

PROIECT DISCIPLINA POO

[Aplicatie de gestiune a unei agentii de turism in C++]

Autor

Ghidoarca Petru-Cristi
Grupa:3122B
Anul de studiu:2

TEMA PROIECT

TEMA SI MOTIVATIA ALEGERII

FORMULAREA TEMEI

Tema ecestui proiect il reprezinta implementarea unei aplicatii de gestiune a unei agentii de turism.

MOTIVAREA ALEGERII TEMEI

In aceasta aplicatie,salvarea datelor despre clienti si ofertele ce le da aceasta agentie de turism pot fi salvate in doua tipuri de fisiere(.txt si .json).

In realizarea aplicatiilor C++,utilizarea fisiereilor text nu poate fi o solutie optima pentru salvarea acestor tipuri de date,fisierele text nu ofera posibilitatea de a sterge datele din interiorul acestora,facand aceasta functionalitate destul de dificil de implementat deoarece trebuie facut o copie a tuturor datelor din acel fisier,cu exceptia celei pe care o vrei sa o stergi,intr-un fisier nou si stergerea complete a fisierului vechi,facand modificarile astea sa fie din ce in ce mai complicat.

Spre deosebire de acest tip de fisier,fisierele .json(JavaScript Oriented Notation)fac posibila modificarea in timp real a datelor acestora fara crearea unui fisier nou si sunt foarte versatile .

Singura utilizare a fisiereilor text in aceasta aplicatie s-a facut in functia de logare a clientilor,prin salvarea datelor de inregistrare si a parolei unice.

CUPRINS

Cuprins

TEMA SI MOTIVATIA ALEGERII.....	2
1. ELEMENTE TEORETICE	4
1.1. DESCRIEREA PROBLEMEI	4
1.2. ABORDAREA TEORETICA A PROBLEMEI.....	5
1.3. ELEMENTE SPECIFICE POO	5
1.4. ALTE CAPITOLE SPECIFICE	6
2. IMPLEMENTARE.....	11
2.1. TEHNOLOGII FOLOSITE	11
2.2. DIAGRAMA DE CLASE.....	12
2.3. ALTE CAPITOLE SPECIFICE	Error! Bookmark not defined.
3. ANALIZA SOLUTIEI IMPLEMENTATE	13
3.1. FORMATUL DATELOR DE I/O	13
3.2. STUDII DE CAZ	13
3.3. PERFORMANTE OBTINUTE	14
4. MANUAL DE UTILIZARE	16
5. CONCLUZII.....	20
6. BIBLIOGRAFIE	21
6.1. CARTI.....	21
6.2. ARTICOLE.....	Error! Bookmark not defined.
6.3. SURSE BIBLIOGRAFICE DIVERSE.....	21

CAPITOLUL I

1. ELEMENTE TEORETICE

1.1. DESCRIEREA PROBLEMEI

Această aplicație reprezintă un program simplu pentru gestionarea unei agenții de turism. Programul permite unui administrator să adauge și să șteargă pachete turistice, să adauge și să șteargă turiști și să vizualizeze rezervările existente. Utilizatorii pot să se înregistreze și să se autentifice, să vizualizeze ofertele turistice, să caute oferte specifice, să facă rezervări și să vizualizeze istoricul rezervărilor.

Programul începe prin afișarea unui meniu de selecție a rolului: administrator sau utilizator. După selectarea rolului, utilizatorul poate introduce parola de administrator pentru a avea acces la funcționalitățile de administrare, sau poate alege să se autentifice sau să se înregistreze ca utilizator obișnuit.

Dacă utilizatorul alege să se autentifice, acesta trebuie să introducă un nume de utilizator și o parolă. Funcția `login()` verifică existența unui fișier de utilizatori și autentifică utilizatorul în funcție de datele introduse. Dacă autentificarea este reușită, utilizatorul poate accesa meniul principal.

Meniul administratorului oferă următoarele opțiuni:

- Adăugare pachet turistic: Administratorul poate introduce numele, descrierea și prețul unui pachet turistic, iar apoi acesta este adăugat în agenția de turism.
- Afișare pachete turistice: Se afișează lista cu toate pachetele turistice disponibile în agenție.
- Adăugare turist: Administratorul poate introduce numele și prenumele unui turist, iar acesta este adăugat în lista de turiști.
- Ștergere turist: Administratorul poate introduce ID-ul unui turist pentru a-l șterge din lista de turiști.
- Vizualizare rezervări: Se afișează lista cu toate rezervările existente.
- Afișare turiști: Se afișează lista cu toți turiștii înregistrați.
- Ștergere pachet turistic: Administratorul poate introduce ID-ul unui pachet turistic pentru a-l șterge din agenție.

Meniul utilizatorului oferă următoarele opțiuni:

- Vizualizare oferte turistice: Se afișează lista cu toate pachetele turistice disponibile în agenție.
- Căutare oferte turistice: Utilizatorul poate căuta oferte turistice în funcție de un criteriu specific.
- Rezervare pachete turistice: Utilizatorul poate selecta un pachet turistic și să facă o rezervare.
- Vizualizare istoric rezervări: Se afișează istoricul rezervărilor utilizatorului.
- Deconectare: Utilizatorul se poate deconecta din contul său.

În funcția `main()`, utilizatorii pot selecta opțiunile dorite prin intermediul unui meniu și pot interacționa cu programul până când aleg să se deconecteze.

1.2. ABORDAREA TEORETICA A PROBLEMEI

Tema are drept scop implementarea unei aplicații de turism care să permită administrarea pachetelor turistice și rezervarea acestora. Codul prezentat conține două fișiere, `main.cpp` și `Classes.cpp`, care conțin implementarea funcționalităților necesare pentru temă.

Elemente teoretice utilizate:

- **Operarea cu fișiere:** În cod se realizează operații de citire și scriere în fișiere text pentru stocarea datelor utilizatorilor și a rezervărilor. Funcția `login()` verifică dacă un utilizator există în "users.txt", iar funcția `registerUser()` salvează datele unui utilizator în fișierul respectiv. De asemenea, funcția `stergeTurist()` din clasa `AgentieTurism` șterge turistul dorit din vectorul `m_turisti` și actualizează "turist.json". Pentru a lucra cu fișiere în limbajul C++, se folosesc clasele `ifstream` (pentru citire) și `ofstream` (pentru scriere), care permit deschiderea și manipularea fișierelor.
- **Modul text:** Interacțiunea cu utilizatorul se realizează prin intermediul consolei, unde se afișează un meniu și se așteaptă introducerea unei opțiuni de la tastatură. Mesajele text sunt afișate utilizând funcția `cout` din biblioteca `<iostream>`, iar citirea opțiunilor introduse de utilizator se realizează cu ajutorul funcției `cin`.

1.3. ELEMENTE SPECIFICE POO

Elementele specifice programării orientate pe obiecte (POO) aplicate în rezolvarea problemei prezentate includ:

1. **Clasele:** În aplicație, sunt definite mai multe clase, cum ar fi clasa "Turist", clasa "PachetTuristic" și clasa "AgentieTurism". Fiecare clasă definește atribute

specifice și metode asociate pentru a manipula și gestiona datele și funcționalitățile specifice.

2. **Obiecte:** Utilizarea claselor permite crearea de obiecte specifice în timpul rulării programului. De exemplu, obiectele de tip "Turist", "PachetTuristic" și "AgentieTurism" sunt create și utilizate în codul dat pentru gestionarea tuturor informațiilor în legătura cu turiștii, pachetele turistice și agențiile de turism.
3. **Encapsularea:** Clasele programului utilizează encapsularea pentru ascunderea datelor interne și expunerea prin intermediul unor metode publice. Atributele clasei sunt definite ca variabile de tip privat, iar metodele publice sunt utilizate pentru accesarea și manipularea acestor atribute. Un exemplu ar fi clasa "Turist" ce are metodele "getNum()", "getPrenume()" și "getId()" pentru accesarea atributelor private ale unui obiect dat de clasa "Turist".
4. **Constructorii:** Constructorii sunt utilizați pentru initializarea obiectelor claselor respective cu valorile initiale specifice. În cod, clasa "Turist" are un constructor definit care primește numele și prenumele turistului și inițializează atributele corespunzătoare ale obiectului.
5. **Metodele de acces (getters) și metodele de modificare (setters):** Aceste metode permit accesarea și modificarea valorilor atributelor private ale unui obiect. De exemplu, în clasa "Turist", metodele "getNum()", "getPrenume()" și "getId()" sunt metode de acces, care returnează valorile atributelor private.
6. **Mostenirea:** Clasa "AgentieTurism" moștenește clasa "PachetTuristic", ceea ce înseamnă că "AgentieTurism" primește toate atributele și metodele definite în "PachetTuristic" și poate adăuga sau suprascrie comportamente specifice.
7. **Polimorfism:** Polimorfismul este folosit în program prin suprascrierea metodelor din clasele derivate. De exemplu, clasa "AgentieTurism" moștenește clasa "PachetTuristic" și suprascrie metoda "afiseazaPachete()" pentru a afișa informații specifice unui pachet.

Acestea sunt doar câteva dintre elementele specifice programării orientate pe obiecte utilizate în aplicație.

1.4. ALTE CAPITOLE SPECIFICE

1.4.1. *Serializare și deserializare JSON*

Serializarea și deserializarea **JSON** sunt procese utilizate pentru a converti obiecte sau structuri de date în format **JSON** și invers. **JSON (JavaScript Object Notation)** este un format de date ce poate fi ușor de citit și de generat de către persoane și de interpretat de către mașini. Este utilizat frecvent în aplicațiile web pentru a transfera și stoca date structurate.

Pentru aplicarea serializării și deserializării JSON în aplicație, se va utiliza biblioteca **nlohmann/json**, ce este o bibliotecă a limbajului C++ folosită pentru manipularea JSON.

Această bibliotecă permite serializarea și deserializarea a obiectelor C++ în format JSON într-un mod mai ușor.

Pentru a face serializarea și deserializarea **JSON** în cadrul aplicației, trebuie efectuate următoarele acțiuni:

1. **Includerea bibliotecii `nlohmann/json`:** La începutul fișierului `main.cpp`, trebuie adăugată linia `#include <nlohmann/json.hpp>` pentru a include biblioteca JSON.
2. **Serializarea obiectului "bookings":** Pentru a serializa obiectul vector "bookings" în format JSON, vom crea un obiect JSON de tip array și apoi vom itera prin fiecare rezervare și vom adăuga informațiile relevante în obiectul JSON creat. De exemplu, pentru fiecare rezervare, putem crea un obiect JSON cu cheile "**name**", "**quantity**" și "**totalPrice**" și valori corespunzătoare. Apoi, vom adăuga acest obiect JSON în array-ul JSON. La final, putem utiliza funcția "**json::dump()**" pentru a serializa obiectul JSON într-un șir de caractere și putem scrie acest șir de caractere într-un fișier JSON utilizând funcția "`std::ofstream`". Pentru a serializa datele utilizatorului într-un obiect JSON, poți face următoarele:

```
// Clasa reprezentand o oferta turistica
class TouristOffer {
public:
    TouristOffer(string name, string description, int price, int ID)
        : name(name), description(description), price(price), id(ID)
    {}

    string getName() const { return name; }
    string getDescription() const { return description; }
    double getPrice() const { return price; }

private:
    string name;
    string description;
    int price, id;
};

// Clasa reprezentand o rezervare
class Booking {
public:
    Booking(TouristOffer offer, int quantity)
        : offer(offer), quantity(quantity) {}

    TouristOffer getOffer() const { return offer; }
    int getQuantity() const { return quantity; }
    double getTotalPrice() const { return offer.getPrice() * quantity; }

    // Serializeaza rezervarea la JSON
    json toJson() const {
        return {
            {"Numele Ofertei", offer.getName()},
            {"Cantitatea", quantity},
            {"Pret Total", getTotalPrice()}
        };
    }

private:
    TouristOffer offer;
    int quantity;
};

// Funcție pentru efectuarea unei rezervări
void makeBooking(vector<Booking>& bookings) {
```

```

// Încarca ofertele turistice disponibile dintr-un fișier JSON
std::ifstream file("C:/Users/Asus/Desktop/pachete_turistice.json");
nlohmann::json j;
file >> j;

// Creeaza un vector de obiecte TouristOffer din datele JSON
vector<TouristOffer> offers;
for (const auto& offer : j) {
    TouristOffer touristOffer(offer["nume"], offer["descriere"],
offer["pret"], offer["id"]);
    offers.push_back(touristOffer);
}

// Arată ofertele turistice disponibile
for (int i = 0; i < offers.size(); ++i) {
    cout << "[" << i << "]" ";
    cout << "Nume: " << offers[i].getName() << endl;
    cout << "Descriere: " << offers[i].getDescription() << endl;
    cout << "Pret: " << offers[i].getPrice() << endl;
}

// Obține oferta de alegere a utilizatorului
int offerIndex;
cout << "Introdu numarul ofertei: ";
cin >> offerIndex;

// Obține cantitatea dorită de utilizator
int quantity;
cout << "Introdu cantitatea: ";
cin >> quantity;

// Creeza un obiect de rezervare și îl adauga la vectorul
rezervărilor
Booking booking(offers[offerIndex], quantity);
bookings.push_back(booking);

cout << "Rezervare facuta!" << endl;
}

// Funcție pentru scrierea rezervărilor în fișierul JSON
void writeBookingsToJson(const vector<Booking>& bookings, const string&
filename) {
    ofstream file(filename);
    if (file.is_open()) {
        json data;
        for (const auto& booking : bookings) {
            data.push_back(booking.toJson());
        }
        file << data.dump(4);
    }
}

```

3. **Deserializarea obiectului "bookings":** Pentru a deserializa un fișier JSON și a obține un obiect vector "bookings" din el, vom a citi și a parse fișierul JSON și vom obține un obiect JSON corespunzător. Apoi, vom itera prin fiecare obiect JSON din array și vom extrage informațiile relevante pentru fiecare rezervare. Vom utiliza aceste informații pentru a crea obiecte de tip Booking și le vom adăuga în vectorul "bookings". De exemplu, pentru a deserializa datele utilizatorului, poți face următoarele:


```

Functia din Classes.cpp:// Funcție pentru citirea rezervărilor din
fișierul JSON
void readBookingsFromJson(vector<Booking>& bookings, const string&
filename) {
    ifstream file(filename);
    if (file.is_open()) {
        json data;
        file >> data;

        for (const auto& bookingData : data) {
            string offerName = bookingData["Numele Ofertei"];
            int quantity = bookingData["Cantitatea"];
            double totalPrice = bookingData["Pret Total"];

            TouristOffer touristOffer(offerName, "", totalPrice, 0);
            Booking booking(touristOffer, quantity);
            bookings.push_back(booking);
        }

        file.close();
    }
}

Secventa de cod din main.cpp:
readBookingsFromJson(bookings, "C:/Users/Asus/Desktop/rezervari.json");
if (bookings.empty()) {
    cout << "Fara rezervari." << endl;
} else {
    cout << "Rezervari:" << endl;
    for (const auto& booking : bookings) {
        cout << "Oferta: " << booking.getOffer().getName() << endl;
        cout << "Cantitate: " << booking.getQuantity() << endl;
        cout << "Pret total: " << booking.getTotalPrice() << endl <<
endl;
    }
}

```

1.4.2. STL (Standard Template Library)

Pentru a rezolva problema, codul utilizează clasa **AgentieTurism** și alte clase ce gestionează informațiile despre pachetele turistice, turiști și rezervări. În plus, se mai utilizează și STL (Standard Template Library) pentru a stoca și manipula aceste date.

În cadrul clasei **AgentieTurism**, sunt definite următoarele funcționalități:

2. **Adăugare turist:** Metoda **adaugaTurist** primește un obiect de tip **Turist** și îl adaugă în vectorul **m_turisti** care stochează turiștii agenției. De asemenea, se actualizează și fișierul JSON "turist.json" cu datele actualizate.
3. **Ștergere turist:** Metoda **stergeTurist** primește un id de turist și elimină turistul corespunzător din vectorul **m_turisti**. De asemenea, se actualizează și fișierul JSON "turist.json" cu datele actualizate.
4. **Adăugare pachet turistic:** Metoda **adaugaPachetTuristic** primește un obiect de tip **PachetTuristic** și îl adaugă în vectorul de pachete turistice al agenției. De asemenea, se actualizează și fișierul JSON "pachete_turistice.json" cu datele actualizate.
5. **Ștergere pachet turistic:** Metoda **stergePachetTuristic** elimină pachetul corespunzător din vectorul de pachete turistice al agenției în funcție de id-ul pachetului turistic primit. De asemenea, se actualizează și fișierul JSON "pachete_turistice.json" cu datele actualizate.
6. **Afișare pachete turistice:** Metoda **afiseazaPachete** are rolul de a afișa informațiile despre toate pachetele turistice din vectorul de pachete turistice al agenției.

7. **Afișare turiști:** Aceasta metoda (**afiseazaTuristi**) afișează toate informațiile date în legătura cu toți turiștii din vectorul **m_turisti**.
8. **Căutare pachete turistice:** Metoda **CautaPacheteTuristice** primește un caracter sau șir de caractere și în funcție de acesta caută pachetele turistice care conțin șirul în nume sau descriere. În final afișează informațiile despre pachetele turistice găsite.
9. **Înregistrare rezervare:** Metoda **makeBooking** primește rezervările și permite utilizatorului să înregistreze o rezervare pentru un anumit pachet turistic. Rezervarea constă în selectarea unui pachet turistic și specificarea cantității după care este adăugată în vectorul de rezervări.
10. **Scriere rezervări în fișier JSON:** Metoda **writeBookingsToJson** primește vectorul de rezervări și actualizează fișierul JSON "rezervari.json" cu datele actualizate.
11. **Citire rezervări din fișier JSON:** Metoda **readBookingsFromJson** are rolul de a citi rezervările din fișierul JSON "rezervari.json" și le adaugă lor într-un vector.

Acestea sunt doar câteva funcționalități din cadrul clasei **AgentieTurism**. De asemenea, este posibil să existe și alte funcționalități și clase auxiliare, în funcție de cerințele și structura specifică a programului.

1.4.3. Manipularea fișierelor

Manipularea fișierelor este utilizată în acest cod pentru a stoca și accesa datele utilizatorilor și rezervările în fișiere. Mai jos sunt explicate în detaliu funcțiile care efectuează aceste operații:

1. **bool login():** Această funcție se ocupă de autentificarea utilizatorului. Ea deschide "users.txt" pentru a verifica dacă există un utilizator cu datele introduse (numele de utilizator și parola). În cazul în care utilizatorul este autentificat cu succes, funcția returnează **true**, altfel returnează **false**.
 2. **void registerUser():** Această funcție permite înregistrarea unui nou utilizator în aplicație. Aceasta primește informațiile de la utilizator și le salvează în fișierul "users.txt". Utilizatorul este creat sub forma unei structuri **User** și informațiile sale sunt scrise în fișier folosind un obiect de tip **ofstream**.
 3. **void stergeTurist(int id):** Această funcție este utilizată în clasa **AgentieTurism** și elimină un turist din lista de turiști pe baza ID-ului introdus. În plus față de eliminarea turistului din vectorul **m_turisti**, funcția încarcă datele JSON din fișierul "turist.json", elimină datele turistului cu ID-ul specificat și rescrie obiectul JSON actualizat înapoi în fișier.
 4. **void adaugaTurist(const Turist& turist):** Această funcție este utilizată în **AgentieTurism** și adaugă un turist în lista de turiști. Funcția adaugă turistul la vectorul **m_turisti** și apoi creează un obiect JSON și îl populează cu datele turistului. Obiectul JSON este apoi scris în fișierul "turist.json".
 5. **void writeBookingsToJson(const vector<Booking>& bookings, const string& fileName):** Această funcție primește un vector de rezervări și un nume de fișier și scrie rezervările în format JSON în fișierul specificat. Funcția utilizează biblioteca **"nlohmann/json"** pentru a crea și manipula obiecte JSON și folosește un obiect de tip **ofstream** pentru a scrie datele în fișier.
-

CAPITOLUL II

12. IMPLEMENTARE

În cadrul rezolvării acestei teme, au fost definite mai multe clase și funcții pentru gestionarea diferitelor operațiuni necesare în aplicație.

Structura `'User'` este definită pentru stocarea informațiilor despre utilizatori, cum ar fi numele, prenumele, numele de utilizator, parola, data nașterii și adresa.

Funcția `'login()'` permite utilizatorilor să se autentifice prin introducerea numelui de utilizator și a parolei. Ea deschide un fișier de utilizatori și verifică dacă datele introduse sunt aceleași cu cele existente în fișier. Dacă acestea se aseamănă, se afișează un mesaj de autentificare reușită; altfel, se afișează un mesaj de eroare.

Funcția `'registerUser()'` permite utilizatorilor să se înregistreze, colectând informațiile necesare despre ei și salvându-le într-un fișier de utilizatori. Dacă fișierul nu poate fi deschis, se afișează un mesaj de eroare.

Funcția `'main()'` este punctul de intrare în program. La început, se cere să se selecteze o opțiune dintre `"Administrator"` și `"Utilizator"`. În funcție de opțiunea aleasă, utilizatorul este direcționat către diferite ramuri ale codului.

În cazul în care utilizatorul selectează opțiunea `"Administrator"`, i se cere să introducă parola de administrator data în cod. Dacă parola introdusă corespunde parolei din cod, utilizatorul are acces meniul de administrator ce are mai multe opțiuni, cum ar fi adăugarea unui pachet turistic, afișarea pachetelor turistice existente, adăugarea și ștergerea de turiști etc.

În cazul în care utilizatorul vrea să selecteze opțiunea `"Utilizator"`, acesta are opțiunea de a se autentifica sau înregistra. Dacă utilizatorul alege să se autentifice, se folosește funcția `'login()'` pentru a verifica datele introduse. Dacă autentificarea este reușită, utilizatorul are acces la un meniu principal ce are diferite opțiuni ca vizualizarea ofertelor turistice, căutarea ofertelor, rezervarea pachetelor și vizualizarea istoricului rezervărilor.

12.1. TEHNOLOGII FOLOSITE

Tehnologiile și limbajele de programare utilizate în construcția acestui proiect sunt limbajul de programare C++ și biblioteca standard a acestuia. În plus, se mai utilizează și biblioteca `'<iostream>'` pentru citirea și scrierea în consolă, `'<string>'` pentru lucrul cu șiruri de caractere, `'<fstream>'` pentru lucrul cu fișiere, `'<vector>'` pentru gestionarea vectorilor, `'<algorithm>'` pentru efectuarea de operații algoritmice, precum ștergerea și căutarea unui element într-un vector și pentru serializarea și deserializarea datelor în sau din formatul JSON, este folosită biblioteca **JSON** pentru C++, numită **nlohmann/json**.

Cu toate acestea, codul poate fi compilat și rulat într-un mediu de dezvoltare C++ precum **Microsoft Visual Studio**, **Xcode**, **Code::Blocks** sau în linia de comandă utilizând compilatoare C++ precum **g++** sau **clang++**. În cazul de față s-a utilizat aplicația **Clion** pentru implementarea acestei aplicații.

12.2. DIAGRAMA DE CLASE, SCHEMA BLOC, WORKFLOW



Figure 1. Digrama UML a claselor aplicatiei

CAPITOLUL III

13. ANALIZA SOLUTIEI IMPLEMENTATE

13.1. FORMATUL DATELOR DE I/O

1. Care este formatul datelor de intrare și cum trebuie să le interpretăm?

Pentru înregistrarea unui utilizator, se introduc următoarelor informații:

- **Numele utilizatorului**
- **Prenumele utilizatorului**
- **Nume de utilizator**
- **Parola**
- **Data nașterii**
- **Adresa**

Pentru autentificarea utilizatorului, se așteaptă introducerea numelui de utilizator și a parolei.

Pentru selectarea unei opțiuni în meniul principal, se așteaptă introducerea unui număr întreg.

2. Care sunt datele de ieșire și în ce format sunt prezentate?

- **Înregistrarea unui utilizator:** Se afișează un mesaj de succes sau de eroare.
- **Autentificarea utilizatorului:** Se afișează un mesaj de succes sau de eroare.
- **Vizualizarea ofertelor turistice:** Se afișează numele, descrierea și prețul fiecărui pachet turistic.
- **Căutarea ofertelor turistice:** Se afișează numele, descrierea și prețul pachetelor turistice care corespund criteriilor de căutare.
- **Rezervarea pachetelor turistice:** Se salvează rezervările într-un fișier JSON.
- **Vizualizarea istoricului rezervărilor:** Se afișează detaliile rezervărilor anterioare, inclusiv numele pachetului turistic, cantitatea și prețul total.

13.2. STUDIU DE CAZ

Acesta este un cod sursă în limbajul C++ care implementează o aplicație de gestionare a unei agenții de turism. Codul are două structuri principale: `User` și `Turist`, precum și clasele `PachetTuristic` și `AgentieTurism`.

Opțiunea de administrator necesită introducerea unei parole. Dacă parola este corectă, este afișat un meniu pentru administrator cu opțiuni precum adăugarea unui pachet turistic, afișarea pachetelor turistice, adăugarea unui turist, ștergerea unui turist, afișarea rezervărilor și ștergerea unui pachet turistic.

Opțiunea de utilizator permite unui utilizator să se autentifice sau să se înregistreze. Dacă utilizatorul se autentifică cu succes, este afișat un meniu pentru utilizator cu opțiuni precum

vizualizarea ofertelor turistice, căutarea ofertelor turistice, rezervarea pachetelor turistice, vizualizarea istoricului rezervărilor și deconectarea.

Codul folosește fișiere pentru a salva datele utilizatorilor și pachetelor turistice. Utilizatorii sunt stocați într-un fișier "users.txt", iar pachetele turistice sunt stocate într-un fișier "pachete_turistice.json".

13.3. PERFORMANTE OBTINUTE

Acest program conține mai multe aplicații pentru administrator și utilizator.

Aplicații pentru administrator:

- **Adăugare pachet turistic:** Administratorul poate introduce numele, descrierea și prețul unui pachet turistic, care va fi adăugat în lista de pachete turistice ale agenției.
- **Afișare pachete turistice:** Administratorul poate afișa lista de pachete turistice existente în agenție.
- **Adăugare turist:** Administratorul poate introduce numele și prenumele unui turist, care va fi adăugat în lista de turiști ai agenției.
- **Ștergere turist:** Administratorul poate introduce ID-ul unui turist și acesta va fi eliminat din lista de turiști ai agenției.
- **Afișare turiști:** Administratorul poate afișa lista de turiști existenți în agenție.
- **Ștergere pachet turistic:** Administratorul poate introduce ID-ul unui pachet turistic și acesta va fi eliminat din lista de pachete turistice ale agenției.

Aplicații pentru utilizator:

- **Logare:** Utilizatorul poate introduce un nume de utilizator și o parolă pentru a se autentifica în aplicație.
- **Vizualizare oferte turistice:** Utilizatorul poate vizualiza lista de pachete turistice existente în agenție.
- **Căutare oferte turistice:** Utilizatorul poate căuta pachete turistice după nume și descriere.
- **Rezervare pachete turistice:** Utilizatorul poate efectua rezervări pentru pachetele turistice disponibile, specificând cantitatea dorită. Rezervările vor fi salvate într-un fișier JSON.
- **Vizualizare istoric rezervări:** Utilizatorul poate vizualiza istoricul rezervărilor sale, care sunt citite din fișierul JSON.

Codul folosește o serie de clase și structuri pentru a organiza datele. Clasa "Turist" reprezintă un turist și conține informații despre nume, prenume și ID. Clasa "PachetTuristic" reprezintă un pachet turistic și conține informații despre nume, descriere, preț și ID. Clasa "AgentieTurism" reprezintă agenția de turism și conține metode pentru adăugarea și ștergerea de turiști și pachete turistice, precum și pentru afișarea acestora.

Codul folosește, de asemenea, biblioteca "nlohmann/json" pentru a manipula datele în format JSON. Aceasta este utilizată pentru citirea și scrierea rezervărilor dintr-un fișier JSON.

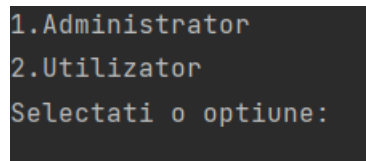
Performanțele codului pot fi evaluate în funcție de scopul specific al aplicației și de necesitățile utilizatorilor.

CAPITOLUL IV

14. MANUAL DE UTILIZARE

Interfața aplicației poate fi descrisă astfel:

1. La pornire, se afișează un meniu cu două opțiuni: **"Administrator"** și **"Utilizator"**.



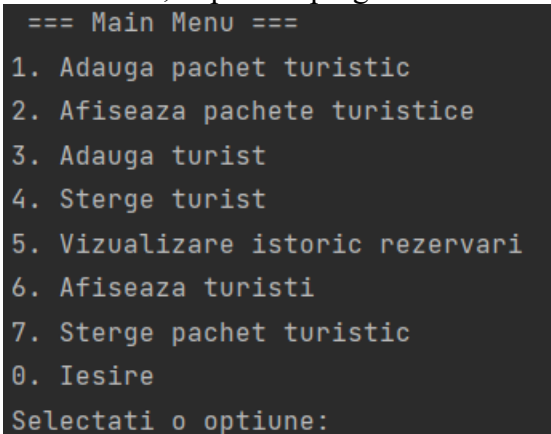
```
1.Administrator
2.Utilizator
Selectati o optiune:
```

Figure 2 *Meniul de administrare*

2. Utilizatorul trebuie să aleaga una dintre opțiuni prin introducerea unui număr corespunzător opțiunii dorite.

3. Dacă utilizatorul selectează opțiunea **"Administrator"**, programul va cere introducerea unei parole pentru autentificare.

- Dacă parola este introdusă corectă, se va afișa un meniu de administrare cu mai multe opțiuni.
- Dacă parola este introdusă gresit, se vor permite maxim 3 încercări de autentificare, după care programul se va închide.



```
=== Main Menu ===
1. Adauga pachet turistic
2. Afiseaza pachete turistice
3. Adauga turist
4. Sterge turist
5. Vizualizare istoric rezervari
6. Afiseaza turisti
7. Sterge pachet turistic
0. Iesire
Selectati o optiune:
```

Figure 3 *Meniul administratorului*

4. Dacă utilizatorul alege opțiunea **"Utilizator"**, programul va afișa un submeniu cu două opțiuni: **"Logare"** și **"Înregistrare"**.


```
1. Logare
2. Inregistrare
Selectati o optiune:1
```

Figure 4 Meniul utilizatorului ce permite sa faca un cont nou sau sa se logheze cu un cont vechi

- Dacă utilizatorul selectează opțiunea **"Logare"**, acesta va trebui să introducă un nume de utilizator și o parolă pentru autentificare.

```
Username:abraham
Parola:gam
Username sau parola invalida!
```

Figure 5 Logarea

- Dacă datele introduse sunt corecte, utilizatorul va avea acces la opțiunile meniului principal.

```
=== Main Menu ===
1. Vizualizare oferte turistice
2. Cautare oferte turistice
3. Rezervare pachete turistice
4. Vizualizare istoric rezervari
5. Deconectare
Selectati o optiune:|
```

Figure 6 Meniul utilizatorului

- Dacă datele introduse sunt gresite, se va afișa un mesaj de eroare și se va reveni la meniul **"Utilizator"**.
- Dacă utilizatorul alege opțiunea **"Înregistrare"**, acesta va trebui să introducă mai multe informații personale pentru a crea un cont nou.

```
Numele:Ghidoarca
Prenumele:Petru-Cristi
Username:Dovahkiin
Parola:gtarmin
Data Nasterii:23.03.2002
Adresa:street
Inregistrare facuta cu succes!
```

Figure 7 Inregistrarea unui utilizator nou

- După completarea informațiilor necesare, contul va fi înregistrat și utilizatorul va primi un mesaj de confirmare corespunzător.

Datele de intrare pentru autentificarea utilizatorului sunt numele de utilizator și parola introduse de utilizator la logare sau înregistrare.

La ieșire, programul poate furniza următoarele informații/format:

- Mesaje de succes sau de eroare pentru acțiunile întreprinse (exemplu: **"Autentificare făcută cu succes!"**, **"Înregistrare făcută cu succes!"**, **"Parolă incorectă!"**).

- Rezultatele acțiunilor utilizatorului, cum ar fi afișarea pachetelor turistice disponibile, rezervărilor efectuate sau istoricului rezervărilor.
- Meniuri cu opțiuni disponibile pentru utilizator, afișate sub formă de numere pentru alegerea opțiunii dorite.

Operațiile ce pot fi efectuate de către utilizatori, respectiv logica acestor operații, sunt următoarele:

1. Autentificare:

- Utilizatorul introduce numele de utilizator și parola pentru a se autentifica.
- Programul verifică existența utilizatorului și corectitudinea parolei în baza de date sau fișierul "users.txt".
- Dacă autentificarea este reușită, utilizatorul este direcționat către meniul principal.
- Dacă autentificarea nu este reușită, utilizatorul primește un mesaj de eroare și are posibilitatea de a încerca din nou sau de a reveni la meniul anterior.

2. Înregistrare:

- Utilizatorul completează informațiile personale necesare pentru crearea unui cont nou, cum ar fi nume, prenume, adresă de e-mail, etc.
- Programul verifică dacă numele de utilizator este disponibil (nu este deja înregistrat).
- Dacă numele de utilizator este disponibil, contul utilizatorului este creat și informațiile sale sunt salvate în baza de date sau fișierul "users.txt".
- Utilizatorul primește un mesaj de confirmare și este îndrumat să se autentifice cu noile date.

3. Meniul principal pentru utilizatori:

- După autentificare cu succes, utilizatorul are acces la un meniu principal cu diverse opțiuni, cum ar fi:
 - Afișarea pachetelor turistice disponibile: Utilizatorul poate vedea o listă de pachete turistice disponibile, inclusiv destinații, prețuri și detalii.
 - Efectuarea unei rezervări: Utilizatorul poate selecta un pachet turistic și efectua o rezervare, furnizând informații suplimentare, cum ar fi data plecării, numărul de persoane etc.
 - Vizualizarea rezervărilor efectuate: Utilizatorul poate accesa o listă cu rezervările pe care le-a efectuat anterior, afișând informații relevante despre fiecare rezervare.
 - Modificarea/ștergerea rezervărilor: Utilizatorul poate modifica sau șterge rezervările existente, introducând numărul de identificare al rezervării și efectuând modificările corespunzătoare.
 - Delogare: Utilizatorul poate alege această opțiune pentru a se deloga și a reveni la meniul de autentificare.

Operații pentru administrator:

1. Autentificare ca administrator:

- Administratorul introduce o parolă specială pentru a se autentifica ca administrator.
- Programul verifică corectitudinea parolei și permite accesul la meniul de administrare dacă autentificarea este reușită.
- Dacă autentificarea nu este reușită, administratorul primește un mesaj de eroare și are posibilitatea de a încerca din nou sau de a reveni la meniul anterior.

2.Meniul de administrare:

- După autentificarea cu succes, administratorul are acces la un meniu de administrare cu opțiuni precum:
 - Adăugarea unui pachet turistic: Administratorul poate introduce informații despre un nou pachet turistic, cum ar fi destinația, descrierea, prețul, etc., și poate salva aceste informații în baza de date sau fișierul "packages.txt".
 - Ștergerea unui pachet turistic: Administratorul poate selecta un pachet turistic existent și îl poate șterge din baza de date sau fișierul "packages.txt".
 - Modificarea unui pachet turistic: Administratorul poate selecta un pachet turistic existent și poate modifica informațiile despre acesta, cum ar fi descrierea, prețul, etc.
 - Vizualizarea tuturor rezervărilor: Administratorul poate accesa o listă cu toate rezervările efectuate de utilizatori, afișând informații relevante despre fiecare rezervare.
 - Delogare: Administratorul poate alege această opțiune pentru a se deloga și a reveni la meniul de autentificare.

CAPITOLUL V

15. CONCLUZII

Concluziile implementării soluției:

Avantajele:

- Soluția implementată oferă un sistem de autentificare pentru utilizatori, permițându-le să se înregistreze și să se conecteze în aplicație.
- Utilizatorii pot vizualiza pachetele turistice disponibile și pot căuta pachete specifice.
- Utilizatorii pot rezerva pachete turistice și pot vedea istoricul rezervărilor lor.
- Administratorii au acces la un meniu special care le permite să adauge și să șteargă pachete turistice, să adauge și să șteargă turiști și să vizualizeze rezervările existente.
- Informațiile despre utilizatori, pachete turistice și rezervări sunt salvate în fișiere JSON pentru stocare și persistență.

Dezavantajele:

- Implementarea actuală nu include validarea datelor introduse de utilizatori, ceea ce poate duce la introducerea de date incorecte sau incomplete.
- Nu există verificări suplimentare pentru a evita crearea de pachete turistice duplicate sau rezervări multiple pentru același pachet.
- Soluția nu include un sistem de gestionare a erorilor robust, care să facă față situațiilor neprevăzute sau erorilor de execuție.

CAPITOLUL VI

16. BIBLIOGRAFIE

1.1. CARTI

- [RCP.1] "Effective Modern C++: 42 Specific Ways to Improve Your Use of C++11 and C++14" by Scott Meyers.
- [RCP.2] "C++ Primer" by Stanley B. Lippman, Josée Lajoie, and Barbara E. Moo.
- [RCP.3] "Effective C++: 55 Specific Ways to Improve Your Programs and Designs" by Scott Meyers.

1.2. SURSE BIBLIOGRAFICE DIVERSE

- [RCP.4] <https://devdocs.io/cpp/>
- [RCP.5] <https://learn.microsoft.com/en-us/cpp/?view=msvc-170>
- [RCP.6] <http://www.cplusplus.com>
- [RCP.7] <https://0x00000000.dev/reflection-serializer>
- [RCP.8] https://www.w3schools.com/cpp/cpp_files.asp
- [RCP.9] <https://en.wikipedia.org/wiki>