**Programming Language Research Practical Project 3**

Algonquin College

School of Advanced Technology

Computer Programming

Onur Önel

Mazin Abou-Seido

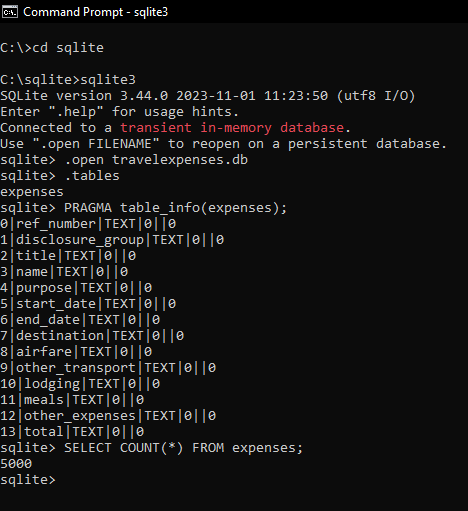
23F\_CST8333\_360

Submitted September 30, 2023

A technical report submitted to Algonquin College in partial fulfillment

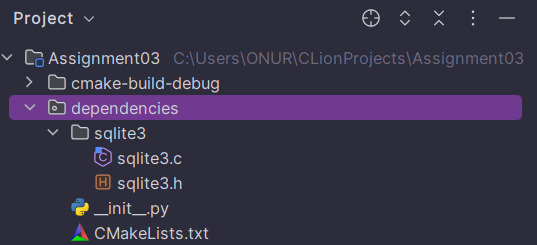
**Evidence of Learning**

Based on Practical Project 03 requirements I shifted from File-IO CSV data management to Database System Management using SQLITE3, which is lightweight, disk-based database that doesn't require a separate server process. I did this by first creating a database table, ensuring that each field in the table corresponded to a column in my existing dataset. Downloaded SQLite from official website <https://www.sqlite.org/draft/download.html>

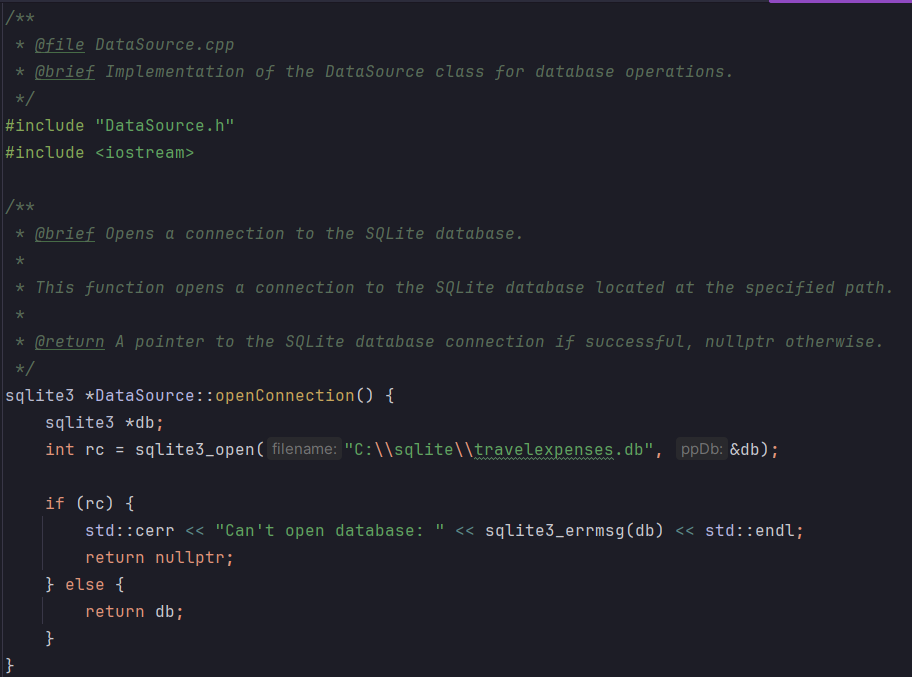


Finally, I completed the task of populating this database with data from my dataset. I utilized the `. mode csv` command followed by the `. import 'path.csv' expenses; ` command.

After successfully installing SQLite, I copied the sqlite3.c and sqlite3.h files into the dependency’s directory, enabling their inclusion and use in my project.



Moreover, I focused on the database connectivity through sqlite3. I did so by carefully selecting an appropriate database driver that allowed my program to interact seamlessly with the database. This setup enabled me to perform Create, Read, Update, and Delete operations directly on the database, which was a significant improvement over the previous file-based CSV system.



https://www.tutorialspoint.com/sqlite/sqlite\_c\_cpp.htm

**Project Structure**

**Model (TravelRecordDAO, DataSource, TravelRecordDTO)**

The Model component manages the data and business logic of the application. It includes:

* **TravelRecordDAO:** Responsible for direct interactions with the database. This class contains methods for CRUD (Create, Read, Update, Delete) operations on travel records.
* **DataSource:** Manages the database connection, specifically to the SQLite database.
* **TravelRecordDTO:** A Data Transfer Object that represents the structure of travel records. It is used to transfer data between the database and other parts of the application.

**View (UserInterface)**

The View component is responsible for the user interface. It interacts with the user, displaying information and capturing user inputs. In the application, the UserInterface class handles:

* Displaying menus and options to the user.
* Collecting user inputs for different functionalities like adding, updating, or deleting travel records.
* Invoking methods based on user choices.

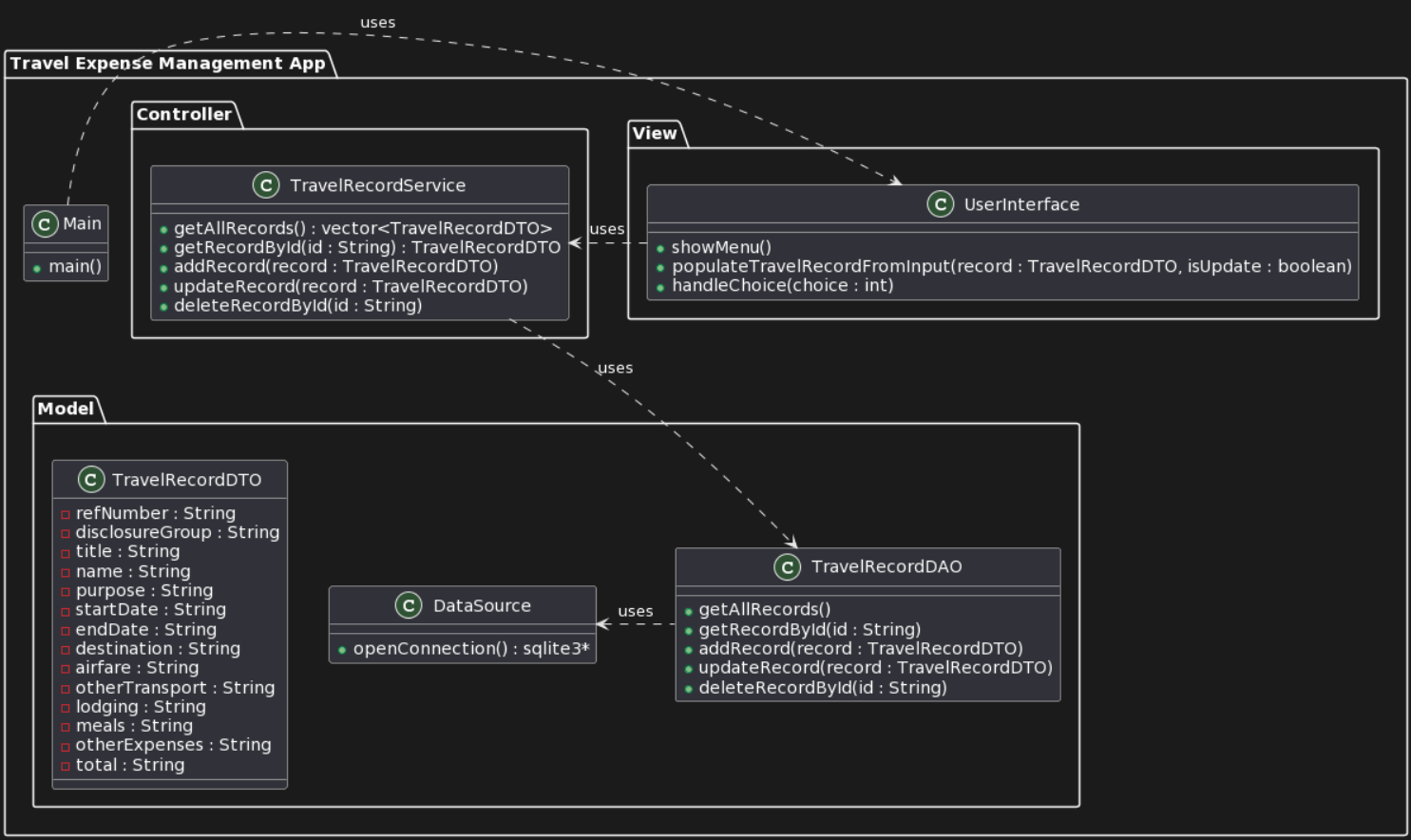
**Controller (TravelRecordService)**

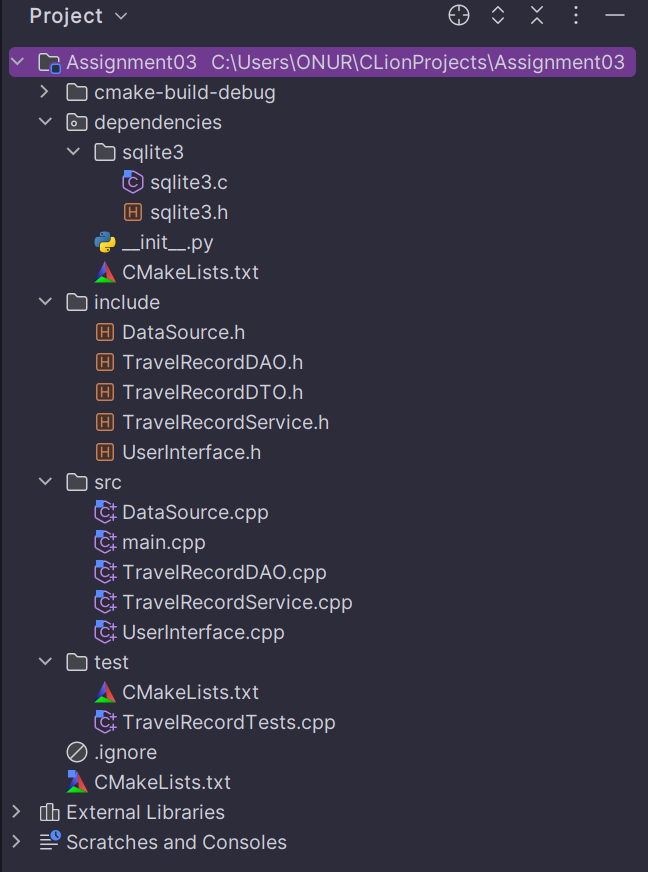
The Controller acts as an intermediary between the Model and the View. It processes user inputs and manipulates data using the Model component. The TravelRecordService in the application:

* Uses TravelRecordDAO for database operations.
* Processes data and handles the business logic.
* Sends processed data back to the UserInterface.

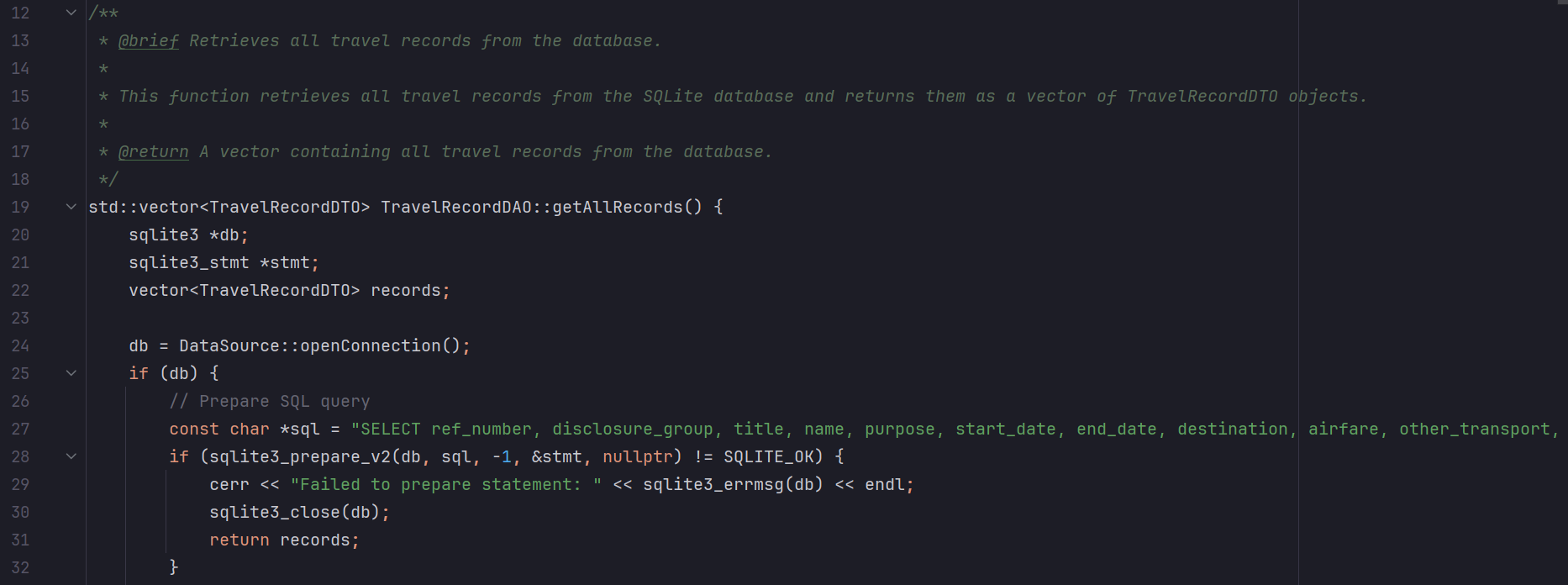
**Main (Entry Point)**

The Main function serves as the entry point of the application. It initializes necessary components like TravelRecordDAO, TravelRecordService, and UserInterface. It triggers the start of the application by calling the showMenu() method on the UserInterface.



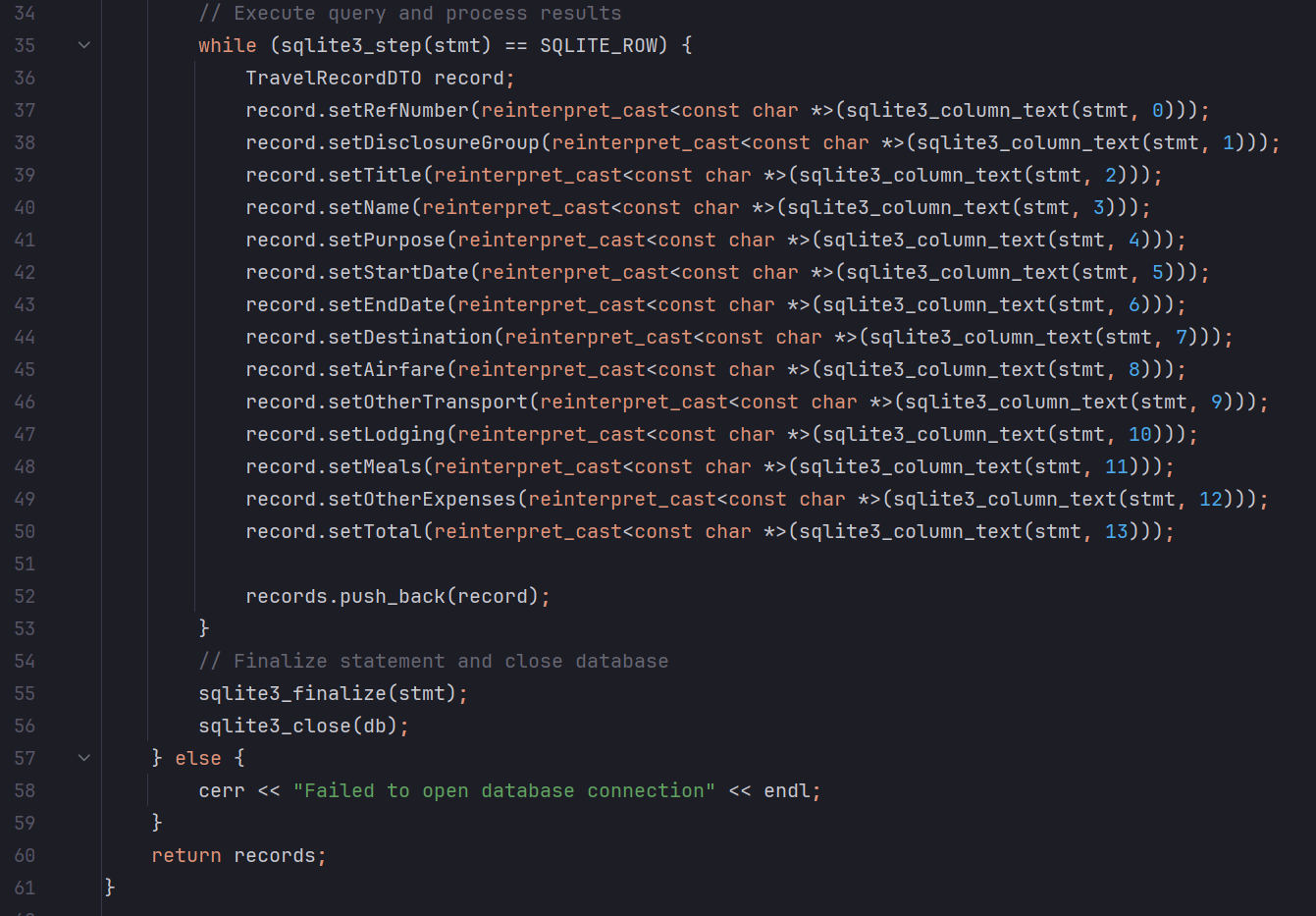
****

**TravelRecordDAO.cpp**

****

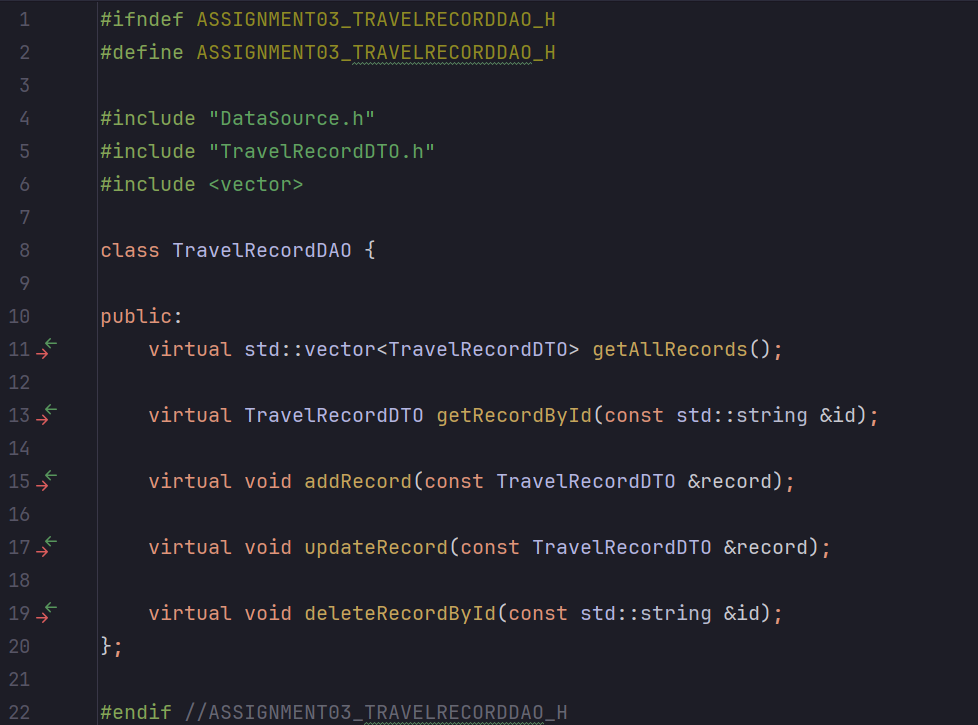
* **sqlite3 \*db;** - This variable represents a pointer to an SQLite database. It is used to establish a connection to the database, allowing for the execution of SQL queries.
* **sqlite3\_stmt \*stmt;** - This variable represents a pointer to an SQLite statement. It is used to prepare and execute SQL statements within the database.
* **vector<TravelRecordDTO> records;** - This variable is a container that will hold instances of the TravelRecordDTO data structure. It is intended to store the results retrieved from the database.

It begins by establishing a connection to the database using openConnection() function. If the connection is successful, it proceeds to prepare an SQL query to select specific columns from the "expenses" table, ordering the results by "ref\_number." Nextly, checks whether the SQL query preparation was successful using sqlite3\_prepare\_v2(). If the preparation is successful, it indicates readiness to execute the query. However, if an error occurs during the preparation, it prints an error message, closes the database connection, and returns an empty result.

****

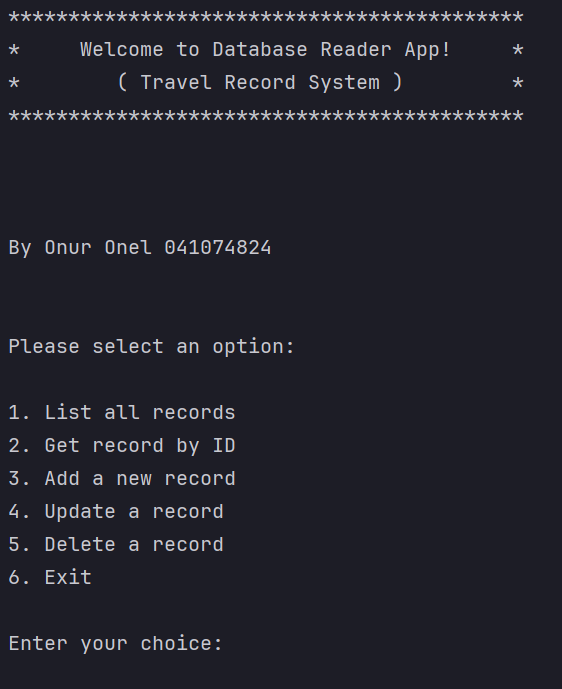
https://www.sqlite.org/c3ref/step.html

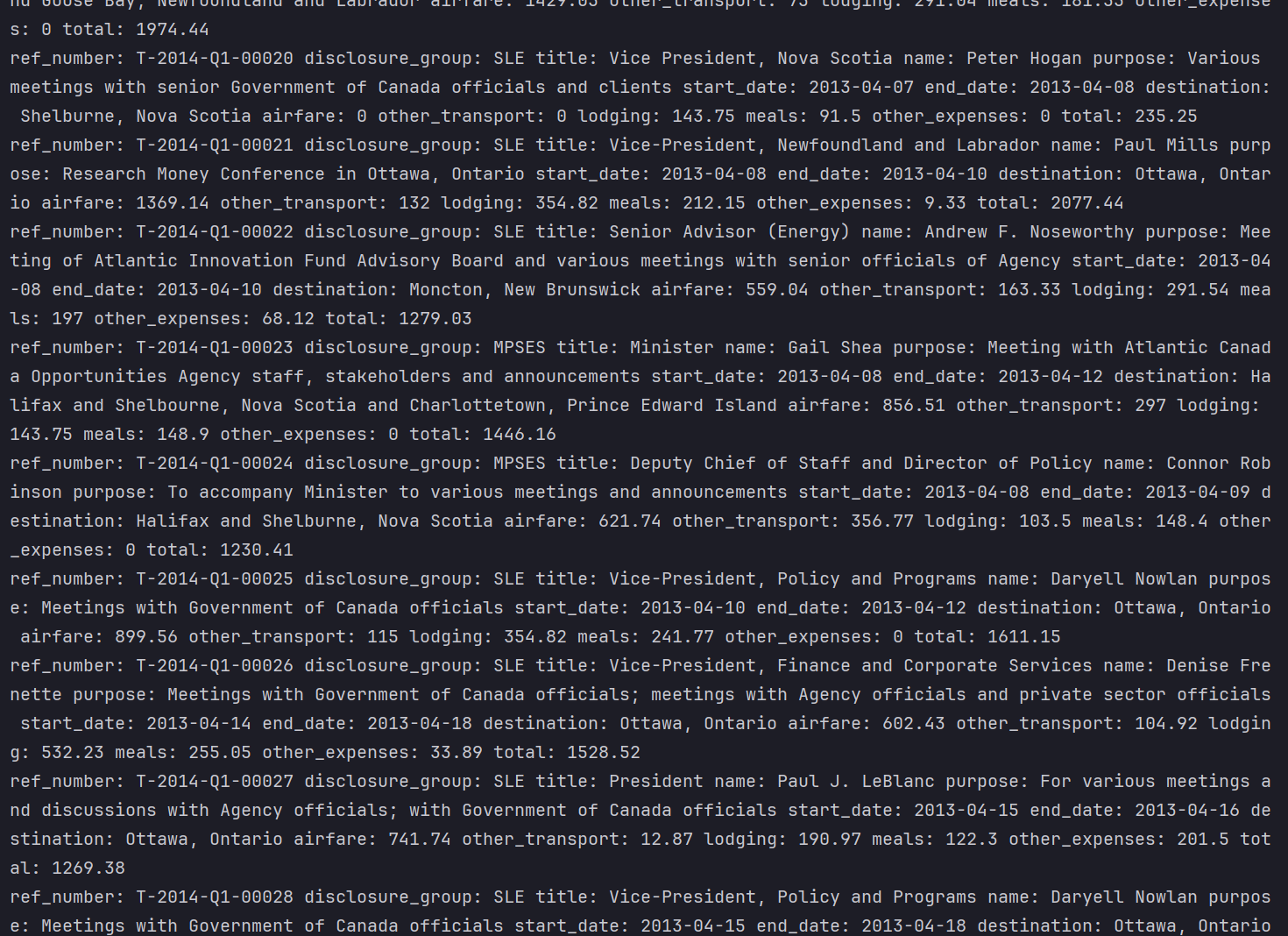
The Function lastly retrieves travel records from SQLite database and stores them in a vector of TravelRecordDTO objects. It executes an SQL query to fetch the records, processes each row, creates TravelRecordDTO instances with the retrieved data, and adds them to the vector. After processing all records, it finalizes the SQL statement and closes the database connection. If there's an issue opening the database connection, it outputs an error message. The vector of travel records is then returned as the result of this operation.

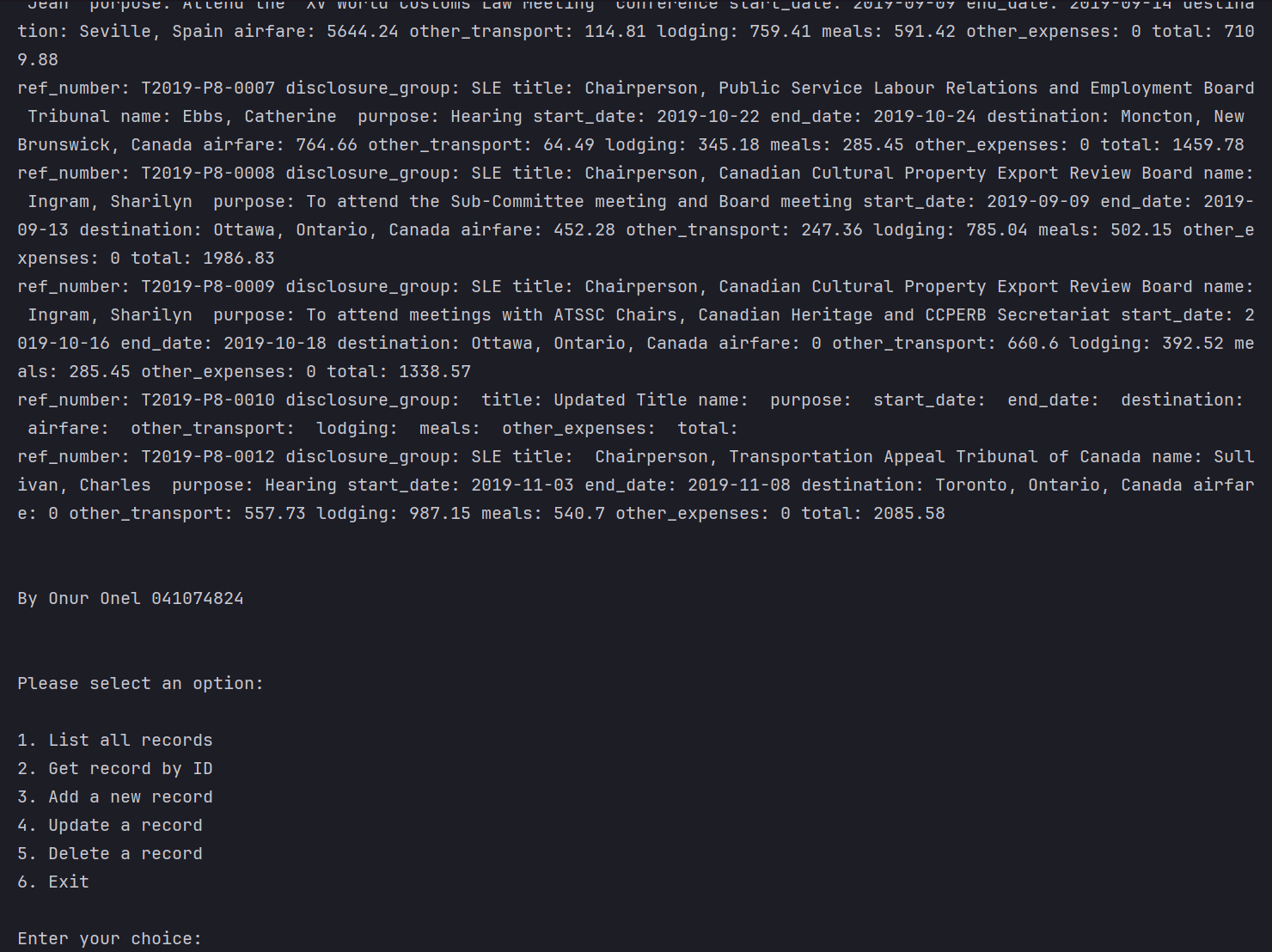
****

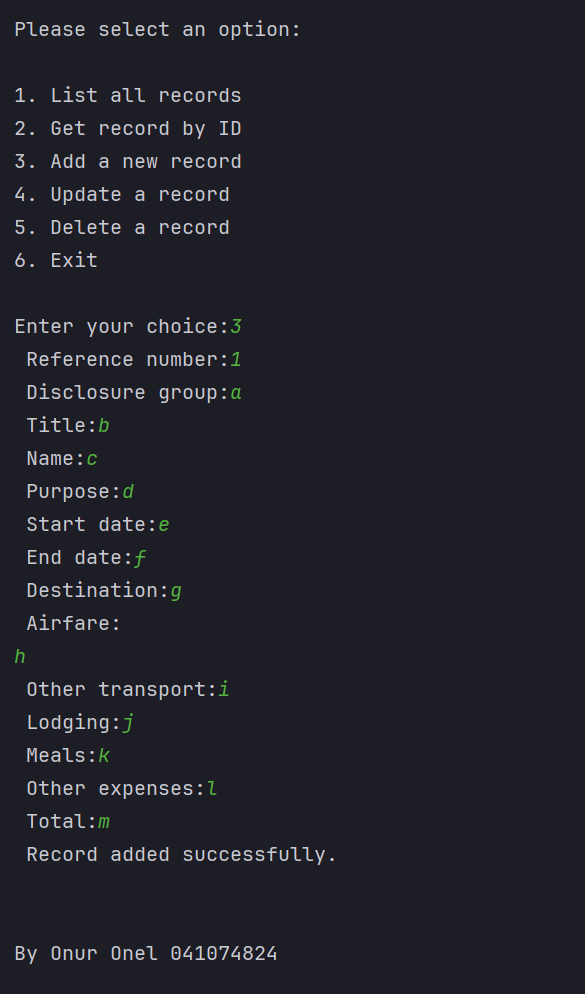
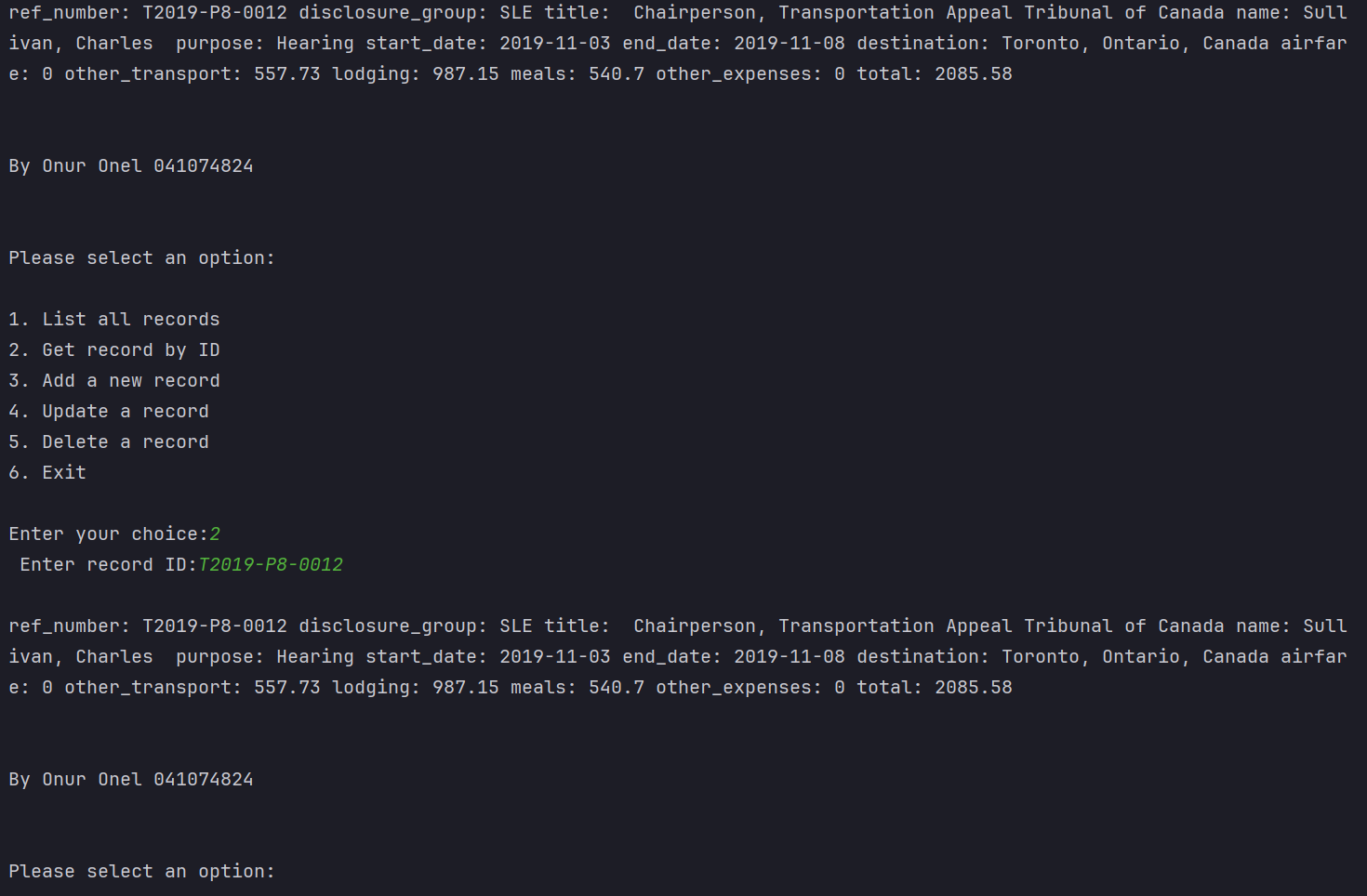
As previously discussed, the 'get all records' function serves as the foundational blueprint for other CRUD (Create, Read, Update, Delete) operations within the TravelRecordDAO class. While each CRUD function shares the core approach of establishing a database connection, preparing SQL statements, and executing them, they also incorporate specific adjustments tailored to their respective tasks. These modifications are subtle yet crucial, as they determine whether we retrieve, insert, update, or delete travel records. This consistent yet adaptable methodology ensures the efficiency and integrity of database operations, making it easier to manage travel records in a systematic and reliable manner.

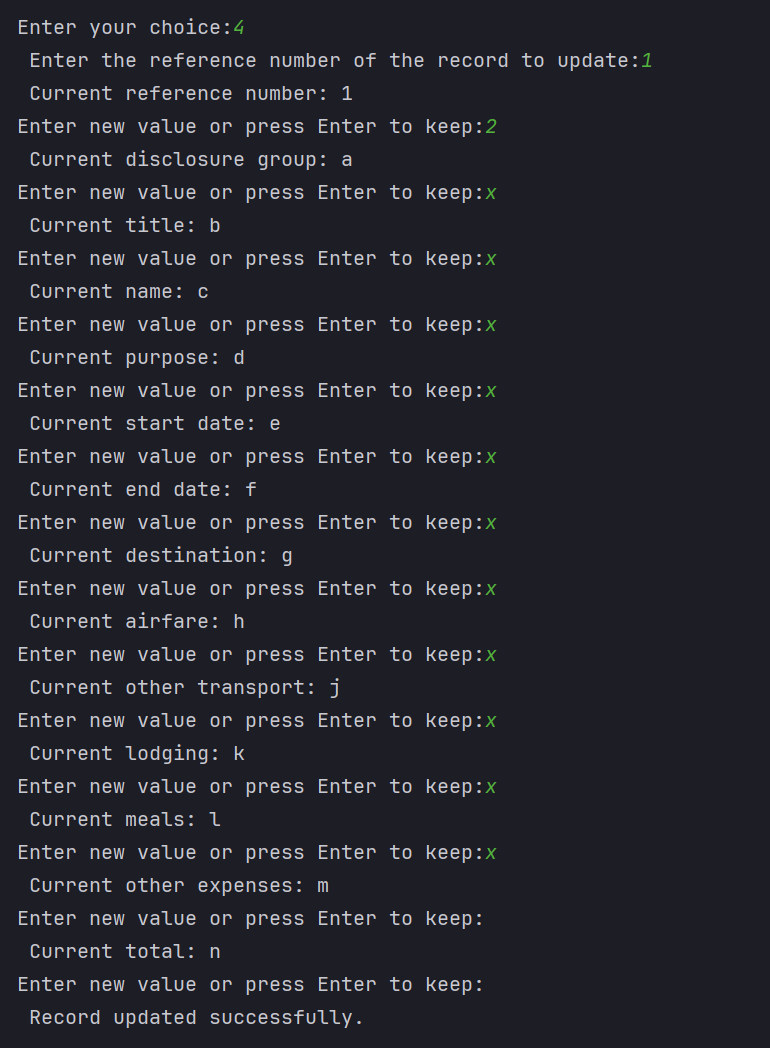
**Program Demonstration via Screen Shots**

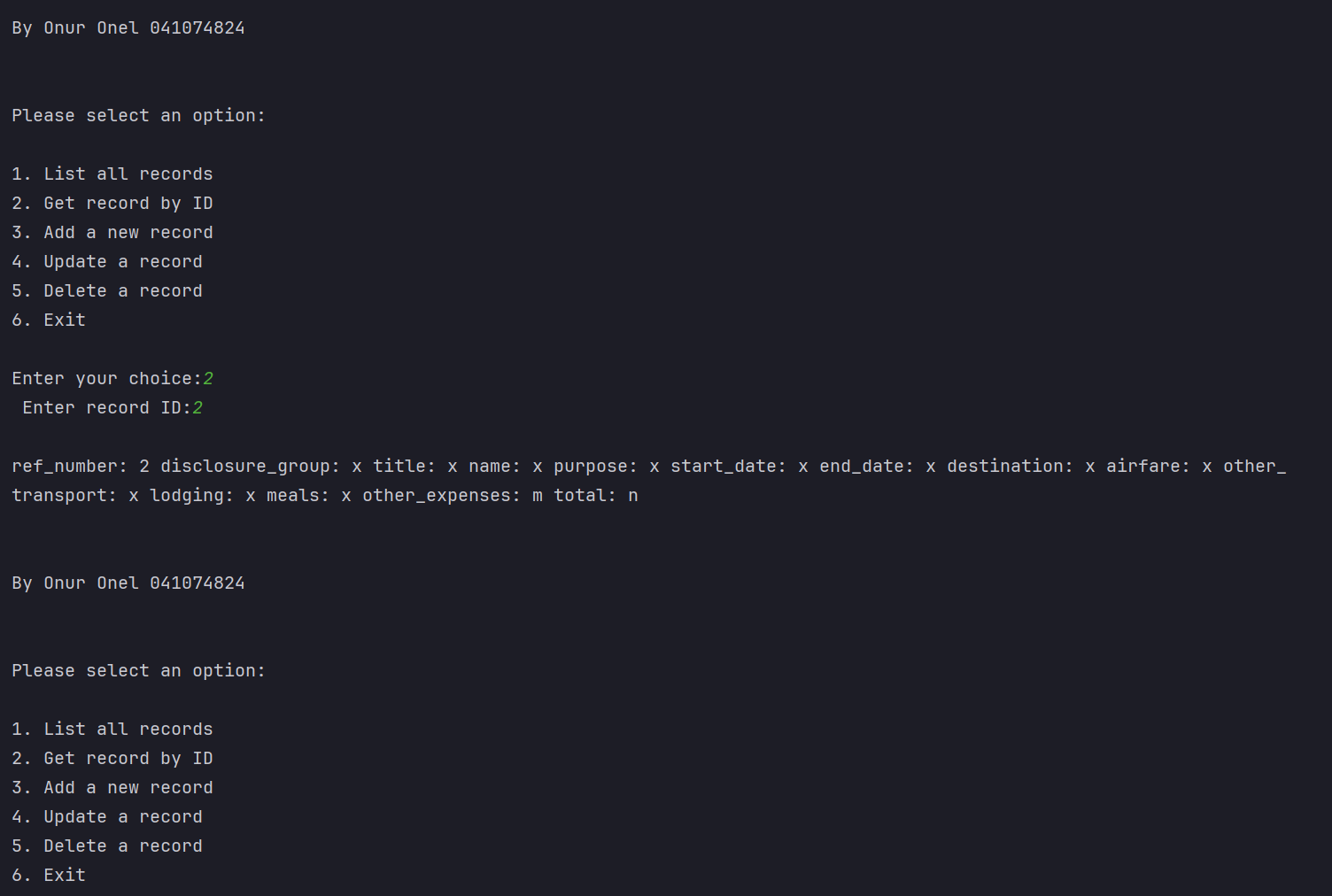
****

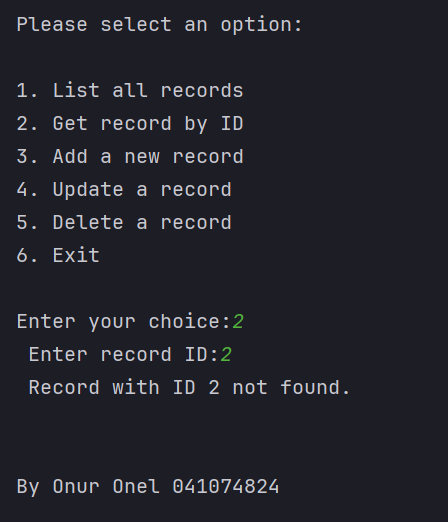
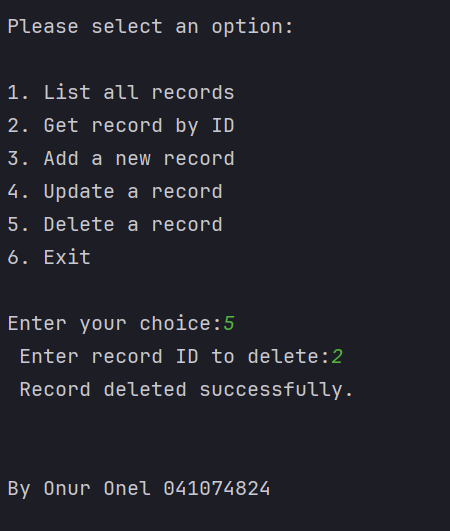
****

****

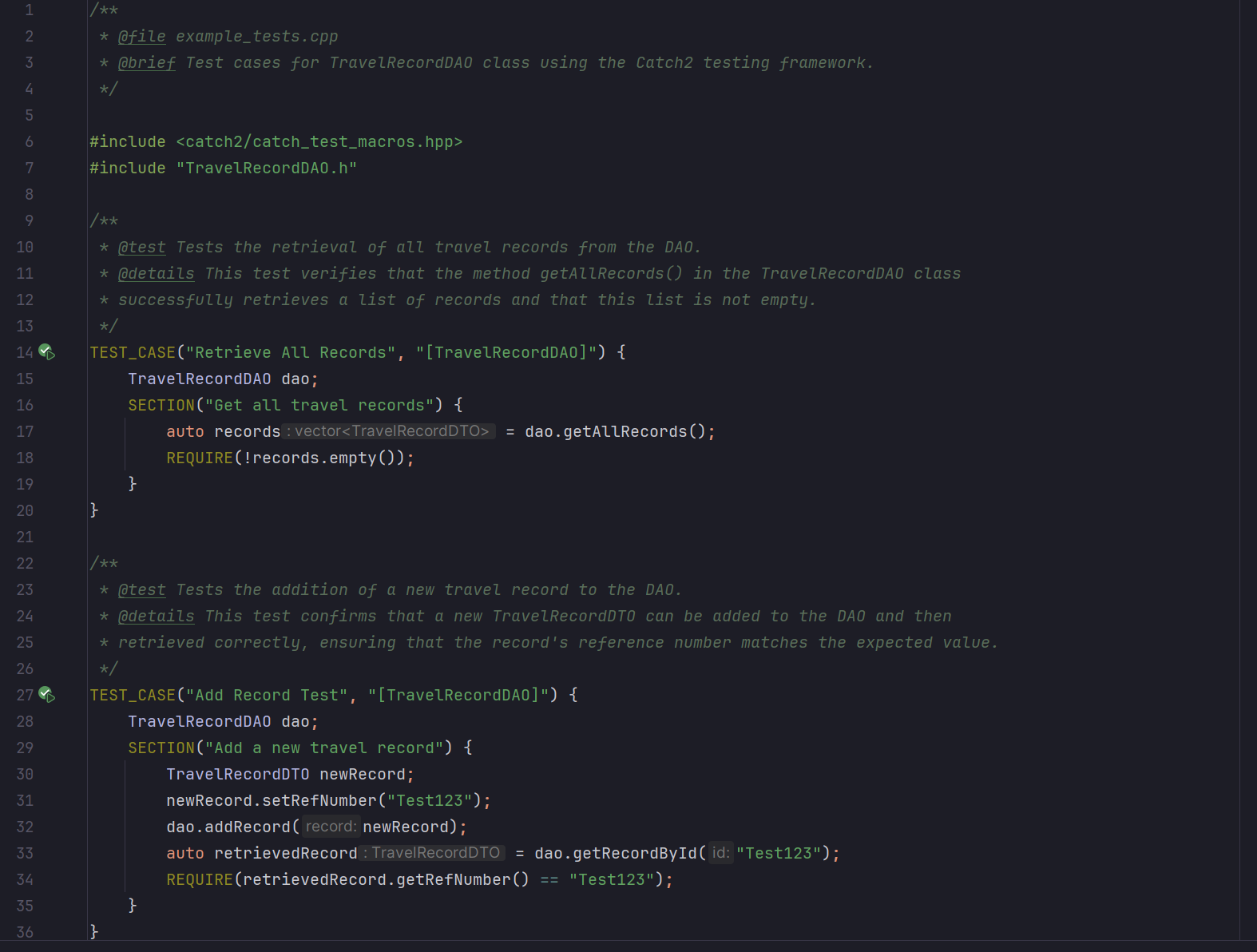
****

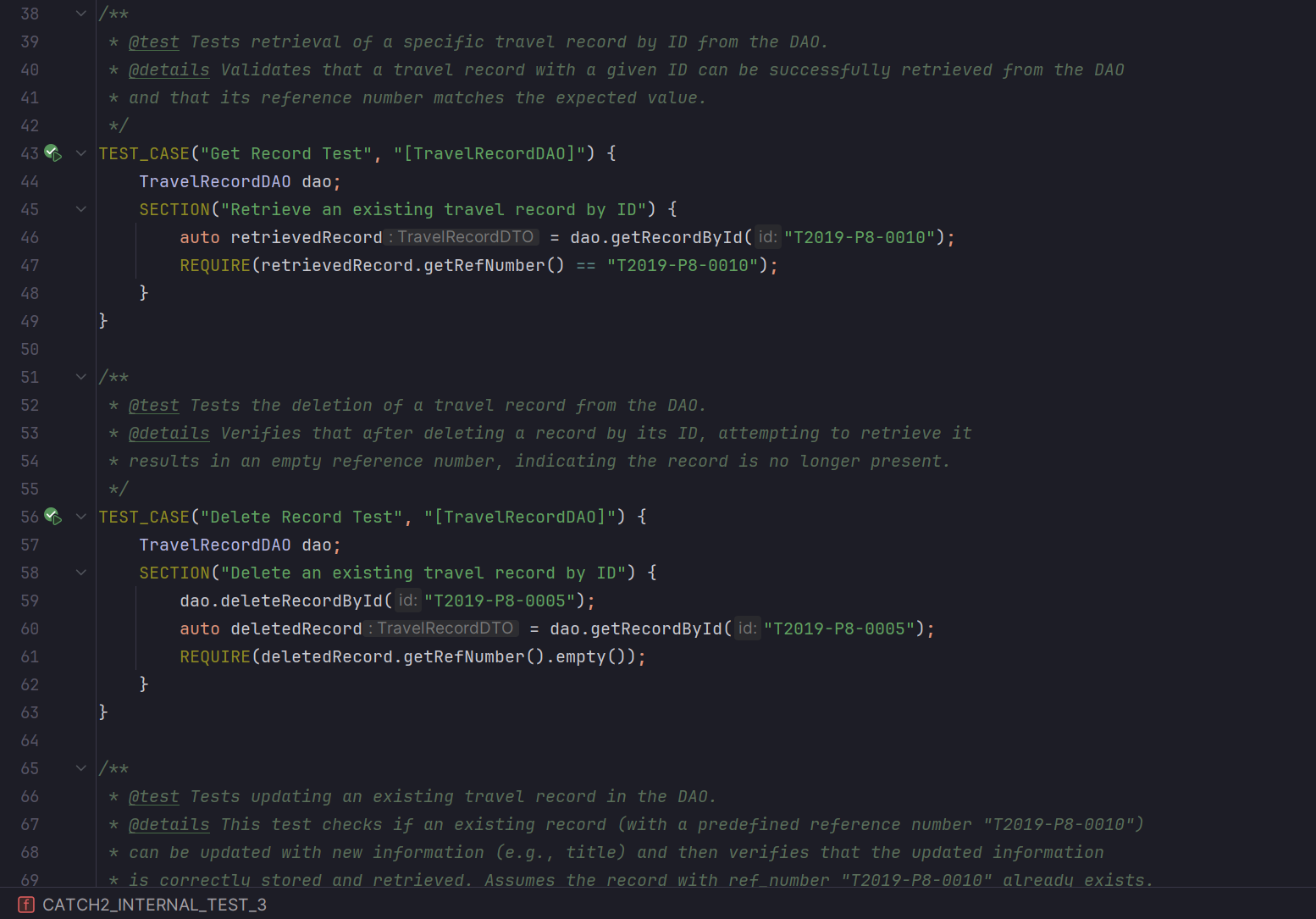
****

****

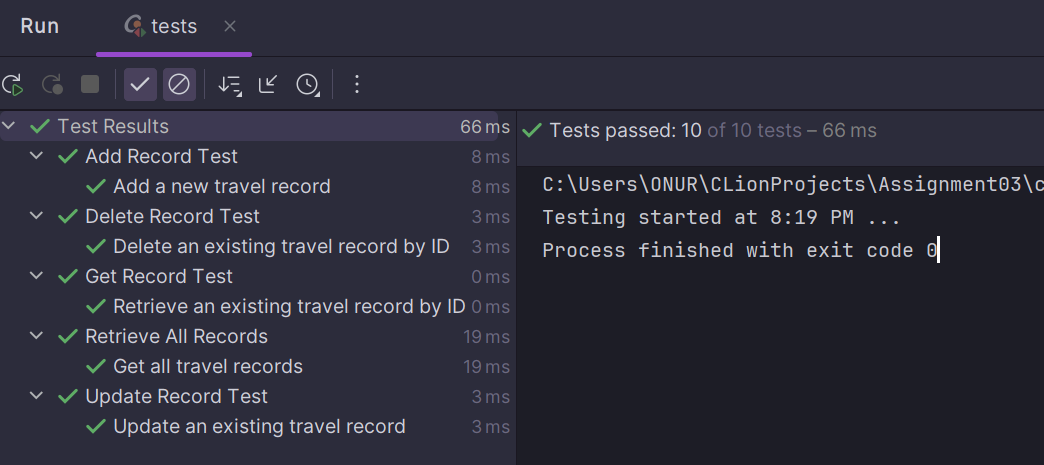
****

**Unit Testing Demonstration via Screen Shots**

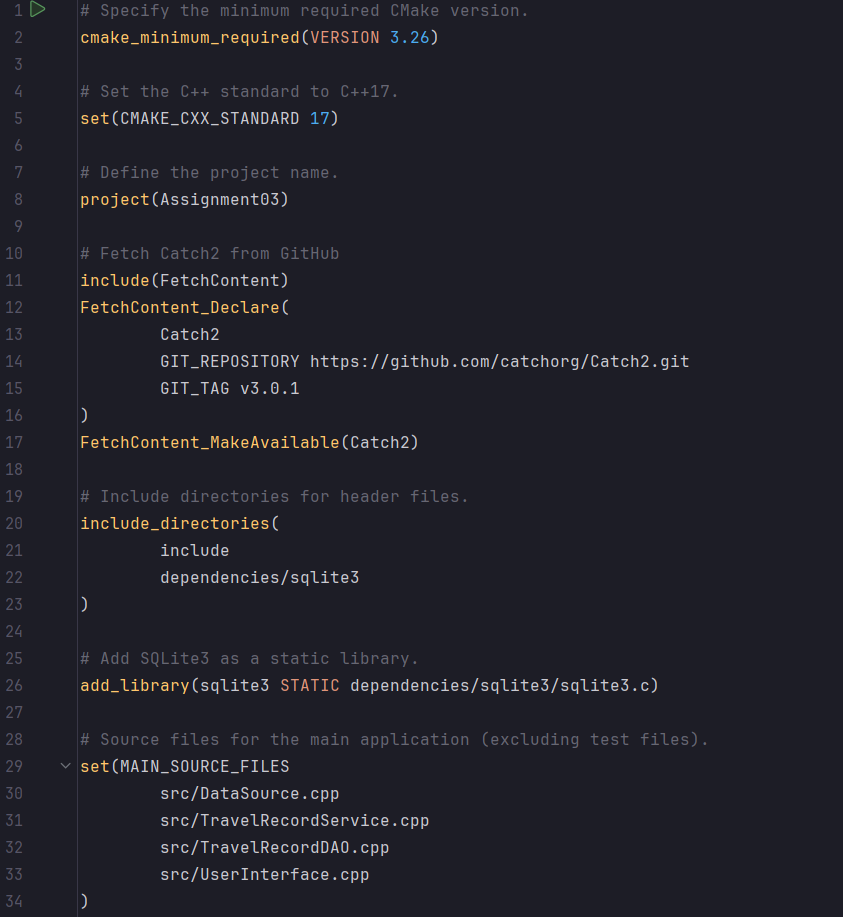
****

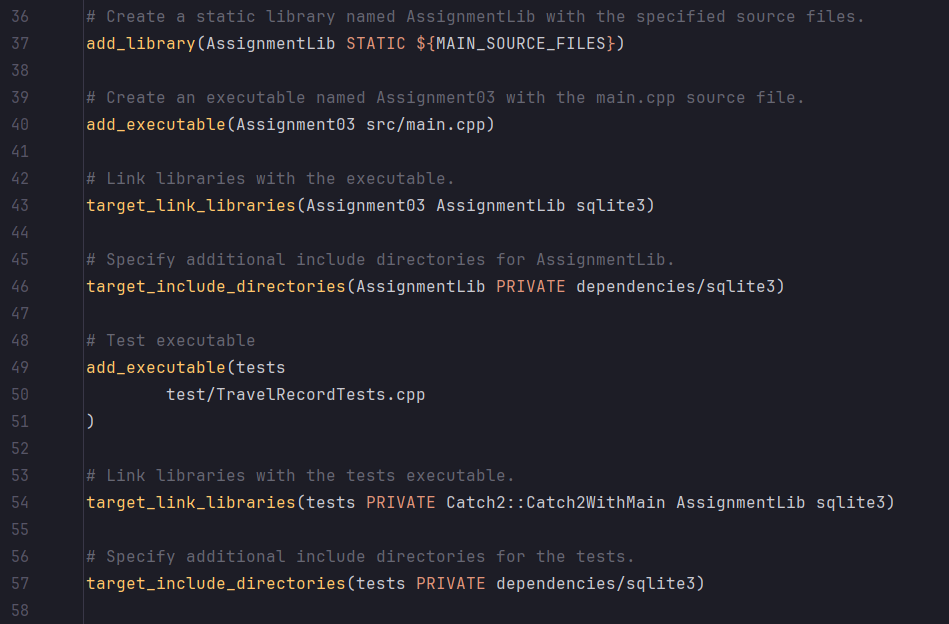
****

****

****

**Assignment03\CMakeLists.txt**

****

****

**References**

[1] SQLite download page, https://www.sqlite.org/draft/download.html

[2] “SQLite - C/C++,” Online Tutorials, Courses, and eBooks Library, https://www.tutorialspoint.com/sqlite/sqlite\_c\_cpp.htm

[3] Evaluate an SQL statement, https://www.sqlite.org/c3ref/step.html

[4] “SQLite: Clion,” CLion Help, https://www.jetbrains.com/help/clion/sqlite.html#connect-to-sqlite-database

**Source Code**

#ifndef ASSIGNMENT03\_DATASOURCE\_H  
#define ASSIGNMENT03\_DATASOURCE\_H  
  
#include <sqlite3.h>  
  
class DataSource {  
public:  
 static sqlite3 \*openConnection()**;**}**;**#endif // ASSIGNMENT03\_DATASOURCE\_H

/\*\*  
 \* @file DataSource.cpp  
 \* @brief Implementation of the DataSource class for database operations.  
 \*/  
  
#include "DataSource.h"  
#include <iostream>  
  
/\*\*  
 \* @brief Opens a connection to the SQLite database.  
 \*  
 \* This function opens a connection to the SQLite database located at the specified path.  
 \*  
 \* @return A pointer to the SQLite database connection if successful, nullptr otherwise.  
 \*/  
sqlite3 \*DataSource::openConnection() {  
 sqlite3 \*db**;** int rc = sqlite3\_open("C:\\sqlite\\travelexpenses.db"**,** &db)**;** if (rc) {  
 std::cerr << "Can't open database: " << sqlite3\_errmsg(db) << std::endl**;** return nullptr**;** } else {  
 return db**;** }  
}

#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)**;**}**;**#endif //ASSIGNMENT03\_TRAVELRECORDDAO\_H

/\*\*  
 \* @file TravelRecordDAO.cpp  
 \* @brief Implementation of the TravelRecordDAO class for database operations.  
 \*/  
  
#include <algorithm>  
#include <iostream>  
#include "../include/TravelRecordDAO.h"  
  
using namespace std**;**/\*\*  
 \* @brief Retrieves all travel records from the database.  
 \*  
 \* This function retrieves all travel records from the SQLite database and returns them as a vector of TravelRecordDTO objects.  
 \*  
 \* @return A vector containing all travel records from the database.  
 \*/  
std::vector<TravelRecordDTO> TravelRecordDAO::getAllRecords() {  
 sqlite3 \*db**;** sqlite3\_stmt \*stmt**;** vector<TravelRecordDTO> records**;** db = DataSource::openConnection()**;** if (db) {  
 // Prepare SQL query  
 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 expenses 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**;** }  
  
 // Execute query and process results  
 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)))**;** record.setAirfare(reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt**,** 8)))**;** record.setOtherTransport(reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt**,** 9)))**;** record.setLodging(reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt**,** 10)))**;** record.setMeals(reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt**,** 11)))**;** record.setOtherExpenses(reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt**,** 12)))**;** record.setTotal(reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt**,** 13)))**;** records.push\_back(record)**;** }  
 // Finalize statement and close database  
 sqlite3\_finalize(stmt)**;** sqlite3\_close(db)**;** } else {  
 cerr << "Failed to open database connection" << endl**;** }  
 return records**;**}  
  
/\*\*  
 \* @brief Retrieves a travel record by its unique identifier.  
 \*  
 \* This function retrieves a travel record from the SQLite database by its unique identifier (ref\_number).  
 \*  
 \* @param id The unique identifier of the travel record to retrieve.  
 \* @return The travel record with the specified identifier, or an empty record if not found.  
 \*/  
TravelRecordDTO TravelRecordDAO::getRecordById(const std::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 expenses 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**;** }  
  
 // Bind the id to the SQL query  
 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)))**;** record.setAirfare(reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt**,** 8)))**;** record.setOtherTransport(reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt**,** 9)))**;** record.setLodging(reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt**,** 10)))**;** record.setMeals(reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt**,** 11)))**;** record.setOtherExpenses(reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt**,** 12)))**;** record.setTotal(reinterpret\_cast<const char \*>(sqlite3\_column\_text(stmt**,** 13)))**;** }  
 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 function adds a new travel record to the SQLite database based on the provided TravelRecordDTO.  
 \*  
 \* @param record The travel record to be added to the database.  
 \*/  
void TravelRecordDAO::addRecord(const TravelRecordDTO &record) {  
 sqlite3 \*db**;** sqlite3\_stmt \*stmt**;** db = DataSource::openConnection()**;** if (db) {  
 const char \*sql = "INSERT INTO expenses (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**;** }  
 // Bind values from TravelRecordDTO to the SQL query  
 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\_text(stmt**,** 9**,** record.getAirfare().c\_str()**,** -1**,** SQLITE\_TRANSIENT)**;** sqlite3\_bind\_text(stmt**,** 10**,** record.getOtherTransport().c\_str()**,** -1**,** SQLITE\_TRANSIENT)**;** sqlite3\_bind\_text(stmt**,** 11**,** record.getLodging().c\_str()**,** -1**,** SQLITE\_TRANSIENT)**;** sqlite3\_bind\_text(stmt**,** 12**,** record.getMeals().c\_str()**,** -1**,** SQLITE\_TRANSIENT)**;** sqlite3\_bind\_text(stmt**,** 13**,** record.getOtherExpenses().c\_str()**,** -1**,** SQLITE\_TRANSIENT)**;** sqlite3\_bind\_text(stmt**,** 14**,** record.getTotal().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 Updates an existing travel record in the database.  
 \*  
 \* This function updates an existing travel record in the SQLite database based on the provided TravelRecordDTO.  
 \*  
 \* @param record The updated travel record to be saved in the database.  
 \*/  
void TravelRecordDAO::updateRecord(const TravelRecordDTO &record) {  
 sqlite3 \*db**;** sqlite3\_stmt \*stmt**;** db = DataSource::openConnection()**;** if (db) {  
 const char \*sql = "UPDATE expenses 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**;** }  
  
 // Correcting the order of binding  
 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\_text(stmt**,** 8**,** record.getAirfare().c\_str()**,** -1**,** SQLITE\_TRANSIENT)**;** sqlite3\_bind\_text(stmt**,** 9**,** record.getOtherTransport().c\_str()**,** -1**,** SQLITE\_TRANSIENT)**;** sqlite3\_bind\_text(stmt**,** 10**,** record.getLodging().c\_str()**,** -1**,** SQLITE\_TRANSIENT)**;** sqlite3\_bind\_text(stmt**,** 11**,** record.getMeals().c\_str()**,** -1**,** SQLITE\_TRANSIENT)**;** sqlite3\_bind\_text(stmt**,** 12**,** record.getOtherExpenses().c\_str()**,** -1**,** SQLITE\_TRANSIENT)**;** sqlite3\_bind\_text(stmt**,** 13**,** record.getTotal().c\_str()**,** -1**,** SQLITE\_TRANSIENT)**;** sqlite3\_bind\_text(stmt**,** 14**,** record.getRefNumber().c\_str()**,** -1**,** SQLITE\_TRANSIENT)**;** // Execute the statement  
 if (sqlite3\_step(stmt) != SQLITE\_DONE) {  
 cerr << "Failed to execute statement: " << sqlite3\_errmsg(db) << endl**;** }  
  
 // Finalize the statement  
 sqlite3\_finalize(stmt)**;** sqlite3\_close(db)**;** } else {  
 cerr << "Failed to open database connection" << endl**;** }  
}  
  
/\*\*  
 \* @brief Deletes a travel record from the database by its unique identifier.  
 \*  
 \* This function deletes a travel record from the SQLite database based on its unique identifier (ref\_number).  
 \*  
 \* @param id The unique identifier of the travel record to delete.  
 \*/  
void TravelRecordDAO::deleteRecordById(const std::string &id) {  
 sqlite3 \*db**;** sqlite3\_stmt \*stmt**;** db = DataSource::openConnection()**;** if (db) {  
 const char \*sql = "DELETE FROM expenses 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**;** }  
 // Bind the id to the SQL query  
 sqlite3\_bind\_text(stmt**,** 1**,** id.c\_str()**,** -1**,** SQLITE\_STATIC)**;** // Execute the statement  
 if (sqlite3\_step(stmt) != SQLITE\_DONE) {  
 cerr << "Failed to execute statement: " << sqlite3\_errmsg(db) << endl**;** }  
  
 // Finalize the statement  
 sqlite3\_finalize(stmt)**;** sqlite3\_close(db)**;** } else {  
 cerr << "Failed to open database connection" << endl**;** }  
}

#ifndef ASSIGNMENT03\_TRAVELRECORDDTO\_H  
#define ASSIGNMENT03\_TRAVELRECORDDTO\_H  
  
#include <string>  
#include <ios>  
#include <iomanip>  
#include <ostream>  
  
class TravelRecordDTO {  
  
private:  
 std::string ref\_number**;** std::string disclosure\_group**;** std::string title**;** std::string name**;** std::string purpose**;** std::string start\_date**;** std::string end\_date**;** std::string destination**;** std::string airfare**;** std::string other\_transport**;** std::string lodging**;** std::string meals**;** std::string other\_expenses**;** std::string total**;**public:  
  
 const std::string &getRefNumber() const {  
 return ref\_number**;** }  
  
 void setRefNumber(const std::string &refNumber) {  
 ref\_number = refNumber**;** }  
  
 const std::string &getDisclosureGroup() const {  
 return disclosure\_group**;** }  
  
 void setDisclosureGroup(const std::string &disclosureGroup) {  
 disclosure\_group = disclosureGroup**;** }  
  
 const std::string &getTitle() const {  
 return title**;** }  
  
 void setTitle(const std::string &title) {  
 TravelRecordDTO::title = title**;** }  
  
 const std::string &getName() const {  
 return name**;** }  
  
 void setName(const std::string &name) {  
 TravelRecordDTO::name = name**;** }  
  
 const std::string &getPurpose() const {  
 return purpose**;** }  
  
 void setPurpose(const std::string &purpose) {  
 TravelRecordDTO::purpose = purpose**;** }  
  
 const std::string &getStartDate() const {  
 return start\_date**;** }  
  
 void setStartDate(const std::string &startDate) {  
 start\_date = startDate**;** }  
  
 const std::string &getEndDate() const {  
 return end\_date**;** }  
  
 void setEndDate(const std::string &endDate) {  
 end\_date = endDate**;** }  
  
 const std::string &getDestination() const {  
 return destination**;** }  
  
 void setDestination(const std::string &destination) {  
 TravelRecordDTO::destination = destination**;** }  
  
 const std::string &getAirfare() const {  
 return airfare**;** }  
  
 void setAirfare(const std::string &airfare) {  
 TravelRecordDTO::airfare = airfare**;** }  
  
 const std::string &getOtherTransport() const {  
 return other\_transport**;** }  
  
 void setOtherTransport(const std::string &otherTransport) {  
 other\_transport = otherTransport**;** }  
  
 const std::string &getLodging() const {  
 return lodging**;** }  
  
 void setLodging(const std::string &lodging) {  
 TravelRecordDTO::lodging = lodging**;** }  
  
 const std::string &getMeals() const {  
 return meals**;** }  
  
 void setMeals(const std::string &meals) {  
 TravelRecordDTO::meals = meals**;** }  
  
 const std::string &getOtherExpenses() const {  
 return other\_expenses**;** }  
  
 void setOtherExpenses(const std::string &otherExpenses) {  
 other\_expenses = otherExpenses**;** }  
  
 const std::string &getTotal() const {  
 return total**;** }  
  
 void setTotal(const std::string &total) {  
 TravelRecordDTO::total = total**;** }  
  
 friend std::ostream &operator<<(std::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

#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)**;**}**;**#endif //ASSIGNMENT03\_TRAVELRECORDSERVICE\_H

/\*\*  
 \* @file TravelRecordService.cpp  
 \* @brief Implementation of the TravelRecordService class for managing travel records.  
 \*/  
  
#include "../include/TravelRecordService.h"  
#include <stdexcept>  
  
/\*\*  
 \* @brief Constructs a TravelRecordService object.  
 \*  
 \* This constructor initializes a TravelRecordService object with a provided TravelRecordDAO instance.  
 \*  
 \* @param dao The TravelRecordDAO instance to be used for data access.  
 \*/  
TravelRecordService::TravelRecordService(const TravelRecordDAO &dao) : dao(dao) {  
}  
  
/\*\*  
 \* @brief Retrieves all travel records.  
 \*  
 \* This function retrieves all travel records from the data source and returns them as a vector of TravelRecordDTO objects.  
 \*  
 \* @return A vector containing all travel records.  
 \*/  
std::vector<TravelRecordDTO> TravelRecordService::getAllRecords() {  
 return dao.getAllRecords()**;**}  
  
/\*\*  
 \* @brief Retrieves a travel record by its unique identifier.  
 \*  
 \* This function retrieves a specific travel record from the data source by its unique identifier (ref\_number).  
 \*  
 \* @param id The unique identifier of the travel record to retrieve.  
 \* @return The travel record with the specified identifier.  
 \*/  
TravelRecordDTO TravelRecordService::getRecordById(const std::string &id) {  
 return dao.getRecordById(id)**;**}  
  
/\*\*  
 \* @brief Adds a new travel record.  
 \*  
 \* This function adds a new travel record to the data source based on the provided TravelRecordDTO.  
 \*  
 \* @param record The travel record to be added.  
 \*/  
void TravelRecordService::addRecord(const TravelRecordDTO &record) {  
 dao.addRecord(record)**;**}  
  
/\*\*  
 \* @brief Updates an existing travel record.  
 \*  
 \* This function updates an existing travel record in the data source based on the provided TravelRecordDTO.  
 \*  
 \* @param record The updated travel record to be saved.  
 \*/  
void TravelRecordService::updateRecord(const TravelRecordDTO &record) {  
 dao.updateRecord(record)**;**}  
  
/\*\*  
 \* @brief Deletes a travel record by its unique identifier.  
 \*  
 \* This function deletes a specific travel record from the data source by its unique identifier (ref\_number).  
 \*  
 \* @param id The unique identifier of the travel record to delete.  
 \*/  
void TravelRecordService::deleteRecordById(const std::string &id) {  
 dao.deleteRecordById(id)**;**}

#ifndef ASSIGNMENT03\_USERINTERFACE\_H  
#define ASSIGNMENT03\_USERINTERFACE\_H  
  
#include <iostream>  
#include <string>  
#include "TravelRecordService.h"  
  
class UserInterface {  
private:  
 TravelRecordService service**;** void handleChoice(int choice)**;**public:  
 explicit UserInterface(const TravelRecordService &service) : service(service) {}  
  
 void showMenu()**;** void populateTravelRecordFromInput(TravelRecordDTO &record**,** bool isUpdate = false)**;**}**;**#endif //ASSIGNMENT03\_USERINTERFACE\_H

/\*\*  
 \* @file UserInterface.cpp  
 \* @brief Implementation of the UserInterface class for managing user interactions.  
 \*/  
#include "UserInterface.h"  
  
using namespace std**;**/\*\*  
 \* @brief Displays the main menu and handles user interactions.  
 \*/  
void UserInterface::showMenu() {  
 cout << "\n"**;** cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl**;** cout << "\* Welcome to Database Reader App! \*" << endl**;** cout << "\* ( Travel Record System ) \*" << endl**;** cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl**;** cout << "\n"**;** while (true) {  
 cout << "\n\nBy Onur Onel 041074824\n\n" << endl**;** cout << "Please 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. Exit\n\n"**;** cout << "Enter your choice: "**;** int choice**;** cin >> choice**;** if (choice == 6) break**;** handleChoice(choice)**;** }  
}  
  
/\*\*  
 \* @brief Populates a TravelRecordDTO object from user input.  
 \*  
 \* @param record The TravelRecordDTO object to populate.  
 \* @param isUpdate Flag indicating whether this is for updating an existing record.  
 \*/  
void UserInterface::populateTravelRecordFromInput(TravelRecordDTO &record**,** bool isUpdate) {  
 std::string input**;** if (isUpdate) {  
 std::cout << "Current reference number: " << record.getRefNumber()**;** std::cout << "\nEnter new value or press Enter to keep: "**;** } else {  
 std::cout << "Reference number: "**;** }  
 std::getline(std::cin >> std::ws**,** input)**;** if (!input.empty()) {  
 record.setRefNumber(input)**;** }  
  
 if (isUpdate) {  
 std::cout << "Current disclosure group: " << record.getDisclosureGroup()**;** std::cout << "\nEnter new value or press Enter to keep: "**;** } else {  
 std::cout << "Disclosure group: "**;** }  
 std::getline(std::cin**,** input)**;** if (!input.empty()) {  
 record.setDisclosureGroup(input)**;** }  
  
 if (isUpdate) {  
 std::cout << "Current title: " << record.getTitle()**;** std::cout << "\nEnter new value or press Enter to keep: "**;** } else {  
 std::cout << "Title: "**;** }  
 std::getline(std::cin**,** input)**;** if (!input.empty()) {  
 record.setTitle(input)**;** }  
  
 if (isUpdate) {  
 std::cout << "Current name: " << record.getName()**;** std::cout << "\nEnter new value or press Enter to keep: "**;** } else {  
 std::cout << "Name: "**;** }  
 std::getline(std::cin**,** input)**;** if (!input.empty()) {  
 record.setName(input)**;** }  
 if (isUpdate) {  
 std::cout << "Current purpose: " << record.getPurpose()**;** std::cout << "\nEnter new value or press Enter to keep: "**;** } else {  
 std::cout << "Purpose: "**;** }  
 std::getline(std::cin**,** input)**;** if (!input.empty()) {  
 record.setPurpose(input)**;** }  
  
 if (isUpdate) {  
 std::cout << "Current start date: " << record.getStartDate()**;** std::cout << "\nEnter new value or press Enter to keep: "**;** } else {  
 std::cout << "Start date: "**;** }  
 std::getline(std::cin**,** input)**;** if (!input.empty()) {  
 record.setStartDate(input)**;** }  
  
 if (isUpdate) {  
 std::cout << "Current end date: " << record.getEndDate()**;** std::cout << "\nEnter new value or press Enter to keep: "**;** } else {  
 std::cout << "End date: "**;** }  
 std::getline(std::cin**,** input)**;** if (!input.empty()) {  
 record.setEndDate(input)**;** }  
  
 if (isUpdate) {  
 std::cout << "Current destination: " << record.getDestination()**;** std::cout << "\nEnter new value or press Enter to keep: "**;** } else {  
 std::cout << "Destination: "**;** }  
 std::getline(std::cin**,** input)**;** if (!input.empty()) {  
 record.setDestination(input)**;** }  
  
 if (isUpdate) {  
 std::cout << "Current airfare: " << record.getAirfare()**;** std::cout << "\nEnter new value or press Enter to keep: "**;** } else {  
 std::cout << "Airfare: "**;** }  
 std::getline(std::cin**,** input)**;** if (!input.empty()) {  
 record.setAirfare(input)**;** }  
  
 if (isUpdate) {  
 std::cout << "Current other transport: " << record.getOtherTransport()**;** std::cout << "\nEnter new value or press Enter to keep: "**;** } else {  
 std::cout << "Other transport: "**;** }  
 std::getline(std::cin**,** input)**;** if (!input.empty()) {  
 record.setOtherTransport(input)**;** }  
  
 if (isUpdate) {  
 std::cout << "Current lodging: " << record.getLodging()**;** std::cout << "\nEnter new value or press Enter to keep: "**;** } else {  
 std::cout << "Lodging: "**;** }  
 std::getline(std::cin**,** input)**;** if (!input.empty()) {  
 record.setLodging(input)**;** }  
  
 if (isUpdate) {  
 std::cout << "Current meals: " << record.getMeals()**;** std::cout << "\nEnter new value or press Enter to keep: "**;** } else {  
 std::cout << "Meals: "**;** }  
 std::getline(std::cin**,** input)**;** if (!input.empty()) {  
 record.setMeals(input)**;** }  
  
 if (isUpdate) {  
 std::cout << "Current other expenses: " << record.getOtherExpenses()**;** std::cout << "\nEnter new value or press Enter to keep: "**;** } else {  
 std::cout << "Other expenses: "**;** }  
 std::getline(std::cin**,** input)**;** if (!input.empty()) {  
 record.setOtherExpenses(input)**;** }  
  
 if (isUpdate) {  
 std::cout << "Current total: " << record.getTotal()**;** std::cout << "\nEnter new value or press Enter to keep: "**;** } else {  
 std::cout << "Total: "**;** }  
 std::getline(std::cin**,** input)**;** if (!input.empty()) {  
 record.setTotal(input)**;** }  
}  
  
/\*\*  
 \* @brief Handles the user's choice from the main menu.  
 \*  
 \* @param choice The user's choice.  
 \*/  
void UserInterface::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**;** }  
 default:  
 cout << "Invalid choice. Please try again.\n"**;** }  
}

/\*\*  
 \* @file main.cpp  
 \* @brief Entry point of the Travel Expense Management application.  
 \*/  
  
#include "TravelRecordDAO.h"  
#include "TravelRecordService.h"  
#include "UserInterface.h"  
  
/\*\*  
 \* @brief The main function of the Travel Expense Management application.  
 \*  
 \* This function initializes the necessary objects, including the TravelRecordDAO, TravelRecordService,  
 \* and UserInterface, to start the application. It then calls the showMenu() method of the UserInterface  
 \* to display the main menu and initiate user interactions.  
 \*  
 \* @return 0 on successful execution of the application.  
 \*/  
int main() {  
 TravelRecordDAO dao**;** // Create a TravelRecordDAO object  
 TravelRecordService service(dao)**;** // Create a TravelRecordService object using the DAO  
 UserInterface ui(service)**;** // Create a UserInterface object using the service  
  
 ui.showMenu()**;** // Display the main menu to the user  
  
 return 0**;** // Exit the application with a status code of 0  
}