**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ КЫРГЫЗСКОЙ РЕСПУБЛИКИ**

**КЫРГЫЗСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСТИТЕТ**

**им. И.Раззакова**

**ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ**

Кафедра «**Программное обеспечение компьютерных систем**»

Направление: 710400 «**Программная инженерия**»

Дисциплина: ««**Объектно-ориентированное программирование**»»

Курсовая работа

Тема: “ TravelExplore”

Выполнил: Кудайбердиев Эрлан

Группа: ПИ-2-21

Проверил: Мусабаев Э. Б.

Бишкек – 2024

Оглавление

[Введение 3](#_Toc169273480)

[ГЛАВА 1. АНАЛИТИЧЕСКАЯ ЧАСТЬ 4](#_Toc169273481)

[1.1. Описание проекта 4](#_Toc169273482)

[1.2. Анализ и формирование требований 4](#_Toc169273483)

[1.2.1. Требование к функциям 4](#_Toc169273484)

[1.2.2. Функциональные требование 4](#_Toc169273485)

[1.2.3. Нефункциональные требование 5](#_Toc169273486)

[1.3. Требования к видам обеспечения 5](#_Toc169273487)

[1.3.1. Требования безопасности 5](#_Toc169273488)

[1.3.2. Требования к надежности 5](#_Toc169273489)

[1.3.3. Требования к лингвистическому обеспечению системы 5](#_Toc169273490)

[1.3.4. Требования к информационной системе 6](#_Toc169273491)

[1.3.5. Требования к пользовательской части 6](#_Toc169273492)

[1.3. Диаграмма вариантов использования 6](#_Toc169273493)

[1.3. Диаграмма классов 7](#_Toc169273494)

[1.4. Описание классов и их назначения 7](#_Toc169273495)

[1.4.1. Абстрактный класс User 7](#_Toc169273496)

[1.4.2. Класс Customer 9](#_Toc169273497)

[1.4.3. Класс Admin 9](#_Toc169273498)

[1.4.4. Класс UserManager 10](#_Toc169273499)

[1.4.5. Класс AttractionType 13](#_Toc169273500)

[1.4.6. Класс Attraction 14](#_Toc169273501)

[1.4.7. Класс AttractionManager 16](#_Toc169273502)

[1.4.8. Класс Tour 20](#_Toc169273503)

[1.4.9. Класс TourManager 22](#_Toc169273504)

[1.4.10. Класс Booking 26](#_Toc169273505)

[1.4.11. Класс BookingManager 27](#_Toc169273506)

[1.4.12. Класс Review 29](#_Toc169273507)

[1.4.13. Класс ReviewManager 30](#_Toc169273508)

[1.5. Хранения данных в проекте 32](#_Toc169273509)

[1.6. Скриншоты результатов работ 33](#_Toc169273510)

[1.7. Заключение 37](#_Toc169273511)

# Введение

Кыргызстан является уникальной страной который может поразить любого путешественника. Это настоящий рай и для любителей экстрима, и для жаждущих уединения и покоя на лоне природы, и для исследователей культуры. Туризм в Кыргызстане год от года набирает обороты, тем самым параллельно увеличивается количество туристов.

Существующие туристические платформы предоставляют огромное количество информации о достопримечательностях, мероприятиях и маршрутах, но часто упускают из виду индивидуальные потребности и интересы каждого путешественника. Отсутствие персонализации приводит к стандартизации туристического опыта, делая его менее насыщенным и удовлетворительным для конкретного пользователя.

Туристы сталкиваются с проблемой выбора среди множества предложений, не учитывающих их уникальные предпочтения, стили путешествий и индивидуальные интересы. Как результат, путешественники рискуют пропустить те уникальные моменты и места, которые делают каждое путешествие неповторимым. Также, часто отсутствует удобный механизм для бронирования всех необходимых услуг, что может создавать неудобства и дополнительные трудности в процессе планирования.

**Спецификация проблемы:**

**1. Отсутствие персонализации:** Существующие туристические платформы часто игнорируют уникальные интересы и предпочтения путешественников. Это приводит к тому, что пользователи сталкиваются с информационным шумом, не способным выделить для них подходящие и запоминающиеся места и маршруты.

**2. Стандартизация туристического опыта:** Большинство существующих сервисов ориентировано на общепринятые маршруты и популярные достопримечательности, что влечет за собой риск упущения уникальных и неизведанных впечатлений, специфичных для каждого пользователя.

**3. Неудобство в выборе:** Путешественники часто сталкиваются с избытком информации и трудностями в выборе наиболее подходящих мест и мероприятий. Это приводит к потере времени на поиск и отсутствию уверенности в правильности выбора.

**4. Неэффективность процесса бронирования:** Отсутствие интегрированных систем бронирования услуг создает дополнительные сложности для пользователей, которые вынуждены искать и бронировать различные услуги отдельно, что снижает удобство и эффективность процесса.

**5. Ограниченный доступ к уникальным местам:** Множество потенциально удивительных и малоизвестных мест остаются вне внимания из-за ориентации платформ на популярные туристические объекты.

# ГЛАВА 1. АНАЛИТИЧЕСКАЯ ЧАСТЬ

# 1.1. Описание проекта

**Основной целью** проекта является предоставления пользователям уникальных и запоминающихся туристических достопримечательности, с учетом их интересов и потребностей, а также обеспечение удобства бронирования.

**Наименование работы:** “ TravelExplore”

**Исполнитель:** Кудайбердиев Эрлан Бусурманкулович

# 1.2. **Анализ и формирование требований**

## 1.2.1. Требование к функциям

**Для пользователя:**

* Регистрация
* Авторизация
* Просмотр/добавление достопримечательности
* Поиск достопримечательности по произвольному шаблону
* Поиск туров по произвольному шаблону
* Просмотр/бронирование туров
* Просмотр/добавление отзывов

**Для администратора:**

* Регистрация
* Авторизация
* Просмотр всех пользователей
* Просмотр/добавление/обновление/удаление достопримечательности
* Поиск достопримечательности по произвольному шаблону
* Поиск туров по произвольному шаблону
* Просмотр/бронирование/удаление туров
* Просмотр/добавление отзывов

## 1.2.2. Функциональные требование

* Программа проверять на идентификацию/аутентификацию/авторизацию пользователя.
* Программа должна обеспечить ввод данных (местоположениям, даты и стоимость) для поиска туров
* Программа должна обеспечить ввод данных (названия, тип, местоположения ) для поиска достопримечательности
* Вывод на консоль стоимость, время бронирования тура.
* Обеспечить сохранение результатов добавления, редактирования, удаления информацию о достопримечательности в БД (данные хранятся в текстовых файлах).

## 1.2.3. Нефункциональные требование

* Система должна иметь меню на консоли
* Программа должна иметь хорошо организованный код
* Система должна быть способна обслуживать необходимое количество пользователей без снижения производительности.

# 1.3. Требования к видам обеспечения

## 1.3.1. Требования безопасности

Система должна обеспечивать соответствие с критериями оценивания по установленным в Государственном и предметных стандартах

## 1.3.2. Требования к надежности

Система должна сохранять работоспособность и обеспечивать восстановление своих функций при возникновении следующих внештатных ситуаций:

– при сбоях в системе электроснабжения аппаратной части, приводящих к перезагрузке ОС, восстановление программы должно происходить после перезапуска ОС и запуска исполняемого файла системы;

– при ошибках в работе аппаратных средств (кроме носителей данных и программ) восстановление функции системы возлагается на ОС;

– при ошибках, связанных с программным обеспечением (ОС и драйверы устройств), восстановление работоспособности возлагается на ОС.

## 1.3.3. Требования к лингвистическому обеспечению системы

Все прикладное программное обеспечение системы для организации взаимодействия с пользователем должно использовать только русский язык.

## 1.3.4. Требования к информационной системе

Используемое при разработке программное обеспечение, библиотеки программных кодов, СУБД должны иметь широкое распространение, быть общедоступными и использоваться в промышленных масштабах.

В качестве основы для разработки должен использоваться Visual Studio, язык программирования Visual Basic. В качестве серверной операционной системы должна использоваться ОС семейства Windows 7 и выше.

## 

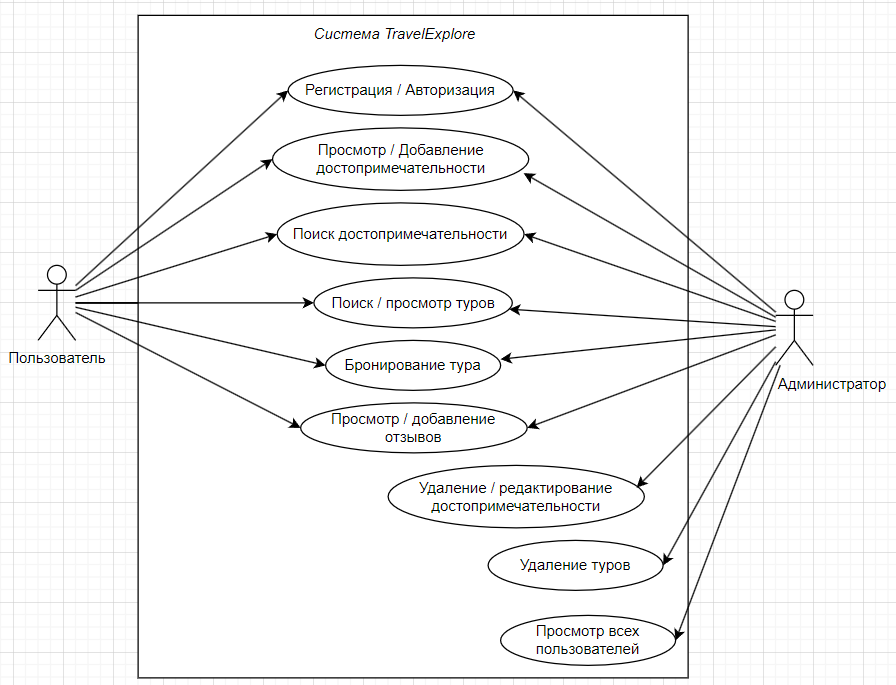
## 1.3.5. Требования к пользовательской части

Пользователь должен использовать:

* процессор – intel core I5 и выше;
* оперативная память – 2гб. и выше;
* память – 862мб. И выше;
* операционная система – Windows OS, Linux OS.

# 1.3. Диаграмма вариантов использования

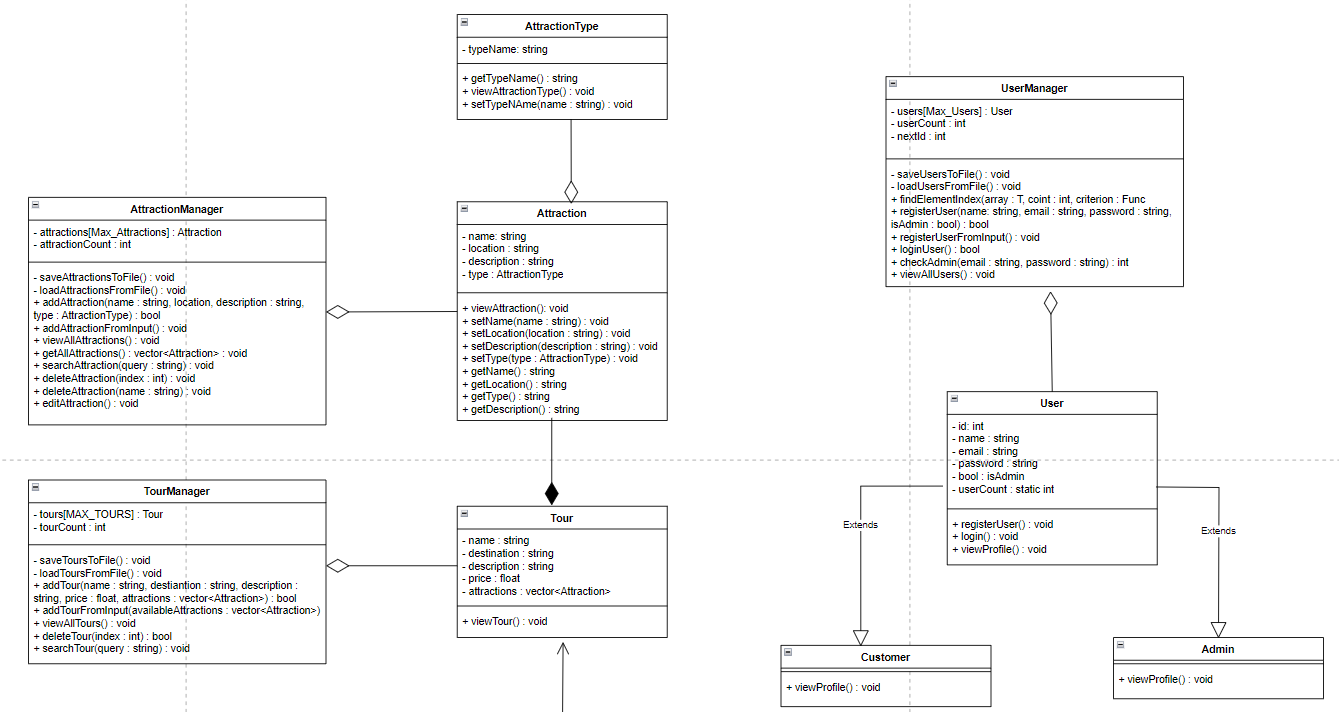
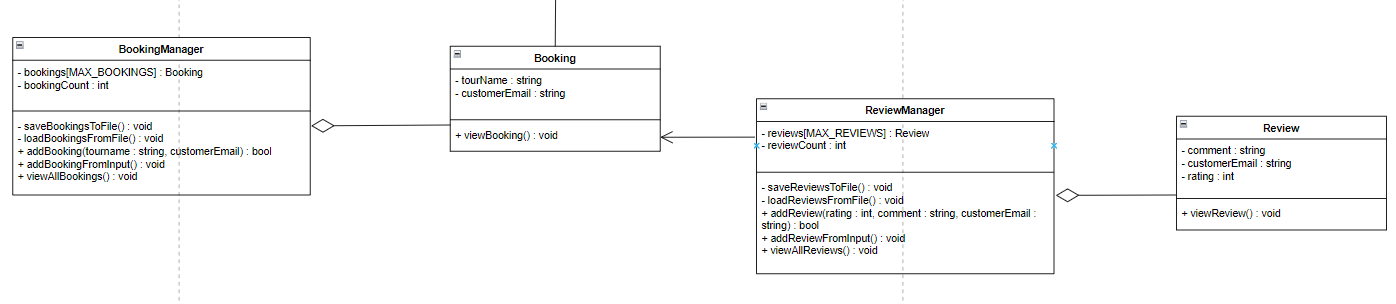
Ниже на рис 1. представлена диаграмма вариантов использование для системы TravelExplore.



*Рисунок 1. Диаграмма вариантов использования*

# 1.3. Диаграмма классов

Диаграмма классов для системы Travelexplore представлено ниже на рисунке 2.

**

*Рисунок 2. Диаграмма классов*

Система содержит 13 классов: AttractionType, Attraction, AttrationManager, User, UserManager, Admin, Customer, Tour, TourManager, Review, ReviewManager, Booking, BookingManager.

# 1.4. Описание классов и их назначения

## 1.4.1. Абстрактный класс User

Назначение класса: User является базовым классом для всех типов пользователей. Хранит базовую информацию о пользователе (id, имя, email, пароль, роль). Поддерживает функции для регистрации, входа в систему и просмотра профиля. Имеет геттеры и сеттеры для доступа к приватным данным.

Описание методов класса:

|  |  |  |
| --- | --- | --- |
| № | Сигнатура | Назначение |
| 1 | User() | Конструктор по умолчанию. Нужен для инициализации объекта User. |
| 2 | User(int id, const string& name, const string& email, const string& password, const bool& a) | Этот конструктор инициализирует объект User с заданными параметрами: id, name, email, password и isAdmin. |
| 3 | void registerUser() | Этот метод предполагает реализацию логики регистрации пользователя. |
| 4 | void login() | Метод login() выводит сообщение о успешном входе пользователя в систему. Выводит имя пользователя, которое было установлено в момент создания объекта. |
| 5 | virtual void viewProfile() const | Этот виртуальный метод viewProfile() выводит информацию о профиле пользователя. В базовом классе User выводится имя и электронная почта пользователя. Виртуальный метод позволяет подклассам (Customer, Admin) переопределить этот метод для добавления дополнительной информации. |
| 6 | ~User() | Деструктор |

Так как поля этого класса приватные и чтобы получить доступ к этим полям, для каждого поле есть методы get и set.

Листинг класса:

class User {

protected:

int id;

string name;

string email;

string password;

bool isAdmin;

static int userCount;

public:

User() : id(-1) {

++userCount;

}

User(int id, const string& name, const string& email, const string& password, const bool& a)

: id(id), name(name), email(email), password(password), isAdmin(a) {

++userCount;

}

void registerUser() {

}

void login() {

cout << "\nВы успешно вошли в систему: " << name << endl << endl;

}

virtual void viewProfile() const {

cout << "Профиль пользователя - Имя: " << name << ", Электронная почта: " << email << endl;

}

int getId() const { return id; }

string getName() const { return name; }

string getEmail() const { return email; }

string getPassword() const { return password; }

bool getIsAdmin() const { return isAdmin; }

void setId(int id) { this->id = id; }

void setName(const string& name) { this->name = name; }

void setEmail(const string& email) { this->email = email; }

void setPassword(const string& password) { this->password = password; }

void setIsAdmin(const bool& a) { this->isAdmin = a; }

~User() {

--userCount;

}

};

## 1.4.2. Класс Customer

Назначение класса: Customer является наследником класса User. Представляет профиль клиента, переопределяя метод viewProfile() для отображения специфичной информации.

Описание методов класса:

|  |  |  |
| --- | --- | --- |
| № | Сигнатура | Назначение |
| 1 | Customer() | Конструктор по умолчанию. Нужен для инициализации объекта Customer. |
| 2 | Customer(int id, const string& name, const string& email, const string& password, const bool& a) | Этот конструктор инициализирует объект User с заданными параметрами. |
| 3 | void viewProfile() const override | Этот метод переопределяет виртуальный метод viewProfile() из базового класса User. Он выводит информацию о профиле пользователя, включая имя и электронную почту. |

Листинг класса:

class Customer : public User {

public:

Customer() : User() {}

Customer(int id, const string& name, const string& email, const string& password, const bool& a)

: User(id, name, email, password, a) {}

void viewProfile() const override {

cout << "Профиль клиента - Имя: " << name << ", Электронная почта: " << email << endl;

}

};

## 1.4.3. Класс Admin

Назначение класса: Admin является наследником класса User. Представляет профиль администратора, переопределяя метод viewProfile() для отображения специфичной информации.

Описание методов класса:

|  |  |  |
| --- | --- | --- |
| № | Сигнатура | Назначение |
| 1 | Admin() | Конструктор по умолчанию. Нужен для инициализации объекта Customer. |
| 2 | Admin(int id, const string& name, const string& email, const string& password, const bool& a) | Этот конструктор инициализирует объект User с заданными параметрами. |
| 3 | void viewProfile() const override | Этот метод переопределяет виртуальный метод viewProfile() из базового класса User. Он выводит информацию о профиле администратора, включая имя и электронную почту. |

Листинг класса:

class Admin : public User {

public:

Admin() : User() {}

Admin(int id, const string& name, const string& email, const string& password, const bool& a)

: User(id, name, email, password, a) {}

void viewProfile() const override {

cout << "Профиль администратора - Имя: " << name << ", Электронная почта: " << email << endl;

}

};

## 1.4.4. Класс UserManager

Назначение класса: Класс UserManager отвечает за управление пользователями системы, включая их регистрацию, аутентификацию, сохранение в файл и загрузку из файла.

Описание полей и методов класса:

|  |  |  |
| --- | --- | --- |
| № | Сигнатура | Назначение |
| 1 | User users[MAX\_USERS]; | Массив объектов типа User, предназначенный для хранения зарегистрированных пользователей. |
| 2 | int userCount; | Текущее количество пользователей в массиве users. |
| 3 | int nextId; | Следующий доступный ID для нового пользователя. |
| 4 | UserManager() | Конструктор по умолчанию |
| 5 | int findElementIndex(const T\* array, int count, Func criterion) | Метод для нахождения позиции пользователя в массиве. |
| 6 | bool registerUser(const string& name, const string& email, const string& password, bool isAdmin) | Регистрирует нового пользователя в системе. Метод проверяет доступность места для новых пользователей и уникальность email. В зависимости от значения isAdmin, создает объект типа Admin или Customer и добавляет его в массив пользователей. Затем сохраняет обновленный список пользователей в файл. |
| 7 | void registerUserFromInput() | Предназначен для регистрации пользователя с вводом данных с клавиатуры. |
| 8 | void viewAllUsers() const | Выводит информацию о всех зарегистрированных пользователях, вызывая метод viewProfile() для каждого из них. |
| 9 | bool loginUser(const string& email, const string& password) | Проверяет наличие пользователя с заданным email и паролем в системе. Если пользователь найден, метод вызывает функцию login() для входа в систему. |
| 10 | int checkAdmin(const string& email, const string& password) | Проверяет, имеет ли пользователь с указанным email и паролем статус администратора. Возвращает true, если пользователь является администратором, и false в противном случае. |
| 11 | void saveUsersToFile() | Cохраняет данные всех пользователей в файл users.txt. Каждый аттрибут разделяется символом ‘|’. |
| 12 | void loadUsersFromFile() | Загружает данные из файла users.txt, заполняя массив users и обновляя userCount. |

Так как поля этого класса приватные и чтобы получить доступ к этим полям, для каждого поле есть методы get и set.

Листинг класса:

class UserManager {

private:

User users[MAX\_USERS];

int userCount;

int nextId;

void saveUsersToFile() {

ofstream file("users.txt");

if (file.is\_open()) {

for (int i = 0; i < userCount; ++i) {

file << users[i].getId() << "|"

<< users[i].getName() << "|"

<< users[i].getEmail() << "|"

<< users[i].getPassword() << "|"

<< users[i].getIsAdmin() << endl;

}

file.close();

}

}

void loadUsersFromFile() {

ifstream file("users.txt");

if (file.is\_open()) {

userCount = 0;

string line;

while (getline(file, line)) {

int id;

string name, email, password;

bool isAdmin;

size\_t pos = 0;

string token;

vector<string> tokens;

while ((pos = line.find('|')) != string::npos) {

token = line.substr(0, pos);

tokens.push\_back(token);

line.erase(0, pos + 1);

}

tokens.push\_back(line);

if (tokens.size() == 5) {

id = stoi(tokens[0]);

name = tokens[1];

email = tokens[2];

password = tokens[3];

isAdmin = (tokens[4] == "1");

users[userCount].setId(id);

users[userCount].setName(name);

users[userCount].setEmail(email);

users[userCount].setPassword(password);

users[userCount].setIsAdmin(isAdmin);

userCount++;

nextId = id + 1;

}

}

file.close();

}

}

public:

UserManager() : userCount(0), nextId(1) {

loadUsersFromFile();

}

template <typename T, typename Func>

int findElementIndex(const T\* array, int count, Func criterion) {

for (int i = 0; i < count; ++i) {

if (criterion(array[i])) {

return i;

}

}

return -1;

}

bool registerUser(const string& name, const string& email, const string& password, bool isAdmin) {

if (userCount >= MAX\_USERS) {

cout << "Достигнуто максимальное количество пользователей." << endl;

return false;

}

for (int i = 0; i < userCount; ++i) {

if (users[i].getEmail() == email) {

cout << "Пользователь с таким email уже существует." << endl;

return false;

}

}

int newId = nextId++;

if (isAdmin) {

Admin admin(newId, name, email, password, isAdmin);

admin.registerUser();

users[userCount++] = admin;

}

else {

Customer customer(newId, name, email, password, isAdmin);

customer.registerUser();

users[userCount++] = customer;

}

saveUsersToFile();

return true;

}

void registerUserFromInput() {

string name, email, password;

bool isAdmin;

cout << "Введите имя пользователя: ";

cin >> name;

cout << "Введите адрес электронной почты: ";

cin >> email;

cout << "Введите пароль: ";

cin >> password;

cout << "Является ли пользователь администратором? (1/0): ";

cin >> isAdmin;

cin.ignore();

registerUser(name, email, password, isAdmin);

cout << "\nРегистрация прошла успешно!!!\n";

cout << endl;

}

bool loginUser(const string& email, const string& password) {

for (int i = 0; i < userCount; ++i) {

if (users[i].getEmail() == email && users[i].getPassword() == password) {

users[i].login();

return true;

}

}

return false;

}

int checkAdmin(const string& email, const string& password) {

for (int i = 0; i < userCount; ++i) {

if (users[i].getEmail() == email && users[i].getPassword() == password) {

return users[i].getIsAdmin();

}

}

}

void viewAllUsers() const {

for (int i = 0; i < userCount; ++i) {

users[i].viewProfile();

}

}

};

## 1.4.5. Класс AttractionType

Назначение класса: Этот класс прост в использовании и предназначен для представления и управления типами достопримечательностей, путем установки и получения их названий, а также для отображения информации о типе.

Описание полей и методов класса:

|  |  |  |
| --- | --- | --- |
| № | Сигнатура | Назначение |
| 1 | AttractionType() | Конструктор по умолчанию |
| 2 | string typeName; | Поле для типа достопримечательности |
| 3 | AttractionType(const string& typeName) : typeName(typeName) | Конструктор с параметрами |
| 4 | UserManager() | Конструктор по умолчанию |
| 5 | void viewAttractionType() const | Этот метод выводит на экран информацию о типе достопримечательности, используя текущее значение typeName. |

Так как поля этого класса приватные и чтобы получить доступ к этим полям, для каждого поле есть методы get и set.

Листинг класса:

class AttractionType {

private:

string typeName;

public:

AttractionType() : typeName("") {}

AttractionType(const string& typeName) : typeName(typeName) {}

string getTypeName() const {

return typeName;

}

void viewAttractionType() const {

cout << "Тип достопримечательности: " << typeName << endl;

}

void setTypeName(const string& name) {

typeName = name;

}

};

## 1.4.6. Класс Attraction

Назначение класса: Класс Attraction представляет собой модель достопримечательности с четырьмя приватными полями и несколькими публичными методами для доступа к данным и управления ими.Описание полей и методов класса:

Описание полей и методов класса:

|  |  |  |
| --- | --- | --- |
| № | Сигнатура | Назначение |
| 1 | string name; | Хранит название достопримечательности. |
| 2 | string location; | Содержит информацию о местоположении достопримечательности. |
| 3 | string description; | Содержит описание достопримечательности. |
| 4 | AttractionType type; | Представляет тип достопримечательности, который определяет тип достопримечательности (например, историческое место, парк, музей и т.д.). |
| 5 | Attraction() | Конструктор по умолчанию |
| 6 | Attraction(const string& name, const string& location, const string& description, const AttractionType& type) | Конструктор с параметрами |
| 7 | void viewAttraction() const | Выводит информацию о достопримечательности, включая ее название, местоположение, описание и тип. |

Так как поля этого класса приватные и чтобы получить доступ к этим полям, для каждого поле есть методы get и set.

Листинг класса:

class Attraction {

private:

string name;

string location;

string description;

AttractionType type;

public:

Attraction() : name(""), location(""), description(""), type("") {}

Attraction(const string& name, const string& location, const string& description, const AttractionType& type)

: name(name), location(location), description(description), type(type) {}

void viewAttraction() const {

cout << "Достопримечательность - Название: " << name << ", Местоположение: " << location

<< ", Описание: " << description << ", Тип: " << type.getTypeName() << endl;

}

void setName(const string& name) {

this->name = name;

}

void setLocation(const string& location) {

this->location = location;

}

void setDescription(const string& description) {

this->description = description;

}

void setType(const AttractionType& type) {

this->type = type;

}

string getName() const {

return name;

}

string getLocation() const {

return location;

}

string getDescription() const {

return description;

}

AttractionType getType() const {

return type;

}

};

## 1.4.7. Класс AttractionManager

Назначение класса: Класс AttractionManager предоставляет полный набор методов для управления коллекцией достопримечательностей, включая их добавление, просмотр, поиск, редактирование и удаление, а также сохранение и загрузку данных из файла.

Описание полей и методов класса:

|  |  |  |
| --- | --- | --- |
| № | Сигнатура | Назначение |
| 1 | Attraction attractions[MAX\_ATTRACTIONS]; | Массив, хранящий объекты типа Attraction. MAX\_ATTRACTIONS - это константа, задающая максимальное количество достопримечательностей, которые могут быть сохранены. |
| 2 | int attractionCount; | Счетчик текущего количества достопримечательностей, хранящихся в массиве attractions. |
| 3 | void saveAttractionsToFile() | Cохраняет данные всех достопримечательностей в файл attractions.txt. Каждый аттрибут разделяется символом ‘|’. |
| 4 | void loadAttractionsFromFile() | Загружает данные из файла attractions.txt, заполняя массив attractions и обновляя attractionCount. |
| 5 | AttractionManager() | Конструктор по умолчанию |
| 6 | bool addAttraction(const string& name, const string& location, const string& description, const AttractionType& type) | Добавляет новую достопримечательность в массив. Если количество достопримечательностей достигает MAX\_ATTRACTIONS, выбрасывается исключение MaxAttractionsException. После добавления вызывается saveAttractionsToFile() для сохранения изменений в файл. |
| 7 | void addAttractionFromInput() | Вводит данные о новой достопримечательности с консоли и добавляет её в массив с использованием метода addAttraction. |
| 8 | void viewAllAttractions() const | Выводит информацию обо всех достопримечательностях, хранящихся в массиве. |
| 9 | vector<Attraction> getAllAttractions() const | Возвращает вектор всех достопримечательностей. |
| 10 | void searchAttraction(const string& query) const | Ищет достопримечательность по названию, местоположению, описанию или типу. Если ничего не найдено, выводит соответствующее сообщение. |
| 11 | void deleteAttraction(int index) | Удаляет достопримечательность по индексу. Если индекс некорректен, выводит сообщение об ошибке. После удаления вызывает saveAttractionsToFile(). |
| 12 | void editAttraction() | Позволяет редактировать данные существующей достопримечательности. Пользователь выбирает индекс достопримечательности для редактирования и затем выбирает, какое поле редактировать. После редактирования данных вызывает saveAttractionsToFile() для сохранения изменений. |

Так как поля этого класса приватные и чтобы получить доступ к этим полям, для каждого поле есть методы get и set.

Листинг класса:

class AttractionManager {

private:

Attraction attractions[MAX\_ATTRACTIONS];

int attractionCount;

void saveAttractionsToFile() {

ofstream file("attractions.txt");

if (file.is\_open()) {

for (int i = 0; i < attractionCount; ++i) {

file << attractions[i].getName() << "|"

<< attractions[i].getLocation() << "|"

<< attractions[i].getDescription() << "|"

<< attractions[i].getType().getTypeName() << endl;

}

file.close();

}

}

void loadAttractionsFromFile() {

ifstream file("attractions.txt");

if (file.is\_open()) {

attractionCount = 0;

string line;

while (getline(file, line)) {

string name, location, description, typeName;

size\_t pos = 0;

string token;

vector<string> tokens;

while ((pos = line.find('|')) != string::npos) {

token = line.substr(0, pos);

tokens.push\_back(token);

line.erase(0, pos + 1);

}

tokens.push\_back(line);

if (tokens.size() == 4) {

name = tokens[0];

location = tokens[1];

description = tokens[2];

typeName = tokens[3];

attractions[attractionCount].setName(name);

attractions[attractionCount].setLocation(location);

attractions[attractionCount].setDescription(description);

attractions[attractionCount].setType(AttractionType(typeName));

attractionCount++;

}

}

file.close();

}

}

public:

AttractionManager() : attractionCount(0) {

loadAttractionsFromFile();

}

bool addAttraction(const std::string& name, const std::string& location, const std::string& description, const AttractionType& type) {

if (attractionCount >= MAX\_ATTRACTIONS) {

throw MaxAttractionsException();

}

attractions[attractionCount++] = Attraction(name, location, description, type);

saveAttractionsToFile();

return true;

}

void addAttractionFromInput() {

string name, location, description, typeName;

cout << "Введите название достопримечательности: ";

cin >> name;

cout << "Введите местоположение достопримечательности: ";

cin >> location;

cout << "Введите описание достопримечательности: ";

cin >> description;

cout << "Введите тип достопримечательности: ";

cin >> typeName;

addAttraction(name, location, description, AttractionType(typeName));

cout << "\nДостопримечательность успешно добавлена!\n";

}

void viewAllAttractions() const {

for (int i = 0; i < attractionCount; ++i) {

attractions[i].viewAttraction();

}

}

vector<Attraction> getAllAttractions() const {

return vector<Attraction>(attractions, attractions + attractionCount);

}

void searchAttraction(const string& query) const {

bool found = false;

for (int i = 0; i < attractionCount; ++i) {

if (attractions[i].getName() == query ||

attractions[i].getLocation() == query ||

attractions[i].getDescription() == query ||

attractions[i].getType().getTypeName() == query) {

attractions[i].viewAttraction();

found = true;

}

}

if (!found) {

cout << "Достопримечательность не найдена.\n" << endl;

}

}

void deleteAttraction(int index) {

if (index < 0 || index >= attractionCount) {

cout << "\nНеверный индекс достопримечательности!!!\n" << endl;

return;

}

for (int i = index; i < attractionCount - 1; ++i) {

attractions[i] = attractions[i + 1];

}

attractionCount--;

cout << "\nДостопримечательность успешно удалена!!!\n" << endl;

saveAttractionsToFile();

}

void deleteAttraction(const std::string& name) {

int index = -1;

for (int i = 0; i < attractionCount; ++i) {

if (attractions[i].getName() == name) {

index = i;

break;

}

}

if (index == -1) {

std::cout << "\nДостопримечательность с указанным названием не найдена!!!\n" << std::endl;

return;

}

deleteAttraction(index);

}

void editAttraction() {

viewAllAttractions();

int index;

cout << "Введите индекс достопримечательности для редактирования: ";

cin >> index;

index = index - 1;

if (index < 0 || index >= attractionCount) {

cout << "Неверный индекс достопримечательности.\n";

return;

}

cout << "\t 1. Редактировать название\n\t 2. Редактировать местоположение\n\t 3. Редактировать описание\n\t 4. Редактировать тип\n";

cout << "Что вы хотите редактировать: ";

short l;

cin >> l;

string name, location, description;

int typeIndex;

AttractionType selectedType;

switch (l) {

case 1:

cout << "Введите новое название достопримечательности: ";

cin.ignore();

getline(cin, name);

attractions[index].setName(name);

break;

case 2:

cout << "Введите новое местоположение достопримечательности: ";

cin.ignore();

getline(cin, location);

attractions[index].setLocation(location);

break;

case 3:

cout << "Введите новое описание достопримечательности: ";

cin.ignore();

getline(cin, description);

attractions[index].setDescription(description);

break;

case 4:

cout << "Выберите новый тип достопримечательности:\n";

cout << "1) Historical\n2) Cultural\n3) Natural\n4) Entertainment\n5) Religious\n6) Architectural\n7) Museum\n";

cout << "Введите номер типа: ";

cin >> typeIndex;

switch (typeIndex) {

case 1:

selectedType = AttractionType("Historical");

break;

case 2:

selectedType = AttractionType("Cultural");

break;

case 3:

selectedType = AttractionType("Natural");

break;

case 4:

selectedType = AttractionType("Entertainment");

break;

case 5:

selectedType = AttractionType("Religious");

break;

case 6:

selectedType = AttractionType("Architectural");

break;

case 7:

selectedType = AttractionType("Museum");

break;

default:

cout << "Неверный тип. Используется тип по умолчанию.\n";

selectedType = AttractionType("Неизвестный");

break;

}

attractions[index].setType(selectedType);

break;

default:

cout << "Неверный выбор. Попробуйте еще раз.\n";

break;

}

cout << "Достопримечательность успешно обновлена.\n";

saveAttractionsToFile();

}

};

## 1.4.8. Класс Tour

Назначение класса: Tour — это класс, представляющий туристический тур. Он содержит информацию о туре, включая название, место назначения, описание, цену и список достопримечательностей.

Описание полей и методов класса:

|  |  |  |
| --- | --- | --- |
| № | Сигнатура | Назначение |
| 1 | string name; | Хранит название тура. |
| 2 | string destination; | Содержит место назначения тура. |
| 3 | string description; | Содержит описание тура. |
| 4 | float price; | Хранит цену тура в сомах. |
| 5 | vector<Attraction> attractions; | Вектор объектов типа Attraction, представляющий достопримечательности, включенные в тур. |
| 6 | Tour() | Конструктор по умолчанию |
| 7 | Tour(const string& name, const string& destination, const string& description, float price, const vector<Attraction>& attractions = {}) | Конструктор с параметрами |
| 8 | void viewTour() const | Выводит информацию о туре, включая его название, место назначения, описание, цену и список достопримечательностей. |
| 9 | vector<Attraction> getAttractions() | Возвращает вектор достопримечательностей, включенных в тур. |

Так как поля этого класса приватные и чтобы получить доступ к этим полям, для каждого поле есть методы get и set.

Листинг класса:

class Tour {

private:

string name;

string destination;

string description;

float price;

vector<Attraction> attractions;

public:

Tour() : name(""), destination(""), description(""), price(0.0) {}

Tour(const string& name, const string& destination, const string& description, float price, const vector<Attraction>& attractions = {})

: name(name), destination(destination), description(description), price(price), attractions(attractions) {}

void viewTour() const {

cout << "Тур - Название: " << name << ", Место: " << destination

<< ", Описание: " << description << ", Цена: " << price << " рублей" << endl;

cout << "Достопримечательности:\n";

for (const auto& attraction : attractions) {

attraction.viewAttraction();

}

}

void setName(const string& name) {

this->name = name;

}

void setDestination(const string& destination) {

this->destination = destination;

}

void setDescription(const string& description) {

this->description = description;

}

void setPrice(float price) {

this->price = price;

}

void setAttractions(const vector<Attraction>& attractions) {

this->attractions = attractions;

}

string getName() const {

return name;

}

string getDestination() const {

return destination;

}

string getDescription() const {

return description;

}

float getPrice() const {

return price;

}

vector<Attraction> getAttractions() const {

return attractions;

}

};

## 1.4.9. Класс TourManager

Назначение класса: TourManager — это класс, предназначенный для управления турами. Он позволяет добавлять, удалять, просматривать и искать туры, а также сохранять и загружать данные о турах из файла.

Описание полей и методов класса:

|  |  |  |
| --- | --- | --- |
| № | Сигнатура | Назначение |
| 1 | Tour tours[MAX\_TOURS]; | Массив объектов класса Tour, который хранит информацию о турах. |
| 2 | int tourCount; | Переменная, которая хранит текущее количество туров. |
| 3 | void saveToursToFile() | Сохраняет все текущие туры в файл "tours.txt". Этот метод открывает файл, записывает данные о каждом туре в файл и затем закрывает файл. |
| 4 | void loadToursFromFile() | Загружает туры из файла "tours.txt". Этот метод открывает файл, считывает данные, инициализирует туры и увеличивает tourCount. |
| 5 | TourManager() | Конструктор по умолчанию |
| 6 | bool addTour(const string& name, const string& destination, const string& description, float price, const vector<Attraction>& attractions) | Добавляет новый тур в массив tours. Если достигнуто максимальное количество туров (MAX\_TOURS), выводится сообщение и возвращается false. В противном случае, тур добавляется в массив, данные сохраняются в файл, и метод возвращает true. |
| 7 | void addTourFromInput(const vector<Attraction>& availableAttractions) | Интерфейс для добавления тура с пользовательским вводом. Метод запрашивает у пользователя данные о туре и его достопримечательностях, затем вызывает метод addTour для добавления тура. |
| 8 | void viewAllTours() const | Отображает информацию обо всех текущих турах. Проходит по массиву tours и вызывает метод viewTour() для каждого тура. |
| 9 | bool deleteTour(int index) | Удаляет тур по указанному индексу. Если индекс некорректен, выводится сообщение об ошибке и возвращается false. В противном случае, тур удаляется, данные сохраняются в файл, и метод возвращает true. |
| 10 | void searchTour(const string& query) const | Выполняет поиск туров по заданному запросу query. Метод проверяет совпадение строки запроса с названием, местом назначения, описанием тура и достопримечательностями. Если найдены совпадения, туры отображаются. |

Так как поля этого класса приватные и чтобы получить доступ к этим полям, для каждого поле есть методы get и set.

Листинг класса:

class TourManager {

private:

Tour tours[MAX\_TOURS];

int tourCount;

void saveToursToFile() {

ofstream file("tours.txt");

if (file.is\_open()) {

for (int i = 0; i < tourCount; ++i) {

file << tours[i].getName() << "|"

<< tours[i].getDestination() << "|"

<< tours[i].getDescription() << "|"

<< tours[i].getPrice() << endl;

}

file.close();

}

}

void loadToursFromFile() {

ifstream file("tours.txt");

if (file.is\_open()) {

tourCount = 0;

string line;

while (getline(file, line)) {

string name, destination, description;

float price;

size\_t pos = 0;

string token;

vector<string> tokens;

while ((pos = line.find('|')) != string::npos) {

token = line.substr(0, pos);

tokens.push\_back(token);

line.erase(0, pos + 1);

}

tokens.push\_back(line);

if (tokens.size() == 4) {

name = tokens[0];

destination = tokens[1];

description = tokens[2];

price = stof(tokens[3]);

tours[tourCount].setName(name);

tours[tourCount].setDestination(destination);

tours[tourCount].setDescription(description);

tours[tourCount].setPrice(price);

tourCount++;

}

}

file.close();

}

}

public:

TourManager() : tourCount(0) {

loadToursFromFile();

}

bool addTour(const string& name, const string& destination, const string& description, float price, const vector<Attraction>& attractions) {

if (tourCount >= MAX\_TOURS) {

cout << "Достигнуто максимальное количество туров." << endl;

return false;

}

tours[tourCount++] = Tour(name, destination, description, price, attractions);

saveToursToFile();

return true;

}

void addTourFromInput(const vector<Attraction>& availableAttractions) {

string name, destination, description;

float price;

int attractionCount;

cout << "Введите название тура: ";

cin >> name;

cout << "Введите место назначения тура: ";

cin >> destination;

cout << "Введите описание тура: ";

cin >> description;

cout << "Введите цену тура: ";

cin >> price;

cout << "Введите количество достопримечательностей для тура: ";

cin >> attractionCount;

vector<Attraction> selectedAttractions;

for (int i = 0; i < attractionCount; ++i) {

int index;

cout << "Введите номер достопримечательности (1-" << availableAttractions.size() << "): ";

cin >> index;

if (index > 0 && index <= availableAttractions.size()) {

selectedAttractions.push\_back(availableAttractions[index - 1]);

}

else {

cout << "Некорректный номер достопримечательности." << endl;

}

}

addTour(name, destination, description, price, selectedAttractions);

cout << "\nТур успешно добавлен!\n";

}

void viewAllTours() const {

for (int i = 0; i < tourCount; ++i) {

tours[i].viewTour();

}

}

bool deleteTour(int index) {

if (index < 0 || index >= tourCount) {

cout << "Некорректный индекс тура." << endl;

return false;

}

for (int i = index; i < tourCount - 1; ++i) {

tours[i] = tours[i + 1];

}

tourCount--;

saveToursToFile();

cout << "Тур успешно удален." << endl;

return true;

}

void searchTour(const string& query) const {

bool found = false;

for (int i = 0; i < tourCount; ++i) {

if (tours[i].getName().find(query) != string::npos ||

tours[i].getDestination().find(query) != string::npos ||

tours[i].getDescription().find(query) != string::npos) {

tours[i].viewTour();

found = true;

}

else {

for (const auto& attraction : tours[i].getAttractions()) {

if (attraction.getName().find(query) != string::npos ||

attraction.getDescription().find(query) != string::npos) {

tours[i].viewTour();

found = true;

break;

}

}

}

}

if (!found) {

cout << "Туры не найдены.\n";

}

}

};

## 1.4.10. Класс Booking

Назначение класса: Класс Booking предназначен для представления информации о бронировании туристического тура. Он включает в себя данные о туре и электронной почте клиента, связанного с бронированием.

Описание полей и методов класса:

|  |  |  |
| --- | --- | --- |
| № | Сигнатура | Назначение |
| 1 | string tourName; | Название забронированного тура. |
| 2 | string customerEmail; | Электронная почта клиента, который забронировал тур. |
| 3 | Booking() | Конструктор по умолчанию |
| 4 | Booking(const string& tourName, const string& customerEmail) | Конструктор с параметрами |
| 5 | void viewBooking() const | Отображает информацию о бронировании, включая название тура и электронную почту клиента. |

Так как поля этого класса приватные и чтобы получить доступ к этим полям, для каждого поле есть методы get и set.

Листинг класса:

class Booking {

private:

string tourName;

string customerEmail;

public:

Booking() : tourName(""), customerEmail("") {}

Booking(const string& tourName, const string& customerEmail)

: tourName(tourName), customerEmail(customerEmail) {}

void viewBooking() const {

cout << "Бронирование - Название тура: " << tourName << ", Электронная почта клиента: " << customerEmail << endl;

}

void setTourName(const string& tourName) {

this->tourName = tourName;

}

void setCustomerEmail(const string& customerEmail) {

this->customerEmail = customerEmail;

}

string getTourName() const {

return tourName;

}

string getCustomerEmail() const {

return customerEmail;

}

};

## 1.4.11. Класс BookingManager

Назначение класса: BookingManager — это класс, предназначенный для управления бронированиями туристических туров. Он позволяет добавлять, просматривать и сохранять бронирования, а также загружать их из файла.

Описание полей и методов класса:

|  |  |  |
| --- | --- | --- |
| № | Сигнатура | Назначение |
| 1 | Booking bookings[MAX\_BOOKINGS]; | Массив объектов класса Booking, который хранит информацию о бронированиях. |
| 2 | int bookingCount; | Переменная, которая хранит текущее количество бронирований. |
| 3 | void saveBookingsToFile() | Сохраняет все текущие бронирования в файл "bookings.txt". Этот метод открывает файл, записывает данные о каждом бронировании в файл и затем закрывает файл. |
| 4 | void loadBookingsFromFile() | Загружает бронирования из файла "bookings.txt". Этот метод открывает файл, считывает данные, инициализирует бронирования и увеличивает bookingCount. |
| 5 | BookingManager() | Конструктор по умолчанию |
| 6 | bool addBooking(const string& tourName, const string& customerEmail) | Добавляет новое бронирование в массив bookings. Если достигнуто максимальное количество бронирований (MAX\_BOOKINGS), выводится сообщение и возвращается false. В противном случае, бронирование добавляется в массив, данные сохраняются в файл, и метод возвращает true. |
| 7 | void addBookingFromInput() | Интерфейс для добавления бронирования с пользовательским вводом. Метод запрашивает у пользователя данные о бронировании и затем вызывает метод addBooking для добавления бронирования. |
| 8 | void viewAllBookings() const | Отображает информацию о всех текущих бронированиях. Проходит по массиву bookings и вызывает метод viewBooking() для каждого бронирования. |

Так как поля этого класса приватные и чтобы получить доступ к этим полям, для каждого поле есть методы get и set.

Листинг класса:

class BookingManager {

private:

Booking bookings[MAX\_BOOKINGS];

int bookingCount;

void saveBookingsToFile() {

ofstream file("bookings.txt");

if (file.is\_open()) {

for (int i = 0; i < bookingCount; ++i) {

file << bookings[i].getTourName() << "|"

<< bookings[i].getCustomerEmail() << endl;

}

file.close();

}

}

void loadBookingsFromFile() {

ifstream file("bookings.txt");

if (file.is\_open()) {

bookingCount = 0;

string line;

while (getline(file, line)) {

string tourName, customerEmail;

size\_t pos = 0;

string token;

vector<string> tokens;

while ((pos = line.find('|')) != string::npos) {

token = line.substr(0, pos);

tokens.push\_back(token);

line.erase(0, pos + 1);

}

tokens.push\_back(line);

if (tokens.size() == 2) {

tourName = tokens[0];

customerEmail = tokens[1];

bookings[bookingCount].setTourName(tourName);

bookings[bookingCount].setCustomerEmail(customerEmail);

bookingCount++;

}

}

file.close();

}

}

public:

BookingManager() : bookingCount(0) {

loadBookingsFromFile();

}

bool addBooking(const string& tourName, const string& customerEmail) {

if (bookingCount >= MAX\_BOOKINGS) {

cout << "Достигнуто максимальное количество бронирований." << endl;

return false;

}

bookings[bookingCount++] = Booking(tourName, customerEmail);

saveBookingsToFile();

return true;

}

void addBookingFromInput() {

string tourName, customerEmail;

cout << "Введите название тура: ";

cin >> tourName;

cout << "Введите электронную почту клиента: ";

cin >> customerEmail;

addBooking(tourName, customerEmail);

cout << "\nБронирование успешно добавлено!\n";

}

void viewAllBookings() const {

for (int i = 0; i < bookingCount; ++i) {

bookings[i].viewBooking();

}

}

};

## 1.4.12. Класс Review

Назначение класса: Класс Review предназначен для представления отзывов клиентов о турах. Он включает в себя рейтинг, комментарий и электронную почту клиента, который оставил отзыв.

Описание полей и методов класса:

|  |  |  |
| --- | --- | --- |
| № | Сигнатура | Назначение |
| 1 | int rating; | Оценка, данная клиентом (например, от 1 до 5) |
| 2 | string comment; | Текстовый комментарий клиента. |
| 3 | string customerEmail; | Электронная почта клиента, который оставил отзыв. |
| 4 | Review() | Конструктор по умолчанию |
| 5 | Review(int rating, const string& comment, const string& customerEmail) | Конструктор с параметрами |
| 6 | void viewReview() const | Отображает информацию об отзыве, включая оценку, комментарий и электронную почту клиента. |

Так как поля этого класса приватные и чтобы получить доступ к этим полям, для каждого поле есть методы get и set.

Листинг класса:

class Review {

private:

int rating;

string comment;

string customerEmail;

public:

Review() : rating(0), comment(""), customerEmail("") {}

Review(int rating, const string& comment, const string& customerEmail)

: rating(rating), comment(comment), customerEmail(customerEmail) {}

void viewReview() const {

cout << "Отзыв - Оценка: " << rating << ", Комментарий: " << comment << ", Электронная почта клиента: " << customerEmail << endl;

}

void setRating(int rating) {

this->rating = rating;

}

void setComment(const string& comment) {

this->comment = comment;

}

void setCustomerEmail(const string& customerEmail) {

this->customerEmail = customerEmail;

}

int getRating() const {

return rating;

}

string getComment() const {

return comment;

}

string getCustomerEmail() const {

return customerEmail;

}

};

## 1.4.13. Класс ReviewManager

Назначение класса: Класс ReviewManager управляет отзывами клиентов о турах. Он позволяет добавлять новые отзывы, просматривать существующие отзывы и сохранять их в файл для последующего использования.

Описание полей и методов класса:

|  |  |  |
| --- | --- | --- |
| № | Сигнатура | Назначение |
| 1 | Review reviews[MAX\_REVIEWS]; | Массив объектов класса Review, который хранит информацию об отзывах. |
| 2 | int reviewCount; | Переменная, которая хранит текущее количество отзывов. |
| 3 | void saveReviewsToFile() | Сохраняет все текущие отзывы в файл "reviews.txt". Метод открывает файл, записывает данные о каждом отзыве и затем закрывает файл. |
| 4 | void loadReviewsFromFile() | Загружает отзывы из файла "reviews.txt". Метод открывает файл, считывает данные, инициализирует отзывы и увеличивает reviewCount. |
| 5 | ReviewManager() | Конструктор по умолчанию |
| 6 | bool addReview(int rating, const string& comment, const string& customerEmail) | Добавляет новый отзыв в массив reviews. Если достигнуто максимальное количество отзывов (MAX\_REVIEWS), выводится сообщение и возвращается false. В противном случае, отзыв добавляется в массив, данные сохраняются в файл, и метод возвращает true. |
| 7 | void addReviewFromInput() | Интерфейс для добавления отзыва с пользовательским вводом. Метод запрашивает у пользователя рейтинг, комментарий и электронную почту, а затем вызывает метод addReview для добавления отзыва. |
| 8 | void viewAllReviews() const | Отображает информацию о всех текущих отзывах. Проходит по массиву reviews и вызывает метод viewReview() для каждого отзыва. |

Так как поля этого класса приватные и чтобы получить доступ к этим полям, для каждого поле есть методы get и set.

Листинг класса:

class ReviewManager {

private:

Review reviews[MAX\_REVIEWS];

int reviewCount;

void saveReviewsToFile() {

ofstream file("reviews.txt");

if (file.is\_open()) {

for (int i = 0; i < reviewCount; ++i) {

file << reviews[i].getRating() << " | "

<< reviews[i].getComment() << " | "

<< reviews[i].getCustomerEmail() << endl;

}

file.close();

}

}

void loadReviewsFromFile() {

ifstream file("reviews.txt");

if (file.is\_open()) {

reviewCount = 0;

string line;

while (getline(file, line)) {

int rating;

string comment, customerEmail;

size\_t pos = 0;

string token;

vector<string> tokens;

while ((pos = line.find('|')) != string::npos) {

token = line.substr(0, pos);

tokens.push\_back(token);

line.erase(0, pos + 1);

}

tokens.push\_back(line);

if (tokens.size() == 3) {

rating = stoi(tokens[0]);

comment = tokens[1];

customerEmail = tokens[2];

reviews[reviewCount].setRating(rating);

reviews[reviewCount].setComment(comment);

reviews[reviewCount].setCustomerEmail(customerEmail);

reviewCount++;

}

}

file.close();

}

}

public:

ReviewManager() : reviewCount(0) {

loadReviewsFromFile();

}

bool addReview(int rating, const string& comment, const string& customerEmail) {

if (reviewCount >= MAX\_REVIEWS) {

cout << "Достигнуто максимальное количество отзывов." << endl;

return false;

}

reviews[reviewCount++] = Review(rating, comment, customerEmail);

saveReviewsToFile();

return true;

}

void addReviewFromInput() {

int rating;

string comment, customerEmail;

cout << "Введите рейтинг (от 1 до 5): ";

cin >> rating;

cout << "Введите комментарий: ";

cin >> comment;

cout << "Введите электронную почту клиента: ";

cin >> customerEmail;

addReview(rating, comment, customerEmail);

cout << "\nОтзыв успешно добавлен!\n";

}

void viewAllReviews() const {

for (int i = 0; i < reviewCount; ++i) {

reviews[i].viewReview();

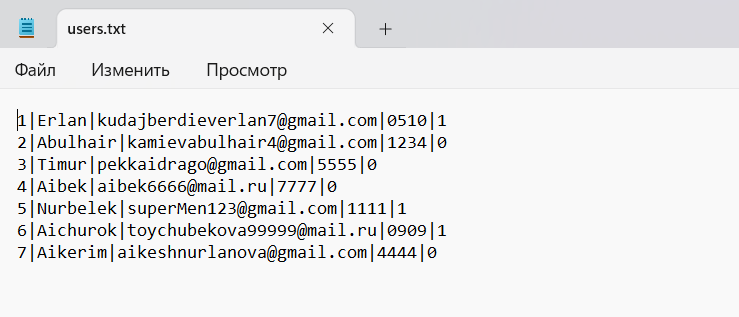
}

}

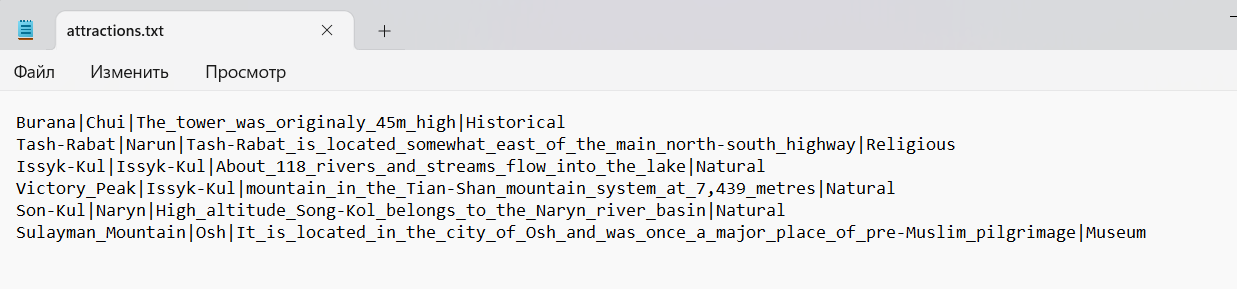
};

# 1.5. Хранения данных в проекте

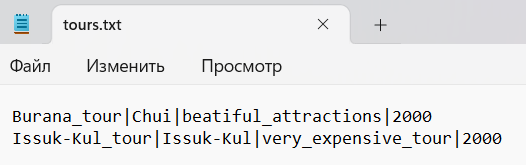
В представленном проекте данные хранятся в текстовых файлах: users.txt, tours.txt, bookings.txt и reviews.txt.



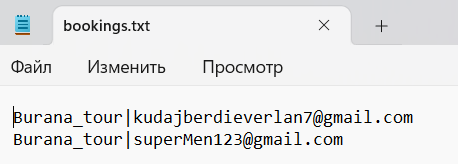
*Рисунок 3. Данные в user.txt*



*Рисунок 4. Данные в attractions.txt*



*Рисунок 5. Данные в tours.txt*



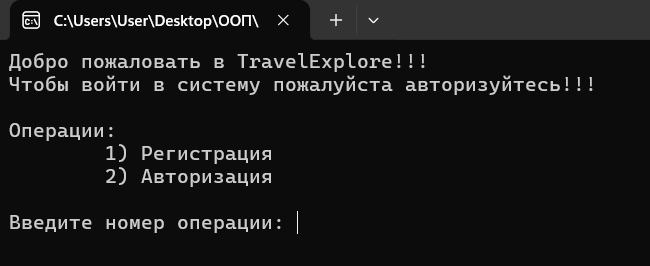
*Рисунок 6. Данные в bookings.txt*

Запись и чтения из текстового файла осуществляется с помощью библиотеки fstream:

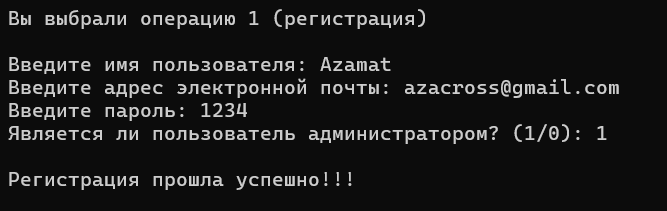
#include <fstream>

# 1.6. Скриншоты результатов работ

Главное окно для авторизации и регистрации пользователя:

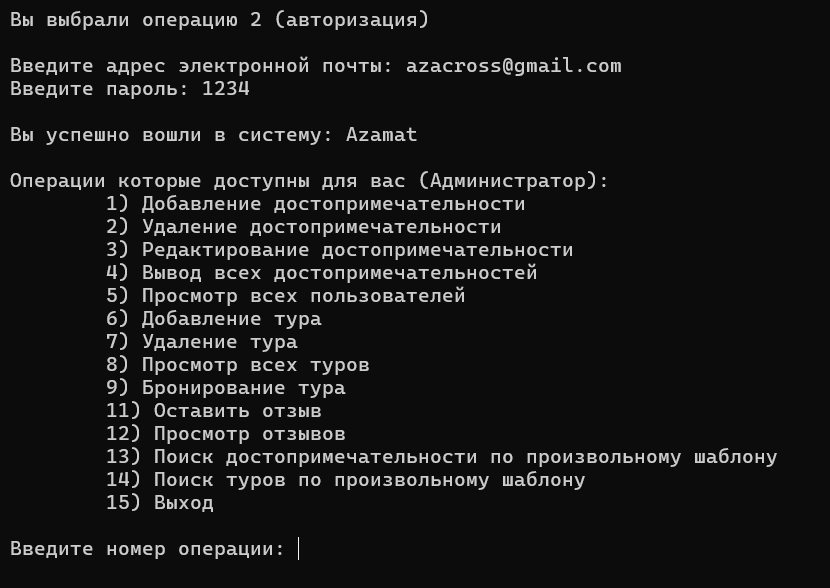


*Рисунок 7. Главное окно*

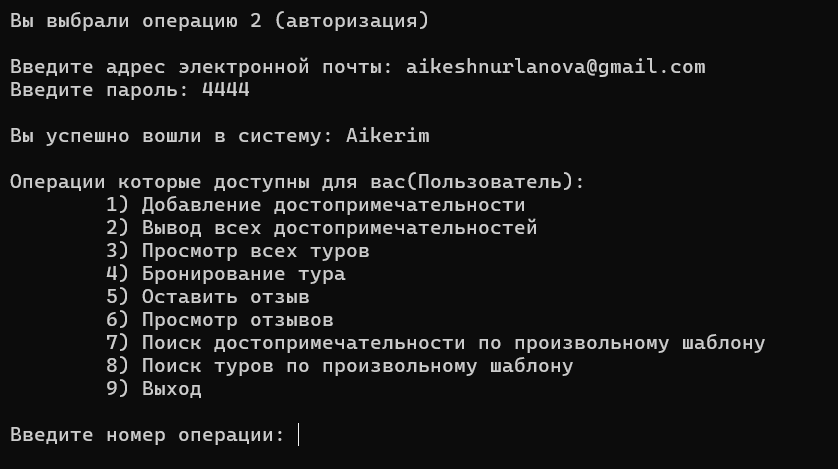
**

*Рисунок 8. Регистрация пользователя администратора*

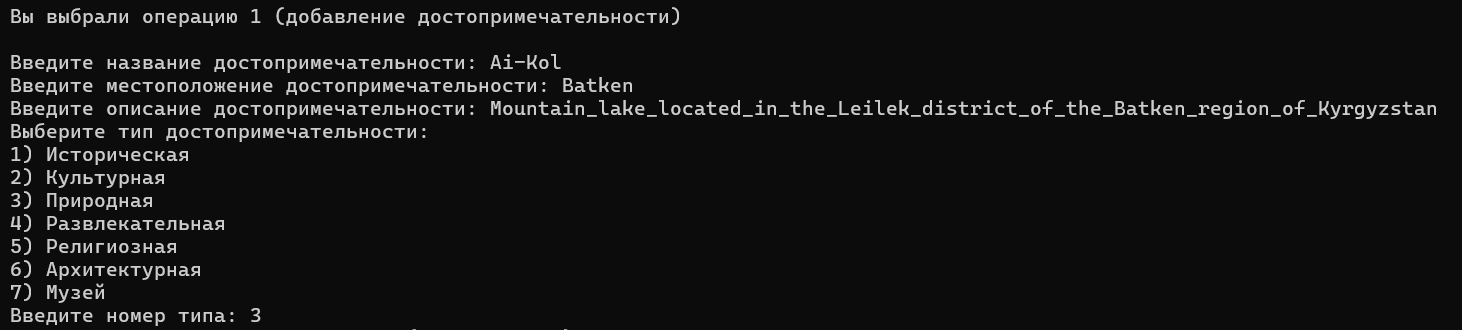
После успешной регистрации пользователь должен авторизоваться для входа систему.

**

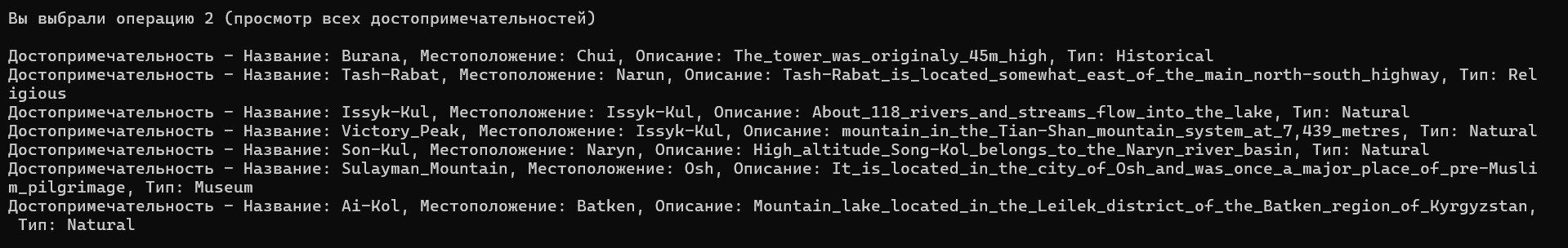
*Рисунок 9. Авторизация администратора и меню операции для него*

**

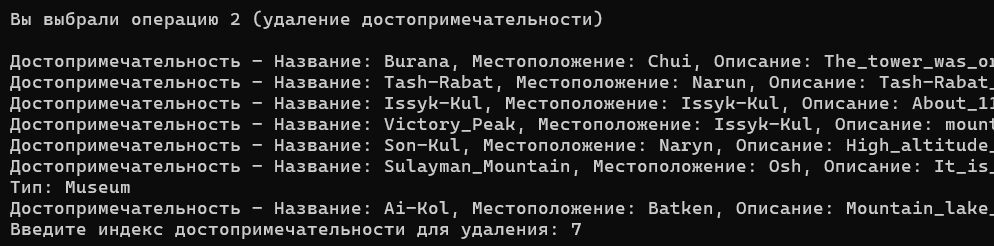
*Рисунок 10. Авторизация пользователя и меню операции для него*

**

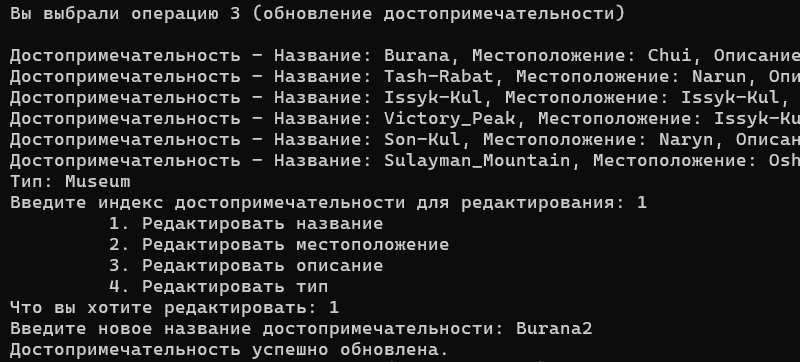
*Рисунок 11. Добавление достопримечательности*



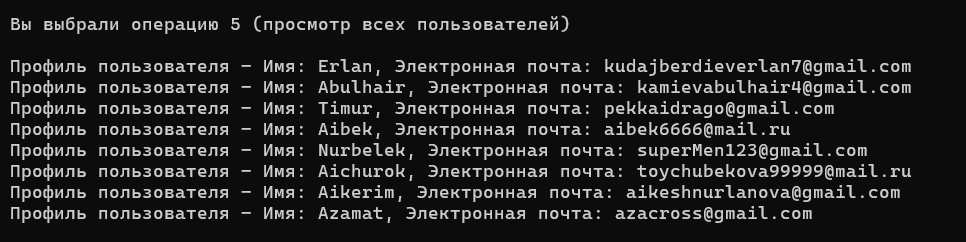
*Рисунок 12. Просмотр всех достопримечательностей*

**

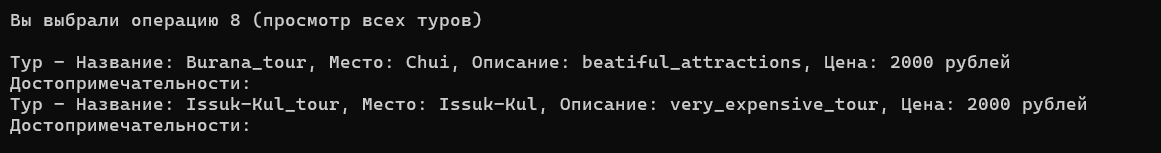
*Рисунок 13. Удаление достопримечательности*

**

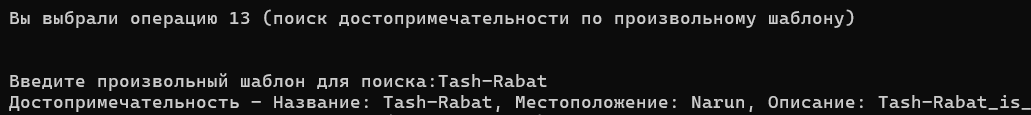
*Рисунок 14. Обновление достопримечательности*

**

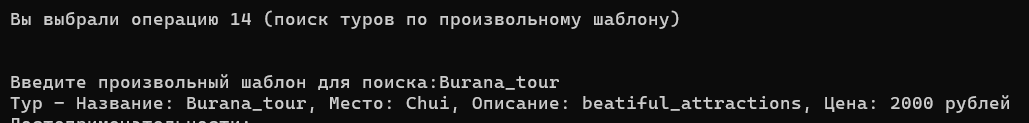
*Рисунок 15. Просмотр всех пользователей*

**

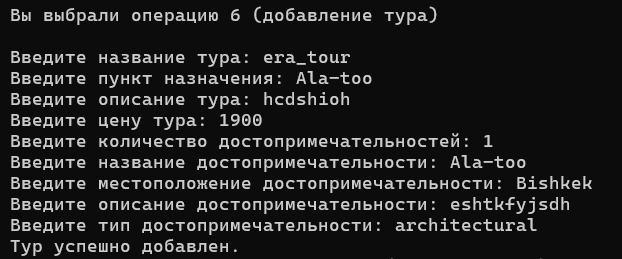
*Рисунок 16. Просмотр всех туров*

**

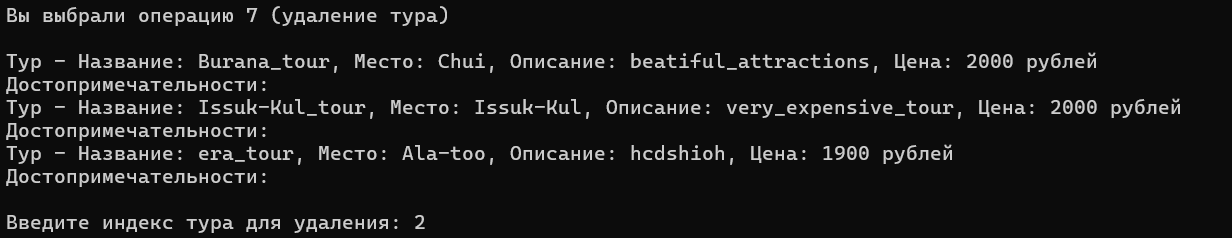
*Рисунок 16. Поиск достопримечательности по произвольному шаблону*

**

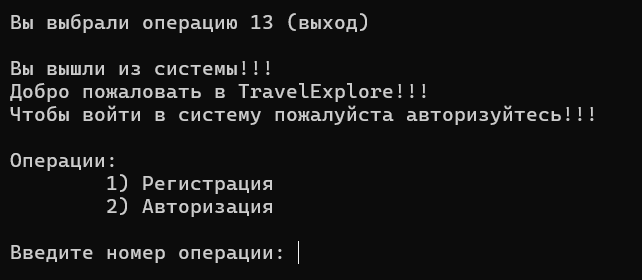
*Рисунок 17. Поиск тура по произвольному шаблону*

**

*Рисунок 17. Добавление тура*

**

*Рисунок 18. Удаление тура*

**

*Рисунок 19. Выход из системы*

# 1.7. Заключение

В своей работы я использовал все основные концепции объектно-ориентированного программирования: абстракцию, инкапсуляцию, наследование и полиморфизм. В результате разработки данной программы я улучшил навыки применения принципов объектно-ориентированного программирования (ООП) для создания эффективных и структурированных решений. Кроме того, я расширили свои знания и опыт работы с языком программирования C++, который предоставляет широкие возможности для создания производительного и надежного программного обеспечения.

Код проекта на GitHub можно посмотреть по ссылке:

https://github.com/erlan2003/OOP.git