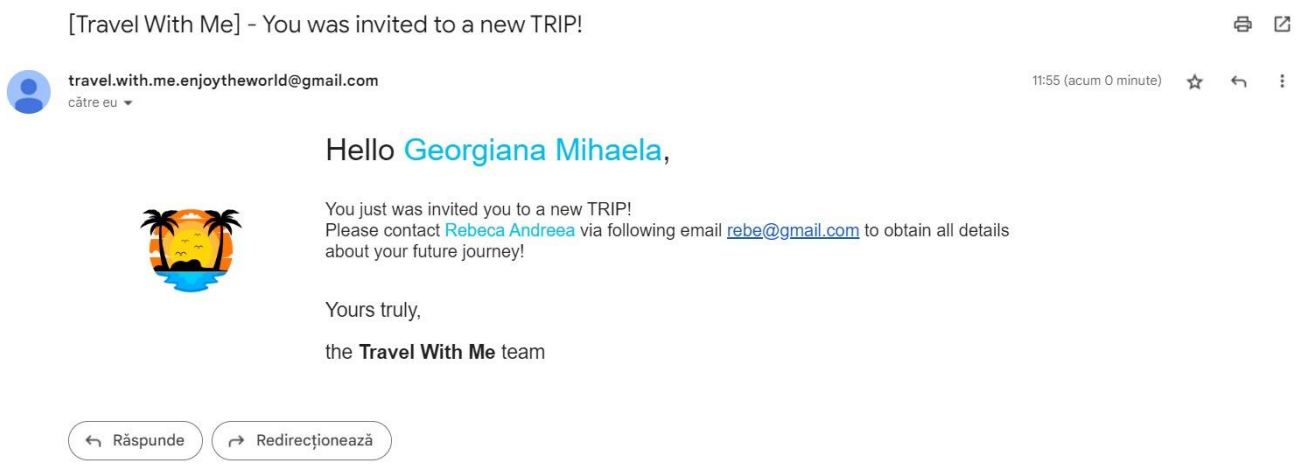

Fig. 20: Exemplu de email

După cum se poate observa în figura 20, există un cont specific al platformei prin care toate email-uile generate sunt trimise: travel.with.me.enjoytheworld@gmail.com.

Deșii funcționalitatea de bază este reprezentată de căutarea unor noi parteneri de călătorie, există și funcționalități adiționale, pentru a facilita căutarea vacanțelor care se vor produce ulterior. Pentru scopul menționat am integrat două pagini: *Forum* și *Travel Ideas*.

Travel Ideas este o pagină care încorporează site-ul web Tripadvisor. Pagina afișată cuprinde o listă cu cele mai vizitate locuri din lume, însă meniul din partea de sus permite navigarea în orice pagină din platforma. Consider un avantaj integrarea efectuată deoarece permite găsirea unor informații relevante rapid și eficient.

Pagina Forum, este un mod rapid și simplu care te conectează cu călătorii din întreaga lume pentru ajutor și sfaturi. Comunitatea de călători este una prietenoasă și dornică de ajutor, astfel, orice întrebare își va găsii răspuns.

Pentru a nu încărca baza de date cu conversațiile din forum, acesta folosește spațiul de stocare al browser-ului pentru a reține fiecare mesaj cu detaliile aferente, timp de o zi.

Concluzii

Proiectul Travel With Me a implicat utilizarea bazelor de date și a tehnologiilor React și Spring Boot pentru dezvoltarea unei aplicații eficiente și funcționale. Pentru a asigura o structură eficientă a codului, s-au aplicat diverse tehnici de programare orientată pe obiecte, s-au utilizat șabloane de proiectare, precum Factory Pattern, Model-View-Controller și Template Method Pattern, dar și algoritmi de analiză. Aceste abordări au permis crearea unei aplicații bine organizate, ușor de întreținut și extensibilă.

Rezultatul final al proiectului este o aplicație reușită, care îndeplinește obiectivele propuse. Interfața intuitivă dezvoltată în React oferă o experiență plăcută utilizatorilor, iar backend-ul implementat în Spring Boot asigură performanța și securitatea aplicației.

Aplicația finală îndeplinește următoarele obiective:

- Crearea, editarea și ștergerea unui cont
- Adăugarea și ștergerea de interese
- Adăugarea, modificarea și ștergerea descrierii
- Adăugarea și ștergerea de poze
- Forum pentru discuții
- Modificarea pozei de profil
- Căutarea și vizualizarea altor utilizatori
- Acceptarea sau respingerea unui utilizator în lista ta
- Trimitere de email pentru inițierea discuției.

În viitor atât securitatea cât și funcționalitățile aplicației pot fii înflorite. Ca și ulterioare îmbunătățiri aș urmării aceste caracteristici și obiective:

- Adăugarea de recenzii per utilizator
- Crearea de excursii la care utilizatorii pot solicita înscrierea
- Creșterea securității prin validarea emailului

Bibliografie

[1] Flavio Copes. *The react beginner's handbook.* Self-publishing, 2020.

[2] Anusheh Zohair Mustafeez. *What is VS Code?* 2023.
https://www.educative.io/answers/what-is-visual-studio-code.

[3] Chantelle Little. *An introduction to web development technologies.* Septembrie 2019.
https://tillerdigital.com/blog/an-introduction-to-web-development-technologies.

[4] *IntelliJ IDEA overview.* Februarie 2023.
https://www.jetbrains.com/help/idea/getting-started.html.

[5] Jamie Chan. *Learn Java in one day and LEARN IT WELL 2nd Edition.* LFC Publishing, 2022

[6] Laura Lemay,. *An Introduction to Java Programming.* Sams.net, 2010.

[7] *LIQUIBASE DBMS Migration Tools.* Medisys , 2018.

[8] Martin Mois. *Hibernate Tutorial.* Exelixis Media, 2015.

[9] Sri vidya college of engineering & technology . „Lecture note, Introduction to OOP ." 2019.

[10] Vijay Kumari. *Java Basics.* Februarie 2023.
https://www.c-sharpcorner.com/article/a-complete-java-basics-tutorial/.

[11] Webb, Phillip, Dave Syer, Josh Long, și Stéphane Nicoll.
https://docs.spring.io/spring-boot/docs/current/reference/pdf/spring-boot-reference.pdf.

Listă de figuri