## **Abstract**

This project develops an information system for managing a museum with a console interface. The system is implemented in **C++** using **MySQL** as the database management system. The program enables the registration of visitors, ticket sales, and the administration of tours, exhibitions, employees, and other objects of museum infrastructure. A multi-level authorization system with access control is implemented for administrators, managers, and regular users.

## **Contents**

* Abstract
* Introduction
* System Requirements
* Conceptual Class Model
* Interaction Diagrams
* Data Model
* Database Queries
* Class Diagram
* Program Testing
* Conclusion
* References
* Appendix A — Program Listing
* Appendix B — SQL Script for Table Creation

## **Introduction**

The aim of the project is to create a **console application** that automates the main processes of museum management, including ticket sales, tour planning, employee management, and exhibition administration. The system should be scalable and easy to use.

## **System Requirements**

### **Functional Requirements:**

* User login with roles (admin, manager, visitor)
* Viewing, adding, editing, deleting information about:  
  + Tickets
  + Visitors
  + Guides
  + Tours
  + Museums
  + Sponsors
  + Authors
  + Exhibitions
  + Ticket sales
  + Museum halls
* Ticket purchase without registration

### **Non-Functional Requirements:**

* Console interface
* Data stored in a **MySQL database**
* User-friendly interface
* Well-structured, modular code

## **Conceptual Class Model**

**Main classes**:

#### **1. Visitor**

* ID: int
* Name: string
* Email: string
* Ticket\_ID: int

#### **2. Ticket**

* ID: int
* Ticket\_Type: string
* Price: double
* Visitor\_Category: string
* Tour\_ID: int (nullable)
* Museum\_ID: int
* Sale\_Date: date (nullable)

#### **3. Tour**

* ID: int
* Name: string
* Tour\_Date: date
* Guide\_ID: int
* Museum\_ID: int

#### **4. Guide**

* ID: int
* Name: string
* Contact\_Info: string
* Years\_Experience: int
* Museum\_ID: int

#### **5. Museum**

* ID: int
* Name: string
* Address: string
* Contact\_Info: string

#### **6. Exhibition**

* ID: int
* Name: string
* Start\_Date: date
* End\_Date: date
* Price: double
* Museum\_ID: int
* Hall\_ID: int

#### **7. Author**

* ID: int
* Name: string
* Birth\_Date: date
* Nationality: string

#### **8. ExhibitionAuthor**

* Exhibition\_ID: int
* Author\_ID: int

#### **9. User**

* ID: int
* Username: string
* Password: string (hashed)
* Role: ENUM("admin", "manager", "user")

#### **10. Sponsor**

* ID: int
* Sponsor\_Name: string
* Contact\_Info: string
* Contribution\_Amount: double
* Contribution\_Date: date
* Museum\_ID: int

#### **11. Seller**

* ID: int
* Name: string
* Email: string
* Museum\_ID: int

#### **12. TicketSale**

* ID: int
* Ticket\_ID: int
* Seller\_ID: int
* Sale\_Date: date
* Quantity: int

#### **13. Hall**

* ID: int
* Name: string
* Museum\_ID: int
* Capacity: int

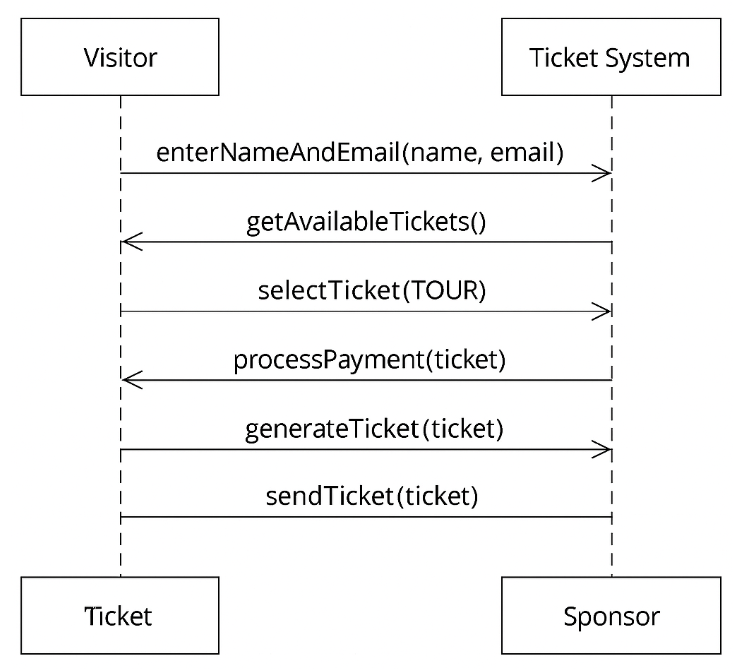
**Connections between classes**:

* Visitor — 1:1 → Ticket
* Ticket — *N:1* → Museum
* Ticket — *N:1* → Tour (nullable)
* Tour — *N:1* → Guide
* Tour — *N:1* → Museum
* Guide — *N:1* → Museum
* Exhibition — *N:1* → Museum, *N:1* → Hall
* ExhibitionAuthor — *N:M* → Exhibition ↔ Author
* TicketSale — *N:1* → Ticket, *N:1* → Seller
* Seller, Sponsor — *N:1* → Museum
* User — independent authorized unit with roles

## **Interaction Diagrams**

Interaction between the **user and system** during login and data viewing involves:

* Entering credentials
* Redirecting to the correct role-based menu
* Choosing operations (e.g., Add, View, Delete)



**Data Model**

## 

## **Database Queries**

Queries include:

* Viewing all exhibitions: SELECT \* FROM Exhibition;
* Inserting new visitor: INSERT INTO Visitor (...) VALUES (...);
* Deleting a ticket: DELETE FROM Ticket WHERE ID = ?;
* Updating guide info: UPDATE Guide SET ... WHERE ID = ?;

sql::PreparedStatement\* pstmt = db.getConnection()->prepareStatement(

"SELECT t.ID, t.Ticket\_Type, t.Price, t.Visitor\_Category, "

"tour.Name AS Tour\_Name, museum.Name AS Museum\_Name "

"FROM Ticket t "

"LEFT JOIN Tour tour ON t.Tour\_ID = tour.ID "

"LEFT JOIN Museum museum ON t.Museum\_ID = museum.ID"

);

sql::PreparedStatement\* pstmt = db.getConnection()->prepareStatement(

"SELECT Price FROM Ticket WHERE ID = ?");

sql::PreparedStatement\* insertStmt = db.getConnection()->prepareStatement(

"INSERT INTO Visitor (Ticket\_ID, Name, Email) VALUES (?, ?, ?)");

sql::PreparedStatement\* pstmt = db.getConnection()->prepareStatement(

"SELECT v.Name, v.Email, t.Ticket\_Type, t.Visitor\_Category, t.Price, t.Sale\_Date, "

"tour.Name AS Tour\_Name, mu.Name AS Museum\_Name "

"FROM visitor v "

"JOIN ticket t ON v.Ticket\_ID = t.ID "

"LEFT JOIN Tour tour ON t.Tour\_ID = tour.ID "

"LEFT JOIN Museum mu ON t.Museum\_ID = mu.ID "

"WHERE v.Name = ? AND v.Email = ?"

);

sql::PreparedStatement\* pstmt = db.getConnection()->prepareStatement(

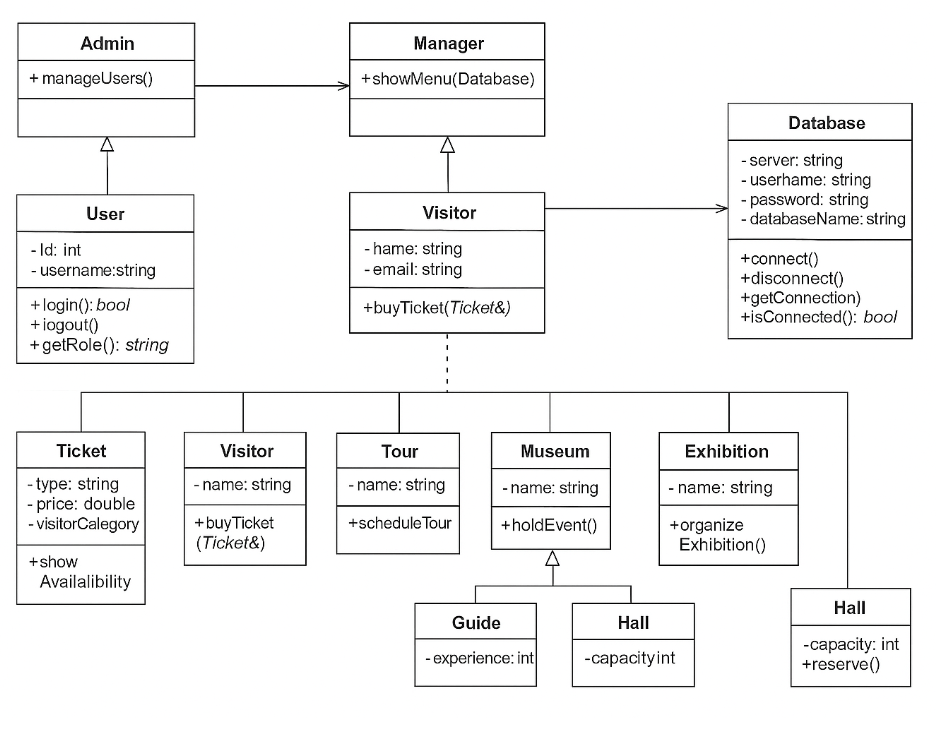
"SELECT tu.ID, tu.Date, "

"Guide.Name AS Guide\_Name "

"FROM Tour tu "

"LEFT JOIN Guide guide ON tu.Guide\_ID = Guide.ID");

**Class Diagram**



## **Program Testing**

### **1. Testing Individual Methods (White-Box Testing)**

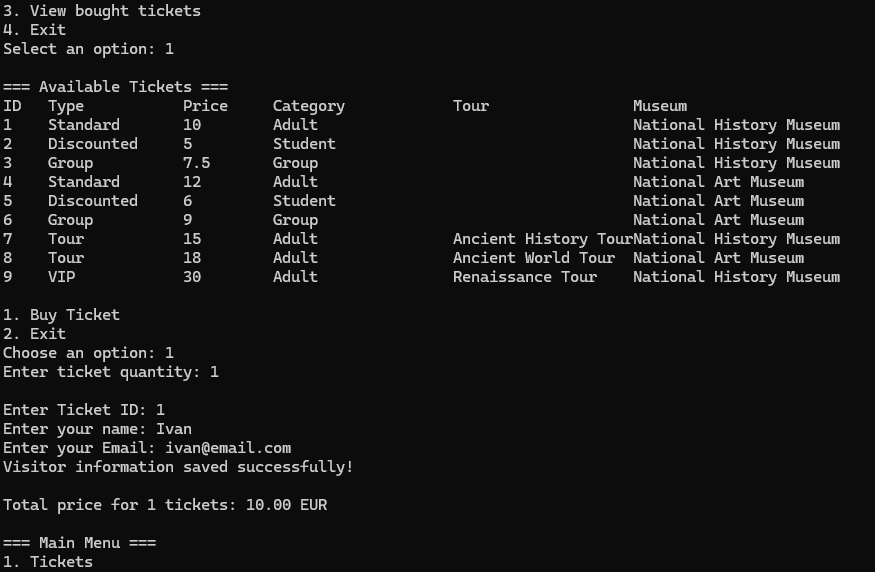
**Goal**: To verify the correctness of the logic within each method.

✅ **Checklist for White-Box Testing** (method TicketSale):

* Is the name read correctly?
* Is the email read correctly?
* Is the SQL INSERT query executed successfully?
* Are exceptions handled properly?

🔍 **Test Case:**

| **Method** | **Input Data** | **Expected Result** |
| --- | --- | --- |
| TicketSale | Name: "Ivan", Email: "ivan@email.com", Ticket\_ID: 1 | Message displayed: “Visitor added successfully” |
| TicketSale | Invalid email, non-existent ID | SQL error message displayed |





### **2. Class Testing (Gray-Box Testing)**

**Goal**: To verify the interaction of methods within classes.

✅ **Checklist for Gray-Box Testing** (AdminVisitor):

* Is the Database object passed correctly?
* Are ViewVisitor, AddVisitor, and DeleteVisitor methods correctly linked to the menu?
* Is the state of the table updated after calling AddVisitor()?

🔍 **Test Cases:**

| **#** | **Class** | **Called Methods** | **Input Data** | **Expected Result** |
| --- | --- | --- | --- | --- |
| 1 | AdminVisitor | VisitorMenu → AddVisitor | Name: A, Email: B | New visitor is added |
| 2 | AdminVisitor | VisitorMenu → ViewVisitor | - | Visitor information shown |



### **3. Application Testing (Black-Box Testing)**

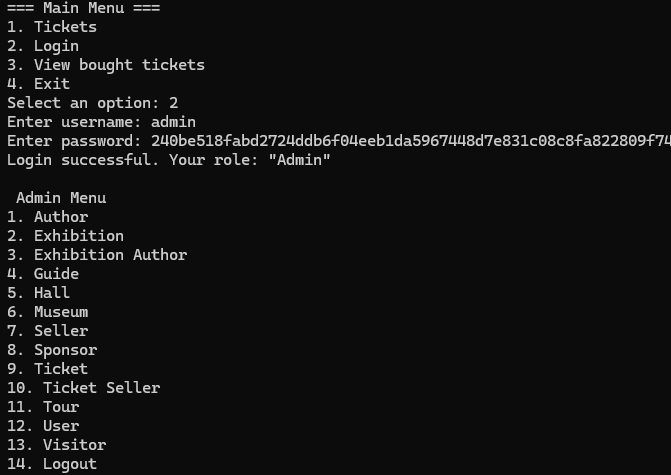
**Goal**: To test the functionality without access to source code.

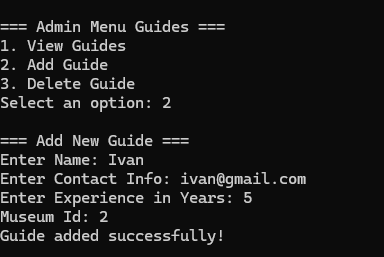
✅ **Checklist**:

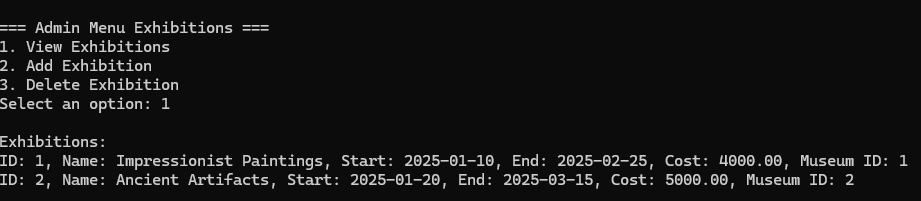
* Is the main menu displayed?
* Is login possible?
* Can a new guide be added?
* Are all exhibitions displayed?
* Does ticket purchase work correctly?

🔍 **Test Cases:**

| **#** | **Scenario** | **Steps** | **Expected Result** |
| --- | --- | --- | --- |
| 1 | Login as Administrator | Select "Login" → Enter username and password | Admin menu is shown |
| 2 | View Tours | In the menu, choose "View Tours" | A list of tours is displayed |









## **Conclusion**

As a result, a fully functioning **console-based museum information system** has been implemented. The system handles most administrative processes efficiently. It offers user-friendly data handling, authorization-based access, and is scalable.

## **References**

* **MySQL Documentation**
* **Official C++ Documentation**
* **YouTube tutorials on C++**
* **OpenAI ChatGPT — technical consulting**
* **MySQL Connector/C++ Documentation**

## **Program Listing**

**├── main.cpp**

#include "Manager.h"

#include "Database.h"

#include "Ticket.h"

#include "Auth.h"

#include "TicketSale.h"

#include "Admin.h" // якщо є окрема логіка

#include "ViewTicketsVisitors.h"

#include <iostream>

using namespace std;

void showMainMenu(Database& db) {

int choice;

string role = "";

while (true) {

cout << "\n=== Main Menu ===\n";

cout << "1. Tickets\n";

cout << "2. Login\n";

cout << "3. View bought tickets\n";

cout << "4. Exit\n";

cout << "Select an option: ";

cin >> choice;

switch (choice) {

case 1:

TicketMenu::showTicketsMenu(db);

break;

case 2:

if (Auth::login(db, role)) {

Auth::handleRoleMenu(db, role);

}

break;

case 3:

ViewTicketsVisitors::ViewTickets(db);

break;

case 4:

cout << "Goodbye!\n";

return;

default:

cout << "Invalid option. Please try again.\n";

}

}

}

int main() {

try {

Database db;

db.connect();

if (!db.isConnected()) {

cout << "Database connection failed!" << endl;

return 1;

}

showMainMenu(db);

}

catch (sql::SQLException& e) {

cerr << "Database error: " << e.what() << endl;

return 1;

}

return 0;

}

**├── database**

**│ ├── Database.h**

#ifndef DATABASE\_H

#define DATABASE\_H

#include <cppconn/driver.h>

#include <cppconn/connection.h>

#include <string>

class Database {

private:

std::string server;

std::string username;

std::string password;

std::string database\_Name;

sql::Driver\* driver;

sql::Connection\* connection;

public:

Database(

const std::string& server = "tcp://127.0.0.1:3306",

const std::string& username = "root",

const std::string& password = "4321Qwer",

const std::string& database\_Name = "museum"

);

~Database();

void connect();

void disconnect();

bool isConnected() const;

sql::Connection\* getConnection();

};

#endif

**│ └── Database.cpp**

#include "Database.h"

#include <iostream>

Database::Database(const std::string& server, const std::string& username, const std::string& password, const std::string& database\_Name)

: server(server), username(username), password(password), database\_Name(database\_Name), driver(nullptr), connection(nullptr) {

}

Database::~Database() {

disconnect();

}

void Database::connect() {

try {

driver = get\_driver\_instance();

connection = driver->connect(server, username, password);

connection->setSchema(database\_Name);

std::cout << "Connected to the database successfully!" << std::endl;

}

catch (sql::SQLException& e) {

std::cerr << "SQL Error: " << e.what() << " (SQLState: " << e.getSQLState() << ")" << std::endl;

}

}

void Database::disconnect() {

if (connection) {

delete connection;

connection = nullptr;

std::cout << "Disconnected from the database." << std::endl;

}

}

sql::Connection\* Database::getConnection() {

if (!connection) {

connect();

}

return connection;

}

bool Database::isConnected() const {

return connection != nullptr;

}

**├── Users**

**│ ├── Manager**

**│ │ ├── Manager.h**

#ifndef MANAGER\_H

#define MANAGER\_H

#include "Database.h"

class Manager {

public:

static void ManagerMenu(Database& db);

};

#endif

**│ │ ├── Manager.cpp**

#include "Manager.h"

#include "ManagerTicket.h"

#include "ManagerVisitor.h"

#include "ManagerTicketSale.h"

#include "ManagerTour.h"

#include "ManagerExhibition.h"

#include <iostream>

using namespace std;

void Manager::ManagerMenu(Database& db) {

int choice;

while (true) {

cout << "\n=== Manager Menu ===\n";

cout << "1. Tickets\n";

cout << "2. Visitors\n";

cout << "3. Tours\n";

cout << "4. Exhibitions\n";

cout << "5. Logout\n";

cout << "Select an option: ";

cin >> choice;

switch (choice) {

case 1:

ManagerTicket::menu(db);

break;

case 2:

ManagerVisitor::menu(db);

break;

case 3:

ManagerTour::menu(db);

break;

case 4:

ManagerExhibition::menu(db);

break;

case 5:

return;

default:

cout << "Invalid choice. Try again.\n";

}

}

}

**│ │ ├── ManagerTicket.h**

#pragma once

#ifndef MANAGER\_TICKET\_H

#define MANAGER\_TICKET\_H

#include "Database.h"

namespace ManagerTicket {

void menu(Database& db);

void viewTickets(Database& db);

void editTicketPrice(Database& db);

}

#endif

**│ │ ├── ManagerTicket.cpp**

#include "ManagerTicket.h"

#include <iostream>

#include <cppconn/prepared\_statement.h>

using namespace std;

void ManagerTicket::menu(Database& db) {

int choice;

cout << "\n=== Manager Menu Tickets ===\n";

cout << "1. View Tickets\n";

cout << "2. Edit Ticket Price\n";

cout << "Select an option: ";

cin >> choice;

switch (choice) {

case 1:

viewTickets(db);

break;

case 2:

editTicketPrice(db);

break;

default:

cout << "Invalid choice.\n";

}

}

void ManagerTicket::viewTickets(Database& db) {

try {

sql::Statement\* stmt = db.getConnection()->createStatement();

sql::ResultSet\* res = stmt->executeQuery("SELECT ID, Ticket\_Type, Price FROM ticket");

cout << "\nTickets:\n";

while (res->next()) {

cout << "ID: " << res->getInt("ID")

<< ", Type: " << res->getString("Ticket\_Type")

<< ", Price: " << res->getDouble("Price") << endl;

}

delete res;

delete stmt;

}

catch (sql::SQLException& e) {

cerr << "SQL Error: " << e.what() << endl;

}

}

void ManagerTicket::editTicketPrice(Database& db) {

int ticket\_id;

double new\_price;

cout << "Enter Ticket ID: ";

cin >> ticket\_id;

cout << "Enter New Price: ";

cin >> new\_price;

try {

sql::PreparedStatement\* pstmt = db.getConnection()->prepareStatement(

"UPDATE ticket SET price = ? WHERE id = ?"

);

pstmt->setDouble(1, new\_price);

pstmt->setInt(2, ticket\_id);

pstmt->executeUpdate();

cout << "Price updated successfully.\n";

delete pstmt;

}

catch (sql::SQLException& e) {

cout << "SQL Error: " << e.what() << endl;

}

}

**│ │ ├── ManagerVisitor.h**

#pragma once

#ifndef MANAGER\_VISITOR\_H

#define MANAGER\_VISITOR\_H

#include "Database.h"

namespace ManagerVisitor {

void menu(Database& db);

void viewVisitors(Database& db);

void deleteVisitor(Database& db);

}

#endif

**│ │ ├── ManagerVisitor.cpp**

#include "ManagerVisitor.h"

#include <iostream>

#include <cppconn/prepared\_statement.h>

using namespace std;

void ManagerVisitor::menu(Database& db) {

int choice;

cout << "\n=== Manager Menu Visitors ===\n";

cout << "1. View Visitors\n";

cout << "2. Delete Visitor\n";

cout << "Select an option: ";

cin >> choice;

switch (choice) {

case 1:

viewVisitors(db);

break;

case 2:

deleteVisitor(db);

break;

default:

cout << "Invalid choice.\n";

}

}

void ManagerVisitor::viewVisitors(Database& db) {

try {

sql::Statement\* stmt = db.getConnection()->createStatement();

sql::ResultSet\* res = stmt->executeQuery("SELECT \* FROM visitor");

cout << "\nVisitors:\n";

while (res->next()) {

cout << "ID: " << res->getInt("ID")

<< ", Name: " << res->getString("Name")

<< ", Email: " << res->getString("Email") << endl;

}

delete res;

delete stmt;

}

catch (sql::SQLException& e) {

cout << "SQL Error: " << e.what() << endl;

}

}

void ManagerVisitor::deleteVisitor(Database& db) {

int id;

cout << "Enter Visitor ID to delete: ";

cin >> id;

try {

sql::PreparedStatement\* pstmt = db.getConnection()->prepareStatement(

"DELETE FROM visitor WHERE id = ?"

);

pstmt->setInt(1, id);

int affected = pstmt->executeUpdate();

if (affected > 0)

cout << "Visitor deleted successfully.\n";

else

cout << "No visitor found with that ID.\n";

delete pstmt;

}

catch (sql::SQLException& e) {

cout << "SQL Error: " << e.what() << endl;

}

}

**│ │ ├── ManagerTicketSale.h**

#pragma once

#ifndef MANAGER\_TICKET\_SALE\_H

#define MANAGER\_TICKET\_SALE\_H

#include "Database.h"

namespace ManagerTicketSale {

void menu(Database& db);

void viewTicketSales(Database& db);

void editSalesDate(Database& db);

void editNumberOfTickets(Database& db);

}

#endif

**│ │ ├── ManagerTicketSale.cpp**

#include "ManagerTicketSale.h"

#include <iostream>

#include <cppconn/prepared\_statement.h>

using namespace std;

void ManagerTicketSale::menu(Database& db) {

int choice;

cout << "\n=== Manager Menu Ticket Sales ===\n";

cout << "1. View Ticket Sales\n";

cout << "2. Edit Sales Date\n";

cout << "3. Edit Number of Tickets\n";

cout << "Select an option: ";

cin >> choice;

switch (choice) {

case 1:

viewTicketSales(db);

break;

case 2:

editSalesDate(db);

break;

case 3:

editNumberOfTickets(db);

break;

default:

cout << "Invalid choice.\n";

}

}

void ManagerTicketSale::viewTicketSales(Database& db) {

try {

sql::Statement\* stmt = db.getConnection()->createStatement();

sql::ResultSet\* res = stmt->executeQuery("SELECT \* FROM ticket\_sale");

cout << "\nTicket Sales:\n";

while (res->next()) {

cout << "Seller ID: " << res->getInt("ID")

<< ", Ticket ID: " << res->getInt("Ticket\_ID") << endl;

}

delete res;

delete stmt;

}

catch (sql::SQLException& e) {

cout << "SQL Error: " << e.what() << endl;

}

}

void ManagerTicketSale::editSalesDate(Database& db) {

int id;

string new\_date;

cout << "Enter Sale ID: ";

cin >> id;

cout << "Enter New Sale Date (YYYY-MM-DD): ";

cin >> new\_date;

try {

sql::PreparedStatement\* pstmt = db.getConnection()->prepareStatement(

"UPDATE ticket\_sale SET sale\_date = ? WHERE id = ?"

);

pstmt->setString(1, new\_date);

pstmt->setInt(2, id);

pstmt->executeUpdate();

cout << "Sale date updated.\n";

delete pstmt;

}

catch (sql::SQLException& e) {

cout << "SQL Error: " << e.what() << endl;

}

}

void ManagerTicketSale::editNumberOfTickets(Database& db) {

int id, quantity;

cout << "Enter Sale ID: ";

cin >> id;

cout << "Enter New Quantity: ";

cin >> quantity;

try {

sql::PreparedStatement\* pstmt = db.getConnection()->prepareStatement(

"UPDATE ticket\_sale SET quantity = ? WHERE id = ?"

);

pstmt->setInt(1, quantity);

pstmt->setInt(2, id);

pstmt->executeUpdate();

cout << "Quantity updated.\n";

delete pstmt;

}

catch (sql::SQLException& e) {

cout << "SQL Error: " << e.what() << endl;

}

}

**│ │ ├── ManagerTour.h**

#pragma once

#ifndef MANAGER\_TOUR\_H

#define MANAGER\_TOUR\_H

#include "Database.h"

namespace ManagerTour {

void menu(Database& db);

void viewTours(Database& db);

void editGuideId(Database& db);

void editTourDate(Database& db);

}

#endif

**│ │ ├── ManagerTour.cpp**

#include "ManagerTour.h"

#include <iostream>

#include <cppconn/prepared\_statement.h>

using namespace std;

void ManagerTour::menu(Database& db) {

int choice;

cout << "\n=== Manager Menu Tours ===\n";

cout << "1. View Tours\n";

cout << "2. Edit Guide ID\n";

cout << "3. Edit Tour Date\n";

cout << "Select an option: ";

cin >> choice;

switch (choice) {

case 1:

viewTours(db);

break;

case 2:

editGuideId(db);

break;

case 3:

editTourDate(db);

break;

default:

cout << "Invalid choice.\n";

}

}

void ManagerTour::viewTours(Database& db) {

try {

sql::PreparedStatement\* pstmt = db.getConnection()->prepareStatement(

"SELECT tu.ID, tu.Date, "

"Guide.Name AS Guide\_Name "

"FROM Tour tu "

"LEFT JOIN Guide guide ON tu.Guide\_ID = Guide.ID");

sql::ResultSet\* res = pstmt->executeQuery();

cout << "\nTours:\n";

while (res->next()) {

cout << "Tour ID: " << res->getInt("ID")

<< ", Tour Date: " << res->getString("Date")

<< ", Guide Name: " << res->getString("Guide\_Name") << endl;

}

delete res;

delete pstmt;

}

catch (sql::SQLException& e) {

cout << "SQL Error: " << e.what() << endl;

}

}

void ManagerTour::editGuideId(Database& db) {

int tour\_id, new\_guide\_id;

cout << "Enter Tour ID: ";

cin >> tour\_id;

cout << "Enter New Guide ID: ";

cin >> new\_guide\_id;

try {

sql::PreparedStatement\* pstmt = db.getConnection()->prepareStatement(

"UPDATE tour SET guide\_id = ? WHERE id = ?"

);

pstmt->setInt(1, new\_guide\_id);

pstmt->setInt(2, tour\_id);

pstmt->executeUpdate();

cout << "Guide ID updated.\n";

delete pstmt;

}

catch (sql::SQLException& e) {

cout << "SQL Error: " << e.what() << endl;

}

}

void ManagerTour::editTourDate(Database& db) {

int tour\_id;

string new\_date;

cout << "Enter Tour ID: ";

cin >> tour\_id;

cout << "Enter New Tour Date (YYYY-MM-DD): ";

cin >> new\_date;

try {

sql::PreparedStatement\* pstmt = db.getConnection()->prepareStatement(

"UPDATE tour SET Date = ? WHERE id = ?"

);

pstmt->setString(1, new\_date);

pstmt->setInt(2, tour\_id);

pstmt->executeUpdate();

cout << "Tour date updated.\n";

delete pstmt;

}

catch (sql::SQLException& e) {

cout << "SQL Error: " << e.what() << endl;

}

}

**│ │ └── ManagerExhibition.h**

#pragma once

#ifndef MANAGER\_EXHIBITION\_H

#define MANAGER\_EXHIBITION\_H

#include "Database.h"

namespace ManagerExhibition {

void menu(Database& db);

void viewExhibitions(Database& db);

void editStartDate(Database& db);

void editEndDate(Database& db);

void editExhibitionPrice(Database& db);

}

#endif

**│ │ └── ManagerExhibition.cpp**

#include "ManagerExhibition.h"

#include <iostream>

#include <cppconn/prepared\_statement.h>

using namespace std;

void ManagerExhibition::menu(Database& db) {

int choice;

cout << "\n=== Manager Menu Exhibition ===\n";

cout << "1. View Exhibition\n";

cout << "2. Edit Exhibition Start Date\n";

cout << "3. Edit Exhibition End Date\n";

cout << "4. Edit Exhibition Price\n";

cout << "Select an option: ";

cin >> choice;

switch (choice) {

case 1:

viewExhibitions(db);

break;

case 2:

editStartDate(db);

break;

case 3:

editEndDate(db);

break;

case 4:

editExhibitionPrice(db);

break;

default:

cout << "Invalid choice.\n";

}

}

void ManagerExhibition::viewExhibitions(Database& db) {

try {

sql::Statement\* stmt = db.getConnection()->createStatement();

sql::ResultSet\* res = stmt->executeQuery("SELECT \* FROM exhibition");

cout << "\nExhibitions:\n";

while (res->next()) {

cout << "ID: " << res->getInt("ID")

<< ", Name: " << res->getString("Name")

<< ", Start: " << res->getString("Start\_Date")

<< ", End: " << res->getString("End\_Date")

<< ", Cost: " << res->getDouble("Cost") << endl;

}

delete res;

delete stmt;

}

catch (sql::SQLException& e) {

cout << "SQL Error: " << e.what() << endl;

}

}

void ManagerExhibition::editStartDate(Database& db) {

int id;

string new\_date;

cout << "Enter Exhibition ID: ";

cin >> id;

cout << "Enter New Start Date (YYYY-MM-DD): ";

cin >> new\_date;

try {

sql::PreparedStatement\* pstmt = db.getConnection()->prepareStatement(

"UPDATE exhibition SET start\_date = ? WHERE id = ?"

);

pstmt->setString(1, new\_date);

pstmt->setInt(2, id);

pstmt->executeUpdate();

cout << "Start date updated.\n";

delete pstmt;

}

catch (sql::SQLException& e) {

cout << "SQL Error: " << e.what() << endl;

}

}

void ManagerExhibition::editEndDate(Database& db) {

int id;

string new\_date;

cout << "Enter Exhibition ID: ";

cin >> id;

cout << "Enter New End Date (YYYY-MM-DD): ";

cin >> new\_date;

try {

sql::PreparedStatement\* pstmt = db.getConnection()->prepareStatement(

"UPDATE exhibition SET end\_date = ? WHERE id = ?"

);

pstmt->setString(1, new\_date);

pstmt->setInt(2, id);

pstmt->executeUpdate();

cout << "End date updated.\n";

delete pstmt;

}

catch (sql::SQLException& e) {

cout << "SQL Error: " << e.what() << endl;

}

}

void ManagerExhibition::editExhibitionPrice(Database& db) {

int id;

double new\_price;

cout << "Enter Exhibition ID: ";

cin >> id;

cout << "Enter New Cost: ";

cin >> new\_price;

try {

sql::PreparedStatement\* pstmt = db.getConnection()->prepareStatement(

"UPDATE exhibition SET Cost = ? WHERE id = ?"

);

pstmt->setDouble(1, new\_price);

pstmt->setInt(2, id);

pstmt->executeUpdate();

cout << "Cost updated.\n";

delete pstmt;

}

catch (sql::SQLException& e) {

cout << "SQL Error: " << e.what() << endl;

}

}

**│ ├── Admin**

**│ │ ├── Admin.h**

#ifndef ADMIN\_H

#define ADMIN\_H

#include "Database.h"

class Admin {

public:

static void AdminMenu(Database& db);

};

#endif

**│ │ ├── Admin.cpp**

#include "Admin.h"

#include "AdminAuthor.h"

#include "AdminExhibition.h"

#include "AdminExhibitionAuthor.h"

#include "AdminGuide.h"

#include "AdminHall.h"

#include "AdminMuseum.h"

#include "AdminSeller.h"

#include "AdminSponsor.h"

#include "AdminTicket.h"

#include "AdminTicketSeller.h"

#include "AdminTour.h"

#include "AdminUser.h"

#include "AdminVisitor.h"

#include <iostream>

using namespace std;

void Admin::AdminMenu(Database& db)

{

int choice;

while (true)

{

cout << "\n Admin Menu \n";

cout << "1. Author \n";

cout << "2. Exhibition \n";

cout << "3. Exhibition Author\n";

cout << "4. Guide \n";

cout << "5. Hall \n";

cout << "6. Museum\n";

cout << "7. Seller\n";

cout << "8. Sponsor\n";

cout << "9. Ticket\n";

cout << "10. Ticket Seller\n";

cout << "11. Tour\n";

cout << "12. User\n";

cout << "13. Visitor\n";

cout << "14. Logout\n";

cout << "Enter choice: ";

cin >> choice;

switch (choice)

{

case 1:

AdminAuthor::AuthorMenu(db);

break;

case 2:

AdminExhibition::ExhibitionMenu(db);

break;

case 3:

AdminExhibitionAuthor::ExhibitionAuthorMenu(db);

break;

case 4:

AdminGuide::GuideMenu(db);

break;

case 5:

AdminHall::HallMenu(db);

break;

case 6:

AdminMuseum::MuseumMenu(db);

break;

case 7:

AdminSeller::SellerMenu(db);

break;

case 8:

AdminSponsor::SponsorMenu(db);

break;

case 9:

AdminTicket::TicketMenu(db);

break;

case 10:

AdminTicketSeller::TicketSellerMenu(db);

break;

case 11:

AdminTour::TourMenu(db);

break;

case 12:

AdminUser::UserMenu(db);

break;

case 13:

AdminVisitor::VisitorMenu(db);

break;

case 14:

return;

default:

cout << "Invalid choice. Try again. \n";

}

}

}

**│ │ ├──AdminAuthor.h**

#pragma once

#ifndef ADMIN\_AUTHOR\_H

#define ADMIN\_AUTHOR\_H

#include "Database.h"

namespace AdminAuthor

{

void AuthorMenu(Database& db);

void ViewAuthor(Database& db);

void AddAuthor(Database& db);

void DeleteAuthor(Database& db);

}

#endif

**│ │ ├──AdminAuthor.cpp**

#include "AdminAuthor.h"

#include <string>

#include <iostream>

#include <cppconn/prepared\_statement.h>

using namespace std;

void AdminAuthor::AuthorMenu(Database& db) {

int choice;

cout << "\n=== Admin Menu Authors ===\n";

cout << "1. View Authors\n";

cout << "2. Add Author\n";

cout << "3. Delete Author\n";

cout << "Select an option: ";

cin >> choice;

switch (choice) {

case 1:

ViewAuthor(db);

break;

case 2:

AddAuthor(db);

break;

case 3:

DeleteAuthor(db);

break;

default:

cout << "Invalid choice.\n";

}

}

void AdminAuthor::ViewAuthor(Database& db) {

try {

sql::Statement\* stmt = db.getConnection()->createStatement();

sql::ResultSet\* res = stmt->executeQuery("SELECT \* FROM Author");

cout << "\nAuthors:\n";

while (res->next()) {

cout << "ID: " << res->getInt("ID")

<< ", Name: " << res->getString("Name")

<< ", Email: " << res->getString("Email")

<< ", Contract: " << res->getString("Agreement") << endl;

}

delete res;

delete stmt;

}

catch (sql::SQLException& e) {

cout << "SQL Error: " << e.what() << endl;

}

}

void AdminAuthor::AddAuthor(Database& db) {

string name, email, contract;

cout << "\n=== Add New Author ===\n";

cin.ignore();

cout << "Enter Name: ";

getline(cin, name);

cout << "Enter Email: ";

getline(cin, email);

cout << "Enter Contract: ";

getline(cin, contract);

try {

sql::PreparedStatement\* pstmt = db.getConnection()->prepareStatement(

"INSERT INTO Author (Name, Email, Agreement) VALUES (?, ?, ?)"

);

pstmt->setString(1, name);

pstmt->setString(2, email);

pstmt->setString(3, contract);

pstmt->executeUpdate();

delete pstmt;

cout << "Author added successfully!\n";

}

catch (sql::SQLException& e) {

cout << "SQL Error: " << e.what() << endl;

}

}

void AdminAuthor::DeleteAuthor(Database& db) {

int id;

cout << "Enter Author ID to delete: ";

cin >> id;

try {

sql::PreparedStatement\* pstmt = db.getConnection()->prepareStatement(

"DELETE FROM Author WHERE ID = ?"

);

pstmt->setInt(1, id);

int affected = pstmt->executeUpdate();

if (affected > 0)

cout << "Author deleted successfully.\n";

else

cout << "No author found with that ID.\n";

delete pstmt;

}

catch (sql::SQLException& e) {

cout << "SQL Error: " << e.what() << endl;

}

}

**│ │ ├──AdminExhibition.h**

#pragma once

#ifndef ADMIN\_EXHIBITION\_H

#define ADMIN\_EXHIBITION\_H

#include "Database.h"

namespace AdminExhibition

{

void ExhibitionMenu(Database& db);

void ViewExhibition(Database& db);

void AddExhibition(Database& db);

void DeleteExhibition(Database& db);

}

#endif

**│ │ ├──AdminExhibition.cpp**

#include "AdminExhibition.h"

#include <iostream>

#include <cppconn/prepared\_statement.h>

using namespace std;

void AdminExhibition::ExhibitionMenu(Database& db) {

int choice;

cout << "\n=== Admin Menu Exhibitions ===\n";

cout << "1. View Exhibitions\n";

cout << "2. Add Exhibition\n";

cout << "3. Delete Exhibition\n";

cout << "Select an option: ";

cin >> choice;

switch (choice) {

case 1: ViewExhibition(db); break;

case 2: AddExhibition(db); break;

case 3: DeleteExhibition(db); break;

default: cout << "Invalid choice.\n";

}

}

void AdminExhibition::ViewExhibition(Database& db) {

try {

sql::Statement\* stmt = db.getConnection()->createStatement();

sql::ResultSet\* res = stmt->executeQuery("SELECT \* FROM Exhibition");

cout << "\nExhibitions:\n";

while (res->next()) {

cout << "ID: " << res->getInt("ID")

<< ", Name: " << res->getString("Name")

<< ", Start: " << res->getString("Start\_Date")

<< ", End: " << res->getString("End\_Date")

<< ", Cost: " << res->getDouble("Cost")

<< ", Museum ID: " << res->getInt("Museum\_ID") << endl;

}

delete res;

delete stmt;

}

catch (sql::SQLException& e) {

cout << "SQL Error: " << e.what() << endl;

}

}

void AdminExhibition::AddExhibition(Database& db) {

string name, start, end;

double cost;

int museum\_id;

cout << "\n=== Add New Exhibition ===\n";

cin.ignore();

cout << "Enter Name: ";

getline(cin, name);

cout << "Enter Start Date (YYYY-MM-DD): ";

getline(cin, start);

cout << "Enter End Date (YYYY-MM-DD): ";

getline(cin, end);

cout << "Enter Cost: ";

cin >> cost;

cout << "Enter Museum ID: ";

cin >> museum\_id;

try {

sql::PreparedStatement\* pstmt = db.getConnection()->prepareStatement(

"INSERT INTO Exhibition (Name, Start\_Date, End\_Date, Cost, Museum\_ID) VALUES (?, ?, ?, ?, ?)"

);

pstmt->setString(1, name);

pstmt->setString(2, start);

pstmt->setString(3, end);

pstmt->setDouble(4, cost);

pstmt->setInt(5, museum\_id);

pstmt->executeUpdate();

delete pstmt;

cout << "Exhibition added successfully!\n";

}

catch (sql::SQLException& e) {

cout << "SQL Error: " << e.what() << "\n";

}

}

void AdminExhibition::DeleteExhibition(Database& db) {

int id;

cout << "Enter Exhibition ID to delete: ";

cin >> id;

try {

sql::PreparedStatement\* pstmt = db.getConnection()->prepareStatement(

"DELETE FROM Exhibition WHERE ID = ?"

);

pstmt->setInt(1, id);

int affected = pstmt->executeUpdate();

delete pstmt;

if (affected > 0)

cout << "Exhibition deleted successfully.\n";

else

cout << "No exhibition found with that ID.\n";

}

catch (sql::SQLException& e) {

cout << "SQL Error: " << e.what() << "\n";

}

}

**│ │ ├──AdminExhibitionAuthor.h**

#pragma once

#ifndef ADMIN\_EXHIBITION\_AUTHOR\_H

#define ADMIN\_EXHIBITION\_AUTHOR\_H

#include "Database.h"

namespace AdminExhibitionAuthor

{

void ExhibitionAuthorMenu(Database& db);

void ViewExhibitionAuthor(Database& db);

void AddExhibitionAuthor(Database& db);

void DeleteExhibitionAuthor(Database& db);

}

#endif

**│ │ ├──AdminExhibitionAuthor.cpp**

#include "AdminExhibitionAuthor.h"

#include <string>

#include <iostream>

#include <cppconn/prepared\_statement.h>

using namespace std;

void AdminExhibitionAuthor::ExhibitionAuthorMenu(Database& db) {

int choice;

cout << "\n=== Admin Menu ExhibitionAuthor ===\n";

cout << "1. View ExhibitionAuthor\n";

cout << "2. Add ExhibitionAuthor\n";

cout << "3. Delete ExhibitionAuthor\n";

cout << "Select an option: ";

cin >> choice;

switch (choice) {

case 1:

ViewExhibitionAuthor(db);

break;

case 2:

AddExhibitionAuthor(db);

break;

case 3:

DeleteExhibitionAuthor(db);

break;

default:

cout << "Invalid choice.\n";

}

}

void AdminExhibitionAuthor::ViewExhibitionAuthor(Database& db) {

try {

sql::Statement\* stmt = db.getConnection()->createStatement();

sql::ResultSet\* res = stmt->executeQuery("SELECT \* FROM Exhibition\_Author");

cout << "\nExhibitionAuthor Records:\n";

while (res->next()) {

cout << "Exhibition ID: " << res->getInt("Exhibition\_ID")

<< ", Author ID: " << res->getInt("Author\_ID")

<< ", Share: " << res->getInt("Share") << endl;

}

delete res;

delete stmt;

}

catch (sql::SQLException& e) {

cout << "SQL Error: " << e.what() << endl;

}

}

void AdminExhibitionAuthor::AddExhibitionAuthor(Database& db) {

int exhibition\_id, author\_id, share;

cout << "\n=== Add New ExhibitionAuthor ===\n";

cout << "Enter Exhibition ID: ";

cin >> exhibition\_id;

cout << "Enter Author ID: ";

cin >> author\_id;

cout << "Enter Share Percentage: ";

cin >> share;

try {

sql::PreparedStatement\* pstmt = db.getConnection()->prepareStatement(

"INSERT INTO Exhibition\_Author (Exhibition\_ID, Author\_ID, Share) VALUES (?, ?, ?)"

);

pstmt->setInt(1, exhibition\_id);

pstmt->setInt(2, author\_id);

pstmt->setInt(3, share);

pstmt->executeUpdate();

delete pstmt;

cout << "ExhibitionAuthor entry added successfully!\n";

}

catch (sql::SQLException& e) {

cout << "SQL Error: " << e.what() << endl;

}

}

void AdminExhibitionAuthor::DeleteExhibitionAuthor(Database& db) {

int exhibition\_id, author\_id;

cout << "Enter Exhibition ID: ";

cin >> exhibition\_id;

cout << "Enter Author ID: ";

cin >> author\_id;

try {

sql::PreparedStatement\* pstmt = db.getConnection()->prepareStatement(

"DELETE FROM Exhibition\_Author WHERE Exhibition\_ID = ? AND Author\_ID = ?"

);

pstmt->setInt(1, exhibition\_id);

pstmt->setInt(2, author\_id);

int affected = pstmt->executeUpdate();

if (affected > 0)

cout << "ExhibitionAuthor entry deleted successfully.\n";

else

cout << "No matching entry found.\n";

delete pstmt;

}

catch (sql::SQLException& e) {

cout << "SQL Error: " << e.what() << endl;

}

}

**│ │ ├──AdminGuide.h**

#pragma once

#ifndef ADMIN\_GUIDE\_H

#define ADMIN\_GUIDE\_H

#include "Database.h"

namespace AdminGuide

{

void GuideMenu(Database& db);

void ViewGuide(Database& db);

void AddGuide(Database& db);

void DeleteGuide(Database& db);

}

#endif

**│ │ ├──AdminGuide.cpp**

#include "AdminGuide.h"

#include <string>

#include <iostream>

#include <cppconn/prepared\_statement.h>

using namespace std;

void AdminGuide::GuideMenu(Database& db) {

int choice;

cout << "\n=== Admin Menu Guides ===\n";

cout << "1. View Guides\n";

cout << "2. Add Guide\n";

cout << "3. Delete Guide\n";

cout << "Select an option: ";

cin >> choice;

switch (choice) {

case 1:

ViewGuide(db);

break;

case 2:

AddGuide(db);

break;

case 3:

DeleteGuide(db);

break;

default:

cout << "Invalid choice.\n";

}

}

void AdminGuide::ViewGuide(Database& db) {

try {

sql::Statement\* stmt = db.getConnection()->createStatement();

sql::ResultSet\* res = stmt->executeQuery("SELECT \* FROM Guide");

cout << "\nGuides:\n";

while (res->next()) {

cout << "ID: " << res->getInt("ID")

<< ", Name: " << res->getString("Name")

<< ", Contact Info: " << res->getString("Contact\_Info")

<< ", Experience in Years: " << res->getInt("Years\_Experience") << " years" << endl;

cout << ", Museum ID: " << res->getInt("Museum\_ID") << endl;

}

delete res;

delete stmt;

}

catch (sql::SQLException& e) {

cout << "SQL Error: " << e.what() << endl;

}

}

void AdminGuide::AddGuide(Database& db) {

string name, contactInfo;

int experience, museumId;

cout << "\n=== Add New Guide ===\n";

cout << "Enter Name: ";

cin.ignore();

getline(cin, name);

cout << "Enter Contact Info: ";

getline(cin, contactInfo);

cout << "Enter Experience in Years: ";

cin >> experience;

cout << "Museum Id: ";

cin >> museumId;

try {

sql::PreparedStatement\* pstmt = db.getConnection()->prepareStatement(

"INSERT INTO guide (Name, Contact\_Info, Years\_Experience, Museum\_ID) VALUES (?, ?, ?, ?)"

);

pstmt->setString(1, name);

pstmt->setString(2, contactInfo);

pstmt->setInt(3, experience);

pstmt->setInt(4, museumId);

pstmt->executeUpdate();

delete pstmt;

cout << "Guide added successfully!\n";

}

catch (sql::SQLException& e) {

cerr << "SQL Error: " << e.what() << endl;

}

}

void AdminGuide::DeleteGuide(Database& db) {

int id;

cout << "Enter Guide ID to delete: ";

cin >> id;

try {

sql::PreparedStatement\* pstmt = db.getConnection()->prepareStatement(

"DELETE FROM Guide WHERE ID = ?"

);

pstmt->setInt(1, id);

int affected = pstmt->executeUpdate();

if (affected > 0)

cout << "Guide deleted successfully.\n";

else

cout << "No guide found with that ID.\n";

delete pstmt;

}

catch (sql::SQLException& e) {

cout << "SQL Error: " << e.what() << endl;

}

}

**│ │ ├──AdminHall.h**

#pragma once

#ifndef ADMIN\_HALL\_H

#define ADMIN\_HALL\_H

#include "Database.h"

namespace AdminHall

{

void HallMenu(Database& db);

void ViewHall(Database& db);

void AddHall(Database& db);

void DeleteHall(Database& db);

}

#endif

**│ │ ├──AdminHall.cpp**

#include "AdminHall.h"

#include <string>

#include <iostream>

#include <cppconn/prepared\_statement.h>

using namespace std;

void AdminHall::HallMenu(Database& db) {

int choice;

cout << "\n=== Admin Menu Halls ===\n";

cout << "1. View Halls\n";

cout << "2. Add Hall\n";

cout << "3. Delete Hall\n";

cout << "Select an option: ";

cin >> choice;

switch (choice) {

case 1:

ViewHall(db);

break;

case 2:

AddHall(db);

break;

case 3:

DeleteHall(db);

break;

default:

cout << "Invalid choice.\n";

}

}

void AdminHall::ViewHall(Database& db) {

try {

sql::Statement\* stmt = db.getConnection()->createStatement();

sql::ResultSet\* res = stmt->executeQuery("SELECT \* FROM Hall");

cout << "\nHalls:\n";

while (res->next()) {

cout << "ID: " << res->getInt("ID")

<< ", Name: " << res->getString("Hall\_Name")

<< ", Capacity: " << res->getInt("Capacity")

<< ", Museum ID: " << res->getInt("Museum\_ID") << endl;

}

delete res;

delete stmt;

}

catch (sql::SQLException& e) {

cout << "SQL Error: " << e.what() << endl;

}

}

void AdminHall::AddHall(Database& db) {

string name;

int museum\_id, capacity;

cout << "\n=== Add New Hall ===\n";

cout << "Enter Hall Name: ";

cin.ignore();

getline(cin, name);

cout << "Enter Hall Capacity: ";

cin >> capacity;

cout << "Enter Museum ID: ";

cin >> museum\_id;

try {

sql::PreparedStatement\* pstmt = db.getConnection()->prepareStatement(

"INSERT INTO hall (Hall\_Name, Capacity, Museum\_ID) VALUES (?, ?, ?)"

);

pstmt->setString(1, name);

pstmt->setInt(2, capacity);

pstmt->setInt(3, museum\_id);

pstmt->executeUpdate();

delete pstmt;

cout << "Hall added successfully!\n";

}

catch (sql::SQLException& e) {

cerr << "SQL Error: " << e.what() << endl;

}

}

void AdminHall::DeleteHall(Database& db) {

int id;

cout << "Enter Hall ID to delete: ";

cin >> id;

try {

sql::PreparedStatement\* pstmt = db.getConnection()->prepareStatement(

"DELETE FROM Hall WHERE ID = ?"

);

pstmt->setInt(1, id);

int affected = pstmt->executeUpdate();

if (affected > 0)

cout << "Hall deleted successfully.\n";

else

cout << "No hall found with that ID.\n";

delete pstmt;

}

catch (sql::SQLException& e) {

cout << "SQL Error: " << e.what() << endl;

}

}

**│ │ ├──AdminMuseum.h**

#pragma once

#ifndef ADMIN\_MUSEUM\_H

#define ADMIN\_MUSEUM\_H

#include "Database.h"

namespace AdminMuseum

{

void MuseumMenu(Database& db);

void ViewMuseum(Database& db);

void AddMuseum(Database& db);

void DeleteMuseum(Database& db);

}

#endif

**│ │ ├──AdminMuseum.cpp**

#include "AdminMuseum.h"

#include <string>

#include <iostream>

#include <cppconn/prepared\_statement.h>

using namespace std;

void AdminMuseum::MuseumMenu(Database& db) {

int choice;

cout << "\n=== Admin Menu Museums ===\n";

cout << "1. View Museums\n";

cout << "2. Add Museum\n";

cout << "3. Delete Museum\n";

cout << "Select an option: ";

cin >> choice;

switch (choice) {

case 1:

ViewMuseum(db);

break;

case 2:

AddMuseum(db);

break;

case 3:

DeleteMuseum(db);

break;

default:

cout << "Invalid choice.\n";

}

}

void AdminMuseum::ViewMuseum(Database& db) {

try {

sql::Statement\* stmt = db.getConnection()->createStatement();

sql::ResultSet\* res = stmt->executeQuery("SELECT \* FROM museum");

cout << "\nMuseums:\n";

while (res->next()) {

cout << "ID: " << res->getInt("ID")

<< ", Name: " << res->getString("Name")

<< ", Address: " << res->getString("Address")

<< ", Contact Info: " << res->getString("Contact\_Info") << endl;

}

delete res;

delete stmt;

}

catch (sql::SQLException& e) {

cout << "SQL Error: " << e.what() << endl;

}

}

void AdminMuseum::AddMuseum(Database& db) {

string name, address, contactInfo;

cout << "\n=== Add New Museum ===\n";

cout << "Enter Museum Name: ";

cin.ignore();

getline(cin, name);

cout << "Enter Museum Address: ";

getline(cin, address);

cout << "Enter Contact Info: ";

cin >> contactInfo;

try {

sql::PreparedStatement\* pstmt = db.getConnection()->prepareStatement(

"INSERT INTO museum (Name, Address, Contact\_Info) VALUES (?, ?, ?)"

);

pstmt->setString(1, name);

pstmt->setString(2, address);

pstmt->setString(3, contactInfo);

pstmt->executeUpdate();

delete pstmt;

cout << "Museum added successfully!\n";

}

catch (sql::SQLException& e) {

cerr << "SQL Error: " << e.what() << endl;

}

}

void AdminMuseum::DeleteMuseum(Database& db) {

int id;

cout << "Enter Museum ID to delete: ";

cin >> id;

try {

sql::PreparedStatement\* pstmt = db.getConnection()->prepareStatement(

"DELETE FROM Museum WHERE ID = ?"

);

pstmt->setInt(1, id);

int affected = pstmt->executeUpdate();

if (affected > 0)

cout << "Museum deleted successfully.\n";

else

cout << "No museum found with that ID.\n";

delete pstmt;

}

catch (sql::SQLException& e) {

cout << "SQL Error: " << e.what() << endl;

}

}

**│ │ ├──AdminSeller.h**

#pragma once

#ifndef ADMIN\_SELLER\_H

#define ADMIN\_SELLER\_H

#include "Database.h"

namespace AdminSeller

{

void SellerMenu(Database& db);

void ViewSeller(Database& db);

void AddSeller(Database& db);

void DeleteSeller(Database& db);

}

#endif

**│ │ ├──AdminSeller.cpp**

#include "AdminSeller.h"

#include <string>

#include <iostream>

#include <cppconn/prepared\_statement.h>

using namespace std;

void AdminSeller::SellerMenu(Database& db) {

int choice;

cout << "\n=== Admin Menu Sellers ===\n";

cout << "1. View Sellers\n";

cout << "2. Add Seller\n";

cout << "3. Delete Seller\n";

cout << "Select an option: ";

cin >> choice;

switch (choice) {

case 1:

ViewSeller(db);

break;

case 2:

AddSeller(db);

break;

case 3:

DeleteSeller(db);

break;

default:

cout << "Invalid choice.\n";

}

}

void AdminSeller::ViewSeller(Database& db) {

try {

sql::Statement\* stmt = db.getConnection()->createStatement();

sql::ResultSet\* res = stmt->executeQuery("SELECT \* FROM Seller");

cout << "\nSellers:\n";

while (res->next()) {

cout << "ID: " << res->getInt("ID")

<< ", Name: " << res->getString("Full\_Name")

<< ", Email: " << res->getString("Contact\_Info") << endl;

}

delete res;

delete stmt;

}

catch (sql::SQLException& e) {

cout << "SQL Error: " << e.what() << endl;

}

}

void AdminSeller::AddSeller(Database& db) {

string name, email;

cout << "\n=== Add New Seller ===\n";

cout << "Enter Seller Name: ";

cin.ignore();

getline(cin, name);

cout << "Enter Seller Email: ";

getline(cin, email);

try {

sql::PreparedStatement\* pstmt = db.getConnection()->prepareStatement(

"INSERT INTO Seller (Full\_Name, Contact\_Info) VALUES (?, ?)"

);

pstmt->setString(1, name);

pstmt->setString(2, email);

pstmt->executeUpdate();

delete pstmt;

cout << "Seller added successfully!\n";

}

catch (sql::SQLException& e) {

cerr << "SQL Error: " << e.what() << endl;

}

}

void AdminSeller::DeleteSeller(Database& db) {

int id;

cout << "Enter Seller ID to delete: ";

cin >> id;

try {

sql::PreparedStatement\* pstmt = db.getConnection()->prepareStatement(

"DELETE FROM Seller WHERE ID = ?"

);

pstmt->setInt(1, id);

int affected = pstmt->executeUpdate();

if (affected > 0)

cout << "Seller deleted successfully.\n";

else

cout << "No seller found with that ID.\n";

delete pstmt;

}

catch (sql::SQLException& e) {

cout << "SQL Error: " << e.what() << endl;

}

}

**│ │ ├──AdminSponsor.h**

#pragma once

#ifndef ADMIN\_SPONSOR\_H

#define ADMIN\_SPONSOR\_H

#include "Database.h"

namespace AdminSponsor

{

void SponsorMenu(Database& db);

void ViewSponsor(Database& db);

void AddSponsor(Database& db);

void DeleteSponsor(Database& db);

}

#endif

**│ │ ├──AdminSponsor.cpp**

#include "AdminSponsor.h"

#include <string>

#include <iostream>

#include <cppconn/prepared\_statement.h>

using namespace std;

void AdminSponsor::SponsorMenu(Database& db) {

int choice;

cout << "\n=== Admin Menu Sponsors ===\n";

cout << "1. View Sponsors\n";

cout << "2. Add Sponsor\n";

cout << "3. Delete Sponsor\n";

cout << "Select an option: ";

cin >> choice;

switch (choice) {

case 1:

ViewSponsor(db);

break;

case 2:

AddSponsor(db);

break;

case 3:

DeleteSponsor(db);

break;

default:

cout << "Invalid choice.\n";

}

}

void AdminSponsor::ViewSponsor(Database& db) {

try {

sql::Statement\* stmt = db.getConnection()->createStatement();

sql::ResultSet\* res = stmt->executeQuery("SELECT \* FROM Sponsor");

cout << "\nSponsors:\n";

while (res->next()) {

cout << "ID: " << res->getInt("ID")

<< ", Name: " << res->getString("Sponsor\_Name")

<< ", Contact Info: " << res->getString("Contact\_Info")

<< ", Amount: " << res->getDouble("Contribution\_Amount")

<< ", Contribution Date: " << res->getString("Contribution\_Date")

<< ", Museum ID: " << res->getString("Museum\_ID") << endl;

}

delete res;

delete stmt;

}

catch (sql::SQLException& e) {

cout << "SQL Error: " << e.what() << endl;

}

}

void AdminSponsor::AddSponsor(Database& db) {

string name, contactInfo, contributionDate;

double amount;

int museumId;

cout << "\n=== Add New Sponsor ===\n";

cout << "Enter Sponsor Name: ";

cin.ignore();

getline(cin, name);

cout << "Enter Contact Info: ";

getline(cin, contactInfo);

cout << "Enter Sponsorship Amount: ";

cin >> amount;

cin.ignore();

cout << "Enter Contributuion Date (YYYY-MM-DD): ";

getline(cin, contributionDate);

cout << "Enter Museum ID: ";

cin >> museumId;

try {

sql::PreparedStatement\* pstmt = db.getConnection()->prepareStatement(

"INSERT INTO Sponsor (Sponsor\_Name, Contact\_Info, Contribution\_Amount, Contribution\_Date, Museum\_ID) VALUES (?, ?, ?, ?, ?)"

);

pstmt->setString(1, name);

pstmt->setString(2, contactInfo);

pstmt->setDouble(3, amount);

pstmt->setString(4, contributionDate);

pstmt->setInt(5, museumId);

pstmt->executeUpdate();

delete pstmt;

cout << "Sponsor added successfully!\n";

}

catch (sql::SQLException& e) {

cerr << "SQL Error: " << e.what() << endl;

}

}

void AdminSponsor::DeleteSponsor(Database& db) {

int id;

cout << "Enter Sponsor ID to delete: ";

cin >> id;

try {

sql::PreparedStatement\* pstmt = db.getConnection()->prepareStatement(

"DELETE FROM Sponsor WHERE ID = ?"

);

pstmt->setInt(1, id);

int affected = pstmt->executeUpdate();

if (affected > 0)

cout << "Sponsor deleted successfully.\n";

else

cout << "No sponsor found with that ID.\n";

delete pstmt;

}

catch (sql::SQLException& e) {

cout << "SQL Error: " << e.what() << endl;

}

}

**│ │ ├──AdminTicket.h**

#pragma once

#ifndef ADMIN\_TICKET\_H

#define ADMIN\_TICKET\_H

#include "Database.h"

namespace AdminTicket

{

void TicketMenu(Database& db);

void ViewTicket(Database& db);

void AddTicket(Database& db);

void DeleteTicket(Database& db);

}

#endif

**│ │ ├──AdminTicket.cpp**

#include "AdminTicket.h"

#include <string>

#include <iostream>

#include <cppconn/prepared\_statement.h>

using namespace std;

void AdminTicket::TicketMenu(Database& db) {

int choice;

cout << "\n=== Admin Menu Tickets ===\n";

cout << "1. View Tickets\n";

cout << "2. Add Ticket\n";

cout << "3. Delete Ticket\n";

cout << "Select an option: ";

cin >> choice;

switch (choice) {

case 1:

ViewTicket(db);

break;

case 2:

AddTicket(db);

break;

case 3:

DeleteTicket(db);

break;

default:

cout << "Invalid choice.\n";

}

}

void AdminTicket::ViewTicket(Database& db) {

try {

sql::Statement\* stmt = db.getConnection()->createStatement();

sql::ResultSet\* res = stmt->executeQuery("SELECT \* FROM ticket");

cout << "\nTickets:\n";

while (res->next()) {

cout << "ID: " << res->getInt("ID")

<< ", Type: " << res->getString("Ticket\_Type")

<< ", Price: " << res->getDouble("Price")

<< ", Visitor Category: " << res->getString("Visitor\_Category")

<< ", Tour ID: " << res->getInt("Tour\_ID")

<< ", Museum ID: " << res->getInt("Museum\_ID")

<< ", Sale Start Date: " << res->getString("Sale\_Date") << endl;

}

delete res;

delete stmt;

}

catch (sql::SQLException& e) {

cout << "SQL Error: " << e.what() << endl;

}

}

void AdminTicket::AddTicket(Database& db) {

string type, category, saleDate;

double price;

int tour\_id, museum\_id;

cout << "\n=== Add New Ticket ===\n";

cout << "Enter Ticket Type: ";

cin.ignore();

getline(cin, type);

cout << "Enter Visitor Category: ";

getline(cin, category);

cout << "Enter Price: ";

cin >> price;

cout << "Enter Tour ID (or 0 if none): ";

cin >> tour\_id;

cout << "Enter Museum ID: ";

cin >> museum\_id;

cin.ignore();

cout << "Enter Start Sale Date (or 0 if none): ";

getline(cin, saleDate);

try {

sql::PreparedStatement\* pstmt = db.getConnection()->prepareStatement(

"INSERT INTO ticket (Ticket\_Type, Price, Visitor\_Category, Tour\_ID, Museum\_ID, Sale\_Date) VALUES (?, ?, ?, ?, ?, ?)"

);

pstmt->setString(1, type);

pstmt->setDouble(2, price);

pstmt->setString(3, category);

if (tour\_id == 0)

pstmt->setNull(4, sql::DataType::INTEGER);

else

pstmt->setInt(4, tour\_id);

pstmt->setInt(5, museum\_id);

if (saleDate == "0")

pstmt->setNull(6, sql::DataType::DATE);

else

pstmt->setString(6, saleDate);

pstmt->executeUpdate();

delete pstmt;

cout << "Ticket added successfully!\n";

}

catch (sql::SQLException& e) {

cerr << "SQL Error: " << e.what() << endl;

}

}

void AdminTicket::DeleteTicket(Database& db) {

int id;

cout << "Enter Ticket ID to delete: ";

cin >> id;

try {

sql::PreparedStatement\* pstmt = db.getConnection()->prepareStatement(

"DELETE FROM Ticket WHERE ID = ?"

);

pstmt->setInt(1, id);

int affected = pstmt->executeUpdate();

if (affected > 0)

cout << "Ticket deleted successfully.\n";

else

cout << "No ticket found with that ID.\n";

delete pstmt;

}

catch (sql::SQLException& e) {

cout << "SQL Error: " << e.what() << endl;

}

}

**│ │ ├──AdminTicketSeller.h**

#pragma once

#ifndef ADMIN\_TICKET\_SELLER\_H

#define ADMIN\_TICKET\_SELLER\_H

#include "Database.h"

namespace AdminTicketSeller

{

void TicketSellerMenu(Database& db);

void ViewTicketSeller(Database& db);

void AddTicketSeller(Database& db);

void DeleteTicketSeller(Database& db);

}

#endif

**│ │ ├──AdminTicketSeller.cpp**

#include "AdminTicketSeller.h"

#include <string>

#include <iostream>

#include <cppconn/prepared\_statement.h>

using namespace std;

void AdminTicketSeller::TicketSellerMenu(Database& db) {

int choice;

cout << "\n=== Admin Menu Ticket Sellers ===\n";

cout << "1. View Ticket Sellers\n";

cout << "2. Add Ticket Seller\n";

cout << "3. Delete Ticket Seller\n";

cout << "Select an option: ";

cin >> choice;

switch (choice) {

case 1:

ViewTicketSeller(db);

break;

case 2:

AddTicketSeller(db);

break;

case 3:

DeleteTicketSeller(db);

break;

default:

cout << "Invalid choice.\n";

}

}

void AdminTicketSeller::ViewTicketSeller(Database& db) {

try {

sql::Statement\* stmt = db.getConnection()->createStatement();

sql::ResultSet\* res = stmt->executeQuery("SELECT \* FROM Ticket\_Sale");

cout << "\nTicket Sellers:\n";

while (res->next()) {

cout << "ID: " << res->getInt("ID")

<< ", Seller Id: " << res->getString("Seller\_ID")

<< ", Ticket Id: " << res->getString("Ticket\_ID") << endl;

}

delete res;

delete stmt;

}

catch (sql::SQLException& e) {

cout << "SQL Error: " << e.what() << endl;

}

}

void AdminTicketSeller::AddTicketSeller(Database& db) {

int sellerId, ticketId;

cout << "\n=== Add New Ticket Seller ===\n";

cout << "Enter Seller Id: ";

cin.ignore();

cin >> sellerId;

cout << "Enter Ticket Id: ";

cin >> ticketId;

try {

sql::PreparedStatement\* pstmt = db.getConnection()->prepareStatement(

"INSERT INTO Ticket\_Sale (Seller\_ID, Ticket\_ID) VALUES (?, ?)"

);

pstmt->setInt(1, sellerId);

pstmt->setInt(2, ticketId);

pstmt->executeUpdate();

delete pstmt;

cout << "Ticket Seller added successfully!\n";

}

catch (sql::SQLException& e) {

cerr << "SQL Error: " << e.what() << endl;

}

}

void AdminTicketSeller::DeleteTicketSeller(Database& db) {

int id;

cout << "Enter Ticket Seller ID to delete: ";

cin >> id;

try {

sql::PreparedStatement\* pstmt = db.getConnection()->prepareStatement(

"DELETE FROM Ticket\_Sale WHERE ID = ?"

);

pstmt->setInt(1, id);

int affected = pstmt->executeUpdate();

if (affected > 0)

cout << "Ticket Seller deleted successfully.\n";

else

cout << "No ticket seller found with that ID.\n";

delete pstmt;

}

catch (sql::SQLException& e) {

cout << "SQL Error: " << e.what() << endl;

}

}

**│ │ ├──AdminTour.h**

#pragma once

#ifndef ADMIN\_TOUR\_H

#define ADMIN\_TOUR\_H

#include "Database.h"

namespace AdminTour

{

void TourMenu(Database& db);

void ViewTour(Database& db);

void AddTour(Database& db);

void DeleteTour(Database& db);

}

#endif

**│ │ ├──AdminTour.cpp**

#include "AdminTour.h"

#include <string>

#include <iostream>

#include <cppconn/prepared\_statement.h>

using namespace std;

void AdminTour::TourMenu(Database& db) {

int choice;

cout << "\n=== Admin Menu Tour ===\n";

cout << "1. View Tours\n";

cout << "2. Add Tour\n";

cout << "3. Delete Tour\n";

cout << "Select an option: ";

cin >> choice;

switch (choice) {

case 1:

ViewTour(db);

break;

case 2:

AddTour(db);

break;

case 3:

DeleteTour(db);

break;

default:

cout << "Invalid choice.\n";

}

}

void AdminTour::ViewTour(Database& db) {

try {

sql::Statement\* stmt = db.getConnection()->createStatement();

sql::ResultSet\* res = stmt->executeQuery("SELECT \* FROM Tour");

cout << "\nTours:\n";

while (res->next()) {

cout << "ID: " << res->getInt("ID")

<< ", Name: " << res->getString("Name")

<< ", Guide id: " << res->getInt("Guide\_ID")

<< ", Date: " << res->getString("Date")

<< ", Museum id: " << res->getInt("Museum\_ID") << endl;

}

delete res;

delete stmt;

}

catch (sql::SQLException& e) {

cout << "SQL Error: " << e.what() << endl;

}

}

void AdminTour::AddTour(Database& db)

{

std::string name, date;

int guideId, museumId;

std::cout << "\n=== Add New Tour ===\n";

std::cout << "Enter Name: ";

std::cin.ignore(); // clear buffer

std::getline(std::cin, name);

std::cout << "Enter Date (YYYY-MM-DD): ";

std::getline(std::cin, date);

std::cout << "Enter Guide ID: ";

std::cin >> guideId;

std::cout << "Enter Museum ID: ";

std::cin >> museumId;

try {

sql::PreparedStatement\* pstmt = db.getConnection()->prepareStatement(

"INSERT INTO Tour (Name, Date, Guide\_ID, Museum\_ID) VALUES (?, ?, ?, ?)"

);

pstmt->setString(1, name);

pstmt->setString(2, date);

pstmt->setInt(3, guideId);

pstmt->setInt(4, museumId);

pstmt->executeUpdate();

delete pstmt;

std::cout << "User added successfully!\n";

}

catch (sql::SQLException& e) {

std::cerr << "SQL Error: " << e.what() << "\n";

}

}

void AdminTour::DeleteTour(Database& db) {

int id;

cout << "Enter Tour ID to delete: ";

cin >> id;

try {

sql::PreparedStatement\* pstmt = db.getConnection()->prepareStatement(

"DELETE FROM Tour WHERE id = ?"

);

pstmt->setInt(1, id);

int affected = pstmt->executeUpdate();

if (affected > 0)

cout << "Tour deleted successfully.\n";

else

cout << "No user found with that ID.\n";

delete pstmt;

}

catch (sql::SQLException& e) {

cout << "SQL Error: " << e.what() << endl;

}

}

**│ │ ├──AdminUser.h**

#pragma once

#ifndef ADMIN\_USER\_H

#define ADMIN\_USER\_H

#include "Database.h"

namespace AdminUser

{

void UserMenu(Database& db);

void ViewUser(Database& db);

void AddUser(Database& db);

void DeleteUser(Database& db);

}

#endif

**│ │ ├──AdminUser.cpp**

#include "AdminUser.h"

#include <string>

#include <iostream>

#include <cppconn/prepared\_statement.h>

using namespace std;

void AdminUser::UserMenu(Database& db) {

int choice;

cout << "\n=== Admin Menu Users ===\n";

cout << "1. View Users\n";

cout << "2. Add User\n";

cout << "3. Delete User\n";

cout << "Select an option: ";

cin >> choice;

switch (choice) {

case 1:

ViewUser(db);

break;

case 2:

AddUser(db);

break;

case 3:

DeleteUser(db);

break;

default:

cout << "Invalid choice.\n";

}

}

void AdminUser::ViewUser(Database& db) {

try {

sql::Statement\* stmt = db.getConnection()->createStatement();

sql::ResultSet\* res = stmt->executeQuery("SELECT \* FROM User");

cout << "\nUsers:\n";

while (res->next()) {

cout << "ID: " << res->getInt("ID")

<< ", Username: " << res->getString("Username")

<< ", Password: " << res->getInt("Password")

<< ", Role: " << res->getString("Role") << endl;

}

delete res;

delete stmt;

}

catch (sql::SQLException& e) {

cout << "SQL Error: " << e.what() << endl;

}

}

void AdminUser::AddUser(Database& db)

{

std::string username, role, password;

std::cout << "\n=== Add New User ===\n";

std::cout << "Enter Username: ";

std::cin.ignore(); // clear buffer

std::getline(std::cin, username);

std::cout << "Enter Role: ";

std::getline(std::cin, role);

std::cout << "Enter password: ";

std::getline(std::cin, password);

try {

sql::PreparedStatement\* pstmt = db.getConnection()->prepareStatement(

"INSERT INTO User (Username, Role, Password) VALUES (?, ?, ?)"

);

pstmt->setString(1, username);

pstmt->setString(2, role);

pstmt->setString(3, password);

pstmt->executeUpdate();

delete pstmt;

std::cout << "User added successfully!\n";

}

catch (sql::SQLException& e) {

std::cerr << "SQL Error: " << e.what() << "\n";

}

}

void AdminUser::DeleteUser(Database& db) {

int id;

cout << "Enter User ID to delete: ";

cin >> id;

try {

sql::PreparedStatement\* pstmt = db.getConnection()->prepareStatement(

"DELETE FROM User WHERE id = ?"

);

pstmt->setInt(1, id);

int affected = pstmt->executeUpdate();

if (affected > 0)

cout << "User deleted successfully.\n";

else

cout << "No user found with that ID.\n";

delete pstmt;

}

catch (sql::SQLException& e) {

cout << "SQL Error: " << e.what() << endl;

}

}

**│ │ └──AdminVisitor.h**

#pragma once

#ifndef ADMIN\_VISITOR\_H

#define ADMIN\_VISITOR\_H

#include "Database.h"

namespace AdminVisitor

{

void VisitorMenu(Database& db);

void ViewVisitor(Database& db);

void AddVisitor(Database& db);

void DeleteVisitor(Database& db);

}

#endif

**│ │ └──AdminVisitor.cpp**

#include "AdminVisitor.h"

#include <string>

#include <iostream>

#include <cppconn/prepared\_statement.h>

using namespace std;

void AdminVisitor::VisitorMenu(Database& db) {

int choice;

cout << "\n=== Admin Menu Visitors ===\n";

cout << "1. View Visitors\n";

cout << "2. Add Visitor\n";

cout << "3. Delete Visitor\n";

cout << "Select an option: ";

cin >> choice;

switch (choice) {

case 1:

ViewVisitor(db);

break;

case 2:

AddVisitor(db);

break;

case 3:

DeleteVisitor(db);

break;

default:

cout << "Invalid choice.\n";

}

}

void AdminVisitor::ViewVisitor(Database& db) {

try {

sql::Statement\* stmt = db.getConnection()->createStatement();

sql::ResultSet\* res = stmt->executeQuery("SELECT \* FROM visitor");

cout << "\nVisitors:\n";

while (res->next()) {

cout << "ID: " << res->getInt("ID")

<< ", Name: " << res->getString("Name")

<< ", Ticket ID: " << res->getInt("Ticket\_ID")

<< ", Email: " << res->getString("Email") << endl;

}

delete res;

delete stmt;

}

catch (sql::SQLException& e) {

cout << "SQL Error: " << e.what() << endl;

}

}

void AdminVisitor::AddVisitor(Database& db)

{

std::string name, email;

int ticket\_id;

std::cout << "\n=== Add New Visitor ===\n";

std::cout << "Enter Name: ";

std::cin.ignore(); // clear buffer

std::getline(std::cin, name);

std::cout << "Enter Email: ";

std::getline(std::cin, email);

std::cout << "Enter Ticket ID: ";

std::cin >> ticket\_id;

try {

sql::PreparedStatement\* pstmt = db.getConnection()->prepareStatement(

"INSERT INTO Visitor (Name, Email, Ticket\_ID) VALUES (?, ?, ?)"

);

pstmt->setString(1, name);

pstmt->setString(2, email);

pstmt->setInt(3, ticket\_id);

pstmt->executeUpdate();

delete pstmt;

std::cout << "Visitor added successfully!\n";

}

catch (sql::SQLException& e) {

std::cerr << "SQL Error: " << e.what() << "\n";

}

}

void AdminVisitor::DeleteVisitor(Database& db) {

int id;

cout << "Enter Visitor ID to delete: ";

cin >> id;

try {

sql::PreparedStatement\* pstmt = db.getConnection()->prepareStatement(

"DELETE FROM visitor WHERE id = ?"

);

pstmt->setInt(1, id);

int affected = pstmt->executeUpdate();

if (affected > 0)

cout << "Visitor deleted successfully.\n";

else

cout << "No visitor found with that ID.\n";

delete pstmt;

}

catch (sql::SQLException& e) {

cout << "SQL Error: " << e.what() << endl;

}

}

**├── Models**

**│ ├──Auth.cpp**

#include "Admin.h"

#include "Auth.h"

#include "Manager.h"

#include <sstream>

#include <iomanip>

//#include <openssl/sha.h>

#include <cppconn/prepared\_statement.h>

#include <cppconn/resultset.h>

#include <iostream>

using namespace std;

bool Auth::login(Database& db, string& role) {

string username, password;

cout << "Enter username: ";

cin >> username;

cout << "Enter password: ";

cin >> password;

sql::Connection\* conn = db.getConnection();

sql::PreparedStatement\* pstmt;

sql::ResultSet\* res;

try {

pstmt = conn->prepareStatement(

"SELECT role FROM User WHERE username = ? AND password = ?"

);

pstmt->setString(1, username);

pstmt->setString(2, password);

res = pstmt->executeQuery();

/\* role.erase(0, role.find\_first\_not\_of(" \t\n\r\f\v"));

role.erase(role.find\_last\_not\_of(" \t\n\r\f\v") + 1);\*/

if (res->next()) {

role = res->getString("role");

cout << "Login successful. Your role: \"" << role << "\"" << endl;

delete res;

delete pstmt;

return true;

}

else {

cout << "Invalid login or password.\n";

}

delete res;

delete pstmt;

}

catch (sql::SQLException& e) {

cout << "SQL Error: " << e.what() << endl;

}

return false;

}

void Auth::handleRoleMenu(Database& db, std::string& role)

{

if(role == "Admin")

{

Admin::AdminMenu(db);

}

else if (role == "Manager")

{

Manager::ManagerMenu(db);

}

else

{

return;

}

}

**│ ├──Auth.h**

#ifndef AUTH\_H

#define AUTH\_H

#include "Database.h"

#include <string>

class Auth {

public:

static bool login(Database& db, std::string& role);

static void handleRoleMenu(Database& db, std::string& role);

};

#endif // AUTH\_H

**│ ├──Ticket.cpp**

#include "Ticket.h"

#include "TicketSale.h"

#include <cppconn/driver.h>

#include <cppconn/connection.h>

#include <cppconn/resultset.h>

#include <cppconn/prepared\_statement.h>

#include <iostream>

#include <iomanip>

using namespace std;

namespace TicketMenu {

void displayTickets(Database& db) {

try {

sql::PreparedStatement\* pstmt = db.getConnection()->prepareStatement(

"SELECT t.ID, t.Ticket\_Type, t.Price, t.Visitor\_Category, "

"tour.Name AS Tour\_Name, museum.Name AS Museum\_Name "

"FROM Ticket t "

"LEFT JOIN Tour tour ON t.Tour\_ID = tour.ID "

"LEFT JOIN Museum museum ON t.Museum\_ID = museum.ID"

);

sql::ResultSet\* res = pstmt->executeQuery();

cout << "\n=== Available Tickets ===\n";

cout << left << setw(5) << "ID"

<< setw(15) << "Type"

<< setw(10) << "Price"

<< setw(20) << "Category"

<< setw(20) << "Tour"

<< setw(20) << "Museum" << endl;

while (res->next()) {

cout << left << setw(5) << res->getInt("ID")

<< setw(15) << res->getString("Ticket\_Type")

<< setw(10) << res->getDouble("Price")

<< setw(20) << res->getString("Visitor\_Category")

<< setw(20) << res->getString("Tour\_Name")

<< setw(20) << res->getString("Museum\_Name") << endl;

}

delete res;

delete pstmt;

}

catch (sql::SQLException& e) {

cerr << "SQL Error: " << e.what() << endl;

}

}

void showTicketsMenu(Database& db) {

int choice;

// while (true) {

displayTickets(db);

cout << "\n1. Buy Ticket\n";

cout << "2. Exit\n";

cout << "Choose an option: ";

cin >> choice;

switch (choice) {

case 1:

TicketSale::buyTickets(db);

break;

case 2:

return;

default:

cout << "Invalid option. Try again.\n";

}

// }

}

}

**│ ├──Ticket.h**

#ifndef TICKET\_H

#define TICKET\_H

#include "Database.h"

namespace TicketMenu {

void showTicketsMenu(Database& db);

}

#endif

**│ ├──TicketSale.cpp**

#include "TicketSale.h"

#include "Ticket.h"

#include "Database.h"

#include <iomanip>

#include <cppconn/prepared\_statement.h>

#include <cppconn/resultset.h>

#include <cppconn/statement.h>

#include <iostream>

using namespace std;

void TicketSale::buyTickets(Database& db) {

int quantity = 0;

cout << "Enter ticket quantity: ";

cin >> quantity;

double totalPrice = 0.0;

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

int ticket\_id;

string visitor\_name;

string visitor\_email;

cout << "\nEnter Ticket ID: ";

cin >> ticket\_id;

cout << "Enter your name: ";

cin.ignore(); // To ignore leftover newline

getline(cin, visitor\_name);

cout << "Enter your Email: ";

cin >> visitor\_email;

try {

sql::PreparedStatement\* pstmt = db.getConnection()->prepareStatement(

"SELECT Price FROM Ticket WHERE ID = ?");

pstmt->setInt(1, ticket\_id);

sql::ResultSet\* res = pstmt->executeQuery();

if (res->next()) {

double price = res->getDouble("Price");

totalPrice += price;

}

else {

cout << "Ticket ID not found! Price will not be added." << endl;

}

delete res;

delete pstmt;

}

catch (sql::SQLException& e) {

cerr << "SQL Error: " << e.what() << endl;

}

// Insert visitor data

try {

sql::PreparedStatement\* insertStmt = db.getConnection()->prepareStatement(

"INSERT INTO Visitor (Ticket\_ID, Name, Email) VALUES (?, ?, ?)");

insertStmt->setInt(1, ticket\_id);

insertStmt->setString(2, visitor\_name);

insertStmt->setString(3, visitor\_email);

insertStmt->execute();

delete insertStmt;

cout << "Visitor information saved successfully!" << endl;

}

catch (sql::SQLException& e) {

cerr << "SQL Error (insert): " << e.what() << endl;

}

}

cout << fixed << setprecision(2); // Always show 2 decimals

cout << "\nTotal price for " << quantity << " tickets: " << totalPrice << " EUR" << endl;

}

**│ ├──TicketSale.h**

#ifndef TICKETSALE\_H

#define TICKETSALE\_H

#include "Database.h"

#include <iostream>

#include <string>

using namespace std;

class TicketSale {

public:

static void buyTickets(Database& db);

};

#endif

**│ ├──ViewTicketsVisitors.cpp**

#include "ViewTicketsVisitors.h"

#include <string>

#include <iostream>

#include <cppconn/prepared\_statement.h>

using namespace std;

void ViewTicketsVisitors::ViewTickets(Database& db)

{

string name, email;

cout << "Enter name: ";

cin.ignore();

getline(cin, name);

cout << "Enter email: ";

getline(cin, email);

try {

sql::PreparedStatement\* pstmt = db.getConnection()->prepareStatement(

"SELECT v.Name, v.Email, t.Ticket\_Type, t.Visitor\_Category, t.Price, t.Sale\_Date, "

"tour.Name AS Tour\_Name, mu.Name AS Museum\_Name "

"FROM visitor v "

"JOIN ticket t ON v.Ticket\_ID = t.ID "

"LEFT JOIN Tour tour ON t.Tour\_ID = tour.ID "

"LEFT JOIN Museum mu ON t.Museum\_ID = mu.ID "

"WHERE v.Name = ? AND v.Email = ?"

);

pstmt->setString(1, name);

pstmt->setString(2, email);

sql::ResultSet\* res = pstmt->executeQuery();

if (res->next()) {

cout << "\n=== Visitor and Ticket Info ===\n";

cout << "Name: " << res->getString("Name") << endl;

cout << "Email: " << res->getString("Email") << endl;

cout << "Ticket Type: " << res->getString("Ticket\_Type") << endl;

cout << "Category: " << res->getString("Visitor\_Category") << endl;

cout << "Price: " << res->getDouble("Price") << endl;

cout << "Sale Date: " << res->getString("Sale\_Date") << endl;

cout << "Museum Name: " << res->getString("Museum\_Name") << endl;

cout << "Tour Name: " << res->getString("Tour\_Name") << endl;

}

else {

cout << "No visitor found with that name and email.\n";

}

delete res;

delete pstmt;

}

catch (sql::SQLException& e) {

cerr << "SQL Error: " << e.what() << endl;

}

}

**│ └──ViewTicketsVisitors.h**

#pragma once

#ifndef VIEW\_TICKETS\_VISITORS\_H

#define VIEW\_TICKETS\_VISITORS\_H

#include "Database.h"

class ViewTicketsVisitors

{

public:

static void ViewTickets(Database& db);

};

#endif

## **SQL-script table creating**

CREATE TABLE Museum (

ID INT AUTO\_INCREMENT PRIMARY KEY,

Name VARCHAR(255),

Address VARCHAR(255),

Contact\_Info VARCHAR(255)

);

CREATE TABLE Hall (

ID INT AUTO\_INCREMENT PRIMARY KEY,

Hall\_Name VARCHAR(255),

Capacity INT,

Museum\_ID INT,

FOREIGN KEY (Museum\_ID) REFERENCES Museum(ID)

);

CREATE TABLE Guide (

ID INT AUTO\_INCREMENT PRIMARY KEY,

Name VARCHAR(255),

Contact\_Info VARCHAR(255),

Years\_Experience INT,

Museum\_ID INT,

FOREIGN KEY (Museum\_ID) REFERENCES Museum(ID)

);

CREATE TABLE Tour (

ID INT AUTO\_INCREMENT PRIMARY KEY,

Name VARCHAR(255),

Date DATE,

Guide\_ID INT,

Museum\_ID INT,

FOREIGN KEY (Guide\_ID) REFERENCES Guide(ID),

FOREIGN KEY (Museum\_ID) REFERENCES Museum(ID)

);

CREATE TABLE Visitor (

ID INT AUTO\_INCREMENT PRIMARY KEY,

Name VARCHAR(255),

Email VARCHAR(255),

Ticket\_ID INT,

FOREIGN KEY (Ticket\_ID) REFERENCES Ticket(ID)

);

CREATE TABLE Ticket (

ID INT AUTO\_INCREMENT PRIMARY KEY,

Ticket\_Type VARCHAR(100),

Price DOUBLE,

Visitor\_Category VARCHAR(100),

Tour\_ID INT,

Museum\_ID INT,

Sale\_Date DATE,

FOREIGN KEY (Tour\_ID) REFERENCES Tour(ID),

FOREIGN KEY (Museum\_ID) REFERENCES Museum(ID)

);

ALTER TABLE Visitor

ADD FOREIGN KEY (Ticket\_ID) REFERENCES Ticket(ID);

CREATE TABLE Ticket\_Sale (

ID INT AUTO\_INCREMENT PRIMARY KEY,

Seller\_ID INT,

Ticket\_ID INT,

FOREIGN KEY (Ticket\_ID) REFERENCES Ticket(ID),

FOREIGN KEY (Seller\_ID) REFERENCES Seller(ID)

);

CREATE TABLE Author (

ID INT AUTO\_INCREMENT PRIMARY KEY,

Name VARCHAR(255),

Email VARCHAR(255),

Agreement VARCHAR(255)

);

CREATE TABLE Exhibition (

ID INT AUTO\_INCREMENT PRIMARY KEY,

Name VARCHAR(255),

Start\_Date DATE,

End\_Date DATE,

Cost DECIMAL,

Museum\_ID INT,

FOREIGN KEY (Museum\_ID) REFERENCES Museum(ID)

);

CREATE TABLE Exhibition\_Author (

ID INT AUTO\_INCREMENT PRIMARY KEY,

Share DECIMAL,

Exhibition\_ID INT,

Author\_ID INT,

FOREIGN KEY (Exhibition\_ID) REFERENCES Exhibition(ID),

FOREIGN KEY (Author\_ID) REFERENCES Author(ID)

);

CREATE TABLE Sponsor (

ID INT AUTO\_INCREMENT PRIMARY KEY,

Sponsor\_Name VARCHAR(255),

Contact\_Info VARCHAR(255),

Contribution\_Amount DOUBLE,

Contribution\_Date DATE,

Museum\_ID INT,

FOREIGN KEY (Museum\_ID) REFERENCES Museum(ID)

);

CREATE TABLE Seller (

ID INT AUTO\_INCREMENT PRIMARY KEY,

Full\_Name VARCHAR(255),

Contact\_Info VARCHAR(255)

);

CREATE TABLE User (

ID INT AUTO\_INCREMENT PRIMARY KEY,

Username VARCHAR(255),

Role ENUM('admin', 'manager', 'user'),

Password VARCHAR(255)

);