**PERDELIYEVA AKJEMAL**

**CSC AN1, GRUPA B**

Obiectiv:

Obiectivul acestei teme este familiarizarea cu șablonul architectural Model-View-Presenter. Pentru persistența informației se va utiliza o bază de date relațională (SQL Server, MySQL, etc.).

Cerințe:

❖ În faza de analiză se va realiza diagrama cazurilor de utilizare.

❖ În faza de proiectare se va realiza diagrama de clase respectând arhitectura MVP și principiile SOLID, dar și diagrama entitate-relație corespunzătoare bazei de date.

❖ În faza de implementare se va scrie cod pentru îndeplinirea tuturor funcționalităților precizate de diagrama cazurilor de utilizare utilizând:

➢proiectarea dată de diagrama de clase;

➢unul dintre următoarele limbaje de programare: C#, C++, Java, Python.

❖ În faza de testare se vor implementa teste unitate(proiect de testare)corespunzătoare operațiilor interogare a tabelelor bazei de date.

❖ Finalizarea temei va consta în predarea unui director ce va cuprinde:

➢Unfișier cu diagramele UML realizate;

➢Baza de date;

➢Aplicația soft;

➢Documentația (minim 10 pagini)-un fișier care cuprinde:

▪numele studentului, grupa;

▪enunțul problemei;

▪instrumente utilizate;

▪justificarea limbajului de programare ales;

▪descrierea diagramelor UML;

▪descrierea aplicației.

**Problema 15**

Dezvoltați o aplicație care poate fi utilizată de către o agenție de rezervare bilete de tren. Aplicația va avea 3 tipuri de utilizatori: călător, angajat și administrator.

Utilizatorii de tip călător pot efectua următoarele operații fără autentificare:

❖Vizualizarea listei trenurilor după stație de plecare, destinație, durată ordonată după numărul trenurilor;

❖Vizualizarea listei trenurilor dintre 2 locații, inclusiv preț și disponibilitate locuri libere;

❖Căutarea unui tren după număr.

Utilizatorii de tip angajat pot efectua următoarele operații după autentificare:

❖Toate operațiile permise utilizatorilor de tip călător și vânzarea unui bilet către un călător;

❖Operații CRUD în ceea ce privește persistența trenurilor și biletelor vândute;

❖Salvarea listelor cu informații despre trenuri în formatele csv, json, xml, txt;

❖Vizualizarea unor statistici legate de trenuri utilizând grafice (structură radială, structură inelară, de tip coloană, etc.).

Utilizatorii de tip administrator pot efectua următoarele operații după autentificare:

❖Toate operațiile permise utilizatorilor de tip călător;

❖Operații CRUD pentru informațiile legate de utilizatorii care necesită autentificare;

❖Vizualizarea listei utilizatorilor care necesită autentificare;

❖Notificarea fiecărui utilizator care necesită autentificare prin cel puțin 2 variante (email, SMS, WhatsApp, Skype, etc.) la orice modificare a informațiilor de autentificare aferente acelui utilizator.

**Instrumente utilizate**

Pentru dezvoltarea proiectului Agency, am utilizat o serie de instrumente și tehnologii care mi-au permis să implementez cu succes arhitectura MVP (Model-View-Presenter) a aplicației.

1. Maven - Acesta este un instrument de management al dependențelor pentru proiecte Java. Am ales pentru că este un instrument foarte popular în comunitatea Java și ușor de folosit, ceea ce a permis dezvoltatorilor să gestioneze cu ușurință dependențele bibliotecilor din cadrul proiectului.

În plus, Maven-ul a fost utilizat pentru a construi proiectul și pentru a genera rapoarte.

2. Swing și AWT - Acestea sunt framework-uri pentru crearea interfețelor grafice de utilizator în Java. Am folosite pentru a crea interfața grafică a aplicației.

Swing este o librărie GUI (Graphical User Interface) care oferă un set de componente grafice personalizabile pentru crearea de interfețe grafice de utilizator. AWT (Abstract Window Toolkit) este o altă librărie GUI din Java care oferă un set de componente grafice.

Am ales să utilizez Swing și AWT datorită flexibilității lor în ceea ce privește personalizarea interfeței grafice. Aceste framework-uri oferă posibilitatea de a crea interfețe grafice profesionale, cu o experiență de utilizare excelentă pentru utilizatori.

3. IntelliJ IDEA - Acesta este un mediu integrat de dezvoltare Java (IDE) care oferă un set larg de instrumente pentru dezvoltarea de aplicații Java. Am folosit IntelliJ IDEA pentru a dezvolta codul sursă al proiectului.

Am ales IntelliJ IDEA datorită capacității sale de a îmbunătăți productivitatea dezvoltatorilor prin oferirea de instrumente puternice de completare a codului, de depanare și de refactorizare. În plus, IDE-ul oferă o interfață prietenoasă pentru utilizator și este ușor de utilizat.

5. Hibernate - un framework ORM (Object-Relational Mapping) utilizat pentru a facilita interacțiunea cu baza de date. Acesta permite dezvoltatorilor să utilizeze obiecte Java pentru a manipula datele din baza de date relațională, eliminând astfel necesitatea de a scrie cod SQL manual și de a face conversii între tipurile de date Java și cele de la nivelul bazei de date. Hibernate se integrează perfect cu JPA (Java Persistence API) și oferă o multitudine de funcționalități pentru gestionarea tranzacțiilor, a cache-ului, a relațiilor între entități și multe altele.

6. Javax Persistence API - un set de interfețe și clase Java utilizate pentru a defini și gestiona entități (obiecte care sunt stocate în baza de date) într-un mod consistent într-o aplicație Java. API-ul definește o serie de anotări și metode pentru a realiza operațiuni CRUD (create, read, update, delete) pe obiecte și asigură portabilitatea între diferite implementări ale acestuia.

7. MySQL Workbench - o unealtă grafică utilizată pentru a administra bazele de date MySQL. Acesta oferă o interfață intuitivă pentru crearea și gestionarea schemelor bazei de date, tabelelor, indiciilor și a altor obiecte ale bazei de date. MySQL Workbench poate fi utilizat și pentru a executa interogări SQL, a importa și exporta date și a monitoriza performanța bazei de date.

În final, combinând aceste instrumente, am dezvoltat o aplicație MVP (Model-View-Presenter) pentru o agenție de turism. Am utilizat Maven ca sistem de management al proiectelor și depedențelor, Swing și AWT pentru a dezvolta interfața grafică a utilizatorului, Hibernate și Javax Persistence API pentru a manipula datele din baza de date și MySQL Workbench pentru a administra baza de date. Am utilizat IntelliJ IDEA ca mediu integrat de dezvoltare. Această combinație de instrumente a permis dezvoltarea unei aplicații stabile, extensibile și ușor de utilizat pentru utilizatorii finali.

**Justificarea limbajului de programare ales**

Am ales să folosesc Java datorită mai multor motive. În primul rând, Java este unul dintre cele mai populare și fiabile limbaje de programare disponibile astăzi. Este cunoscut pentru securitatea sa, fiabilitatea și portabilitatea sa, ceea ce îl face foarte potrivit pentru dezvoltarea aplicațiilor desktop. De asemenea, Java are o comunitate mare și activă, ceea ce face ca dezvoltarea cu Java să fie ușoară și accesibilă.

În al doilea rând, Java are o sintaxă ușor de înțeles și de scris. Este un limbaj orientat obiect, ceea ce înseamnă că este structurat în jurul conceptelor de obiecte, ceea ce face ca codul să fie ușor de înțeles și de întreținut. Java are, de asemenea, o mulțime de biblioteci și cadre care sunt disponibile pentru a face dezvoltarea mai ușoară și mai rapidă.

În plus, Java este cunoscut pentru faptul că are o performanță ridicată și poate gestiona o cantitate mare de date. Acest lucru este important în proiectul meu, deoarece dezvoltăm o aplicație desktop pentru o agenție, care poate avea nevoie să gestioneze o cantitate mare de date.

În concluzie, am ales să folosesc Java în proiectul meu pentru că este un limbaj popular, fiabil și ușor de înțeles. De asemenea, are o performanță ridicată și o comunitate activă și o mulțime de biblioteci disponibile pentru dezvoltarea rapidă și ușoară a aplicațiilor desktop.

**Descrierea aplicației**

Aplicația mea „Agency” este realizata după cerințele din problema 15. Aplicatia are în total 3 ferestre pentru 3 tipuri de utilizatori calator, angajat si administrator. Administrator si angajat trebuie sa se logeze ca sa poată utiliza aplicația. Codul sursa deține 3 pachete denumite model, presenter și view. Am 3 entități pentru bilete, trenuri si utilizatori. Am și clasele responsabili pentru persistențe entităților.

Clasa "Ticket" reprezintă un model pentru un bilet de tren. Acesta conține informații precum numărul biletului, numărul trenului, locul în tren și prețul biletului. Clasa este marcată cu anotări JPA (Java Persistence API) pentru a fi utilizată cu o bază de date relațională.

Clasa "TicketRepo" reprezintă un repository (repozitoriu) pentru entitățile de tip "Ticket". Aceasta conține metode pentru a gestiona biletul în baza de date, inclusiv pentru a salva, actualiza și șterge bilete, precum și pentru a obține o listă de toate biletele din baza de date. Clasa utilizează JPA pentru a interacționa cu baza de date.

Clasa Train reprezintă o entitate a unei baze de date, cu informații despre trenuri. Aceasta include informații cum ar fi numărul de tren, stația de plecare, destinația, ora de plecare și sosire, precum și capacitatea de locuri.

Clasa TrainRepo este utilizată pentru a accesa și manipula datele din baza de date legate de trenuri. Aceasta conține metode pentru a salva, găsi, actualiza și șterge trenuri din baza de date. Mai specific, ea oferă metode pentru a obține un tren prin ID, pentru a obține o listă de trenuri în funcție de stația de plecare și destinație, pentru a obține toate trenurile din baza de date, pentru a actualiza un tren existent și pentru a șterge un tren existent.

Eu am creat o clasă numită User în cadrul unui pachet numit org.agency.model. Această clasă are câteva proprietăți, precum numele de utilizator (username), parola (password), rolul (role), prenumele (firstName), numele de familie (lastName) și adresa de email (email), dar și un CNP (cod numeric personal), care este unic pentru fiecare utilizator.

Am creat și o clasă numită UserRepo în același pachet, care este utilizată pentru a accesa și manipula obiectele User în baza de date. Am utilizat JPA (Java Persistence API) pentru a face acest lucru.

Mai concret, clasa `AdminPresenter` conține metode care permit adaugarea de utilizatori noi, afisarea utilizatorilor autentificati, afisarea utilizatorilor neautentificati, actualizarea si stergerea de utilizatori existenti. Pentru aceasta, se foloseste o instanta a clasei `UserRepo`, care contine metode pentru a interactiona cu baza de date.

Clasa `AdminView` este interfata grafica a aplicatiei si contine o serie de campuri si butoane care permit utilizatorului sa interactioneze cu aplicatia. Aceasta clasa implementeaza interfata `AdminInterface`, care contine metodele necesare pentru a comunica cu clasa `AdminPresenter`. Astfel, clasa `AdminView` primeste datele de la utilizator si le transmite catre `AdminPresenter` pentru a fi procesate, iar apoi afiseaza rezultatele in interfata grafica.

O imagine care conține diagramă

Descriere generată automat

O imagine care conține diagramă

Descriere generată automat

O imagine care conține diagramă

Descriere generată automat

package org.agency.model;  
  
import org.junit.jupiter.api.\*;  
  
import javax.persistence.EntityManager;  
import javax.persistence.EntityManagerFactory;  
import javax.persistence.Persistence;  
import java.util.List;  
  
import static org.junit.jupiter.api.Assertions.\*;  
  
class TicketRepoTest {  
  
 private TicketRepo ticketRepo;  
 private EntityManagerFactory emf;  
 private EntityManager em;  
  
 @BeforeEach  
 void setUp() {  
 emf = Persistence.createEntityManagerFactory("test");  
 em = emf.createEntityManager();  
 ticketRepo = new TicketRepo();  
 ticketRepo.emf = emf;  
 ticketRepo.em = em;  
 }  
  
 @AfterEach  
 void tearDown() {  
 em.close();  
 emf.close();  
 }  
  
 @Test  
 void save() {  
 // given  
 Ticket ticket = new Ticket(1, 10, 5, 15.0);  
  
 // when  
 ticketRepo.save(ticket);  
  
 // then  
 Ticket retrievedTicket = em.find(Ticket.class, ticket.getTicketNumber());  
 assertNotNull(retrievedTicket);  
 assertEquals(ticket, retrievedTicket);  
 }  
  
 @Test  
 void getAllTickets() {  
 // given  
 Ticket ticket1 = new Ticket(1, 10, 5, 15.0);  
 Ticket ticket2 = new Ticket(2, 12, 7, 17.0);  
 em.getTransaction().begin();  
 em.persist(ticket1);  
 em.persist(ticket2);  
 em.getTransaction().commit();  
  
 // when  
 List<Ticket> tickets = ticketRepo.getAllTickets();  
  
 // then  
 assertNotNull(tickets);  
 assertEquals(2, tickets.size());  
 assertTrue(tickets.contains(ticket1));  
 assertTrue(tickets.contains(ticket2));  
 }  
  
 @Test  
 void deleteTicket() {  
 // given  
 Ticket ticket = new Ticket(1, 10, 5, 15.0);  
 em.getTransaction().begin();  
 em.persist(ticket);  
 em.getTransaction().commit();  
  
 // when  
 ticketRepo.deleteTicket(ticket.getTicketNumber());  
  
 // then  
 Ticket retrievedTicket = em.find(Ticket.class, ticket.getTicketNumber());  
 assertNull(retrievedTicket);  
 }  
  
 @Test  
 void updateTicket() {  
 // given  
 Ticket ticket = new Ticket(1, 10, 5, 15.0);  
 em.getTransaction().begin();  
 em.persist(ticket);  
 em.getTransaction().commit();  
  
 // when  
 ticket.setTrain(11);  
 ticket.setPrice(20.0);  
 ticketRepo.updateTicket(ticket);  
  
 // then  
 Ticket retrievedTicket = em.find(Ticket.class, ticket.getTicketNumber());  
 assertNotNull(retrievedTicket);  
 assertEquals(ticket, retrievedTicket);  
 }  
}