### **PARENTS BACKEND:**

### **Version 1 Of Backend C++ Code**

#### // -----------------------------------------------------------------------------------------------------------------------

#### // Name Gino Torres

#### // Course CMPS 499, Spring 2025

#### // Senior Project

#### //

#### // Purpose:

#### /\*

#### Backend Devleopment for Senior Project:

#### User Input Handling: Collect and validate player details.

#### Age Calculation: Determine player's age as of July 31st.

#### Division Assignment: Assign player to the correct division (7U–13U).

#### Basic Data Storage: Store validated player data in a structured format (can be a simple file or database later).

#### \*/

#### //------------------------------------------------------------------------------------------------------------------------

#### #include <iostream>

#### #include <string>

#### #include <vector>

#### #include <ctime>

#### #include <regex>

#### 

#### using namespace std;

#### 

#### // Structure to store player data

#### struct Player {

#### string name;

#### string dob;

#### string guardian;

#### string phone;

#### string email;

#### string division;

#### };

#### 

#### // Function to validate phone number format

#### bool isValidPhone(const string& phone) {

#### regex phoneRegex("\\d{10}"); // Basic validation for 10-digit numbers

#### return regex\_match(phone, phoneRegex);

#### }

#### 

#### // Function to validate email format

#### bool isValidEmail(const string& email) {

#### regex emailRegex("^[\\w.-]+@[\\w.-]+\\.[a-zA-Z]{2,}$");

#### return regex\_match(email, emailRegex);

#### }

#### 

#### // Function to calculate age as of July 31st of the current year

#### int calculateAge(const string& dob) {

#### int birthYear, birthMonth, birthDay;

#### sscanf(dob.c\_str(), "%d-%d-%d", &birthYear, &birthMonth, &birthDay);

#### 

#### time\_t t = time(nullptr);

#### tm\* now = localtime(&t);

#### int currentYear = now->tm\_year + 1900;

#### int age = currentYear - birthYear;

#### 

#### // Adjust age if birthdate is after July 31st

#### if (birthMonth > 7 || (birthMonth == 7 && birthDay > 31)) {

#### age--;

#### }

#### return age;

#### }

#### 

#### // Function to determine the player's division

#### string assignDivision(int age) {

#### if (age <= 7) return "7U";

#### else if (age <= 8) return "8U";

#### else if (age <= 9) return "9U";

#### else if (age <= 10) return "10U";

#### else if (age <= 11) return "11U";

#### else if (age <= 12) return "12U";

#### else return "13U";

#### }

#### 

#### // Function to register a player

#### void registerPlayer(vector<Player>& players) {

#### Player p;

#### 

#### cout << "Enter Player Name: ";

#### getline(cin, p.name);

#### 

#### cout << "Enter Date of Birth (YYYY-MM-DD): ";

#### getline(cin, p.dob);

#### 

#### cout << "Enter Guardian Name: ";

#### getline(cin, p.guardian);

#### 

#### do {

#### cout << "Enter Phone Number (10 digits): ";

#### getline(cin, p.phone);

#### } while (!isValidPhone(p.phone));

#### 

#### do {

#### cout << "Enter Email: ";

#### getline(cin, p.email);

#### } while (!isValidEmail(p.email));

#### 

#### int age = calculateAge(p.dob);

#### p.division = assignDivision(age);

#### 

#### players.push\_back(p);

#### cout << "\nPlayer registered successfully! Assigned to Division: " << p.division << "\n";

#### }

#### 

#### // Function to display all registered players

#### void displayPlayers(const vector<Player>& players) {

#### if (players.empty()) {

#### cout << "No players registered yet.\n";

#### return;

#### }

#### 

#### cout << "\nRegistered Players:\n";

#### for (const auto& p : players) {

#### cout << "Name: " << p.name << " | DOB: " << p.dob << " | Guardian: " << p.guardian

#### << " | Phone: " << p.phone << " | Email: " << p.email << " | Division: " << p.division << "\n";

#### }

#### }

#### 

#### int main() {

#### vector<Player> players;

#### int choice;

#### 

#### while (true) {

#### cout << "\nYouth Football Registration System";

#### cout << "\n1. Register a Player";

#### cout << "\n2. View Registered Players";

#### cout << "\n3. Exit";

#### cout << "\nEnter your choice: ";

#### cin >> choice;

#### cin.ignore();

#### 

#### switch (choice) {

#### case 1:

#### registerPlayer(players);

#### break;

#### case 2:

#### displayPlayers(players);

#### break;

#### case 3:

#### cout << "Exiting system. Goodbye!\n";

#### return 0;

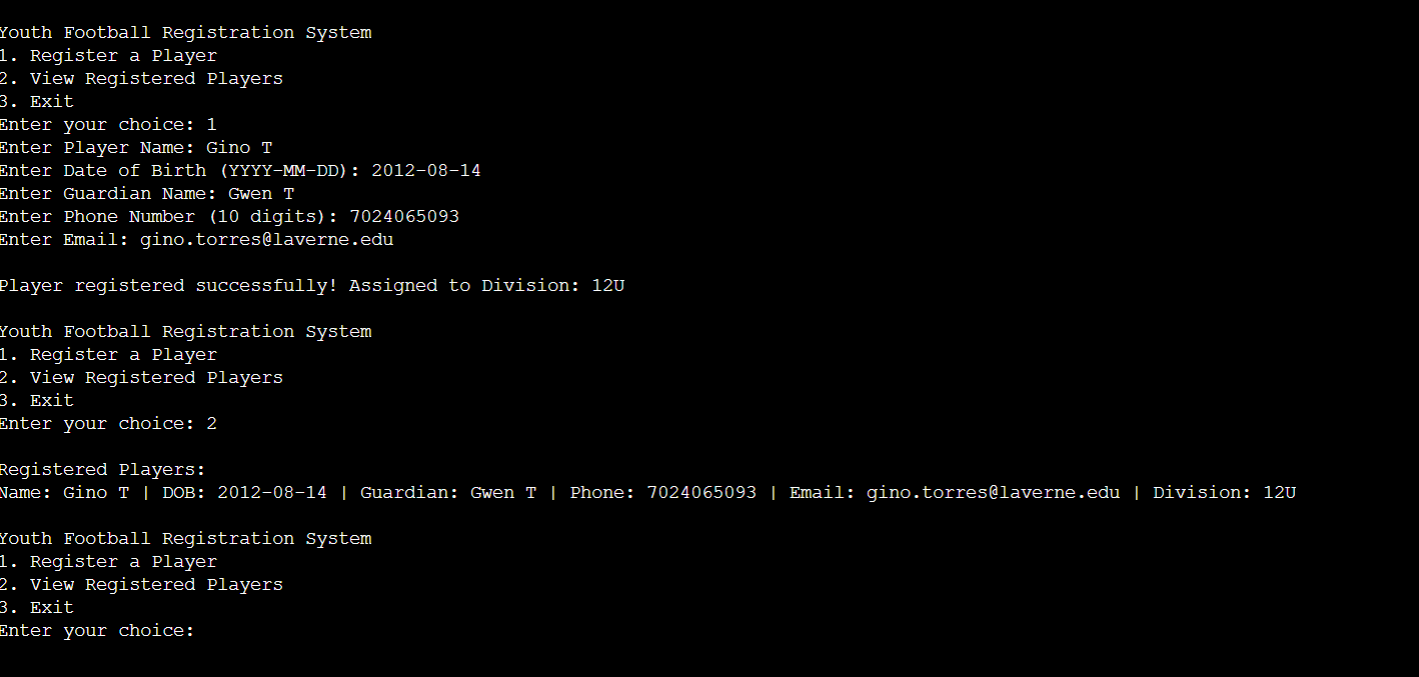
#### default:

#### cout << "Invalid choice. Please try again.\n";

#### }

#### }

#### }

**OUTPUT EXAMPLE: (image)  
**

### **Version 2 Of Backend C++ Code**

// -----------------------------------------------------------------------------------------------------------------------

// Name Gino Torres

// Course CMPS 499, Spring 2025

// Senior Project

//

// Purpose:

/\*

Backend Devleopment for Senior Project:

User Input Handling: Collect and validate player details.

Age Calculation: Determine player's age as of July 31st.

Division Assignment: Assign player to the correct division (7U–13U).

Basic Data Storage: Store validated player data in a structured format (can be a simple file or database later).

Save player data to a file (players.txt) – This ensures data persistence even if the program is restarted.

Store player data in an SQLite database (players.db) – This allows better data management and retrieval.

Load existing players from the database on startup – So previously registered players are not lost.

\*/

//------------------------------------------------------------------------------------------------------------------------

#include <iostream>

#include <fstream>

#include <string>

#include <vector>

#include <ctime>

#include <regex>

#include <sqlite3.h>

using namespace std;

// Structure to store player data

struct Player {

string name;

string dob;

string guardian;

string phone;

string email;

string division;

};

// Function to validate phone number format

bool isValidPhone(const string& phone) {

regex phoneRegex("\\d{10}"); // Basic validation for 10-digit numbers

return regex\_match(phone, phoneRegex);

}

// Function to validate email format

bool isValidEmail(const string& email) {

regex emailRegex("^[\\w.-]+@[\\w.-]+\\.[a-zA-Z]{2,}$");

return regex\_match(email, emailRegex);

}

// Function to calculate age as of July 31st of the current year

int calculateAge(const string& dob) {

int birthYear, birthMonth, birthDay;

sscanf(dob.c\_str(), "%d-%d-%d", &birthYear, &birthMonth, &birthDay);

time\_t t = time(nullptr);

tm\* now = localtime(&t);

int currentYear = now->tm\_year + 1900;

int age = currentYear - birthYear;

// Adjust age if birthdate is after July 31st

if (birthMonth > 7 || (birthMonth == 7 && birthDay > 31)) {

age--;

}

return age;

}

// Function to determine the player's division

string assignDivision(int age) {

if (age <= 7) return "7U";

else if (age <= 8) return "8U";

else if (age <= 9) return "9U";

else if (age <= 10) return "10U";

else if (age <= 11) return "11U";

else if (age <= 12) return "12U";

else return "13U";

}

// Function to initialize SQLite database

void initializeDatabase(sqlite3\* &db) {

sqlite3\_open("players.db", &db);

string createTable = "CREATE TABLE IF NOT EXISTS Players (Name TEXT, DOB TEXT, Guardian TEXT, Phone TEXT, Email TEXT, Division TEXT);";

sqlite3\_exec(db, createTable.c\_str(), nullptr, nullptr, nullptr);

}

// Function to insert player data into SQLite database

void insertPlayerIntoDB(sqlite3\* db, const Player& p) {

string query = "INSERT INTO Players (Name, DOB, Guardian, Phone, Email, Division) VALUES ('" + p.name + "', '" + p.dob + "', '" + p.guardian + "', '" + p.phone + "', '" + p.email + "', '" + p.division + "');";

sqlite3\_exec(db, query.c\_str(), nullptr, nullptr, nullptr);

}

// Function to load players from SQLite database

void loadPlayersFromDB(sqlite3\* db, vector<Player>& players) {

sqlite3\_stmt\* stmt;

string query = "SELECT \* FROM Players;";

if (sqlite3\_prepare\_v2(db, query.c\_str(), -1, &stmt, nullptr) == SQLITE\_OK) {

while (sqlite3\_step(stmt) == SQLITE\_ROW) {

Player p;

p.name = reinterpret\_cast<const char\*>(sqlite3\_column\_text(stmt, 0));

p.dob = reinterpret\_cast<const char\*>(sqlite3\_column\_text(stmt, 1));

p.guardian = reinterpret\_cast<const char\*>(sqlite3\_column\_text(stmt, 2));

p.phone = reinterpret\_cast<const char\*>(sqlite3\_column\_text(stmt, 3));

p.email = reinterpret\_cast<const char\*>(sqlite3\_column\_text(stmt, 4));

p.division = reinterpret\_cast<const char\*>(sqlite3\_column\_text(stmt, 5));

players.push\_back(p);

}

}

sqlite3\_finalize(stmt);

}

// Function to register a player

void registerPlayer(vector<Player>& players, sqlite3\* db) {

Player p;

cout << "Enter Player Name: ";

getline(cin, p.name);

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

getline(cin, p.dob);

cout << "Enter Guardian Name: ";

getline(cin, p.guardian);

do {

cout << "Enter Phone Number (10 digits): ";

getline(cin, p.phone);

} while (!isValidPhone(p.phone));

do {

cout << "Enter Email: ";

getline(cin, p.email);

} while (!isValidEmail(p.email));

int age = calculateAge(p.dob);

p.division = assignDivision(age);

players.push\_back(p);

insertPlayerIntoDB(db, p);

cout << "\nPlayer registered successfully! Assigned to Division: " << p.division << "\n";

}

// Function to display all registered players

void displayPlayers(const vector<Player>& players) {

if (players.empty()) {

cout << "No players registered yet.\n";

return;

}

cout << "\nRegistered Players:\n";

for (const auto& p : players) {

cout << "Name: " << p.name << " | DOB: " << p.dob << " | Guardian: " << p.guardian

<< " | Phone: " << p.phone << " | Email: " << p.email << " | Division: " << p.division << "\n";

}

}

int main() {

sqlite3\* db;

initializeDatabase(db);

vector<Player> players;

loadPlayersFromDB(db, players);

int choice;

while (true) {

cout << "\nYouth Football Registration System";

cout << "\n1. Register a Player";

cout << "\n2. View Registered Players";

cout << "\n3. Exit";

cout << "\nEnter your choice: ";

cin >> choice;

cin.ignore();

switch (choice) {

case 1:

registerPlayer(players, db);

break;

case 2:

displayPlayers(players);

break;

case 3:

cout << "Exiting system. Goodbye!\n";

sqlite3\_close(db);

return 0;

default:

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

}

}

}

**OUTPUT EXAMPLE: (Image)**

****

Had trouble with database connection but with help, I was able to figure out how to get it to work. Had to make a .exe file. (Help from Classmate and GPT). Since I have the backend done, I have been trying for the past few days to get the Frontend working for this using QT but I have not been able to get it to work so far. I may have to change direction with this code by either going into Python and getting a UI using that instead of C++ or G++ or I may have to forget about the front end and just expand on the backend. I am planning, in that case, to make another version of the backend for the Admin’s to give them the ability to easily remove players from the database if needed.

### **Version 3 Of Backend C++ Code**

// -----------------------------------------------------------------------------------------------------------------------

// Name Gino Torres

// Course CMPS 499, Spring 2025

// Senior Project

//

// Purpose:

/\*

Backend Devleopment for Senior Project:

User Input Handling: Collect and validate player details. (Added ID Assignment)

Age Calculation: Determine player's age as of July 31st.

Division Assignment: Assign player to the correct division (7U–13U).

Basic Data Storage: Store validated player data in a structured format (can be a simple file or database later).

Save player data to a file (players.txt) – This ensures data persistence even if the program is restarted.

Store player data in an SQLite database (players.db) – This allows better data management and retrieval.

Load existing players from the database on startup – So previously registered players are not lost.

\*/

//------------------------------------------------------------------------------------------------------------------------

#include <iostream>

#include <fstream>

#include <string>

#include <vector>

#include <ctime>

#include <regex>

#include <sqlite3.h>

using namespace std;

// Structure to store player data

struct Player {

int id; // Unique ID for each player

string name;

string dob;

string guardian;

string phone;

string email;

string division;

};

// Function to validate phone number format

bool isValidPhone(const string& phone) {

regex phoneRegex("\\d{10}");

return regex\_match(phone, phoneRegex);

}

// Function to validate email format

bool isValidEmail(const string& email) {

regex emailRegex("^[\\w.-]+@[\\w.-]+\\.[a-zA-Z]{2,}$");

return regex\_match(email, emailRegex);

}

// Function to calculate age as of July 31st of the current year

int calculateAge(const string& dob) {

int birthYear, birthMonth, birthDay;

sscanf(dob.c\_str(), "%d-%d-%d", &birthYear, &birthMonth, &birthDay);

time\_t t = time(nullptr);

tm\* now = localtime(&t);

int currentYear = now->tm\_year + 1900;

int age = currentYear - birthYear;

if (birthMonth > 7 || (birthMonth == 7 && birthDay > 31)) {

age--;

}

return age;

}

// Function to determine the player's division

string assignDivision(int age) {

if (age <= 7) return "7U";

else if (age <= 8) return "8U";

else if (age <= 9) return "9U";

else if (age <= 10) return "10U";

else if (age <= 11) return "11U";

else if (age <= 12) return "12U";

else return "13U";

}

// Function to initialize SQLite database

void initializeDatabase(sqlite3\*& db) {

sqlite3\_open("registration.db", &db);

string createTable = "CREATE TABLE IF NOT EXISTS Players ("

"ID INTEGER PRIMARY KEY AUTOINCREMENT, "

"Name TEXT, "

"DOB TEXT, "

"Guardian TEXT, "

"Phone TEXT, "

"Email TEXT, "

"Division TEXT);";

sqlite3\_exec(db, createTable.c\_str(), nullptr, nullptr, nullptr);

}

// Function to insert player data into SQLite database

void insertPlayerIntoDB(sqlite3\* db, const Player& p) {

string query = "INSERT INTO Players (Name, DOB, Guardian, Phone, Email, Division) VALUES ('" +

p.name + "', '" + p.dob + "', '" + p.guardian + "', '" + p.phone + "', '" +

p.email + "', '" + p.division + "');";

sqlite3\_exec(db, query.c\_str(), nullptr, nullptr, nullptr);

}

// Function to load players from SQLite database

void loadPlayersFromDB(sqlite3\* db, vector<Player>& players) {

sqlite3\_stmt\* stmt;

string query = "SELECT \* FROM Players;";

if (sqlite3\_prepare\_v2(db, query.c\_str(), -1, &stmt, nullptr) == SQLITE\_OK) {

while (sqlite3\_step(stmt) == SQLITE\_ROW) {

Player p;

p.id = sqlite3\_column\_int(stmt, 0);

p.name = reinterpret\_cast<const char\*>(sqlite3\_column\_text(stmt, 1));

p.dob = reinterpret\_cast<const char\*>(sqlite3\_column\_text(stmt, 2));

p.guardian = reinterpret\_cast<const char\*>(sqlite3\_column\_text(stmt, 3));

p.phone = reinterpret\_cast<const char\*>(sqlite3\_column\_text(stmt, 4));

p.email = reinterpret\_cast<const char\*>(sqlite3\_column\_text(stmt, 5));

p.division = reinterpret\_cast<const char\*>(sqlite3\_column\_text(stmt, 6));

players.push\_back(p);

}

}

sqlite3\_finalize(stmt);

}

// Function to register a player

void registerPlayer(vector<Player>& players, sqlite3\* db) {

Player p;

cout << "Enter Player Name: ";

getline(cin, p.name);

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

getline(cin, p.dob);

cout << "Enter Guardian Name: ";

getline(cin, p.guardian);

do {

cout << "Enter Phone Number (10 digits): ";

getline(cin, p.phone);

} while (!isValidPhone(p.phone));

do {

cout << "Enter Email: ";

getline(cin, p.email);

} while (!isValidEmail(p.email));

int age = calculateAge(p.dob);

p.division = assignDivision(age);

players.push\_back(p);

insertPlayerIntoDB(db, p);

cout << "\nPlayer registered successfully! Assigned to Division: " << p.division << "\n";

}

// Function to display all registered players

void displayPlayers(const vector<Player>& players) {

if (players.empty()) {

cout << "No players registered yet.\n";

return;

}

cout << "\nRegistered Players:\n";

for (const auto& p : players) {

cout << "ID: " << p.id << " | Name: " << p.name << " | DOB: " << p.dob

<< " | Guardian: " << p.guardian << " | Phone: " << p.phone

<< " | Email: " << p.email << " | Division: " << p.division << "\n";

}

}

int main() {

sqlite3\* db;

initializeDatabase(db);

vector<Player> players;

loadPlayersFromDB(db, players);

int choice;

while (true) {

cout << "\nYouth Football Registration System";

cout << "\n1. Register a Player";

cout << "\n2. View Registered Players";

cout << "\n3. Exit";

cout << "\nEnter your choice: ";

cin >> choice;

cin.ignore();

switch (choice) {

case 1:

registerPlayer(players, db);

break;

case 2:

displayPlayers(players);

break;

case 3:

cout << "Exiting system. Goodbye!\n";

sqlite3\_close(db);

return 0;

default:

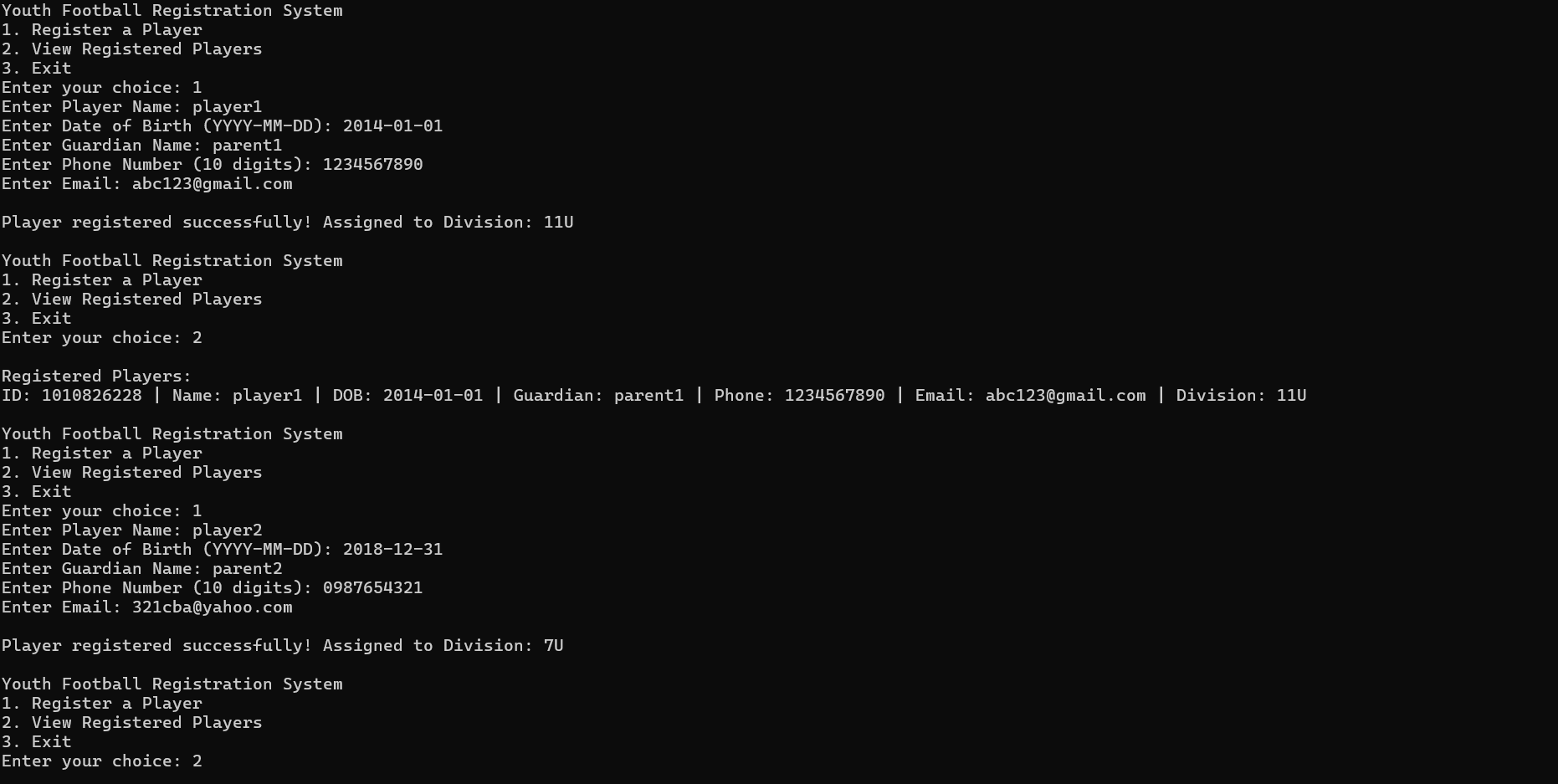
cout << "Invalid choice. Please try again.\n";

}

}

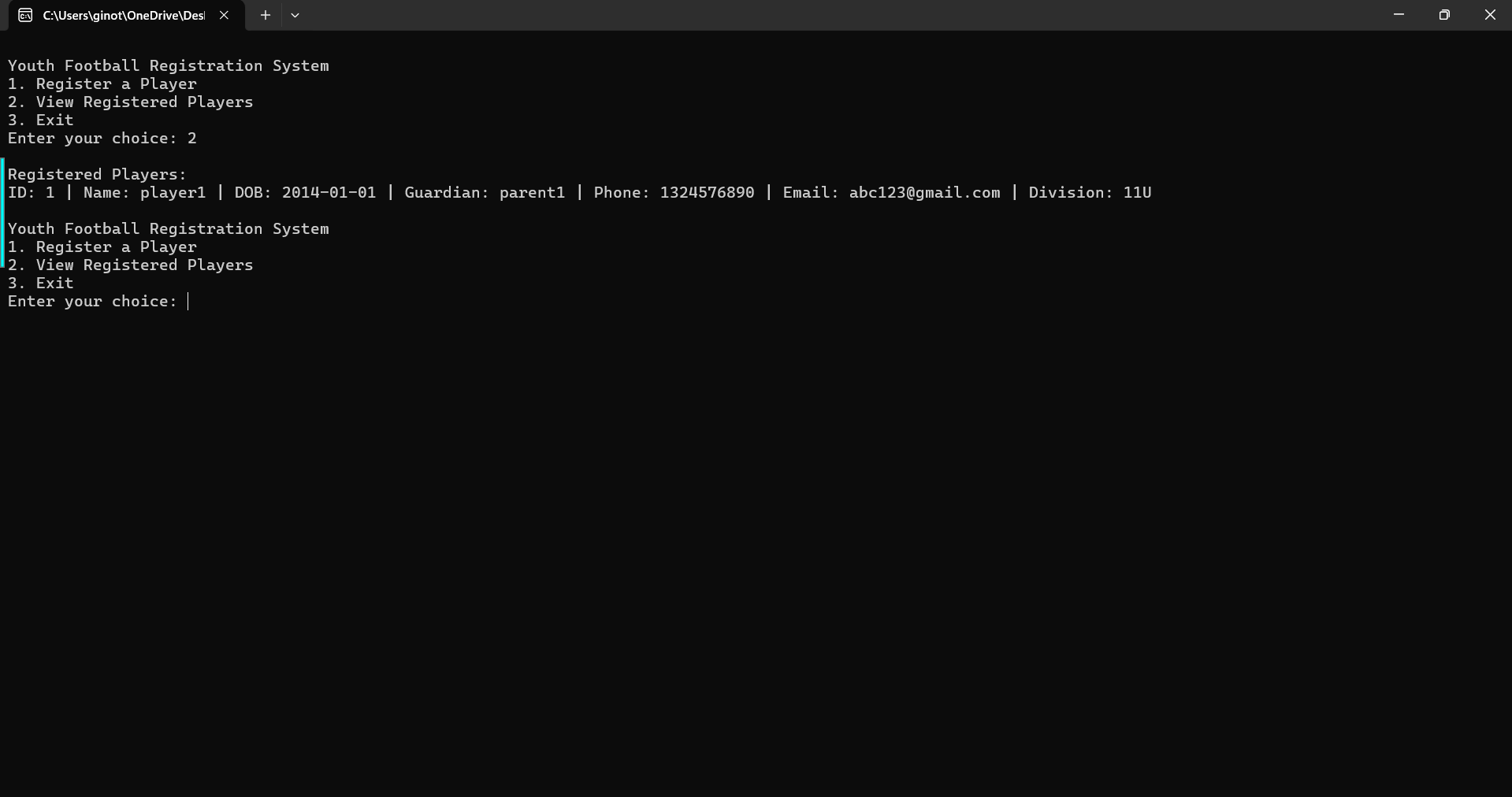
}

**OUTPUT EXAMPLE:**

****

### 

New database was created ‘registration’, so that it makes it easier for the admin to edit specific information about each players and delete them.



After Admin deleted player 2.

### **ADMIN’S BACKEND:**

### **Version 1 Of Backend C++ Code**

// -----------------------------------------------------------------------------------------------------------------------

// Name Gino Torres

// Course CMPS 499, Spring 2025

// Senior Project

//

// Purpose:

/\*

Admin Backend Devleopment for Senior Project:

Admin's that makes it easier for the Admin's to view, access, and edit the database.

Specifically with being able to see the players on the database, edit the information in the database, and remove players from the registration database.

\*/

//------------------------------------------------------------------------------------------------------------------------

#include <iostream>

#include <sqlite3.h>

#include <string>

using namespace std;

// Function prototypes

void viewPlayers(sqlite3 \*db);

void editPlayer(sqlite3 \*db);

void removePlayer(sqlite3 \*db);

int main() {

sqlite3 \*db;

int exit = sqlite3\_open("players.db", &db);

if (exit) {

cerr << "Error opening database: " << sqlite3\_errmsg(db) << std::endl;

return -1;

} else {

cout << "Database opened successfully.\n";

}

int choice;

do {

cout << "\nAdmin Menu:\n";

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

cout << "2. Edit Player Information\n";

cout << "3. Remove Player\n";

cout << "4. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

cin.ignore(); // Clear input buffer

switch (choice) {

case 1:

viewPlayers(db);

break;

case 2:

editPlayer(db);

break;

case 3:

removePlayer(db);

break;

case 4:

cout << "Exiting...\n";

break;

default:

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

}

} while (choice != 4);

sqlite3\_close(db);

return 0;

}

// Function to view all players in the database

void viewPlayers(sqlite3 \*db) {

string query = "SELECT \* FROM players;";

sqlite3\_stmt \*stmt;

if (sqlite3\_prepare\_v2(db, query.c\_str(), -1, &stmt, nullptr) != SQLITE\_OK) {

cerr << "Error executing query: " << sqlite3\_errmsg(db) << endl;

return;

}

cout << "\nPlayers List:\n";

while (sqlite3\_step(stmt) == SQLITE\_ROW) {

int id = sqlite3\_column\_int(stmt, 0);

const char \*name = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 1));

const char \*dob = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 2));

const char \*guardianName = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 3));

const char \*phone = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 4));

const char \*email = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 5));

cout << "ID: " << id << ", Name: " << name << ", DOB: " << dob

<< ", Guardian: " << guardianName << ", Phone: " << phone

<< ", Email: " << email << std::endl;

}

sqlite3\_finalize(stmt);

}

// Function to edit a player's information

void editPlayer(sqlite3 \*db) {

int id;

string column, newValue;

cout << "Enter the Player ID to edit: ";

cin >> id;

cin.ignore();

cout << "Enter the column to edit (name, dob, guardian\_name, phone, email): ";

getline(cin, column);

cout << "Enter the new value: ";

getline(cin, newValue);

string query = "UPDATE players SET " + column + " = ? WHERE id = ?;";

sqlite3\_stmt \*stmt;

if (sqlite3\_prepare\_v2(db, query.c\_str(), -1, &stmt, nullptr) != SQLITE\_OK) {

cerr << "Error preparing query: " << sqlite3\_errmsg(db) << endl;

return;

}

sqlite3\_bind\_text(stmt, 1, newValue.c\_str(), -1, SQLITE\_STATIC);

sqlite3\_bind\_int(stmt, 2, id);

if (sqlite3\_step(stmt) != SQLITE\_DONE) {

cerr << "Error executing query: " << sqlite3\_errmsg(db) << endl;

} else {

cout << "Player information updated successfully.\n";

}

sqlite3\_finalize(stmt);

}

// Function to remove a player from the database

void removePlayer(sqlite3 \*db) {

int id;

cout << "Enter the Player ID to remove: ";

cin >> id;

string query = "DELETE FROM players WHERE id = ?;";

sqlite3\_stmt \*stmt;

if (sqlite3\_prepare\_v2(db, query.c\_str(), -1, &stmt, nullptr) != SQLITE\_OK) {

cerr << "Error preparing query: " << sqlite3\_errmsg(db) << endl;

return;

}

sqlite3\_bind\_int(stmt, 1, id);

if (sqlite3\_step(stmt) != SQLITE\_DONE) {

cerr << "Error executing query: " << sqlite3\_errmsg(db) << endl;

} else {

cout << "Player removed successfully.\n";

}

sqlite3\_finalize(stmt);

}

### **Version 2 Of Backend C++ Code**

**// -----------------------------------------------------------------------------------------------------------------------**

**// Name Gino Torres**

**// Course CMPS 499, Spring 2025**

**// Senior Project**

**//**

**// Purpose:**

**/\***

**Admin Backend Development for Senior Project:**

**Admins can view, access, and edit the database with features to see registered players, edit player information, and remove players.**

**\*/**

**// -----------------------------------------------------------------------------------------------------------------------**

**#include <iostream>**

**#include <sqlite3.h>**

**#include <string>**

**using namespace std;**

**// Function prototypes**

**void viewPlayers(sqlite3 \*db);**

**void editPlayer(sqlite3 \*db);**

**void removePlayer(sqlite3 \*db);**

**int main() {**

**sqlite3 \*db;**

**int exit = sqlite3\_open("registration.db", &db); // Connect to the new database "registration.db"**

**if (exit) {**

**cerr << "Error opening database: " << sqlite3\_errmsg(db) << endl;**

**return -1;**

**} else {**

**cout << "Database opened successfully.\n";**

**}**

**int choice;**

**do {**

**cout << "\nAdmin Menu:\n";**

**cout << "1. View Players\n";**

**cout << "2. Edit Player Information\n";**

**cout << "3. Remove Player\n";**

**cout << "4. Exit\n";**

**cout << "Enter your choice: ";**

**cin >> choice;**

**cin.ignore(); // Clear input buffer**

**switch (choice) {**

**case 1:**

**viewPlayers(db);**

**break;**

**case 2:**

**editPlayer(db);**

**break;**

**case 3:**

**removePlayer(db);**

**break;**

**case 4:**

**cout << "Exiting...\n";**

**break;**

**default:**

**cout << "Invalid choice. Please try again.\n";**

**}**

**} while (choice != 4);**

**sqlite3\_close(db);**

**return 0;**

**}**

**// Function to view all players in the database**

**void viewPlayers(sqlite3 \*db) {**

**string query = "SELECT ROWID, Name, DOB, Guardian, Phone, Email, Division FROM Players;";**

**sqlite3\_stmt \*stmt;**

**if (sqlite3\_prepare\_v2(db, query.c\_str(), -1, &stmt, nullptr) != SQLITE\_OK) {**

**cerr << "Error executing query: " << sqlite3\_errmsg(db) << endl;**

**return;**

**}**

**cout << "\nPlayers List:\n";**

**cout << "ID | Name | DOB | Guardian | Phone | Email | Division\n";**

**cout << "---------------------------------------------------------\n";**

**while (sqlite3\_step(stmt) == SQLITE\_ROW) {**

**int id = sqlite3\_column\_int(stmt, 0);**

**const char \*name = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 1));**

**const char \*dob = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 2));**

**const char \*guardian = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 3));**

**const char \*phone = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 4));**

**const char \*email = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 5));**

**const char \*division = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 6));**

**cout << id << " | " << name << " | " << dob << " | " << guardian**

**<< " | " << phone << " | " << email << " | " << division << "\n";**

**}**

**sqlite3\_finalize(stmt);**

**}**

**// Function to edit a player's information in the database**

**void editPlayer(sqlite3 \*db) {**

**int id;**

**string column, newValue;**

**cout << "Enter Player ID to edit: ";**

**cin >> id;**

**cin.ignore();**

**cout << "Enter the field to edit (Name, DOB, Guardian, Phone, Email, Division): ";**

**getline(cin, column);**

**cout << "Enter the new value: ";**

**getline(cin, newValue);**

**string query = "UPDATE Players SET " + column + " = ? WHERE ROWID = ?;";**

**sqlite3\_stmt \*stmt;**

**if (sqlite3\_prepare\_v2(db, query.c\_str(), -1, &stmt, nullptr) == SQLITE\_OK) {**

**sqlite3\_bind\_text(stmt, 1, newValue.c\_str(), -1, SQLITE\_STATIC);**

**sqlite3\_bind\_int(stmt, 2, id);**

**if (sqlite3\_step(stmt) == SQLITE\_DONE) {**

**cout << "Player information updated successfully.\n";**

**} else {**

**cerr << "Error updating player information: " << sqlite3\_errmsg(db) << endl;**

**}**

**} else {**

**cerr << "Error preparing query: " << sqlite3\_errmsg(db) << endl;**

**}**

**sqlite3\_finalize(stmt);**

**}**

**// Function to remove a player from the database**

**void removePlayer(sqlite3 \*db) {**

**int id;**

**cout << "Enter Player ID to remove: ";**

**cin >> id;**

**cin.ignore();**

**string query = "DELETE FROM Players WHERE ROWID = ?;";**

**sqlite3\_stmt \*stmt;**

**if (sqlite3\_prepare\_v2(db, query.c\_str(), -1, &stmt, nullptr) == SQLITE\_OK) {**

**sqlite3\_bind\_int(stmt, 1, id);**

**if (sqlite3\_step(stmt) == SQLITE\_DONE) {**

**cout << "Player removed successfully.\n";**

**} else {**

**cerr << "Error removing player: " << sqlite3\_errmsg(db) << std::endl;**

**}**

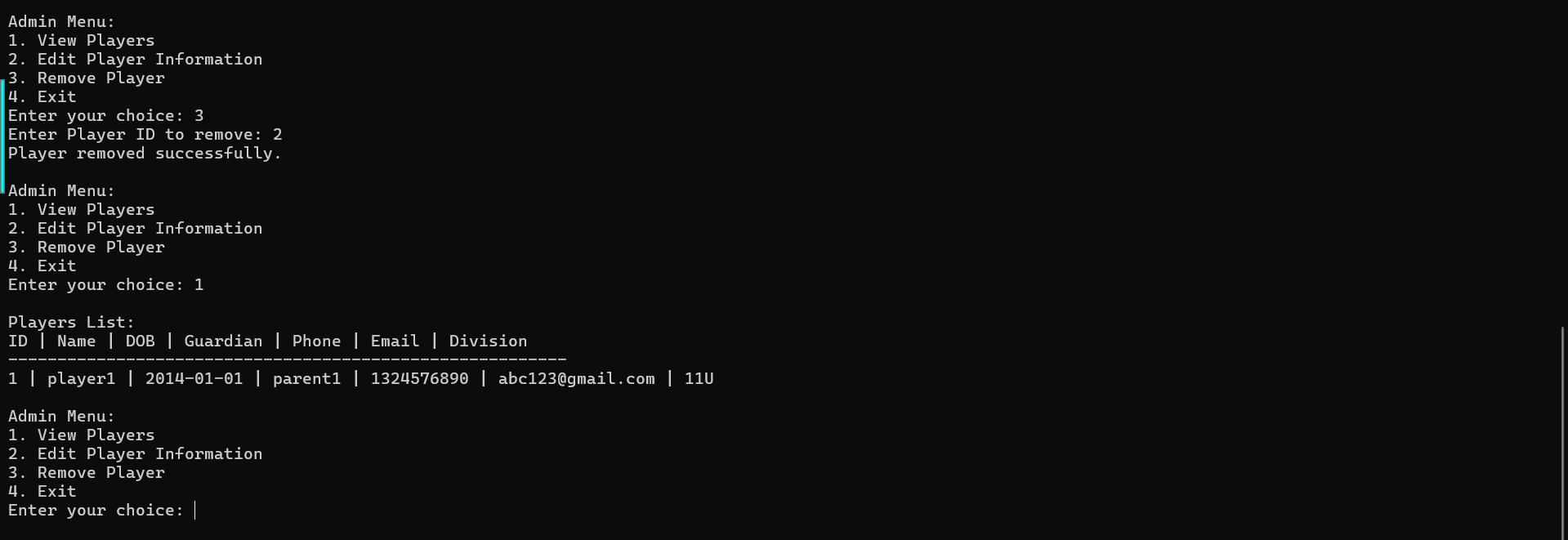
**} else {**

**cerr << "Error preparing query: " << sqlite3\_errmsg(db) << std::endl;**

**}**

**sqlite3\_finalize(stmt);**

**}**

**  
Version 3 Of Backend C++ Code**

**// -----------------------------------------------------------------------------------------------------------------------**

**// Name Gino Torres**

**// Course CMPS 499, Spring 2025**

**// Senior Project**

**//**

**// Purpose:**

**/\***

**Admin Backend Development for Senior Project:**

**Admins must log in with credentials before accessing functionality.**

**Can view, access, edit player information, including division, and remove players.**

**\*/**

**// -----------------------------------------------------------------------------------------------------------------------**

**#include <iostream>**

**#include <sqlite3.h>**

**#include <string>**

**#include <unordered\_set>**

**using namespace std;**

**// Function prototypes**

**void viewPlayers(sqlite3 \*db);**

**void editPlayer(sqlite3 \*db);**

**void removePlayer(sqlite3 \*db);**

**bool login();**

**int main() {**

**// Login authentication**

**if (!login()) {**

**cout << "Too many failed login attempts. Exiting...\n";**

**return 0;**

**}**

**sqlite3 \*db;**

**int exit = sqlite3\_open("registration.db", &db); // Connect to the database**

**if (exit) {**

**cerr << "Error opening database: " << sqlite3\_errmsg(db) << endl;**

**return -1;**

**} else {**

**cout << "Database opened successfully.\n";**

**}**

**int choice;**

**do {**

**cout << "\nAdmin Menu:\n";**

**cout << "1. View Players\n";**

**cout << "2. Edit Player Information\n";**

**cout << "3. Remove Player\n";**

**cout << "4. Exit\n";**

**cout << "Enter your choice: ";**

**cin >> choice;**

**cin.ignore(); // Clear input buffer**

**switch (choice) {**

**case 1:**

**viewPlayers(db);**

**break;**

**case 2:**

**editPlayer(db);**

**break;**

**case 3:**

**removePlayer(db);**

**break;**

**case 4:**

**cout << "Exiting...\n";**

**break;**

**default:**

**cout << "Invalid choice. Please try again.\n";**

**}**

**} while (choice != 4);**

**sqlite3\_close(db);**

**return 0;**

**}**

**// Function to handle admin login**

**bool login() {**

**const string correctUsername = "gtorres88";**

**const string correctPassword = "GFamily88!";**

**string username, password;**

**int attempts = 0;**

**const int maxAttempts = 3;**

**while (attempts < maxAttempts) {**

**cout << "=== Admin Login ===\n";**

**cout << "Username: ";**

**getline(cin, username);**

**cout << "Password: ";**

**getline(cin, password);**

**if (username == correctUsername && password == correctPassword) {**

**cout << "Login successful. Welcome, Admin!\n";**

**return true;**

**} else {**

**cout << "Incorrect username or password. Try again.\n";**

**attempts++;**

**}**

**}**

**return false;**

**}**

**// Function to view all players in the database**

**void viewPlayers(sqlite3 \*db) {**

**string query = "SELECT ROWID, Name, DOB, Guardian, Phone, Email, Division FROM Players;";**

**sqlite3\_stmt \*stmt;**

**if (sqlite3\_prepare\_v2(db, query.c\_str(), -1, &stmt, nullptr) != SQLITE\_OK) {**

**cerr << "Error executing query: " << sqlite3\_errmsg(db) << endl;**

**return;**

**}**

**cout << "\nPlayers List:\n";**

**cout << "ID | Name | DOB | Guardian | Phone | Email | Division\n";**

**cout << "---------------------------------------------------------\n";**

**while (sqlite3\_step(stmt) == SQLITE\_ROW) {**

**int id = sqlite3\_column\_int(stmt, 0);**

**const char \*name = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 1));**

**const char \*dob = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 2));**

**const char \*guardian = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 3));**

**const char \*phone = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 4));**

**const char \*email = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 5));**

**const char \*division = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 6));**

**cout << id << " | " << name << " | " << dob << " | " << guardian**

**<< " | " << phone << " | " << email << " | " << division << "\n";**

**}**

**sqlite3\_finalize(stmt);**

**}**

**// Function to edit a player's information in the database**

**void editPlayer(sqlite3 \*db) {**

**int id;**

**string column, newValue;**

**unordered\_set<string> validFields = {"Name", "DOB", "Guardian", "Phone", "Email", "Division"};**

**cout << "Enter Player ID to edit: ";**

**cin >> id;**

**cin.ignore();**

**cout << "Enter the field to edit (Name, DOB, Guardian, Phone, Email, Division): ";**

**getline(cin, column);**

**// Ensure the input column is valid**

**if (validFields.find(column) == validFields.end()) {**

**cout << "Invalid field. Valid fields are: Name, DOB, Guardian, Phone, Email, Division\n";**

**return;**

**}**

**cout << "Enter the new value: ";**

**getline(cin, newValue);**

**string query = "UPDATE Players SET " + column + " = ? WHERE ROWID = ?;";**

**sqlite3\_stmt \*stmt;**

**if (sqlite3\_prepare\_v2(db, query.c\_str(), -1, &stmt, nullptr) == SQLITE\_OK) {**

**sqlite3\_bind\_text(stmt, 1, newValue.c\_str(), -1, SQLITE\_STATIC);**

**sqlite3\_bind\_int(stmt, 2, id);**

**if (sqlite3\_step(stmt) == SQLITE\_DONE) {**

**cout << "Player information updated successfully.\n";**

**} else {**

**cerr << "Error updating player information: " << sqlite3\_errmsg(db) << endl;**

**}**

**} else {**

**cerr << "Error preparing query: " << sqlite3\_errmsg(db) << endl;**

**}**

**sqlite3\_finalize(stmt);**

**}**

**// Function to remove a player from the database**

**void removePlayer(sqlite3 \*db) {**

**int id;**

**cout << "Enter Player ID to remove: ";**

**cin >> id;**

**cin.ignore();**

**string query = "DELETE FROM Players WHERE ROWID = ?;";**

**sqlite3\_stmt \*stmt;**

**if (sqlite3\_prepare\_v2(db, query.c\_str(), -1, &stmt, nullptr) == SQLITE\_OK) {**

**sqlite3\_bind\_int(stmt, 1, id);**

**if (sqlite3\_step(stmt) == SQLITE\_DONE) {**

**cout << "Player removed successfully.\n";**

**} else {**

**cerr << "Error removing player: " << sqlite3\_errmsg(db) << std::endl;**

**}**

**} else {**

**cerr << "Error preparing query: " << sqlite3\_errmsg(db) << std::endl;**

**}**

**sqlite3\_finalize(stmt);**

**}**

****

**Version 4 Of Backend C++ Code**

**// -----------------------------------------------------------------------------------------------------------------------**

**// Name Gino Torres**

**// Course CMPS 499, Spring 2025**

**// Senior Project**

**//**

**// Purpose:**

**/\***

**Admin Backend Development for Senior Project:**

**Admins must log in with credentials before accessing functionality.**

**Can view, access, edit player information, including division, and remove players.**

**\*/**

**// -----------------------------------------------------------------------------------------------------------------------**

**#include <iostream>**

**#include <sqlite3.h>**

**#include <string>**

**#include <unordered\_set>**

**#include <chrono>**

**#include <thread>**

**#ifdef \_WIN32**

**#include <conio.h>**

**#else**

**#include <termios.h>**

**#include <unistd.h>**

**#endif**

**using namespace std;**

**// Function prototypes**

**void viewPlayers(sqlite3 \*db);**

**void editPlayer(sqlite3 \*db);**

**void removePlayer(sqlite3 \*db);**

**bool login();**

**string getHiddenInput();**

**// Main function**

**int main() {**

**// Login authentication**

**if (!login()) {**

**cout << "Too many failed login attempts. Exiting...\n";**

**return 0;**

**}**

**sqlite3 \*db;**

**int exit = sqlite3\_open("registration.db", &db); // Connect to the database**

**if (exit) {**

**cerr << "Error opening database: " << sqlite3\_errmsg(db) << endl;**

**return -1;**

**} else {**

**cout << "Database opened successfully.\n";**

**}**

**int choice;**

**do {**

**cout << "\nAdmin Menu:\n";**

**cout << "1. View Players\n";**

**cout << "2. Edit Player Information\n";**

**cout << "3. Remove Player\n";**

**cout << "4. Exit\n";**

**cout << "Enter your choice: ";**

**cin >> choice;**

**cin.ignore(); // Clear input buffer**

**switch (choice) {**

**case 1:**

**viewPlayers(db);**

**break;**

**case 2:**

**editPlayer(db);**

**break;**

**case 3:**

**removePlayer(db);**

**break;**

**case 4:**

**cout << "Exiting...\n";**

**break;**

**default:**

**cout << "Invalid choice. Please try again.\n";**

**}**

**} while (choice != 4);**

**sqlite3\_close(db);**

**return 0;**

**}**

**// Function to get password input without echo**

**string getHiddenInput() {**

**string input;**

**char ch;**

**#ifdef \_WIN32**

**while ((ch = \_getch()) != '\r') { // Enter key**

**if (ch == '\b') { // Backspace**

**if (!input.empty()) {**

**input.pop\_back();**

**cout << "\b \b";**

**}**

**} else {**

**input.push\_back(ch);**

**cout << '\*';**

**}**

**}**

**#else**

**termios oldt, newt;**

**tcgetattr(STDIN\_FILENO, &oldt);**

**newt = oldt;**

**newt.c\_lflag &= ~(ECHO);**

**tcsetattr(STDIN\_FILENO, TCSANOW, &newt);**

**getline(cin, input);**

**tcsetattr(STDIN\_FILENO, TCSANOW, &oldt);**

**#endif**

**cout << endl;**

**return input;**

**}**

**// Secure admin login function**

**bool login() {**

**const string correctUsername = "gtorres88";**

**const string correctPassword = "GFamily88!";**

**string username, password;**

**int attempts = 0;**

**const int maxAttempts = 3;**

**int lockoutTime = 30; // seconds**

**while (true) {**

**cout << "\n=== Admin Login ===\n";**

**cout << "Username: ";**

**getline(cin, username);**

**cout << "Password: ";**

**password = getHiddenInput();**

**if (username == correctUsername && password == correctPassword) {**

**cout << "Login successful. Welcome, Admin!\n";**

**return true;**

**} else {**

**cout << "Incorrect username or password.\n";**

**attempts++;**

**if (attempts == maxAttempts) {**

**cout << "Too many failed attempts. Locked out for " << lockoutTime << " seconds.\n";**

**this\_thread::sleep\_for(chrono::seconds(lockoutTime));**

**attempts++; // Go past maxAttempts to start exponential lockouts**

**} else if (attempts > maxAttempts) {**

**cout << "Still incorrect. Locked out for " << lockoutTime << " seconds.\n";**

**this\_thread::sleep\_for(chrono::seconds(lockoutTime));**

**lockoutTime \*= 2;**

**}**

**}**

**}**

**return false;**

**}**

**// Function to view all players**

**void viewPlayers(sqlite3 \*db) {**

**string query = "SELECT ROWID, Name, DOB, Guardian, Phone, Email, Division FROM Players;";**

**sqlite3\_stmt \*stmt;**

**if (sqlite3\_prepare\_v2(db, query.c\_str(), -1, &stmt, nullptr) != SQLITE\_OK) {**

**cerr << "Error executing query: " << sqlite3\_errmsg(db) << endl;**

**return;**

**}**

**cout << "\nPlayers List:\n";**

**cout << "ID | Name | DOB | Guardian | Phone | Email | Division\n";**

**cout << "---------------------------------------------------------\n";**

**while (sqlite3\_step(stmt) == SQLITE\_ROW) {**

**int id = sqlite3\_column\_int(stmt, 0);**

**const char \*name = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 1));**

**const char \*dob = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 2));**

**const char \*guardian = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 3));**

**const char \*phone = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 4));**

**const char \*email = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 5));**

**const char \*division = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 6));**

**cout << id << " | " << name << " | " << dob << " | " << guardian**

**<< " | " << phone << " | " << email << " | " << division << "\n";**

**}**

**sqlite3\_finalize(stmt);**

**}**

**// Function to edit a player's information**

**void editPlayer(sqlite3 \*db) {**

**int id;**

**string column, newValue;**

**unordered\_set<string> validFields = {"Name", "DOB", "Guardian", "Phone", "Email", "Division"};**

**cout << "Enter Player ID to edit: ";**

**cin >> id;**

**cin.ignore();**

**cout << "Enter the field to edit (Name, DOB, Guardian, Phone, Email, Division): ";**

**getline(cin, column);**

**if (validFields.find(column) == validFields.end()) {**

**cout << "Invalid field. Valid fields are: Name, DOB, Guardian, Phone, Email, Division\n";**

**return;**

**}**

**cout << "Enter the new value: ";**

**getline(cin, newValue);**

**string query = "UPDATE Players SET " + column + " = ? WHERE ROWID = ?;";**

**sqlite3\_stmt \*stmt;**

**if (sqlite3\_prepare\_v2(db, query.c\_str(), -1, &stmt, nullptr) == SQLITE\_OK) {**

**sqlite3\_bind\_text(stmt, 1, newValue.c\_str(), -1, SQLITE\_STATIC);**

**sqlite3\_bind\_int(stmt, 2, id);**

**if (sqlite3\_step(stmt) == SQLITE\_DONE) {**

**cout << "Player information updated successfully.\n";**

**} else {**

**cerr << "Error updating player information: " << sqlite3\_errmsg(db) << endl;**

**}**

**} else {**

**cerr << "Error preparing query: " << sqlite3\_errmsg(db) << endl;**

**}**

**sqlite3\_finalize(stmt);**

**}**

**// Function to remove a player**

**void removePlayer(sqlite3 \*db) {**

**int id;**

**cout << "Enter Player ID to remove: ";**

**cin >> id;**

**cin.ignore();**

**string query = "DELETE FROM Players WHERE ROWID = ?;";**

**sqlite3\_stmt \*stmt;**

**if (sqlite3\_prepare\_v2(db, query.c\_str(), -1, &stmt, nullptr) == SQLITE\_OK) {**

**sqlite3\_bind\_int(stmt, 1, id);**

**if (sqlite3\_step(stmt) == SQLITE\_DONE) {**

**cout << "Player removed successfully.\n";**

**} else {**

**cerr << "Error removing player: " << sqlite3\_errmsg(db) << std::endl;**

**}**

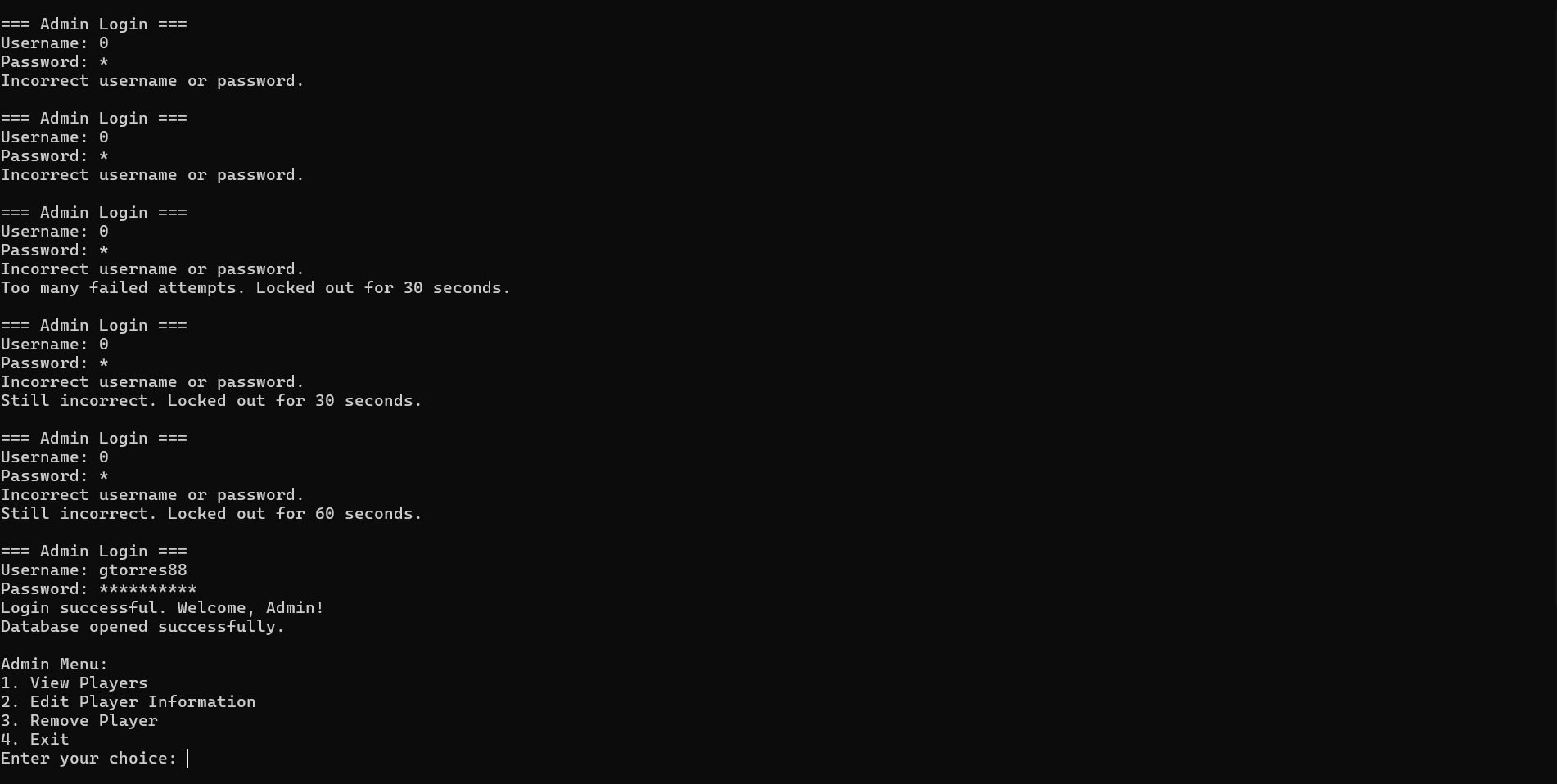
**} else {**

**cerr << "Error preparing query: " << sqlite3\_errmsg(db) << std::endl;**

**}**

**sqlite3\_finalize(stmt);**

**}**

****

**Version 5 Of Backend C++ Code**

**// -----------------------------------------------------------------------------------------------------------------------**

**// Name Gino Torres**

**// Course CMPS 499, Spring 2025**

**// Senior Project**

**//**

**// Purpose:**

**/\***

**Admin Backend Development for Senior Project:**

**Admins must log in with credentials before accessing functionality.**

**Can view, access, edit player information, including division, and remove players.**

**\*/**

**// -----------------------------------------------------------------------------------------------------------------------**

**#include <iostream>**

**#include <sqlite3.h>**

**#include <string>**

**#include <unordered\_set>**

**#include <chrono>**

**#include <thread>**

**#include <ctime>**

**#include <sstream>**

**#ifdef \_WIN32**

**#include <conio.h>**

**#else**

**#include <termios.h>**

**#include <unistd.h>**

**#endif**

**using namespace std;**

**// Function prototypes**

**void viewPlayers(sqlite3 \*db);**

**void editPlayer(sqlite3 \*db);**

**void removePlayer(sqlite3 \*db);**

**bool login();**

**string getHiddenInput();**

**// Function to calculate age as of July 31st**

**int calculateAgeOnJuly31(const string& dob) {**

**int birthYear, birthMonth, birthDay;**

**char dash;**

**istringstream iss(dob);**

**iss >> birthYear >> dash >> birthMonth >> dash >> birthDay;**

**// Get current year**

**time\_t now = time(0);**

**tm \*ltm = localtime(&now);**

**int currentYear = 1900 + ltm->tm\_year;**

**// Calculate age as of July 31**

**int age = currentYear - birthYear;**

**if (birthMonth > 7 || (birthMonth == 7 && birthDay > 31)) {**

**age--; // hasn't had birthday yet this year by July 31**

**}**

**return age;**

**}**

**// Function to determine division based on age**

**string determineDivision(int age) {**

**if (age <= 7) return "7U";**

**if (age == 8) return "8U";**

**if (age == 9) return "9U";**

**if (age == 10) return "10U";**

**if (age == 11) return "11U";**

**if (age == 12) return "12U";**

**return "13U";**

**}**

**// Main function**

**int main() {**

**// Login authentication**

**if (!login()) {**

**cout << "Too many failed login attempts. Exiting...\n";**

**return 0;**

**}**

**sqlite3 \*db;**

**int exit = sqlite3\_open("registration.db", &db); // Connect to the database**

**if (exit) {**

**cerr << "Error opening database: " << sqlite3\_errmsg(db) << endl;**

**return -1;**

**} else {**

**cout << "Database opened successfully.\n";**

**}**

**int choice;**

**do {**

**cout << "\nAdmin Menu:\n";**

**cout << "1. View Players\n";**

**cout << "2. Edit Player Information\n";**

**cout << "3. Remove Player\n";**

**cout << "4. Exit\n";**

**cout << "Enter your choice: ";**

**cin >> choice;**

**cin.ignore(); // Clear input buffer**

**switch (choice) {**

**case 1:**

**viewPlayers(db);**

**break;**

**case 2:**

**editPlayer(db);**

**break;**

**case 3:**

**removePlayer(db);**

**break;**

**case 4:**

**cout << "Exiting...\n";**

**break;**

**default:**

**cout << "Invalid choice. Please try again.\n";**

**}**

**} while (choice != 4);**

**sqlite3\_close(db);**

**return 0;**

**}**

**// Function to get password input without echo**

**string getHiddenInput() {**

**string input;**

**char ch;**

**#ifdef \_WIN32**

**while ((ch = \_getch()) != '\r') { // Enter key**

**if (ch == '\b') { // Backspace**

**if (!input.empty()) {**

**input.pop\_back();**

**cout << "\b \b";**

**}**

**} else {**

**input.push\_back(ch);**

**cout << '\*';**

**}**

**}**

**#else**

**termios oldt, newt;**

**tcgetattr(STDIN\_FILENO, &oldt);**

**newt = oldt;**

**newt.c\_lflag &= ~(ECHO);**

**tcsetattr(STDIN\_FILENO, TCSANOW, &newt);**

**getline(cin, input);**

**tcsetattr(STDIN\_FILENO, TCSANOW, &oldt);**

**#endif**

**cout << endl;**

**return input;**

**}**

**// Secure admin login function**

**bool login() {**

**const string correctUsername = "gtorres88";**

**const string correctPassword = "GFamily88!";**

**string username, password;**

**int attempts = 0;**

**const int maxAttempts = 3;**

**int lockoutTime = 30; // seconds**

**while (true) {**

**cout << "\n=== Admin Login ===\n";**

**cout << "Username: ";**

**getline(cin, username);**

**cout << "Password: ";**

**password = getHiddenInput();**

**if (username == correctUsername && password == correctPassword) {**

**cout << "Login successful. Welcome, Admin!\n";**

**return true;**

**} else {**

**cout << "Incorrect username or password.\n";**

**attempts++;**

**if (attempts == maxAttempts) {**

**cout << "Too many failed attempts. Locked out for " << lockoutTime << " seconds.\n";**

**this\_thread::sleep\_for(chrono::seconds(lockoutTime));**

**attempts++; // Go past maxAttempts to start exponential lockouts**

**} else if (attempts > maxAttempts) {**

**cout << "Still incorrect. Locked out for " << lockoutTime << " seconds.\n";**

**this\_thread::sleep\_for(chrono::seconds(lockoutTime));**

**lockoutTime \*= 2;**

**}**

**}**

**}**

**return false;**

**}**

**// Function to view all players**

**void viewPlayers(sqlite3 \*db) {**

**string query = "SELECT ROWID, Name, DOB, Guardian, Phone, Email, Division FROM Players;";**

**sqlite3\_stmt \*stmt;**

**if (sqlite3\_prepare\_v2(db, query.c\_str(), -1, &stmt, nullptr) != SQLITE\_OK) {**

**cerr << "Error executing query: " << sqlite3\_errmsg(db) << endl;**

**return;**

**}**

**cout << "\nPlayers List:\n";**

**cout << "ID | Name | DOB | Guardian | Phone | Email | Division\n";**

**cout << "---------------------------------------------------------\n";**

**while (sqlite3\_step(stmt) == SQLITE\_ROW) {**

**int id = sqlite3\_column\_int(stmt, 0);**

**const char \*name = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 1));**

**const char \*dob = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 2));**

**const char \*guardian = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 3));**

**const char \*phone = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 4));**

**const char \*email = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 5));**

**const char \*division = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 6));**

**cout << id << " | " << name << " | " << dob << " | " << guardian**

**<< " | " << phone << " | " << email << " | " << division << "\n";**

**}**

**sqlite3\_finalize(stmt);**

**}**

**// Function to edit a player's information**

**void editPlayer(sqlite3 \*db) {**

**int id;**

**string column, newValue;**

**unordered\_set<string> validFields = {"Name", "DOB", "Guardian", "Phone", "Email", "Division"};**

**cout << "Enter Player ID to edit: ";**

**cin >> id;**

**cin.ignore();**

**cout << "Enter the field to edit (Name, DOB, Guardian, Phone, Email, Division): ";**

**getline(cin, column);**

**if (validFields.find(column) == validFields.end()) {**

**cout << "Invalid field. Valid fields are: Name, DOB, Guardian, Phone, Email, Division\n";**

**return;**

**}**

**cout << "Enter the new value: ";**

**getline(cin, newValue);**

**string query;**

**sqlite3\_stmt \*stmt;**

**if (column == "DOB") {**

**// Update DOB**

**query = "UPDATE Players SET DOB = ? WHERE ROWID = ?;";**

**if (sqlite3\_prepare\_v2(db, query.c\_str(), -1, &stmt, nullptr) == SQLITE\_OK) {**

**sqlite3\_bind\_text(stmt, 1, newValue.c\_str(), -1, SQLITE\_STATIC);**

**sqlite3\_bind\_int(stmt, 2, id);**

**if (sqlite3\_step(stmt) == SQLITE\_DONE) {**

**cout << "DOB updated.\n";**

**} else {**

**cerr << "Error updating DOB: " << sqlite3\_errmsg(db) << endl;**

**sqlite3\_finalize(stmt);**

**return;**

**}**

**sqlite3\_finalize(stmt);**

**}**

**// Auto-update Division based on new DOB**

**int age = calculateAgeOnJuly31(newValue);**

**string division = determineDivision(age);**

**query = "UPDATE Players SET Division = ? WHERE ROWID = ?;";**

**if (sqlite3\_prepare\_v2(db, query.c\_str(), -1, &stmt, nullptr) == SQLITE\_OK) {**

**sqlite3\_bind\_text(stmt, 1, division.c\_str(), -1, SQLITE\_STATIC);**

**sqlite3\_bind\_int(stmt, 2, id);**

**if (sqlite3\_step(stmt) == SQLITE\_DONE) {**

**cout << "Division auto-updated based on DOB: " << division << "\n";**

**} else {**

**cerr << "Error updating Division: " << sqlite3\_errmsg(db) << endl;**

**}**

**sqlite3\_finalize(stmt);**

**}**

**} else {**

**// Regular manual update**

**query = "UPDATE Players SET " + column + " = ? WHERE ROWID = ?;";**

**if (sqlite3\_prepare\_v2(db, query.c\_str(), -1, &stmt, nullptr) == SQLITE\_OK) {**

**sqlite3\_bind\_text(stmt, 1, newValue.c\_str(), -1, SQLITE\_STATIC);**

**sqlite3\_bind\_int(stmt, 2, id);**

**if (sqlite3\_step(stmt) == SQLITE\_DONE) {**

**cout << column << " updated successfully.\n";**

**} else {**

**cerr << "Error updating " << column << ": " << sqlite3\_errmsg(db) << endl;**

**}**

**} else {**

**cerr << "Error preparing query: " << sqlite3\_errmsg(db) << endl;**

**}**

**sqlite3\_finalize(stmt);**

**}**

**}**

**// Function to remove a player**

**void removePlayer(sqlite3 \*db) {**

**int id;**

**cout << "Enter Player ID to remove: ";**

**cin >> id;**

**cin.ignore();**

**string query = "DELETE FROM Players WHERE ROWID = ?;";**

**sqlite3\_stmt \*stmt;**

**if (sqlite3\_prepare\_v2(db, query.c\_str(), -1, &stmt, nullptr) == SQLITE\_OK) {**

**sqlite3\_bind\_int(stmt, 1, id);**

**if (sqlite3\_step(stmt) == SQLITE\_DONE) {**

**cout << "Player removed successfully.\n";**

**} else {**

**cerr << "Error removing player: " << sqlite3\_errmsg(db) << std::endl;**

**}**

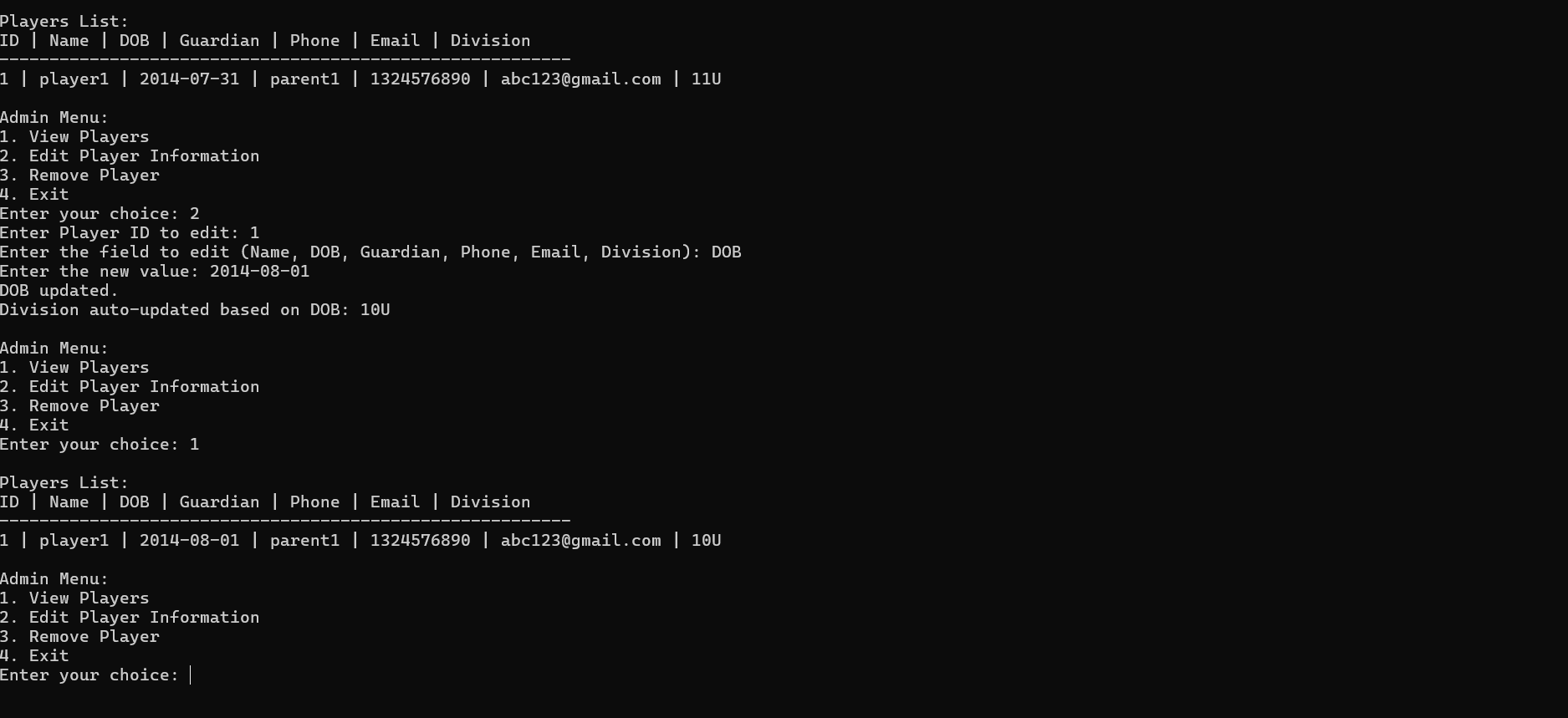
**} else {**

**cerr << "Error preparing query: " << sqlite3\_errmsg(db) << std::endl;**

**}**

**sqlite3\_finalize(stmt);**

**}**

****

**Version 6 Of Backend C++ Code**

**// -----------------------------------------------------------------------------------------------------------------------**

**// Name Gino Torres**

**// Course CMPS 499, Spring 2025**

**// Senior Project**

**//**

**// Purpose:**

**/\***

**Admin Backend Development for Senior Project:**

**Admins must log in with credentials before accessing functionality.**

**Can view, access, edit player information, including division, remove players, and create a CSV file for Offline/Coaches Use.**

**\*/**

**// -----------------------------------------------------------------------------------------------------------------------**

**#include <iostream>**

**#include <sqlite3.h>**

**#include <string>**

**#include <unordered\_set>**

**#include <chrono>**

**#include <thread>**

**#include <ctime>**

**#include <sstream>**

**#include <fstream>**

**#ifdef \_WIN32**

**#include <conio.h>**

**#else**

**#include <termios.h>**

**#include <unistd.h>**

**#endif**

**using namespace std;**

**// Function prototypes**

**void viewPlayers(sqlite3 \*db);**

**void editPlayer(sqlite3 \*db);**

**void removePlayer(sqlite3 \*db);**

**void exportPlayersToCSV(sqlite3 \*db);**

**bool login();**

**string getHiddenInput();**

**// Function to calculate age as of July 31st**

**int calculateAgeOnJuly31(const string& dob) {**

**int birthYear, birthMonth, birthDay;**

**char dash;**

**istringstream iss(dob);**

**iss >> birthYear >> dash >> birthMonth >> dash >> birthDay;**

**// Get current year**

**time\_t now = time(0);**

**tm \*ltm = localtime(&now);**

**int currentYear = 1900 + ltm->tm\_year;**

**// Calculate age as of July 31**

**int age = currentYear - birthYear;**

**if (birthMonth > 7 || (birthMonth == 7 && birthDay > 31)) {**

**age--; // hasn't had birthday yet this year by July 31**

**}**

**return age;**

**}**

**// Function to determine division based on age**

**string determineDivision(int age) {**

**if (age <= 7) return "7U";**

**if (age == 8) return "8U";**

**if (age == 9) return "9U";**

**if (age == 10) return "10U";**

**if (age == 11) return "11U";**

**if (age == 12) return "12U";**

**return "13U";**

**}**

**// Main function**

**int main() {**

**// Login authentication**

**if (!login()) {**

**cout << "Too many failed login attempts. Exiting...\n";**

**return 0;**

**}**

**sqlite3 \*db;**

**int exit = sqlite3\_open("registration.db", &db); // Connect to the database**

**if (exit) {**

**cerr << "Error opening database: " << sqlite3\_errmsg(db) << endl;**

**return -1;**

**} else {**

**cout << "Database opened successfully.\n";**

**}**

**int choice;**

**do {**

**cout << "\nAdmin Menu:\n";**

**cout << "1. View Players\n";**

**cout << "2. Edit Player Information\n";**

**cout << "3. Remove Player\n";**

**cout << "4. Export Players to CSV\n";**

**cout << "5. Exit\n";**

**cout << "Enter your choice: ";**

**cin >> choice;**

**cin.ignore(); // Clear input buffer**

**switch (choice) {**

**case 1:**

**viewPlayers(db);**

**break;**

**case 2:**

**editPlayer(db);**

**break;**

**case 3:**

**removePlayer(db);**

**break;**

**case 4:**

**exportPlayersToCSV(db);**

**break;**

**case 5:**

**cout << "Exiting...\n";**

**break;**

**default:**

**cout << "Invalid choice. Please try again.\n";**

**}**

**} while (choice != 5);**

**sqlite3\_close(db);**

**return 0;**

**}**

**// Function to get password input without echo**

**string getHiddenInput() {**

**string input;**

**char ch;**

**#ifdef \_WIN32**

**while ((ch = \_getch()) != '\r') { // Enter key**

**if (ch == '\b') { // Backspace**

**if (!input.empty()) {**

**input.pop\_back();**

**cout << "\b \b";**

**}**

**} else {**

**input.push\_back(ch);**

**cout << '\*';**

**}**

**}**

**#else**

**termios oldt, newt;**

**tcgetattr(STDIN\_FILENO, &oldt);**

**newt = oldt;**

**newt.c\_lflag &= ~(ECHO);**

**tcsetattr(STDIN\_FILENO, TCSANOW, &newt);**

**getline(cin, input);**

**tcsetattr(STDIN\_FILENO, TCSANOW, &oldt);**

**#endif**

**cout << endl;**

**return input;**

**}**

**// Secure admin login function**

**bool login() {**

**const string correctUsername = "gtorres88";**

**const string correctPassword = "GFamily88!";**

**string username, password;**

**int attempts = 0;**

**const int maxAttempts = 3;**

**int lockoutTime = 30; // seconds**

**while (true) {**

**cout << "\n=== Admin Login ===\n";**

**cout << "Username: ";**

**getline(cin, username);**

**cout << "Password: ";**

**password = getHiddenInput();**

**if (username == correctUsername && password == correctPassword) {**

**cout << "Login successful. Welcome, Admin!\n";**

**return true;**

**} else {**

**cout << "Incorrect username or password.\n";**

**attempts++;**

**if (attempts == maxAttempts) {**

**cout << "Too many failed attempts. Locked out for " << lockoutTime << " seconds.\n";**

**this\_thread::sleep\_for(chrono::seconds(lockoutTime));**

**attempts++; // Go past maxAttempts to start exponential lockouts**

**} else if (attempts > maxAttempts) {**

**cout << "Still incorrect. Locked out for " << lockoutTime << " seconds.\n";**

**this\_thread::sleep\_for(chrono::seconds(lockoutTime));**

**lockoutTime \*= 2;**

**}**

**}**

**}**

**return false;**

**}**

**// Function to view all players**

**void viewPlayers(sqlite3 \*db) {**

**string query = "SELECT ROWID, Name, DOB, Guardian, Phone, Email, Division FROM Players;";**

**sqlite3\_stmt \*stmt;**

**if (sqlite3\_prepare\_v2(db, query.c\_str(), -1, &stmt, nullptr) != SQLITE\_OK) {**

**cerr << "Error executing query: " << sqlite3\_errmsg(db) << endl;**

**return;**

**}**

**cout << "\nPlayers List:\n";**

**cout << "ID | Name | DOB | Guardian | Phone | Email | Division\n";**

**cout << "---------------------------------------------------------\n";**

**while (sqlite3\_step(stmt) == SQLITE\_ROW) {**

**int id = sqlite3\_column\_int(stmt, 0);**

**const char \*name = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 1));**

**const char \*dob = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 2));**

**const char \*guardian = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 3));**

**const char \*phone = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 4));**

**const char \*email = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 5));**

**const char \*division = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 6));**

**cout << id << " | " << name << " | " << dob << " | " << guardian**

**<< " | " << phone << " | " << email << " | " << division << "\n";**

**}**

**sqlite3\_finalize(stmt);**

**}**

**// Function to edit a player's information**

**void editPlayer(sqlite3 \*db) {**

**int id;**

**string column, newValue;**

**unordered\_set<string> validFields = {"Name", "DOB", "Guardian", "Phone", "Email", "Division"};**

**cout << "Enter Player ID to edit: ";**

**cin >> id;**

**cin.ignore();**

**cout << "Enter the field to edit (Name, DOB, Guardian, Phone, Email, Division): ";**

**getline(cin, column);**

**if (validFields.find(column) == validFields.end()) {**

**cout << "Invalid field. Valid fields are: Name, DOB, Guardian, Phone, Email, Division\n";**

**return;**

**}**

**cout << "Enter the new value: ";**

**getline(cin, newValue);**

**string query;**

**sqlite3\_stmt \*stmt;**

**if (column == "DOB") {**

**// Update DOB**

**query = "UPDATE Players SET DOB = ? WHERE ROWID = ?;";**

**if (sqlite3\_prepare\_v2(db, query.c\_str(), -1, &stmt, nullptr) == SQLITE\_OK) {**

**sqlite3\_bind\_text(stmt, 1, newValue.c\_str(), -1, SQLITE\_STATIC);**

**sqlite3\_bind\_int(stmt, 2, id);**

**if (sqlite3\_step(stmt) == SQLITE\_DONE) {**

**cout << "DOB updated.\n";**

**} else {**

**cerr << "Error updating DOB: " << sqlite3\_errmsg(db) << endl;**

**sqlite3\_finalize(stmt);**

**return;**

**}**

**sqlite3\_finalize(stmt);**

**}**

**// Auto-update Division based on new DOB**

**int age = calculateAgeOnJuly31(newValue);**

**string division = determineDivision(age);**

**query = "UPDATE Players SET Division = ? WHERE ROWID = ?;";**

**if (sqlite3\_prepare\_v2(db, query.c\_str(), -1, &stmt, nullptr) == SQLITE\_OK) {**

**sqlite3\_bind\_text(stmt, 1, division.c\_str(), -1, SQLITE\_STATIC);**

**sqlite3\_bind\_int(stmt, 2, id);**

**if (sqlite3\_step(stmt) == SQLITE\_DONE) {**

**cout << "Division auto-updated based on DOB: " << division << "\n";**

**} else {**

**cerr << "Error updating Division: " << sqlite3\_errmsg(db) << endl;**

**}**

**sqlite3\_finalize(stmt);**

**}**

**} else {**

**// Regular manual update**

**query = "UPDATE Players SET " + column + " = ? WHERE ROWID = ?;";**

**if (sqlite3\_prepare\_v2(db, query.c\_str(), -1, &stmt, nullptr) == SQLITE\_OK) {**

**sqlite3\_bind\_text(stmt, 1, newValue.c\_str(), -1, SQLITE\_STATIC);**

**sqlite3\_bind\_int(stmt, 2, id);**

**if (sqlite3\_step(stmt) == SQLITE\_DONE) {**

**cout << column << " updated successfully.\n";**

**} else {**

**cerr << "Error updating " << column << ": " << sqlite3\_errmsg(db) << endl;**

**}**

**} else {**

**cerr << "Error preparing query: " << sqlite3\_errmsg(db) << endl;**

**}**

**sqlite3\_finalize(stmt);**

**}**

**}**

**// Function to remove a player**

**void removePlayer(sqlite3 \*db) {**

**int id;**

**cout << "Enter Player ID to remove: ";**

**cin >> id;**

**cin.ignore();**

**string query = "DELETE FROM Players WHERE ROWID = ?;";**

**sqlite3\_stmt \*stmt;**

**if (sqlite3\_prepare\_v2(db, query.c\_str(), -1, &stmt, nullptr) == SQLITE\_OK) {**

**sqlite3\_bind\_int(stmt, 1, id);**

**if (sqlite3\_step(stmt) == SQLITE\_DONE) {**

**cout << "Player removed successfully.\n";**

**} else {**

**cerr << "Error removing player: " << sqlite3\_errmsg(db) << std::endl;**

**}**

**} else {**

**cerr << "Error preparing query: " << sqlite3\_errmsg(db) << std::endl;**

**}**

**sqlite3\_finalize(stmt);**

**}**

**// Function to export players to CSV File**

**void exportPlayersToCSV(sqlite3 \*db) {**

**string query = "SELECT ROWID, Name, DOB, Guardian, Phone, Email, Division FROM Players;";**

**sqlite3\_stmt \*stmt;**

**if (sqlite3\_prepare\_v2(db, query.c\_str(), -1, &stmt, nullptr) != SQLITE\_OK) {**

**cerr << "Error preparing export query: " << sqlite3\_errmsg(db) << endl;**

**return;**

**}**

**ofstream outFile("players\_export.csv");**

**if (!outFile.is\_open()) {**

**cerr << "Failed to open file for export.\n";**

**sqlite3\_finalize(stmt);**

**return;**

**}**

**// Write CSV headers**

**outFile << "ID,Name,DOB,Guardian,Phone,Email,Division\n";**

**while (sqlite3\_step(stmt) == SQLITE\_ROW) {**

**int id = sqlite3\_column\_int(stmt, 0);**

**const char \*name = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 1));**

**const char \*dob = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 2));**

**const char \*guardian = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 3));**

**const char \*phone = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 4));**

**const char \*email = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 5));**

**const char \*division = reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt, 6));**

**// Write row to CSV**

**outFile << id << ","**

**<< (name ? name : "") << ","**

**<< (dob ? dob : "") << ","**

**<< (guardian ? guardian : "") << ","**

**<< (phone ? phone : "") << ","**

**<< (email ? email : "") << ","**

**<< (division ? division : "") << "\n";**

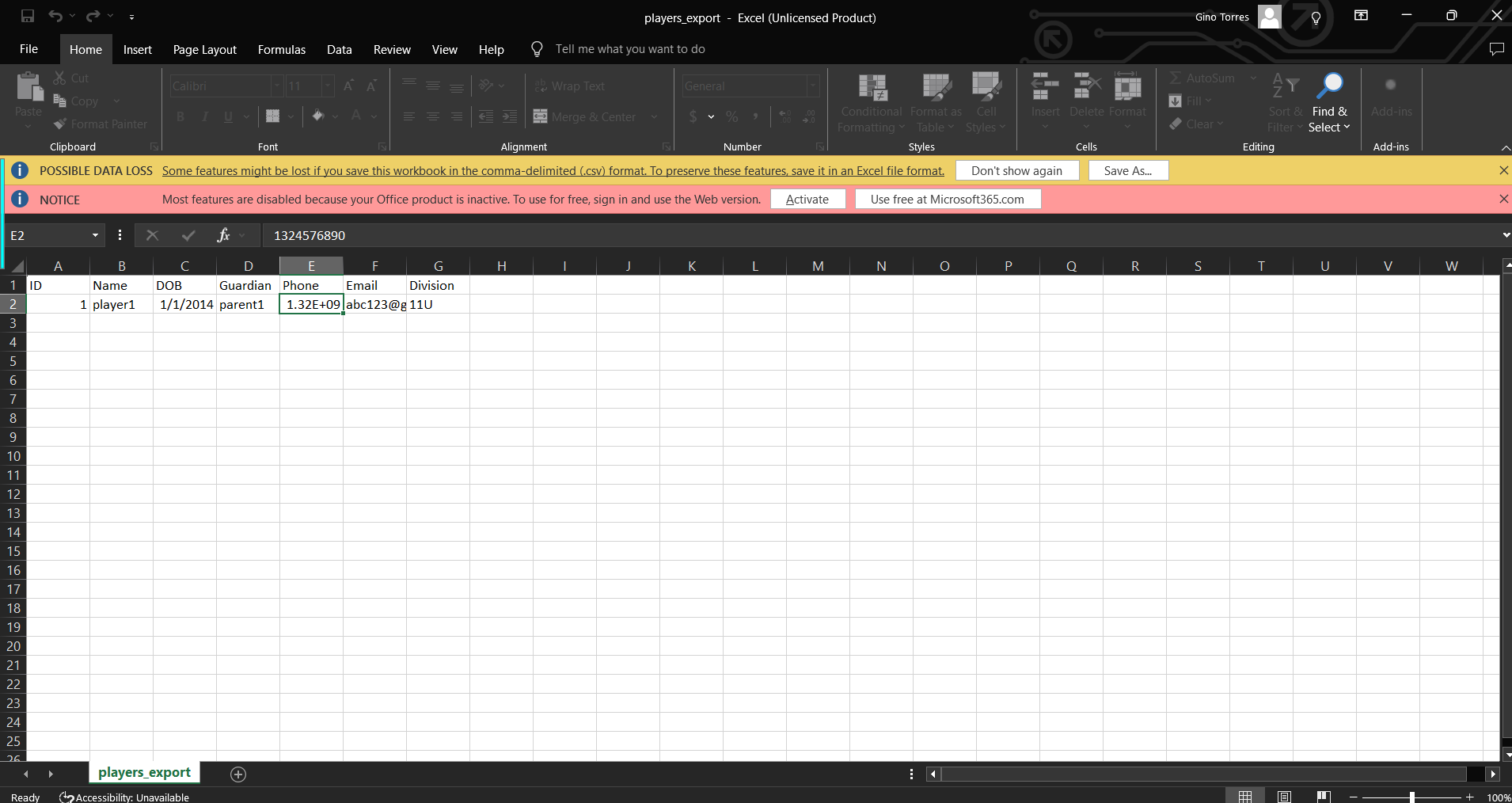
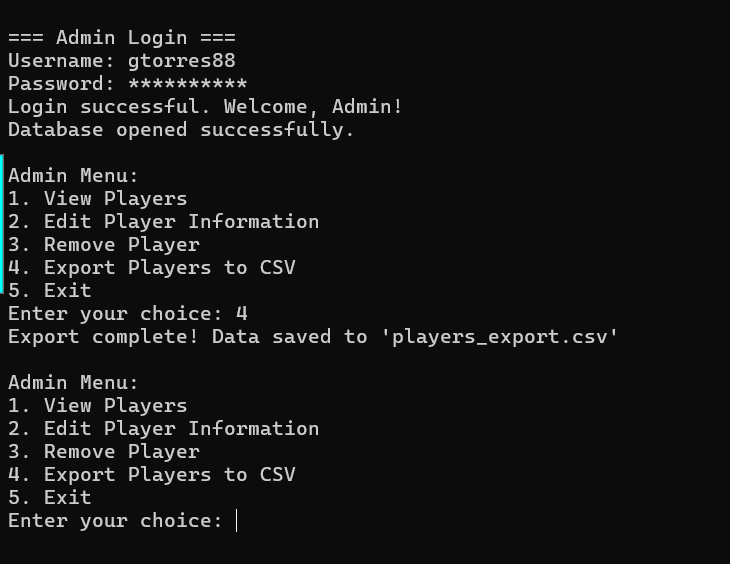
**}**

**sqlite3\_finalize(stmt);**

**outFile.close();**

**cout << "Export complete! Data saved to 'players\_export.csv'\n";**

**}**

****