Programming Language Research Practical Project 4

Algonquin College
School of Advanced Technology
Computer Programming

Onur Önel
Mazin Abou-Seido

23F_CST8333_360

Submitted December 01, 2023

Evidence of Learning

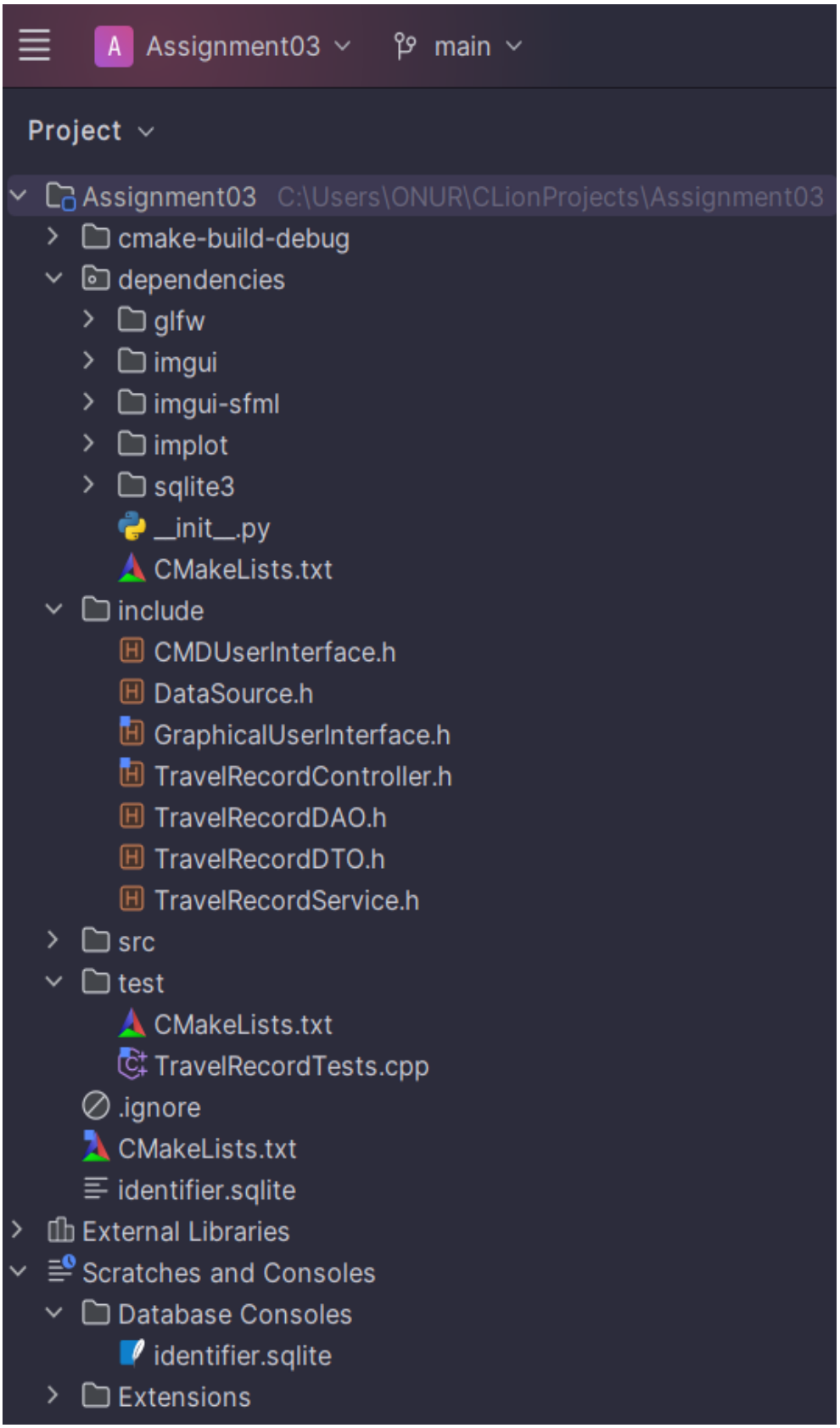
In my last project, I established a foundational connection between SQLite3 and the project using a DataSource class. This bridge facilitated data management and interaction. Within the TravelRecordDTO class, I defined variables tailored to both the database schema and the specific requirements of the project. This step was critical in aligning the application's data structure with the SQLite3. Further, I developed functions that utilize these defined variables, primarily for constructing SQL prepared statements. These functions, intricately linked to the SQLite3 datasource, form the core of the application's data handling capabilities. Additionally, I implemented a command-line interface (CLI) in previous assignment that effectively demonstrates the CRUD (Create, Read, Update, Delete) operations. This interface provides easier way to interact with and visualize the data manipulations directly in the terminal.

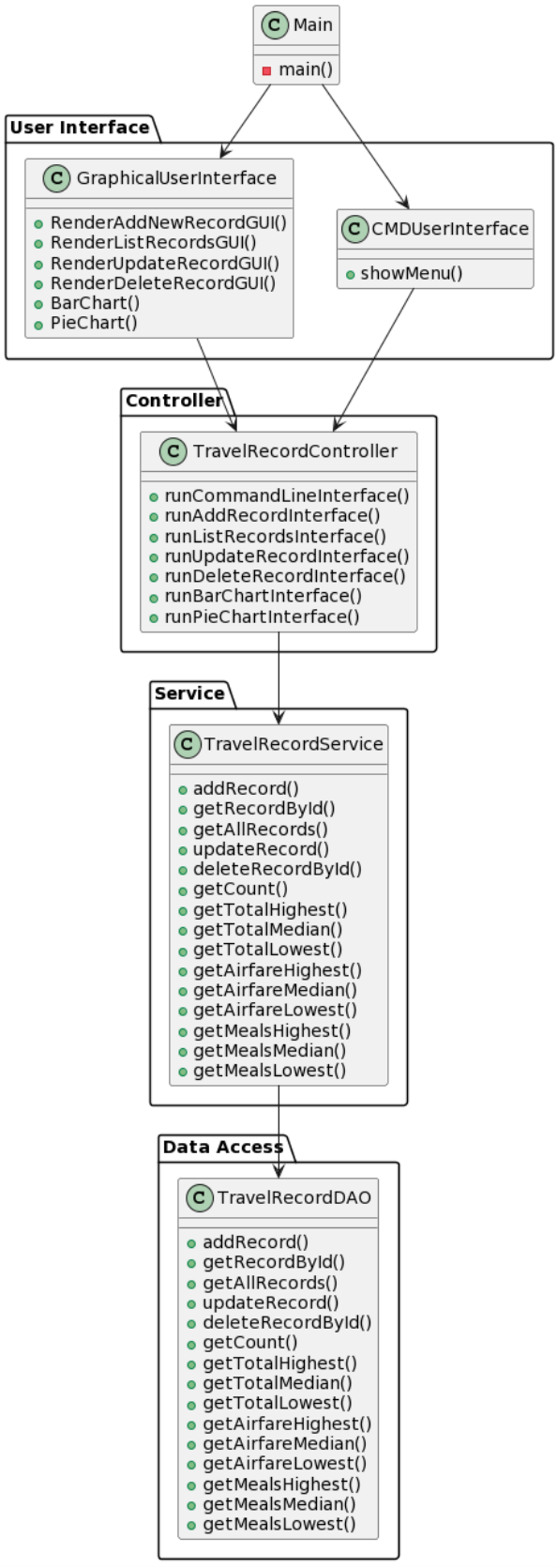
To expand my skills beyond backend development, I ventured into graphical user interfaces by learning IMGUI, IMPLOT, and GLFW, particularly in the context of OpenGL. This learning journey influenced from the requirements of Practical Project 04, which demonstrates data visualization through bar charts and graphs. I successfully integrated these visual elements into the project.

This experience not only increased my skills in C++ GUI applications but also deepened my understanding of essential design patterns like MVC (Model-View-Controller) and advanced C++ language concepts. This journey of learning and implementation was immensely rewarding

References:

1. Ocornut. (n.d.).*DEAR IMGUI: Bloat-free graphical user interface for C++ with minimal dependencies*. GitHub. https://github.com/ocornut/imgui
2. Epezent. (n.d.). *Epezent/Implot: Immediate mode plotting*. GitHub. https://github.com/epezent/implot
3. *Documentation*. GLFW. (n.d.). https://www.glfw.org/documentation.html
4. Doxygen. (n.d.). https://www.doxygen.nl/


Assignment03
main
Project
Assignment03 C:\Users\ONUR\CLionProjects\Assignment03
cmake-build-debug
dependencies
glfw
imgui
imgui-sfml
implot
sqlite3
__init__.py
CMakeLists.txt
include
CMDUserInterface.h
DataSource.h
GraphicalUserInterface.h
TravelRecordController.h
TravelRecordDAO.h
TravelRecordDTO.h
TravelRecordService.h
src
test
CMakeLists.txt
TravelRecordTests.cpp
.ignore
CMakeLists.txt
identifier.sqlite
External Libraries
Scratches and Consoles
Database Consoles
identifier.sqlite
Extensions

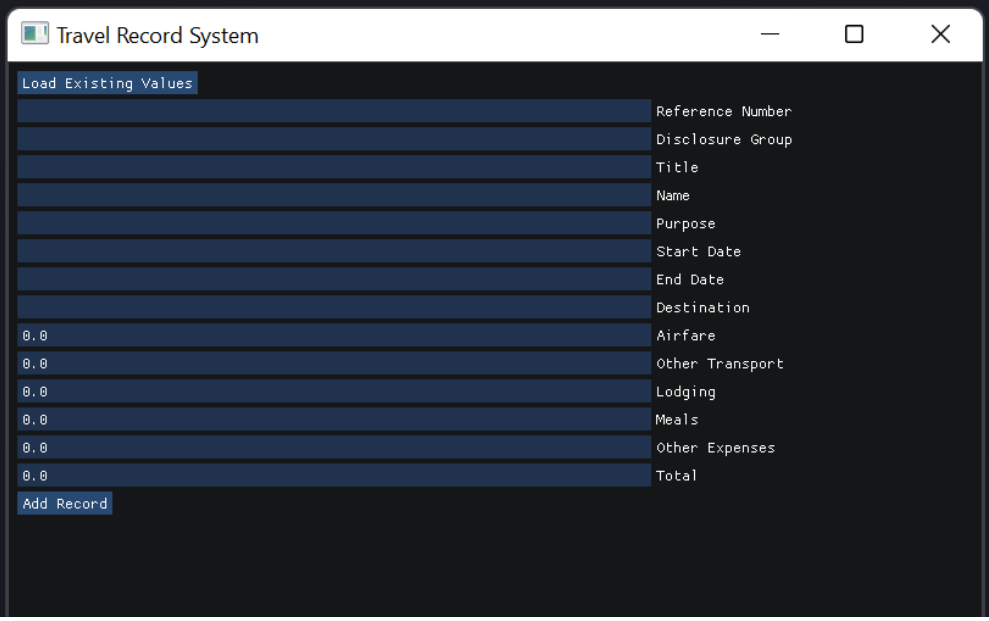

Main
main()
User Interface
GraphicalUserInterface
RenderAddNewRecordGUI()
RenderListRecordsGUI()
RenderUpdateRecordGUI()
RenderDeleteRecordGUI()
BarChart()
PieChart()
CMDUserInterface
showMenu()
Controller
TravelRecordController
runCommandLineInterface()
runAddRecordInterface()
runListRecordsInterface()
runUpdateRecordInterface()
runDeleteRecordInterface()
runBarChartInterface()
runPieChartInterface()
Service
TravelRecordService
addRecord()
getRecordById()
getAllRecords()
updateRecord()
deleteRecordById()
getCount()
getTotalHighest()
getTotalMedian()
getTotalLowest()
getAirfareHighest()
getAirfareMedian()
getAirfareLowest()
getMealsHighest()
getMealsMedian()
getMealsLowest()
Data Access
TravelRecordDAO
addRecord()
getRecordById()
getAllRecords()
updateRecord()
deleteRecordById()
getCount()
getTotalHighest()
getTotalMedian()
getTotalLowest()
getAirfareHighest()
getAirfareMedian()
getAirfareLowest()
getMealsHighest()
getMealsMedian()
getMealsLowest()

Program Demonstration via Screen Shots

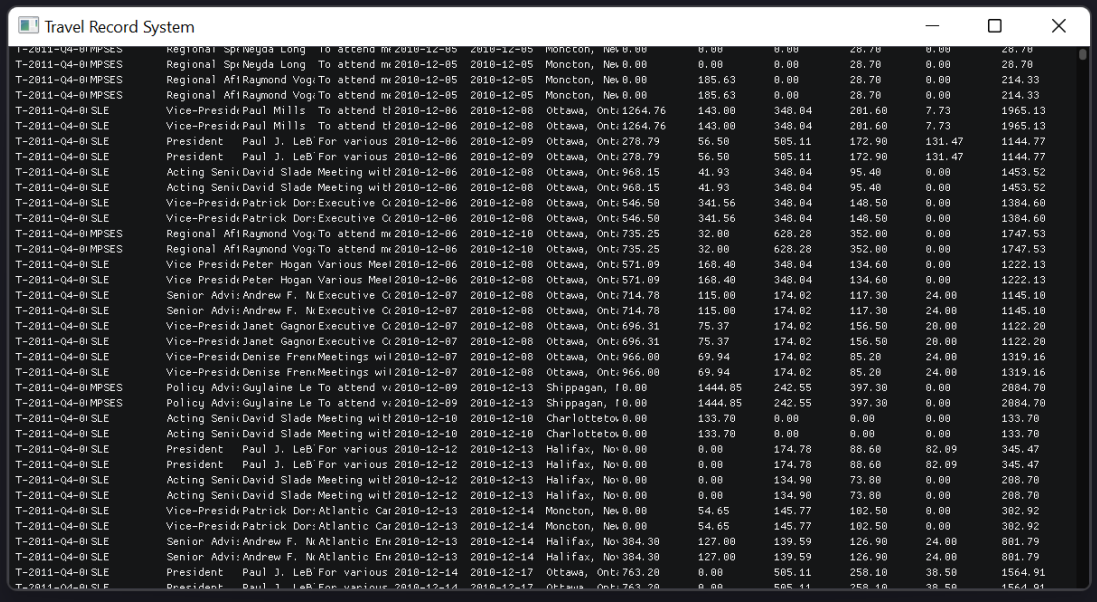
```
*******************************************
*     Welcome to Database Reader App!     *
*        ( Travel Record System )         *
*******************************************

By Onur Onel 041074824

Choose Action:
1. CMD Mode
2. ADD Record
3. LIST Records
4. UPDATE Record
5. DELETE Record
6. Bar Chart
7. Pie Chart
Enter your choice:
```

```
Enter your choice:1


Please select an option:

1. List all records
2. Get record by ID
3. Add a new record
4. Update a record
5. Delete a record
6. Show record count
7. Exit

Enter your choice:6
 Total record count: 10016
```

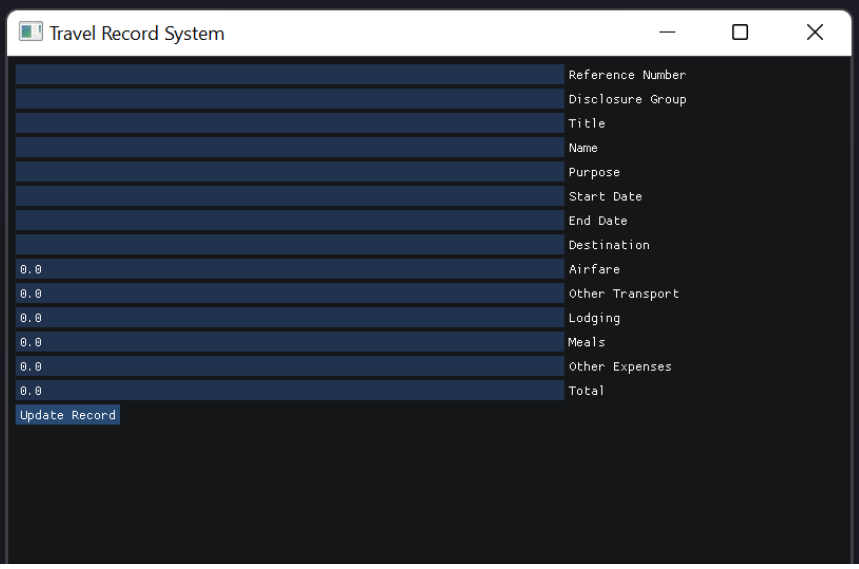
```
****************************************
*     Welcome to Database Reader App!     *
*        ( Travel Record System )         *
****************************************

By Onur Onel 041074824

Choose Action:
1. CMD Mode
2. ADD Record
3. LIST Records
4. UPDATE Record
5. DELETE Record
6. Bar Chart
7. Pie Chart
Enter your choice:2
```



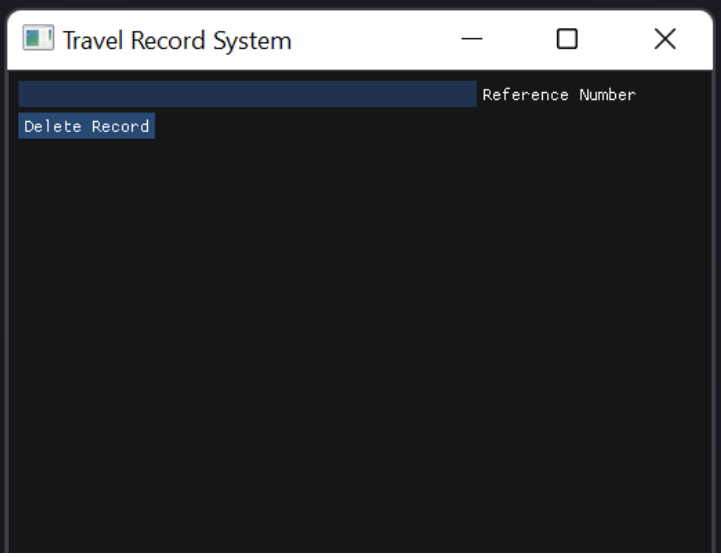

```
****************************************
*     Welcome to Database Reader App!     *
*        ( Travel Record System )         *
****************************************

By Onur Onel 041074824

Choose Action:
1. CMD Mode
2. ADD Record
3. LIST Records
4. UPDATE Record
5. DELETE Record
6. Bar Chart
7. Pie Chart
Enter your choice:3
```



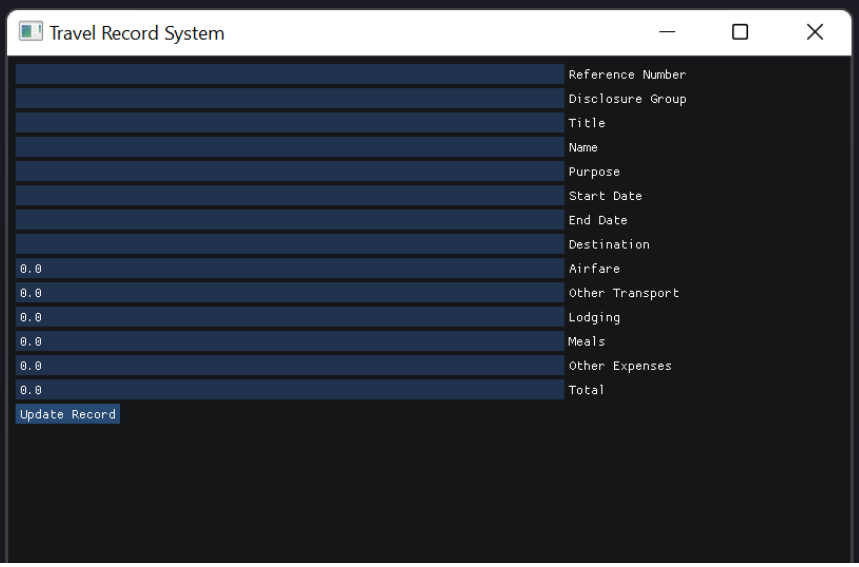
```
****************************************
*     Welcome to Database Reader App!     *
*        ( Travel Record System )         *
****************************************

By Onur Onel 041074824

Choose Action:
1. CMD Mode
2. ADD Record
3. LIST Records
4. UPDATE Record
5. DELETE Record
6. Bar Chart
7. Pie Chart
Enter your choice:4
```



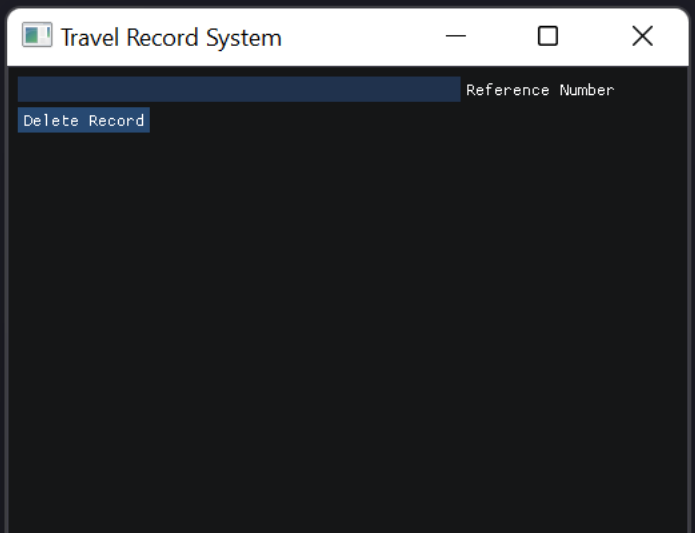

```
****************************************
*     Welcome to Database Reader App!     *
*        ( Travel Record System )         *
****************************************

By Onur Onel 041074824

Choose Action:
1. CMD Mode
2. ADD Record
3. LIST Records
4. UPDATE Record
5. DELETE Record
6. Bar Chart
7. Pie Chart
Enter your choice:5
```



```
****************************************
*     Welcome to Database Reader App!    *
*        ( Travel Record System )        *
****************************************

By Onur Onel 041074824

Choose Action:
1. CMD Mode
2. ADD Record
3. LIST Records
4. UPDATE Record
5. DELETE Record
6. Bar Chart
7. Pie Chart
Enter your choice:4
```

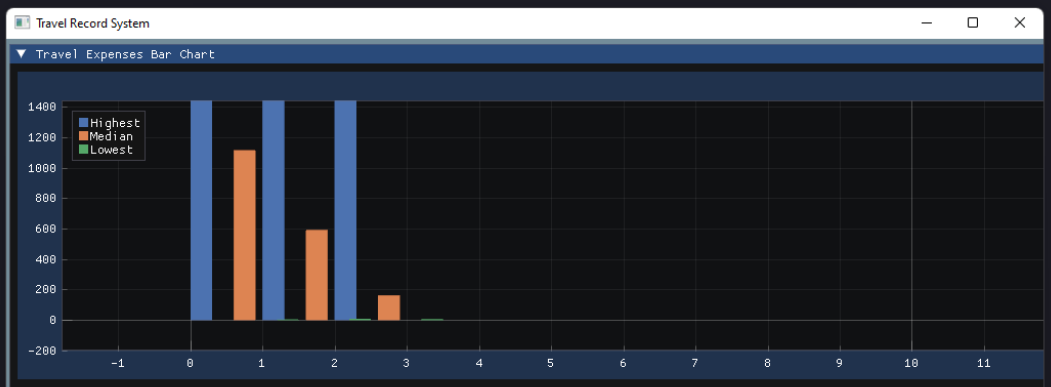



```
*********************************************
*     Welcome to Database Reader App!     *
*        ( Travel Record System )         *
*********************************************

By Onur Onel 041074824

Choose Action:
1. CMD Mode
2. ADD Record
3. LIST Records
4. UPDATE Record
5. DELETE Record
6. Bar Chart
7. Pie Chart
Enter your choice:5
```

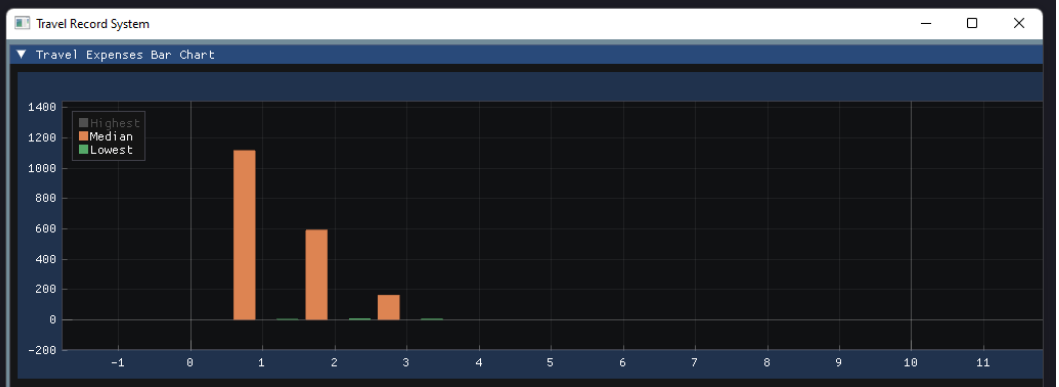


```
*****************************************
*     Welcome to Database Reader App!     *
*        ( Travel Record System )         *
*****************************************

By Onur Onel 041074824

Choose Action:
1. CMD Mode
2. ADD Record
3. LIST Records
4. UPDATE Record
5. DELETE Record
6. Bar Chart
7. Pie Chart
Enter your choice:6
```

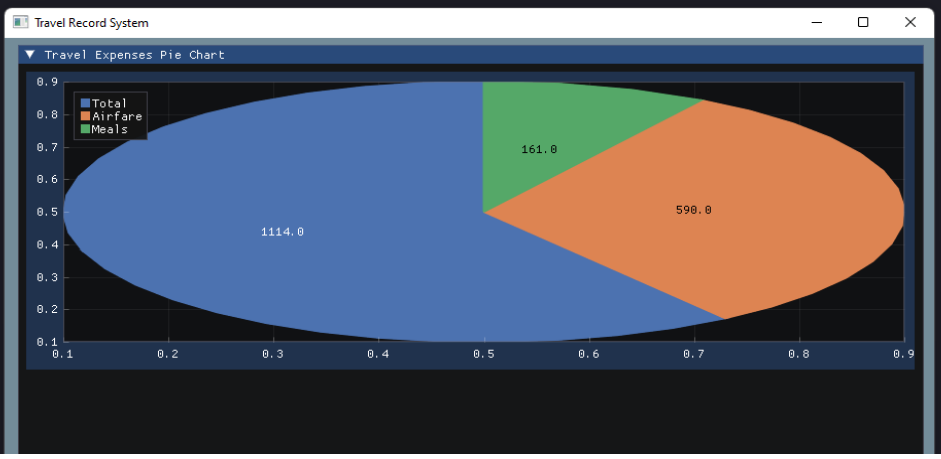

```
*****************************************
*     Welcome to Database Reader App!     *
*        ( Travel Record System )         *
*****************************************

By Onur Onel 041074824

Choose Action:
1. CMD Mode
2. ADD Record
3. LIST Records
4. UPDATE Record
5. DELETE Record
6. Bar Chart
7. Pie Chart
Enter your choice:6
```




```
*****************************************
*     Welcome to Database Reader App!     *
*        ( Travel Record System )         *
*****************************************

By Onur Onel 041074824

Choose Action:
1. CMD Mode
2. ADD Record
3. LIST Records
4. UPDATE Record
5. DELETE Record
6. Bar Chart
7. Pie Chart
Enter your choice:6
```



```
*****************************************
*     Welcome to Database Reader App!     *
*        ( Travel Record System )         *
*****************************************

By Onur Onel 041074824

Choose Action:
1. CMD Mode
2. ADD Record
3. LIST Records
4. UPDATE Record
5. DELETE Record
6. Bar Chart
7. Pie Chart
Enter your choice:7
```



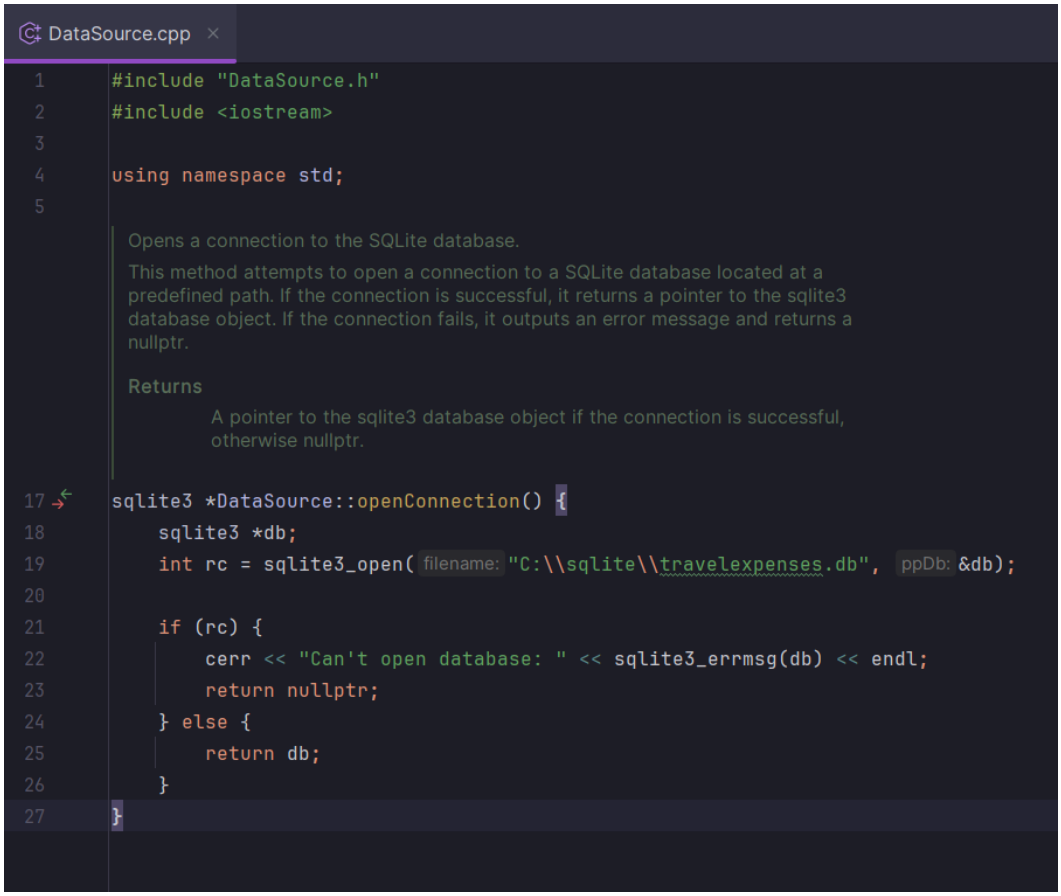
ImPlot

ImPlot is an immediate mode, GPU accelerated plotting library for Dear ImGui. It aims to provide a first-class API that ImGui fans will love. ImPlot is well suited for visualizing program data in real-time or creating interactive plots, and requires minimal code to integrate. Just like ImGui, it does not burden the end user with GUI state management, avoids STL containers and C++ headers, and has no external dependencies except for ImGui itself.

Source Code Commenting Example (Doxygen)

DataSource.cpp

```cpp
#include "DataSource.h"
#include <iostream>

using namespace std;

/**
 * @brief Opens a connection to the SQLite database.
 *
 * This method attempts to open a connection to a SQLite database located at a
 * predefined path. If the connection is successful, it returns a pointer to the
 * sqlite3 database object. If the connection fails, it outputs an error message
 * and returns a nullptr.
 *
 * @return A pointer to the sqlite3 database object if the connection is successful,
 *         otherwise nullptr.
 */
sqlite3 *DataSource::openConnection() {
    sqlite3 *db;
    int rc = sqlite3_open( filename: "C:\\sqlite\\travelexpenses.db",  ppDb: &db);

    if (rc) {
        cerr << "Can't open database: " << sqlite3_errmsg(db) << endl;
        return nullptr;
    } else {
        return db;
    }
}
```



```cpp
#include "../include/TravelRecordService.h"
#include <stdexcept>

using namespace std;

/**
 * Constructor for TravelRecordService.
 *
 * Parameters
 *         dao A reference to TravelRecordDAO object.
 */
TravelRecordService::TravelRecordService(const TravelRecordDAO &dao) : dao(dao) {
}

/**
 * Retrieves all travel records.
 *
 * Returns
 *         A vector of TravelRecordDTO objects.
 */
vector<TravelRecordDTO> TravelRecordService::getAllRecords() {
    return dao.getAllRecords();
}

/**
 * Gets a travel record by its ID.
 *
 * Parameters
 *         id The ID of the travel record.
 *
 * Returns
 *         A TravelRecordDTO object.
 */
TravelRecordDTO TravelRecordService::getRecordById(const std::string &id) {
    return dao.getRecordById(id);
}

/**
 * Adds a new travel record.
 *
 * Parameters
 *         record A TravelRecordDTO object representing the new record.
 */
void TravelRecordService::addRecord(const TravelRecordDTO &record) {
    dao.addRecord(record);
}

/**
 * Updates an existing travel record.
 *
 * Parameters
 *         record A TravelRecordDTO object with updated information.
```

As a user of CLion integrated with Doxygen, I appreciate the seamless combination of powerful IDE functionalities and comprehensive documentation capabilities that significantly enhance my coding efficiency and documentation quality.

Source Code

```cpp
#include <limits>
#include "TravelRecordDAO.h"
#include "TravelRecordService.h"
#include "CMDUserInterface.h"
#include "TravelRecordController.h"

using namespace std;

/**
 * @brief Displays the welcome message for the application.
 */
void displayWelcomeMessage() {
    cout << "\n*****************************************" << endl
         << "*     Welcome to Database Reader App!     *" << endl
         << "*         ( Travel Record System )        *" << endl
         << "*****************************************" << endl
         << "\nBy Onur Onel 041074824\n" << endl;
}

/**
 * @brief Prompts the user to choose an action and retrieves their choice.
 * @return An integer representing the user's choice.
 */
int getUserChoice() {
    cout << "Choose Action: \n"
         << "1. CMD Mode \n"
         << "2. ADD Record \n"
         << "3. LIST Records \n"
         << "4. UPDATE Record \n"
         << "5. DELETE Record \n"
         << "6. Bar Chart \n"
         << "7. Pie Chart \n"
         << "Enter your choice: ";
    int choice;
    while (!(cin >> choice)) {
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        cout << "Invalid input. Please enter a number: ";
    }
    return choice;
}

/**
 * @brief The main function of the application.
 * @return Integer 0 upon successful execution.
 */
int main() {
    displayWelcomeMessage();
    int mode = getUserChoice();

    TravelRecordDAO dao;
    TravelRecordService service(dao);
    TravelRecordController controller(service);
    switch (mode) {
        case 1:
            controller.runCommandLineInterface();
            break;
        case 2:
            controller.runAddRecordInterface();
            break;
        case 3:
```

```
            controller.runListRecordsInterface();
            break;
        case 4:
            controller.runUpdateRecordInterface("1");
            break;
        case 5:
            controller.runDeleteRecordInterface();
            break;
        case 6:
            controller.runBarChartInterface();
            break;
        case 7:
            controller.runPieChartInterface();
            break;
        default:
            cout << "Invalid choice. Exiting." << endl;
            break;
    }
    return 0;
}
#include "DataSource.h"
#include <iostream>

using namespace std;

/**
 * @brief Opens a connection to the SQLite database.
 *
 * This method attempts to open a connection to a SQLite database located at a
 * predefined path. If the connection is successful, it returns a pointer to the
 * sqlite3 database object. If the connection fails, it outputs an error message
 * and returns a nullptr.
 *
 * @return A pointer to the sqlite3 database object if the connection is successful,
 *         otherwise nullptr.
 */
sqlite3 *DataSource::openConnection() {
    sqlite3 *db;
    int rc = sqlite3_open("C:\\sqlite\\travelexpenses.db", &db);

    if (rc) {
        cerr << "Can't open database: " << sqlite3_errmsg(db) << endl;
        return nullptr;
    } else {
        return db;
    }
}
#ifndef ASSIGNMENT03_DATASOURCE_H
#define ASSIGNMENT03_DATASOURCE_H

#include <sqlite3.h>

class DataSource {
public:
    static sqlite3 *openConnection();
};

#endif // ASSIGNMENT03_DATASOURCE_H
#ifndef ASSIGNMENT03_TRAVELRECORDDTO_H
#define ASSIGNMENT03_TRAVELRECORDDTO_H

#include <string>
#include <ios>
```

```cpp
#include <iomanip>
#include <ostream>

using namespace std;

class TravelRecordDTO {

private:
    string ref_number;
    string disclosure_group;
    string title;
    string name;
    string purpose;
    string start_date;
    string end_date;
    string destination;
    double airfare;
    double other_transport;
    double lodging;
    double meals;
    double other_expenses;
    double total;

public:

    const string &getRefNumber() const {
        return ref_number;
    }

    void setRefNumber(const string &refNumber) {
        ref_number = refNumber;
    }

    const string &getDisclosureGroup() const {
        return disclosure_group;
    }

    void setDisclosureGroup(const string &disclosureGroup) {
        disclosure_group = disclosureGroup;
    }

    const string &getTitle() const {
        return title;
    }

    void setTitle(const string &title) {
        TravelRecordDTO::title = title;
    }

    const string &getName() const {
        return name;
    }

    void setName(const string &name) {
        TravelRecordDTO::name = name;
    }

    const string &getPurpose() const {
        return purpose;
    }

    void setPurpose(const string &purpose) {
        TravelRecordDTO::purpose = purpose;
```

```
    }

    const string &getStartDate() const {
        return start_date;
    }

    void setStartDate(const string &startDate) {
        start_date = startDate;
    }

    const string &getEndDate() const {
        return end_date;
    }

    void setEndDate(const string &endDate) {
        end_date = endDate;
    }

    const string &getDestination() const {
        return destination;
    }

    void setDestination(const string &destination) {
        TravelRecordDTO::destination = destination;
    }

    const double &getAirfare() const {
        return airfare;
    }

    void setAirfare(const double &airfare) {
        TravelRecordDTO::airfare = airfare;
    }

    const double &getOtherTransport() const {
        return other_transport;
    }

    void setOtherTransport(const double &otherTransport) {
        other_transport = otherTransport;
    }

    const double &getLodging() const {
        return lodging;
    }

    void setLodging(const double &lodging) {
        TravelRecordDTO::lodging = lodging;
    }

    const double &getMeals() const {
        return meals;
    }

    void setMeals(const double &meals) {
        TravelRecordDTO::meals = meals;
    }

    const double &getOtherExpenses() const {
        return other_expenses;
    }

    void setOtherExpenses(const double &otherExpenses) {
```

```
        other_expenses = otherExpenses;
    }

    const double &getTotal() const {
        return total;
    }

    void setTotal(const double &total) {
        TravelRecordDTO::total = total;
    }

    friend ostream &operator<<(ostream &os, const TravelRecordDTO &dto) {
        os << "ref_number: " << dto.ref_number << " disclosure_group: " <<
dto.disclosure_group << " title: "
           << dto.title << " name: " << dto.name << " purpose: " << dto.purpose << "
start_date: " << dto.start_date
           << " end_date: " << dto.end_date << " destination: " << dto.destination << "
airfare: " << dto.airfare
           << " other_transport: " << dto.other_transport << " lodging: " <<
dto.lodging << " meals: " << dto.meals
           << " other_expenses: " << dto.other_expenses << " total: " << dto.total;
        return os;
    }
};

#endif //ASSIGNMENT03_TRAVELRECORDDTO_H
#include <algorithm>
#include <iostream>
#include "../include/TravelRecordDAO.h"

using namespace std;

/**
 * @brief Retrieves all travel records from the database.
 *
 * This method opens a connection to the database and executes a SQL query to fetch all
travel records.
 * Each record is then converted into a `TravelRecordDTO` object and added to a vector.
 *
 * @return A vector of `TravelRecordDTO` objects representing all the travel records in
the database.
 */
vector<TravelRecordDTO> TravelRecordDAO::getAllRecords() {
    sqlite3 *db;
    sqlite3_stmt *stmt;
    vector<TravelRecordDTO> records;

    db = DataSource::openConnection();
    if (db) {
        const char *sql = "SELECT ref_number, disclosure_group, title, name, purpose,
start_date, end_date, destination, airfare, other_transport, lodging, meals,
other_expenses, total FROM expensesnew ORDER BY ref_number";
        if (sqlite3_prepare_v2(db, sql, -1, &stmt, nullptr) != SQLITE_OK) {
            cerr << "Failed to prepare statement: " << sqlite3_errmsg(db) << endl;
            sqlite3_close(db);
            return records;
        }
        while (sqlite3_step(stmt) == SQLITE_ROW) {
            TravelRecordDTO record;
            record.setRefNumber(reinterpret_cast<const char
*>(sqlite3_column_text(stmt, 0)));
            record.setDisclosureGroup(reinterpret_cast<const char
*>(sqlite3_column_text(stmt, 1)));
```

```
            record.setTitle(reinterpret_cast<const char *>(sqlite3_column_text(stmt,
2)));
            record.setName(reinterpret_cast<const char *>(sqlite3_column_text(stmt,
3)));
            record.setPurpose(reinterpret_cast<const char *>(sqlite3_column_text(stmt,
4)));
            record.setStartDate(reinterpret_cast<const char
*>(sqlite3_column_text(stmt, 5)));
            record.setEndDate(reinterpret_cast<const char *>(sqlite3_column_text(stmt,
6)));
            record.setDestination(reinterpret_cast<const char
*>(sqlite3_column_text(stmt, 7)));

            double airfare = sqlite3_column_double(stmt, 8);
            double other_transport = sqlite3_column_double(stmt, 9);
            double lodging = sqlite3_column_double(stmt, 10);
            double meals = sqlite3_column_double(stmt, 11);
            double other_expenses = sqlite3_column_double(stmt, 12);
            double total = sqlite3_column_double(stmt, 13);

            record.setAirfare(airfare);
            record.setOtherTransport(other_transport);
            record.setLodging(lodging);
            record.setMeals(meals);
            record.setOtherExpenses(other_expenses);
            record.setTotal(total);

            records.push_back(record);
        }
        sqlite3_finalize(stmt);
        sqlite3_close(db);
    } else {
        cerr << "Failed to open database connection" << endl;
    }
    return records;
}

/**
 * @brief Retrieves a specific travel record by its reference number.
 *
 * This method opens a database connection and prepares a SQL statement to fetch a
travel record
 * that matches the provided reference number. It then converts the fetched data into a
`TravelRecordDTO` object.
 *
 * @param id The reference number of the travel record to retrieve.
 * @return A `TravelRecordDTO` object representing the fetched travel record.
 */
TravelRecordDTO TravelRecordDAO::getRecordById(const string &id) {
    sqlite3 *db;
    sqlite3_stmt *stmt;
    TravelRecordDTO record;

    db = DataSource::openConnection();
    if (db) {
        const char *sql = "SELECT ref_number, disclosure_group, title, name, purpose,
start_date, end_date, destination, airfare, other_transport, lodging, meals,
other_expenses, total FROM expensesnew WHERE ref_number = ?";
        if (sqlite3_prepare_v2(db, sql, -1, &stmt, nullptr) != SQLITE_OK) {
            cerr << "Failed to prepare statement: " << sqlite3_errmsg(db) << endl;
            sqlite3_close(db);
            return record;
        }
```

```
        sqlite3_bind_text(stmt, 1, id.c_str(), -1, SQLITE_STATIC);

        if (sqlite3_step(stmt) == SQLITE_ROW) {
            record.setRefNumber(reinterpret_cast<const char
*>(sqlite3_column_text(stmt, 0)));
            record.setDisclosureGroup(reinterpret_cast<const char
*>(sqlite3_column_text(stmt, 1)));
            record.setTitle(reinterpret_cast<const char *>(sqlite3_column_text(stmt,
2)));
            record.setName(reinterpret_cast<const char *>(sqlite3_column_text(stmt,
3)));
            record.setPurpose(reinterpret_cast<const char *>(sqlite3_column_text(stmt,
4)));
            record.setStartDate(reinterpret_cast<const char
*>(sqlite3_column_text(stmt, 5)));
            record.setEndDate(reinterpret_cast<const char *>(sqlite3_column_text(stmt,
6)));
            record.setDestination(reinterpret_cast<const char
*>(sqlite3_column_text(stmt, 7)));

            double airfare = sqlite3_column_double(stmt, 8);
            double other_transport = sqlite3_column_double(stmt, 9);
            double lodging = sqlite3_column_double(stmt, 10);
            double meals = sqlite3_column_double(stmt, 11);
            double other_expenses = sqlite3_column_double(stmt, 12);
            double total = sqlite3_column_double(stmt, 13);

            record.setAirfare(airfare);
            record.setOtherTransport(other_transport);
            record.setLodging(lodging);
            record.setMeals(meals);
            record.setOtherExpenses(other_expenses);
            record.setTotal(total);
        }

        sqlite3_finalize(stmt);
        sqlite3_close(db);
    } else {
        cerr << "Failed to open database connection" << endl;
    }

    return record;
}

/**
 * @brief Adds a new travel record to the database.
 *
 * This method opens a database connection and prepares a SQL statement to insert a new
travel record
 * with the details provided in the `TravelRecordDTO` object.
 *
 * @param record The `TravelRecordDTO` object containing the details of the travel
record to add.
 */
void TravelRecordDAO::addRecord(const TravelRecordDTO &record) {
    sqlite3 *db;
    sqlite3_stmt *stmt;

    db = DataSource::openConnection();
    if (db) {
        const char *sql = "INSERT INTO expensesnew (ref_number, disclosure_group,
title, name, purpose, start_date, end_date, destination, airfare, other_transport,
```

```
lodging, meals, other_expenses, total) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
?)";
        if (sqlite3_prepare_v2(db, sql, -1, &stmt, nullptr) != SQLITE_OK) {
            cerr << "Failed to prepare statement: " << sqlite3_errmsg(db) << endl;
            sqlite3_close(db);
            return;
        }

        sqlite3_bind_text(stmt, 1, record.getRefNumber().c_str(), -1,
SQLITE_TRANSIENT);
        sqlite3_bind_text(stmt, 2, record.getDisclosureGroup().c_str(), -1,
SQLITE_TRANSIENT);
        sqlite3_bind_text(stmt, 3, record.getTitle().c_str(), -1, SQLITE_TRANSIENT);
        sqlite3_bind_text(stmt, 4, record.getName().c_str(), -1, SQLITE_TRANSIENT);
        sqlite3_bind_text(stmt, 5, record.getPurpose().c_str(), -1, SQLITE_TRANSIENT);
        sqlite3_bind_text(stmt, 6, record.getStartDate().c_str(), -1,
SQLITE_TRANSIENT);
        sqlite3_bind_text(stmt, 7, record.getEndDate().c_str(), -1, SQLITE_TRANSIENT);
        sqlite3_bind_text(stmt, 8, record.getDestination().c_str(), -1,
SQLITE_TRANSIENT);

        sqlite3_bind_double(stmt, 9, record.getAirfare());
        sqlite3_bind_double(stmt, 10, record.getOtherTransport());
        sqlite3_bind_double(stmt, 11, record.getLodging());
        sqlite3_bind_double(stmt, 12, record.getMeals());
        sqlite3_bind_double(stmt, 13, record.getOtherExpenses());
        sqlite3_bind_double(stmt, 14, record.getTotal());

        if (sqlite3_step(stmt) != SQLITE_DONE) {
            cerr << "Failed to execute statement: " << sqlite3_errmsg(db) << endl;
        }

        sqlite3_finalize(stmt);
        sqlite3_close(db);
    } else {
        cerr << "Failed to open database connection" << endl;
    }
}

/**
 * @brief Updates an existing travel record in the database.
 *
 * This method opens a database connection and prepares a SQL statement to update an
existing travel record.
 * The record to be updated is identified by the reference number in the provided
`TravelRecordDTO` object.
 *
 * @param record The `TravelRecordDTO` object containing updated details for the travel
record.
 */
void TravelRecordDAO::updateRecord(const TravelRecordDTO &record) {
    sqlite3 *db;
    sqlite3_stmt *stmt;

    db = DataSource::openConnection();
    if (db) {
        const char *sql = "UPDATE expensesnew SET disclosure_group = ?, title = ?, name
= ?, purpose = ?, start_date = ?, end_date = ?, destination = ?, airfare = ?,
other_transport = ?, lodging = ?, meals = ?, other_expenses = ?, total = ? WHERE
ref_number = ?";
        if (sqlite3_prepare_v2(db, sql, -1, &stmt, nullptr) != SQLITE_OK) {
            cerr << "Failed to prepare statement: " << sqlite3_errmsg(db) << endl;
            sqlite3_close(db);
```

```
            return;
        }

        sqlite3_bind_text(stmt, 1, record.getDisclosureGroup().c_str(), -1,
SQLITE_TRANSIENT);
        sqlite3_bind_text(stmt, 2, record.getTitle().c_str(), -1, SQLITE_TRANSIENT);
        sqlite3_bind_text(stmt, 3, record.getName().c_str(), -1, SQLITE_TRANSIENT);
        sqlite3_bind_text(stmt, 4, record.getPurpose().c_str(), -1, SQLITE_TRANSIENT);
        sqlite3_bind_text(stmt, 5, record.getStartDate().c_str(), -1,
SQLITE_TRANSIENT);
        sqlite3_bind_text(stmt, 6, record.getEndDate().c_str(), -1, SQLITE_TRANSIENT);
        sqlite3_bind_text(stmt, 7, record.getDestination().c_str(), -1,
SQLITE_TRANSIENT);

        sqlite3_bind_double(stmt, 8, record.getAirfare());
        sqlite3_bind_double(stmt, 9, record.getOtherTransport());
        sqlite3_bind_double(stmt, 10, record.getLodging());
        sqlite3_bind_double(stmt, 11, record.getMeals());
        sqlite3_bind_double(stmt, 12, record.getOtherExpenses());
        sqlite3_bind_double(stmt, 13, record.getTotal());

        sqlite3_bind_text(stmt, 14, record.getRefNumber().c_str(), -1,
SQLITE_TRANSIENT);

        if (sqlite3_step(stmt) != SQLITE_DONE) {
            cerr << "Failed to execute statement: " << sqlite3_errmsg(db) << endl;
        }

        sqlite3_finalize(stmt);
        sqlite3_close(db);
    } else {
        cerr << "Failed to open database connection" << endl;
    }
}

/**
 * @brief Deletes a travel record from the database by its reference number.
 *
 * This method opens a database connection and prepares a SQL statement to delete a
travel record
 * identified by the provided reference number.
 *
 * @param id The reference number of the travel record to delete.
 */
void TravelRecordDAO::deleteRecordById(const string &id) {
    sqlite3 *db;
    sqlite3_stmt *stmt;

    db = DataSource::openConnection();
    if (db) {
        const char *sql = "DELETE FROM expensesnew WHERE ref_number = ?";
        if (sqlite3_prepare_v2(db, sql, -1, &stmt, nullptr) != SQLITE_OK) {
            cerr << "Failed to prepare statement: " << sqlite3_errmsg(db) << endl;
            sqlite3_close(db);
            return;
        }
        sqlite3_bind_text(stmt, 1, id.c_str(), -1, SQLITE_STATIC);

        if (sqlite3_step(stmt) != SQLITE_DONE) {
            cerr << "Failed to execute statement: " << sqlite3_errmsg(db) << endl;
        }

        sqlite3_finalize(stmt);
```

```
        sqlite3_close(db);
    } else {
        cerr << "Failed to open database connection" << endl;
    }
}

/**
 * @brief Retrieves the total number of travel records in the database.
 *
 * This method opens a database connection and executes a SQL query to count the total
number of travel records.
 *
 * @param record A `TravelRecordDTO` object, potentially used for filtering criteria in
future enhancements.
 * @return The total count of travel records in the database.
 */
int TravelRecordDAO::getCount(const TravelRecordDTO &record) {
    sqlite3 *db;
    sqlite3_stmt *stmt;
    int count = 0;

    db = DataSource::openConnection();
    if (db) {
        const char *sql = "SELECT COUNT(*) FROM expensesnew";
        if (sqlite3_prepare_v2(db, sql, -1, &stmt, nullptr) == SQLITE_OK) {
            if (sqlite3_step(stmt) == SQLITE_ROW) {
                count = static_cast<int>(sqlite3_column_int64(stmt, 0));
            } else {
                cerr << "Failed to execute statement: " << sqlite3_errmsg(db) << endl;
            }
            sqlite3_finalize(stmt);
        } else {
            cerr << "Failed to prepare statement: " << sqlite3_errmsg(db) << endl;
        }
        sqlite3_close(db);
    } else {
        cerr << "Failed to open database connection" << endl;
    }

    return count;
}

/**
 * @brief Retrieves the highest total value among all travel records.
 *
 * Connects to the database and executes a SQL query to find the maximum total value.
 * Only records with valid (non-null) and positive total values are considered.
 *
 * @return The highest total value as an integer, or -1 if no valid records are found.
 */
int TravelRecordDAO::getTotalHighest() {
    sqlite3 *db;
    sqlite3_stmt *stmt;
    int highest = -1;

    db = DataSource::openConnection();
    if (db) {
        const char *sql = "SELECT MAX(CAST(total AS REAL)) FROM expensesnew WHERE total
> 0 AND typeof(total) = 'real';";
        if (sqlite3_prepare_v2(db, sql, -1, &stmt, nullptr) == SQLITE_OK) {
            if (sqlite3_step(stmt) == SQLITE_ROW) {
                if (sqlite3_column_type(stmt, 0) != SQLITE_NULL) {
                    highest = static_cast<int>(sqlite3_column_int64(stmt, 0));
```

```
                }
            } else {
                cerr << "Failed to execute statement: " << sqlite3_errmsg(db) << endl;
            }
            sqlite3_finalize(stmt);
        } else {
            cerr << "Failed to prepare statement: " << sqlite3_errmsg(db) << endl;
        }
        sqlite3_close(db);
    } else {
        cerr << "Failed to open database connection" << endl;
    }

    return highest;
}

/**
 * @brief Retrieves the median total value among all travel records.
 *
 * Connects to the database and executes a SQL query to calculate the median of total
values.
 * The median is computed by averaging the middle values in the sorted list of totals.
 *
 * @return The median total value as a double.
 */
int TravelRecordDAO::getTotalMedian() {
    sqlite3 *db;
    sqlite3_stmt *stmt;
    double median = 0;

    db = DataSource::openConnection();
    if (db) {
        const char *sql = "SELECT AVG(total) FROM (\n"
                          "    SELECT total,\n"
                          "    ROW_NUMBER() OVER (ORDER BY total) AS row_number,\n"
                          "    COUNT(*) OVER () AS total_rows\n"
                          "    FROM expensesnew\n"
                          ") WHERE row_number IN (total_rows / 2, (total_rows + 1) /
2);";
        if (sqlite3_prepare_v2(db, sql, -1, &stmt, nullptr) == SQLITE_OK) {
            if (sqlite3_step(stmt) == SQLITE_ROW) {
                median = sqlite3_column_double(stmt, 0);
            } else {
                cerr << "Failed to execute statement: " << sqlite3_errmsg(db) << endl;
            }
            sqlite3_finalize(stmt);
        } else {
            cerr << "Failed to prepare statement: " << sqlite3_errmsg(db) << endl;
        }
        sqlite3_close(db);
    } else {
        cerr << "Failed to open database connection" << endl;
    }

    return static_cast<int>(median);
}

/**
 * @brief Retrieves the lowest total value among all travel records.
 *
 * Connects to the database and executes a SQL query to find the minimum total value.
 * Only records with valid (non-null) and positive total values are considered.
 *
```

```
 * @return The lowest total value as an integer, or -1 if no valid records are found.
 */
int TravelRecordDAO::getTotalLowest() {
    sqlite3 *db;
    sqlite3_stmt *stmt;
    int lowest = -1;

    db = DataSource::openConnection();
    if (db) {
        const char *sql = "SELECT MIN(total) FROM expensesnew WHERE total > 0";
        if (sqlite3_prepare_v2(db, sql, -1, &stmt, nullptr) == SQLITE_OK) {
            if (sqlite3_step(stmt) == SQLITE_ROW) {
                if (sqlite3_column_type(stmt, 0) != SQLITE_NULL) {
                    lowest = static_cast<int>(sqlite3_column_int64(stmt, 0));
                } else {

                    cerr << "No records found with 'total' greater than 0." << endl;
                    lowest = -1;
                }
            } else {
                cerr << "Failed to execute statement: " << sqlite3_errmsg(db) << endl;
            }
            sqlite3_finalize(stmt);
        } else {
            cerr << "Failed to prepare statement: " << sqlite3_errmsg(db) << endl;
        }
        sqlite3_close(db);
    } else {
        cerr << "Failed to open database connection" << endl;
    }

    return lowest;
}

/**
 * @brief Retrieves the highest airfare value among all travel records.
 *
 * Connects to the database and executes a SQL query to find the maximum airfare value.
 * Only records with valid (non-null) and positive airfare values are considered.
 *
 * @return The highest airfare value as an integer.
 */
int TravelRecordDAO::getAirfareHighest() {
    sqlite3 *db;
    sqlite3_stmt *stmt;
    int highest = 0;

    db = DataSource::openConnection();
    if (db) {
        const char *sql = "SELECT MAX(CAST(airfare AS REAL)) FROM expensesnew WHERE
airfare > 0 AND typeof(airfare) = 'real';";
        if (sqlite3_prepare_v2(db, sql, -1, &stmt, nullptr) == SQLITE_OK) {
            if (sqlite3_step(stmt) == SQLITE_ROW) {
                highest = static_cast<int>(sqlite3_column_int64(stmt, 0));
            } else {
                cerr << "Failed to execute statement: " << sqlite3_errmsg(db) << endl;
            }
            sqlite3_finalize(stmt);
        } else {
            cerr << "Failed to prepare statement: " << sqlite3_errmsg(db) << endl;
        }
        sqlite3_close(db);
    } else {
```

```
        cerr << "Failed to open database connection" << endl;
    }

    return highest;
}

/**
 * @brief Retrieves the median airfare value among all travel records.
 *
 * Connects to the database and executes a SQL query to calculate the median of airfare
values.
 * The median is computed by averaging the middle values in the sorted list of
airfares.
 *
 * @return The median airfare value as a double.
 */
int TravelRecordDAO::getAirfareMedian() {
    sqlite3 *db;
    sqlite3_stmt *stmt;
    double median = 0;

    db = DataSource::openConnection();
    if (db) {
        const char *sql = "SELECT AVG(airfare) FROM (\n"
                          "    SELECT airfare,\n"
                          "    ROW_NUMBER() OVER (ORDER BY airfare) AS row_number,\n"
                          "    COUNT(*) OVER () AS airfare_rows\n"
                          "    FROM expensesnew\n"
                          ") WHERE row_number IN (airfare_rows / 2, (airfare_rows + 1)
/ 2);";
        if (sqlite3_prepare_v2(db, sql, -1, &stmt, nullptr) == SQLITE_OK) {
            if (sqlite3_step(stmt) == SQLITE_ROW) {
                median = sqlite3_column_double(stmt, 0);
            } else {
                cerr << "Failed to execute statement: " << sqlite3_errmsg(db) << endl;
            }
            sqlite3_finalize(stmt);
        } else {
            cerr << "Failed to prepare statement: " << sqlite3_errmsg(db) << endl;
        }
        sqlite3_close(db);
    } else {
        cerr << "Failed to open database connection" << endl;
    }

    return static_cast<int>(median);
}

/**
 * @brief Retrieves the lowest airfare value among all travel records.
 *
 * Connects to the database and executes a SQL query to find the minimum airfare value.
 * Only records with valid (non-null) and positive airfare values are considered.
 *
 * @return The lowest airfare value as an integer, or -1 if no valid records are found.
 */
int TravelRecordDAO::getAirfareLowest() {
    sqlite3 *db;
    sqlite3_stmt *stmt;
    int lowest = -1;

    db = DataSource::openConnection();
    if (db) {
```

```
        const char *sql = "SELECT MIN(airfare) FROM expensesnew WHERE airfare > 0";
        if (sqlite3_prepare_v2(db, sql, -1, &stmt, nullptr) == SQLITE_OK) {
            if (sqlite3_step(stmt) == SQLITE_ROW) {
                if (sqlite3_column_type(stmt, 0) != SQLITE_NULL) {
                    lowest = static_cast<int>(sqlite3_column_int64(stmt, 0));
                } else {

                    cerr << "No records found with 'airfare' greater than 0." << endl;
                    lowest = -1;
                }
            } else {
                cerr << "Failed to execute statement: " << sqlite3_errmsg(db) << endl;
            }
            sqlite3_finalize(stmt);
        } else {
            cerr << "Failed to prepare statement: " << sqlite3_errmsg(db) << endl;
        }
        sqlite3_close(db);
    } else {
        cerr << "Failed to open database connection" << endl;
    }

    return lowest;
}

/**
 * @brief Retrieves the highest meals expense among all travel records.
 *
 * Connects to the database and executes a SQL query to find the maximum meals expense.
 * Only records with valid (non-null) and positive meals values are considered.
 *
 * @return The highest meals expense as an integer.
 */
int TravelRecordDAO::getMealsHighest() {
    sqlite3 *db;
    sqlite3_stmt *stmt;
    int highest = 0;

    db = DataSource::openConnection();
    if (db) {
        const char *sql = "SELECT MAX(CAST(meals AS REAL)) FROM expensesnew WHERE meals
> 0 AND typeof(meals) = 'real';";
        if (sqlite3_prepare_v2(db, sql, -1, &stmt, nullptr) == SQLITE_OK) {
            if (sqlite3_step(stmt) == SQLITE_ROW) {
                highest = static_cast<int>(sqlite3_column_int64(stmt, 0));
            } else {
                cerr << "Failed to execute statement: " << sqlite3_errmsg(db) << endl;
            }
            sqlite3_finalize(stmt);
        } else {
            cerr << "Failed to prepare statement: " << sqlite3_errmsg(db) << endl;
        }
        sqlite3_close(db);
    } else {
        cerr << "Failed to open database connection" << endl;
    }

    return highest;
}

/**
 * @brief Retrieves the highest meals expense among all travel records.
 *
```

```
 * Connects to the database and executes a SQL query to find the maximum meals expense.
 * Only records with valid (non-null) and positive meals values are considered.
 *
 * @return The highest meals expense as an integer.
 */
int TravelRecordDAO::getMealsMedian() {
    sqlite3 *db;
    sqlite3_stmt *stmt;
    double median = 0;

    db = DataSource::openConnection();
    if (db) {
        const char *sql = "SELECT AVG(meals) FROM (\n"
                          "    SELECT meals,\n"
                          "    ROW_NUMBER() OVER (ORDER BY meals) AS row_number,\n"
                          "    COUNT(*) OVER () AS meals_rows\n"
                          "    FROM expensesnew\n"
                          ") WHERE row_number IN (meals_rows / 2, (meals_rows + 1) /
2);";
        if (sqlite3_prepare_v2(db, sql, -1, &stmt, nullptr) == SQLITE_OK) {
            if (sqlite3_step(stmt) == SQLITE_ROW) {
                median = sqlite3_column_double(stmt, 0);
            } else {
                cerr << "Failed to execute statement: " << sqlite3_errmsg(db) << endl;
            }
            sqlite3_finalize(stmt);
        } else {
            cerr << "Failed to prepare statement: " << sqlite3_errmsg(db) << endl;
        }
        sqlite3_close(db);
    } else {
        cerr << "Failed to open database connection" << endl;
    }

    return static_cast<int>(median);
}

/**
 * @brief Retrieves the lowest meals expense among all travel records.
 *
 * Connects to the database and executes a SQL query to find the minimum meals expense.
 * Only records with valid (non-null) and positive meals values are considered.
 *
 * @return The lowest meals expense as an integer, or -1 if no valid records are found.
 */
int TravelRecordDAO::getMealsLowest() {
    sqlite3 *db;
    sqlite3_stmt *stmt;
    int lowest = -1;

    db = DataSource::openConnection();
    if (db) {
        const char *sql = "SELECT MIN(meals) FROM expensesnew WHERE meals > 0";
        if (sqlite3_prepare_v2(db, sql, -1, &stmt, nullptr) == SQLITE_OK) {
            if (sqlite3_step(stmt) == SQLITE_ROW) {
                if (sqlite3_column_type(stmt, 0) != SQLITE_NULL) {
                    lowest = static_cast<int>(sqlite3_column_int64(stmt, 0));
                } else {

                    cerr << "No records found with 'meals' greater than 0." << endl;
                    lowest = -1;
                }
            } else {
```

```
                cerr << "Failed to execute statement: " << sqlite3_errmsg(db) << endl;
            }
            sqlite3_finalize(stmt);
        } else {
            cerr << "Failed to prepare statement: " << sqlite3_errmsg(db) << endl;
        }

        sqlite3_close(db);
    } else {
        cerr << "Failed to open database connection" << endl;
    }

    return lowest;
}



#ifndef ASSIGNMENT03_TRAVELRECORDDAO_H
#define ASSIGNMENT03_TRAVELRECORDDAO_H

#include "DataSource.h"
#include "TravelRecordDTO.h"
#include <vector>

class TravelRecordDAO {

public:
    virtual std::vector<TravelRecordDTO> getAllRecords();

    virtual TravelRecordDTO getRecordById(const std::string &id);

    virtual void addRecord(const TravelRecordDTO &record);

    virtual void updateRecord(const TravelRecordDTO &record);

    virtual void deleteRecordById(const std::string &id);

    virtual int getCount(const TravelRecordDTO &record);

    int getTotalHighest();

    int getTotalMedian();

    int getTotalLowest();

    int getAirfareHighest();

    int getAirfareMedian();

    int getAirfareLowest();

    int getMealsHighest();

    int getMealsMedian();

    int getMealsLowest();
};

#endif //ASSIGNMENT03_TRAVELRECORDDAO_H
#include "../include/TravelRecordService.h"
#include <stdexcept>

using namespace std;
```

```cpp
/**
 * @brief Constructor for TravelRecordService.
 * @param dao A reference to TravelRecordDAO object.
 */
TravelRecordService::TravelRecordService(const TravelRecordDAO &dao) : dao(dao) {
}

/**
 * @brief Retrieves all travel records.
 * @return A vector of TravelRecordDTO objects.
 */
vector<TravelRecordDTO> TravelRecordService::getAllRecords() {
    return dao.getAllRecords();
}

/**
 * @brief Gets a travel record by its ID.
 * @param id The ID of the travel record.
 * @return A TravelRecordDTO object.
 */
TravelRecordDTO TravelRecordService::getRecordById(const std::string &id) {
    return dao.getRecordById(id);
}

/**
 * @brief Adds a new travel record.
 * @param record A TravelRecordDTO object representing the new record.
 */
void TravelRecordService::addRecord(const TravelRecordDTO &record) {
    dao.addRecord(record);
}

/**
 * @brief Updates an existing travel record.
 * @param record A TravelRecordDTO object with updated information.
 */
void TravelRecordService::updateRecord(const TravelRecordDTO &record) {
    dao.updateRecord(record);
}

/**
 * @brief Deletes a travel record by its ID.
 * @param id The ID of the travel record to be deleted.
 */
void TravelRecordService::deleteRecordById(const std::string &id) {
    dao.deleteRecordById(id);
}

/**
 * @brief Gets the count of a specific type of travel record.
 * @param record A TravelRecordDTO object to specify the type of record.
 * @return The count of records.
 */
int TravelRecordService::getCount(const TravelRecordDTO &record) {
    return dao.getCount(record);
}

/**
 * @brief Retrieves the total highest travel record value.
 * @return The highest value.
 */
int TravelRecordService::getTotalHighest() {
```

```
    return dao.getTotalHighest();
}

/**
 * @brief Retrieves the total median travel record value.
 * @return The median value.
 */
int TravelRecordService::getTotalMedian() {
    return dao.getTotalMedian();
}

/**
 * @brief Retrieves the total lowest travel record value.
 * @return The lowest value.
 */
int TravelRecordService::getTotalLowest() {
    return dao.getTotalLowest();
}

/**
 * @brief Retrieves the highest airfare.
 * @return The highest airfare value.
 */
int TravelRecordService::getAirfareHighest() {
    return dao.getAirfareHighest();
}

/**
 * @brief Retrieves the median airfare.
 * @return The median airfare value.
 */
int TravelRecordService::getAirfareMedian() {
    return dao.getAirfareMedian();
}

/**
 * @brief Retrieves the lowest airfare.
 * @return The lowest airfare value.
 */
int TravelRecordService::getAirfareLowest() {
    return dao.getAirfareLowest();
}

/**
 * @brief Retrieves the highest meal expense.
 * @return The highest meal expense value.
 */
int TravelRecordService::getMealsHighest() {
    return dao.getMealsHighest();
}

/**
 * @brief Retrieves the median meal expense.
 * @return The median meal expense value.
 */
int TravelRecordService::getMealsMedian() {
    return dao.getMealsMedian();
}

/**
 * @brief Retrieves the lowest meal expense.
 * @return The lowest meal expense value.
 */
```

```cpp
int TravelRecordService::getMealsLowest() {
    return dao.getMealsLowest();
}

#ifndef ASSIGNMENT03_TRAVELRECORDSERVICE_H
#define ASSIGNMENT03_TRAVELRECORDSERVICE_H

#include "TravelRecordDAO.h"
#include <vector>
#include <string>

class TravelRecordService {
private:
    TravelRecordDAO dao;

public:
    explicit TravelRecordService(const TravelRecordDAO &dao);

    std::vector<TravelRecordDTO> getAllRecords();

    TravelRecordDTO getRecordById(const std::string &id);

    void addRecord(const TravelRecordDTO &record);

    void updateRecord(const TravelRecordDTO &record);

    void deleteRecordById(const std::string &id);

    int getCount(const TravelRecordDTO &record);

    int getTotalHighest();

    int getTotalMedian();

    int getTotalLowest();

    int getAirfareHighest();

    int getAirfareMedian();

    int getAirfareLowest();

    int getMealsHighest();

    int getMealsMedian();

    int getMealsLowest();
};

#endif //ASSIGNMENT03_TRAVELRECORDSERVICE_H
#include "CMDUserInterface.h"
#include <iostream>
#include <string>

using namespace std;

/**
 * @brief Displays the main menu and handles user interaction.
 *
 * This method continuously displays the main menu, allowing the user to choose various
options
 * such as listing all records, adding a new record, updating or deleting a record,
etc.
```

```
 * It uses a while loop for continuous operation and breaks the loop if the user
chooses to exit.
 */
void TerminalUI::showMenu() {
    while (true) {
        cout << "\n\nPlease select an option:\n" << endl;
        cout << "1. List all records\n";
        cout << "2. Get record by ID\n";
        cout << "3. Add a new record\n";
        cout << "4. Update a record\n";
        cout << "5. Delete a record\n";
        cout << "6. Show record count\n";
        cout << "7. Exit\n\n";
        cout << "Enter your choice: ";

        int choice;
        cin >> choice;

        if (choice == 7) {
            cout << "Goodbye!" << endl;
            break;
        }
        handleChoice(choice);
    }
}

/**
 * @brief Populates a TravelRecordDTO object with user input.
 *
 * This method requests user input for different fields of a TravelRecordDTO object.
 * The method supports both creating a new record and updating an existing one.
 *
 * @param record A reference to the TravelRecordDTO object to populate.
 * @param isUpdate A boolean flag to indicate whether this is for updating an existing
record (true) or creating a new one (false).
 */
void TerminalUI::populateTravelRecordFromInput(TravelRecordDTO &record, bool isUpdate)
{
    string input;

    if (isUpdate) {
        cout << "Current reference number: " << record.getRefNumber();
        cout << "\nEnter new value or press Enter to keep: ";
    } else {
        cout << "Reference number: ";
    }
    getline(cin >> ws, input);
    if (!input.empty()) {
        record.setRefNumber(input);
    }

    if (isUpdate) {
        cout << "Current disclosure group: " << record.getDisclosureGroup();
        cout << "\nEnter new value or press Enter to keep: ";
    } else {
        cout << "Disclosure group: ";
    }
    getline(cin, input);
    if (!input.empty()) {
        record.setDisclosureGroup(input);
    }

    if (isUpdate) {
```

```
        cout << "Current title: " << record.getTitle();
        cout << "\nEnter new value or press Enter to keep: ";
    } else {
        cout << "Title: ";
    }
    getline(cin, input);
    if (!input.empty()) {
        record.setTitle(input);
    }

    if (isUpdate) {
        cout << "Current name: " << record.getName();
        cout << "\nEnter new value or press Enter to keep: ";
    } else {
        cout << "Name: ";
    }
    getline(cin, input);
    if (!input.empty()) {
        record.setName(input);
    }
    if (isUpdate) {
        cout << "Current purpose: " << record.getPurpose();
        cout << "\nEnter new value or press Enter to keep: ";
    } else {
        cout << "Purpose: ";
    }
    getline(cin, input);
    if (!input.empty()) {
        record.setPurpose(input);
    }

    if (isUpdate) {
        cout << "Current start date: " << record.getStartDate();
        cout << "\nEnter new value or press Enter to keep: ";
    } else {
        cout << "Start date: ";
    }
    getline(cin, input);
    if (!input.empty()) {
        record.setStartDate(input);
    }

    if (isUpdate) {
        cout << "Current end date: " << record.getEndDate();
        cout << "\nEnter new value or press Enter to keep: ";
    } else {
        cout << "End date: ";
    }
    getline(cin, input);
    if (!input.empty()) {
        record.setEndDate(input);
    }

    if (isUpdate) {
        cout << "Current destination: " << record.getDestination();
        cout << "\nEnter new value or press Enter to keep: ";
    } else {
        cout << "Destination: ";
    }
    getline(cin, input);
    if (!input.empty()) {
        record.setDestination(input);
    }
```

```
    if (isUpdate) {
        cout << "Current airfare: " << record.getAirfare();
        cout << "\nEnter new value or press Enter to keep: ";
    } else {
        cout << "Airfare: ";
    }
    getline(cin, input);
    if (!input.empty()) {
        try {
            double newValue = stod(input);
            record.setAirfare(newValue);
        } catch (const invalid_argument &e) {
            cerr << "Invalid input. Please enter a valid numeric value." << endl;
        }
    }
    if (isUpdate) {
        cout << "Current lodging: " << record.getLodging();
        cout << "\nEnter new value or press Enter to keep: ";
    } else {
        cout << "Lodging: ";
    }
    getline(cin, input);
    if (!input.empty()) {
        try {
            double newValue = stod(input);
            record.setLodging(newValue);
        } catch (const invalid_argument &e) {
            cerr << "Invalid input. Please enter a valid numeric value." << endl;
        }
    }

    if (isUpdate) {
        cout << "Current meals: " << record.getMeals();
        cout << "\nEnter new value or press Enter to keep: ";
    } else {
        cout << "Meals: ";
    }
    getline(cin, input);
    if (!input.empty()) {
        try {
            double newValue = stod(input);
            record.setMeals(newValue);
        } catch (const invalid_argument &e) {
            cerr << "Invalid input. Please enter a valid numeric value." << endl;
        }
    }

    if (isUpdate) {
        cout << "Current other expenses: " << record.getOtherExpenses();
        cout << "\nEnter new value or press Enter to keep: ";
    } else {
        cout << "Other expenses: ";
    }
    getline(cin, input);
    if (!input.empty()) {
        try {
            double newValue = stod(input);
            record.setOtherExpenses(newValue);
        } catch (const invalid_argument &e) {
            cerr << "Invalid input. Please enter a valid numeric value." << endl;
        }
    }
```

```
    if (isUpdate) {
        cout << "Current total: " << record.getTotal();
        cout << "\nEnter new value or press Enter to keep: ";
    } else {
        cout << "Total: ";
    }
    getline(cin, input);
    if (!input.empty()) {
        try {
            double newValue = stod(input);
            record.setTotal(newValue);
        } catch (const invalid_argument &e) {
            cerr << "Invalid input. Please enter a valid numeric value." << endl;
        }
    }
}

/**
 * @brief Handles the user's menu choice.
 *
 * Based on the user's input choice, this method calls the appropriate function to
 * list, add, update, or delete records, or show the record count.
 *
 * @param choice The user's menu choice as an integer.
 */
void TerminalUI::handleChoice(int choice) {
    switch (choice) {
        case 1: {
            auto records = service.getAllRecords();
            for (const auto &record: records) {
                cout << record << endl;
            }
            break;
        }
        case 2: {
            cout << "Enter record ID: ";
            string id;
            cin >> id;
            TravelRecordDTO record = service.getRecordById(id);
            if (record.getRefNumber().empty() && record.getRefNumber() != id) {
                cout << "Record with ID " << id << " not found.\n";
                break;
            } else {
                cout << "\n" << record << endl;
            }
            break;
        }
        case 3: {
            TravelRecordDTO record;
            populateTravelRecordFromInput(record);
            service.addRecord(record);
            cout << "Record added successfully.\n";
            break;
        }
        case 4: {
            cout << "Enter the reference number of the record to update: ";
            string refNumber;
            cin >> refNumber;
            TravelRecordDTO record = service.getRecordById(refNumber);

            if (!record.getRefNumber().empty()) {
                populateTravelRecordFromInput(record, true);
                service.updateRecord(record);
```

```
                cout << "Record updated successfully.\n";
            } else {
                cout << "Record with reference number " << refNumber << " not
found.\n";
            }

            break;
        }

        case 5: {
            cout << "Enter record ID to delete: ";
            string id;
            cin >> id;
            service.deleteRecordById(id);
            cout << "Record deleted successfully.\n";
            break;
        }
        case 6: {
            int count = service.getCount(TravelRecordDTO());
            cout << "Total record count: " << count << endl;
            break;
        }
        default:
            cout << "Invalid choice. Please try again.\n";
            break;
    }
}
#ifndef ASSIGNMENT03_CMDUSERINTERFACE_H
#define ASSIGNMENT03_CMDUSERINTERFACE_H

#include <iostream>
#include <string>
#include "TravelRecordService.h"

class TerminalUI {
private:
    TravelRecordService service;

    void handleChoice(int choice);

public:
    explicit TerminalUI(const TravelRecordService &service) : service(service) {}

    void showMenu();

    void populateTravelRecordFromInput(TravelRecordDTO &record, bool isUpdate = false);
};

#endif //ASSIGNMENT03_CMDUSERINTERFACE_H
#include "TravelRecordService.h"
#include "GraphicalUserInterface.h"
#include <imgui.h>
#include "../dependencies/implot/implot.h"

/**
 * @brief Renders the GUI for adding a new travel record.
 *
 * This method initializes a static TravelRecordDTO object and input buffers to store
user input.
 * It renders a graphical user interface using ImGui for the user to input new travel
record data.
 * The method handles the input for various fields like reference number, name, dates,
costs, etc.,
```

```
 * and adds the new record to the service when the 'Add Record' button is clicked.
 */
void GraphicalUserInterface::RenderAddNewRecordGUI() {
    static TravelRecordDTO newRecord;
    static char refNumberBuffer[100] = "";
    static char disclosureGroupBuffer[100] = "";
    static char titleBuffer[100] = "";
    static char nameBuffer[100] = "";
    static char purposeBuffer[100] = "";
    static char startDateBuffer[100] = "";
    static char endDateBuffer[100] = "";
    static char destinationBuffer[100] = "";
    static double airfare = 0.0;
    static double otherTransport = 0.0;
    static double lodging = 0.0;
    static double meals = 0.0;
    static double otherExpenses = 0.0;
    static double total = 0.0;

    ImVec2 fullscreen = ImGui::GetIO().DisplaySize;

    ImGui::SetNextWindowPos(ImVec2(0, 0));
    ImGui::SetNextWindowSize(fullscreen);

    ImGui::Begin("Add New Record", nullptr,
                 ImGuiWindowFlags_NoTitleBar | ImGuiWindowFlags_NoResize |
ImGuiWindowFlags_NoMove);

    if (ImGui::Button("Load Existing Values")) {
        strcpy(refNumberBuffer, newRecord.getRefNumber().c_str());
        strcpy(disclosureGroupBuffer, newRecord.getDisclosureGroup().c_str());
        strcpy(titleBuffer, newRecord.getTitle().c_str());
        strcpy(nameBuffer, newRecord.getName().c_str());
        strcpy(purposeBuffer, newRecord.getPurpose().c_str());
        strcpy(startDateBuffer, newRecord.getStartDate().c_str());
        strcpy(endDateBuffer, newRecord.getEndDate().c_str());
        strcpy(destinationBuffer, newRecord.getDestination().c_str());
        airfare = newRecord.getAirfare();
        otherTransport = newRecord.getOtherTransport();
        lodging = newRecord.getLodging();
        meals = newRecord.getMeals();
        otherExpenses = newRecord.getOtherExpenses();
        total = newRecord.getTotal();
    }

    ImGui::InputText("Reference Number", refNumberBuffer,
IM_ARRAYSIZE(refNumberBuffer));
    ImGui::InputText("Disclosure Group", disclosureGroupBuffer,
IM_ARRAYSIZE(disclosureGroupBuffer));
    ImGui::InputText("Title", titleBuffer, IM_ARRAYSIZE(titleBuffer));
    ImGui::InputText("Name", nameBuffer, IM_ARRAYSIZE(nameBuffer));
    ImGui::InputText("Purpose", purposeBuffer, IM_ARRAYSIZE(purposeBuffer));
    ImGui::InputText("Start Date", startDateBuffer, IM_ARRAYSIZE(startDateBuffer));
    ImGui::InputText("End Date", endDateBuffer, IM_ARRAYSIZE(endDateBuffer));
    ImGui::InputText("Destination", destinationBuffer,
IM_ARRAYSIZE(destinationBuffer));
    ImGui::InputDouble("Airfare", &airfare, 0.0, 0.0, "%.1f");
    ImGui::InputDouble("Other Transport", &otherTransport, 0.0, 0.0, "%.1f");
    ImGui::InputDouble("Lodging", &lodging, 0.0, 0.0, "%.1f");
    ImGui::InputDouble("Meals", &meals, 0.0, 0.0, "%.1f");
    ImGui::InputDouble("Other Expenses", &otherExpenses, 0.0, 0.0, "%.1f");
    ImGui::InputDouble("Total", &total, 0.0, 0.0, "%.1f");
```

```
    if (ImGui::Button("Add Record")) {
        newRecord.setRefNumber(std::string(refNumberBuffer));
        newRecord.setDisclosureGroup(std::string(disclosureGroupBuffer));
        newRecord.setTitle(std::string(titleBuffer));
        newRecord.setName(std::string(nameBuffer));
        newRecord.setPurpose(std::string(purposeBuffer));
        newRecord.setStartDate(std::string(startDateBuffer));
        newRecord.setEndDate(std::string(endDateBuffer));
        newRecord.setDestination(std::string(destinationBuffer));
        newRecord.setAirfare(airfare);
        newRecord.setOtherTransport(otherTransport);
        newRecord.setLodging(lodging);
        newRecord.setMeals(meals);
        newRecord.setOtherExpenses(otherExpenses);
        newRecord.setTotal(total);

        service.addRecord(newRecord);
    }

    ImGui::End();
}

/**
 * @brief Renders the GUI for listing all travel records.
 *
 * This method retrieves all records from the service and displays them in a table.
 * It uses ImGui to create a graphical user interface, where each record is listed with
its details.
 * The method sets up columns for each field in the TravelRecordDTO and populates them
with data.
 */
void GraphicalUserInterface::RenderListRecordsGUI(TravelRecordService &service) {
    auto records = service.getAllRecords();

    ImVec2 fullscreen = ImGui::GetIO().DisplaySize;

    ImGui::SetNextWindowPos(ImVec2(0, 0));
    ImGui::SetNextWindowSize(fullscreen);

    ImGui::Begin("List of Records", nullptr,
                 ImGuiWindowFlags_NoTitleBar | ImGuiWindowFlags_NoResize |
ImGuiWindowFlags_NoMove);
    if (ImGui::BeginTable("Records", 14)) {
        ImGui::TableSetupColumn("Ref Number");
        ImGui::TableSetupColumn("Disclosure Group");
        ImGui::TableSetupColumn("Title");
        ImGui::TableSetupColumn("Name");
        ImGui::TableSetupColumn("Purpose");
        ImGui::TableSetupColumn("Start Date");
        ImGui::TableSetupColumn("End Date");
        ImGui::TableSetupColumn("Destination");
        ImGui::TableSetupColumn("Airfare");
        ImGui::TableSetupColumn("Other Transport");
        ImGui::TableSetupColumn("Lodging");
        ImGui::TableSetupColumn("Meals");
        ImGui::TableSetupColumn("Other Expenses");
        ImGui::TableSetupColumn("Total");

        for (const auto &record: records) {
            ImGui::TableNextRow();
            ImGui::TableSetColumnIndex(0);
            ImGui::Text("%s", record.getRefNumber().c_str());
            ImGui::TableSetColumnIndex(1);
```

```
            ImGui::Text("%s", record.getDisclosureGroup().c_str());
            ImGui::TableSetColumnIndex(2);
            ImGui::Text("%s", record.getTitle().c_str());
            ImGui::TableSetColumnIndex(3);
            ImGui::Text("%s", record.getName().c_str());
            ImGui::TableSetColumnIndex(4);
            ImGui::Text("%s", record.getPurpose().c_str());
            ImGui::TableSetColumnIndex(5);
            ImGui::Text("%s", record.getStartDate().c_str());
            ImGui::TableSetColumnIndex(6);
            ImGui::Text("%s", record.getEndDate().c_str());
            ImGui::TableSetColumnIndex(7);
            ImGui::Text("%s", record.getDestination().c_str());
            ImGui::TableSetColumnIndex(8);
            ImGui::Text("%.2f", record.getAirfare());
            ImGui::TableSetColumnIndex(9);
            ImGui::Text("%.2f", record.getOtherTransport());
            ImGui::TableSetColumnIndex(10);
            ImGui::Text("%.2f", record.getLodging());
            ImGui::TableSetColumnIndex(11);
            ImGui::Text("%.2f", record.getMeals());
            ImGui::TableSetColumnIndex(12);
            ImGui::Text("%.2f", record.getOtherExpenses());
            ImGui::TableSetColumnIndex(13);
            ImGui::Text("%.2f", record.getTotal());
        }
        ImGui::EndTable();
    }
    ImGui::End();
}

/**
 * @brief Renders the GUI for updating an existing travel record.
 *
 * This method initializes a static TravelRecordDTO object with the record to update
based on the provided reference number.
 * It renders a graphical user interface using ImGui for the user to update the
record's data.
 * The method handles the input for various fields and updates the record in the
service when the 'Update Record' button is clicked.
 *
 * @param refNumber The reference number of the record to be updated.
 */
void GraphicalUserInterface::RenderUpdateRecordGUI(const std::string &refNumber) {
    static TravelRecordDTO recordToUpdate = service.getRecordById(refNumber);
    static char refNumberBuffer[100];
    static char disclosureGroupBuffer[100];
    static char titleBuffer[100];
    static char nameBuffer[100];
    static char purposeBuffer[100];
    static char startDateBuffer[100];
    static char endDateBuffer[100];
    static char destinationBuffer[100];
    static double airfare = 0.0;
    static double otherTransport = 0.0;
    static double lodging = 0.0;
    static double meals = 0.0;
    static double otherExpenses = 0.0;
    static double total = 0.0;

    // Initialize buffers with current values
    strcpy(refNumberBuffer, recordToUpdate.getRefNumber().c_str());
    strcpy(disclosureGroupBuffer, recordToUpdate.getDisclosureGroup().c_str());
```

```
    strcpy(titleBuffer, recordToUpdate.getTitle().c_str());
    strcpy(nameBuffer, recordToUpdate.getName().c_str());
    strcpy(purposeBuffer, recordToUpdate.getPurpose().c_str());
    strcpy(startDateBuffer, recordToUpdate.getStartDate().c_str());
    strcpy(endDateBuffer, recordToUpdate.getEndDate().c_str());
    strcpy(destinationBuffer, recordToUpdate.getDestination().c_str());
    airfare = recordToUpdate.getAirfare();
    otherTransport = recordToUpdate.getOtherTransport();
    lodging = recordToUpdate.getLodging();
    meals = recordToUpdate.getMeals();
    otherExpenses = recordToUpdate.getOtherExpenses();
    total = recordToUpdate.getTotal();

    ImVec2 fullscreen = ImGui::GetIO().DisplaySize;

    ImGui::SetNextWindowPos(ImVec2(0, 0));
    ImGui::SetNextWindowSize(fullscreen);

    ImGui::Begin("Update Record", nullptr,
                 ImGuiWindowFlags_NoTitleBar | ImGuiWindowFlags_NoResize |
ImGuiWindowFlags_NoMove);


    ImGui::InputText("Reference Number", refNumberBuffer,
IM_ARRAYSIZE(refNumberBuffer));
    ImGui::InputText("Disclosure Group", disclosureGroupBuffer,
IM_ARRAYSIZE(disclosureGroupBuffer));
    ImGui::InputText("Title", titleBuffer, IM_ARRAYSIZE(titleBuffer));
    ImGui::InputText("Name", nameBuffer, IM_ARRAYSIZE(nameBuffer));
    ImGui::InputText("Purpose", purposeBuffer, IM_ARRAYSIZE(purposeBuffer));
    ImGui::InputText("Start Date", startDateBuffer, IM_ARRAYSIZE(startDateBuffer));
    ImGui::InputText("End Date", endDateBuffer, IM_ARRAYSIZE(endDateBuffer));
    ImGui::InputText("Destination", destinationBuffer,
IM_ARRAYSIZE(destinationBuffer));
    ImGui::InputDouble("Airfare", &airfare, 0.0, 0.0, "%.1f");
    ImGui::InputDouble("Other Transport", &otherTransport, 0.0, 0.0, "%.1f");
    ImGui::InputDouble("Lodging", &lodging, 0.0, 0.0, "%.1f");
    ImGui::InputDouble("Meals", &meals, 0.0, 0.0, "%.1f");
    ImGui::InputDouble("Other Expenses", &otherExpenses, 0.0, 0.0, "%.1f");
    ImGui::InputDouble("Total", &total, 0.0, 0.0, "%.1f");

    if (ImGui::Button("Update Record")) {
        recordToUpdate.setRefNumber(std::string(refNumberBuffer));
        recordToUpdate.setDisclosureGroup(std::string(disclosureGroupBuffer));
        recordToUpdate.setTitle(std::string(titleBuffer));
        recordToUpdate.setName(std::string(nameBuffer));
        recordToUpdate.setPurpose(std::string(purposeBuffer));
        recordToUpdate.setStartDate(std::string(startDateBuffer));
        recordToUpdate.setEndDate(std::string(endDateBuffer));
        recordToUpdate.setDestination(std::string(destinationBuffer));
        recordToUpdate.setAirfare(airfare);
        recordToUpdate.setOtherTransport(otherTransport);
        recordToUpdate.setLodging(lodging);
        recordToUpdate.setMeals(meals);
        recordToUpdate.setOtherExpenses(otherExpenses);
        recordToUpdate.setTotal(total);

        service.updateRecord(recordToUpdate);
        ImGui::Text("Record updated successfully");
    }

    ImGui::End();
}
```

```cpp
/**
 * @brief Renders the GUI for deleting a travel record.
 *
 * This method provides an interface for the user to input the reference number of the
record they wish to delete.
 * It uses ImGui to create the graphical interface and deletes the record from the
service when the 'Delete Record' button is clicked.
 *
 * @param service The TravelRecordService instance used for deleting the record.
 */
void GraphicalUserInterface::RenderDeleteRecordGUI(TravelRecordService &service) {
    static char refNumberToDelete[50];

    ImVec2 fullscreen = ImGui::GetIO().DisplaySize;

    ImGui::SetNextWindowPos(ImVec2(0, 0));
    ImGui::SetNextWindowSize(fullscreen);

    ImGui::Begin("Delete Record", nullptr,
                 ImGuiWindowFlags_NoTitleBar | ImGuiWindowFlags_NoResize |
ImGuiWindowFlags_NoMove);

    ImGui::InputText("Reference Number", refNumberToDelete,
IM_ARRAYSIZE(refNumberToDelete));

    if (ImGui::Button("Delete Record")) {
        service.deleteRecordById(std::string(refNumberToDelete));
        ImGui::Text("Record deleted successfully");
    }

    ImGui::End();
}

/**
 * @brief Renders a bar chart of travel expenses.
 *
 * This method fetches various travel expense statistics like the highest, median, and
lowest expenses for different categories.
 * It uses ImPlot to render a bar chart visualizing these statistics for comparison.
 *
 * @param service The TravelRecordService instance used for fetching expense
statistics.
 */
void GraphicalUserInterface::BarChart(TravelRecordService &service) {
    int totalHighest = service.getTotalHighest();
    int totalMedian = service.getTotalMedian();
    int totalLowest = service.getTotalLowest();
    int airfareHighest = service.getAirfareHighest();
    int airfareMedian = service.getAirfareMedian();
    int airfareLowest = service.getAirfareLowest();
    int mealsHighest = service.getMealsHighest();
    int mealsMedian = service.getMealsMedian();
    int mealsLowest = service.getMealsLowest();

    ImGui::Begin("Travel Expenses Bar Chart");

    if (ImPlot::BeginPlot("Travel Expenses")) {


        float highestValues[] = {static_cast<float>(totalHighest),
static_cast<float>(airfareHighest),
                                 static_cast<float>(mealsHighest)};
```

```
        float medianValues[] = {static_cast<float>(totalMedian),
static_cast<float>(airfareMedian),
                                static_cast<float>(mealsMedian)};
        float lowestValues[] = {static_cast<float>(totalLowest),
static_cast<float>(airfareLowest),
                                static_cast<float>(mealsLowest)};

        ImPlot::PlotBars("Highest", highestValues, 3, 0.3f, 0.15f);
        ImPlot::PlotBars("Median", medianValues, 3, 0.3f, 0.75f);
        ImPlot::PlotBars("Lowest", lowestValues, 3, 0.3f, 1.35f);

        ImPlot::EndPlot();
    }
    ImGui::End();
}

/**
 * @brief Renders a pie chart of travel expenses.
 *
 * This method fetches median expenses for different categories like total, airfare,
and meals.
 * It uses ImPlot to render a pie chart for a visual representation of the proportion
of each category in total expenses.
 *
 * @param service The TravelRecordService instance used for fetching expense
statistics.
 */
void GraphicalUserInterface::PieChart(TravelRecordService &service) {

    int totalMedian = service.getTotalMedian();
    int airfareMedian = service.getAirfareMedian();
    int mealsMedian = service.getMealsMedian();

    ImGui::Begin("Travel Expenses Pie Chart");


    if (ImPlot::BeginPlot("##PieChart", ImVec2(-1, 0))) {
        const char *categories[] = {"Total", "Airfare", "Meals"};
        double values[] = {static_cast<double>(totalMedian),
static_cast<double>(airfareMedian),
                           static_cast<double>(mealsMedian)};
        int numCategories = sizeof(categories) / sizeof(categories[0]);

        ImPlot::PlotPieChart(categories, values, numCategories, 0.5, 0.5, 0.4);

        ImPlot::EndPlot();
    }
    ImGui::End();
}
#ifndef ASSIGNMENT03_GRAPHICALUSERINTERFACE_H
#define ASSIGNMENT03_GRAPHICALUSERINTERFACE_H

#include "TravelRecordService.h"
#include "TravelRecordDTO.h"
#include <string>

class GraphicalUserInterface {
private:
    TravelRecordService service;
public:
    GraphicalUserInterface(TravelRecordService &service) : service(service) {}

    void RenderAddNewRecordGUI();
```

```
    void RenderListRecordsGUI(TravelRecordService &service);

    void RenderUpdateRecordGUI(const std::string &refNumber);

    void RenderDeleteRecordGUI(TravelRecordService &service);

    void BarChart(TravelRecordService &service);

    void PieChart(TravelRecordService &service);
};

#endif //ASSIGNMENT03_GRAPHICALUSERINTERFACE_H
#include "TravelRecordController.h"
#include "imgui_impl_opengl3.h"
#include "imgui_impl_glfw.h"
#include "../dependencies/implot/implot.h"

using namespace std;

/**
 * @brief Constructs the TravelRecordController object.
 *
 * Initializes the controller with a reference to a TravelRecordService object.
 * Sets up a window pointer for the graphical interface and an instance of
GraphicalUserInterface.
 *
 * @param service Reference to a TravelRecordService object used for managing travel
records.
 */
TravelRecordController::TravelRecordController(TravelRecordService &service)
        : service(service), window(nullptr), gui(service) {}

/**
* @brief Runs the command-line interface for the travel record system.
*
* This method creates an instance of TerminalUI and calls its showMenu method,
* effectively starting the command-line interface.
*/
void TravelRecordController::runCommandLineInterface() {
    TerminalUI ui(service);
    ui.showMenu();
}

/**
 * @brief Runs the graphical interface with a specified render function.
 *
 * This method initializes the window, runs the main loop with the given render
function,
 * and then cleans up the window after the loop terminates.
 *
 * @param renderFunc A function that takes a GraphicalUserInterface reference and
returns void.
 */
void TravelRecordController::runInterface(function<void(GraphicalUserInterface &)>
renderFunc) {
    initializeWindow();
    mainLoop(renderFunc);
    cleanupWindow();
}

/**
 * @brief Runs the interface for adding a new travel record.
```

```
 *
 * This method uses the `runInterface` function with a lambda expression to render the
GUI
 * for adding a new travel record using
`GraphicalUserInterface::RenderAddNewRecordGUI`.
 */
void TravelRecordController::runAddRecordInterface() {
    runInterface([](GraphicalUserInterface &gui) { gui.RenderAddNewRecordGUI(); });
}

/**
 * @brief Runs the interface for updating an existing travel record.
 *
 * This method uses the `runInterface` function with a lambda expression to render the
GUI
 * for updating a travel record, specified by its reference number, using
`GraphicalUserInterface::RenderUpdateRecordGUI`.
 *
 * @param refNumber The reference number of the record to be updated.
 */
void TravelRecordController::runUpdateRecordInterface(const string &refNumber) {
    runInterface([&refNumber](GraphicalUserInterface &gui) {
gui.RenderUpdateRecordGUI(refNumber); });
}

/**
 * @brief Runs the interface for deleting a travel record.
 *
 * This method uses the `runInterface` function with a lambda expression to render the
GUI
 * for deleting a travel record using `GraphicalUserInterface::RenderDeleteRecordGUI`.
 */
void TravelRecordController::runDeleteRecordInterface() {
    runInterface([this](GraphicalUserInterface &gui) {
gui.RenderDeleteRecordGUI(service); });
}

/**
 * @brief Runs the interface for listing all travel records.
 *
 * This method uses the `runInterface` function with a lambda expression to render the
GUI
 * for listing all travel records using `GraphicalUserInterface::RenderListRecordsGUI`.
 */
void TravelRecordController::runListRecordsInterface() {
    runInterface([this](GraphicalUserInterface &gui) {
gui.RenderListRecordsGUI(service); });
}

/**
 * @brief Runs the interface for displaying a bar chart of travel records.
 *
 * This method uses the `runInterface` function with a lambda expression to render a
bar chart GUI
 * showing various statistics of travel records using
`GraphicalUserInterface::BarChart`.
 */
void TravelRecordController::runBarChartInterface() {
    runInterface([this](GraphicalUserInterface &gui) { gui.BarChart(service); });
}

/**
 * @brief Runs the interface for displaying a pie chart of travel records.
```

```
 *
 * This method uses the `runInterface` function with a lambda expression to render a
pie chart GUI
 * showing the distribution of expenses in travel records using
`GraphicalUserInterface::PieChart`.
 */
void TravelRecordController::runPieChartInterface() {
    runInterface([this](GraphicalUserInterface &gui) { gui.PieChart(service); });
}

/**
 * @brief Initializes the GLFW window and ImGui context for rendering the GUI.
 *
 * This method sets up the necessary components for rendering the graphical user
interface.
 * It initializes GLFW, creates a window, sets up ImGui and ImPlot contexts, and
configures their styles.
 */
void TravelRecordController::initializeWindow() {
    if (!glfwInit()) {
        cerr << "Failed to initialize GLFW" << endl;
        exit(-1);
    }

    window = glfwCreateWindow(800, 600, "Travel Record System", NULL, NULL);
    if (window == NULL) {
        cerr << "Failed to create GLFW window" << endl;
        glfwTerminate();
        exit(-1);
    }

    glfwMakeContextCurrent(window);
    IMGUI_CHECKVERSION();
    ImGui::CreateContext();
    ImPlot::CreateContext();
    ImGui::StyleColorsDark();
    ImGui_ImplGlfw_InitForOpenGL(window, true);
    ImGui_ImplOpenGL3_Init("#version 130");
}

/**
 * @brief Cleans up the window and ImGui context after use.
 *
 * This method properly shuts down ImGui, ImPlot, and GLFW, and destroys the created
window.
 * It's called after the main loop of the graphical interface ends.
 */
void TravelRecordController::cleanupWindow() {
    ImGui_ImplOpenGL3_Shutdown();
    ImGui_ImplGlfw_Shutdown();
    ImPlot::DestroyContext();
    ImGui::DestroyContext();
    glfwDestroyWindow(window);
    glfwTerminate();
}

/**
 * @brief Runs the main loop for the graphical user interface.
 *
 * In this loop, GLFW events are polled, ImGui frames are prepared and rendered,
 * and the specified render function is called to render the GUI.
 * The loop continues until the window should close.
 *
```

```
 * @param renderFunction A function that takes a GraphicalUserInterface reference and
returns void.
 */
void TravelRecordController::mainLoop(function<void(GraphicalUserInterface &)>
renderFunction) {
    while (!glfwWindowShouldClose(window)) {
        glfwPollEvents();
        ImGui_ImplOpenGL3_NewFrame();
        ImGui_ImplGlfw_NewFrame();
        ImGui::NewFrame();

        renderFunction(gui);

        ImGui::Render();
        int display_w, display_h;
        glfwGetFramebufferSize(window, &display_w, &display_h);
        glViewport(0, 0, display_w, display_h);
        glClearColor(0.45f, 0.55f, 0.60f, 1.00f);
        glClear(GL_COLOR_BUFFER_BIT);
        ImGui_ImplOpenGL3_RenderDrawData(ImGui::GetDrawData());
        glfwSwapBuffers(window);
    }
}
#ifndef ASSIGNMENT03_TRAVELRECORDCONTROLLER_H
#define ASSIGNMENT03_TRAVELRECORDCONTROLLER_H

#include "TravelRecordService.h"
#include "GraphicalUserInterface.h"
#include "CMDUserInterface.h"
#include "GLFW/glfw3.h"
#include <functional>

class TravelRecordController {
public:
    TravelRecordController(TravelRecordService &service);

    void runCommandLineInterface();

    void runAddRecordInterface();

    void runListRecordsInterface();

    void runUpdateRecordInterface(const std::string &refNumber);

    void runDeleteRecordInterface();

    void runBarChartInterface();

    void runPieChartInterface();

    void initializeWindow();

    void cleanupWindow();

    void mainLoop(std::function<void(GraphicalUserInterface &)> renderFunction);

private:
    TravelRecordService &service;
    GLFWwindow *window;
    GraphicalUserInterface gui;

    void runInterface(std::function<void(GraphicalUserInterface &)> renderFunc);
};
```

```cpp
#endif //ASSIGNMENT03_TRAVELRECORDCONTROLLER_H
#include <limits>
#include "TravelRecordDAO.h"
#include "TravelRecordService.h"
#include "CMDUserInterface.h"
#include "TravelRecordController.h"

using namespace std;

/**
 * @brief Displays the welcome message for the application.
 */
void displayWelcomeMessage() {
    cout << "\n****************************************" << endl
         << "*     Welcome to Database Reader App!     *" << endl
         << "*         ( Travel Record System )        *" << endl
         << "****************************************" << endl
         << "\nBy Onur Onel 041074824\n" << endl;
}

/**
 * @brief Prompts the user to choose an action and retrieves their choice.
 * @return An integer representing the user's choice.
 */
int getUserChoice() {
    cout << "Choose Action: \n"
         << "1. CMD Mode \n"
         << "2. ADD Record \n"
         << "3. LIST Records \n"
         << "4. UPDATE Record \n"
         << "5. DELETE Record \n"
         << "6. Bar Chart \n"
         << "7. Pie Chart \n"
         << "Enter your choice: ";
    int choice;
    while (!(cin >> choice)) {
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        cout << "Invalid input. Please enter a number: ";
    }
    return choice;
}

/**
 * @brief The main function of the application.
 * @return Integer 0 upon successful execution.
 */
int main() {
    displayWelcomeMessage();
    int mode = getUserChoice();

    TravelRecordDAO dao;
    TravelRecordService service(dao);
    TravelRecordController controller(service);
    switch (mode) {
        case 1:
            controller.runCommandLineInterface();
            break;
        case 2:
            controller.runAddRecordInterface();
            break;
```

```
        case 3:
            controller.runListRecordsInterface();
            break;
        case 4:
            controller.runUpdateRecordInterface("1");
            break;
        case 5:
            controller.runDeleteRecordInterface();
            break;
        case 6:
            controller.runBarChartInterface();
            break;
        case 7:
            controller.runPieChartInterface();
            break;
        default:
            cout << "Invalid choice. Exiting." << endl;
            break;
    }
    return 0;
}
```