Programming Language Research Practical Project 2

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

Throughout this project, I delved deeply into the structures of the Model-View-Controller (MVC) architecture. This provided me with a comprehensive understanding of how components interact in a system, allowing for better development and maintenance processes.

Furthermore, I familiarized myself with the compilers and the significant role of CMakeList.txt in the compilation process. This file serves as a blueprint for the build system, directing how various software elements should be assembled and integrated.

My journey made me delve into library management and dependencies. In terms of how to link libraries efficiently, my learning experience made me significantly advanced in my knowledge and usage in compilers.

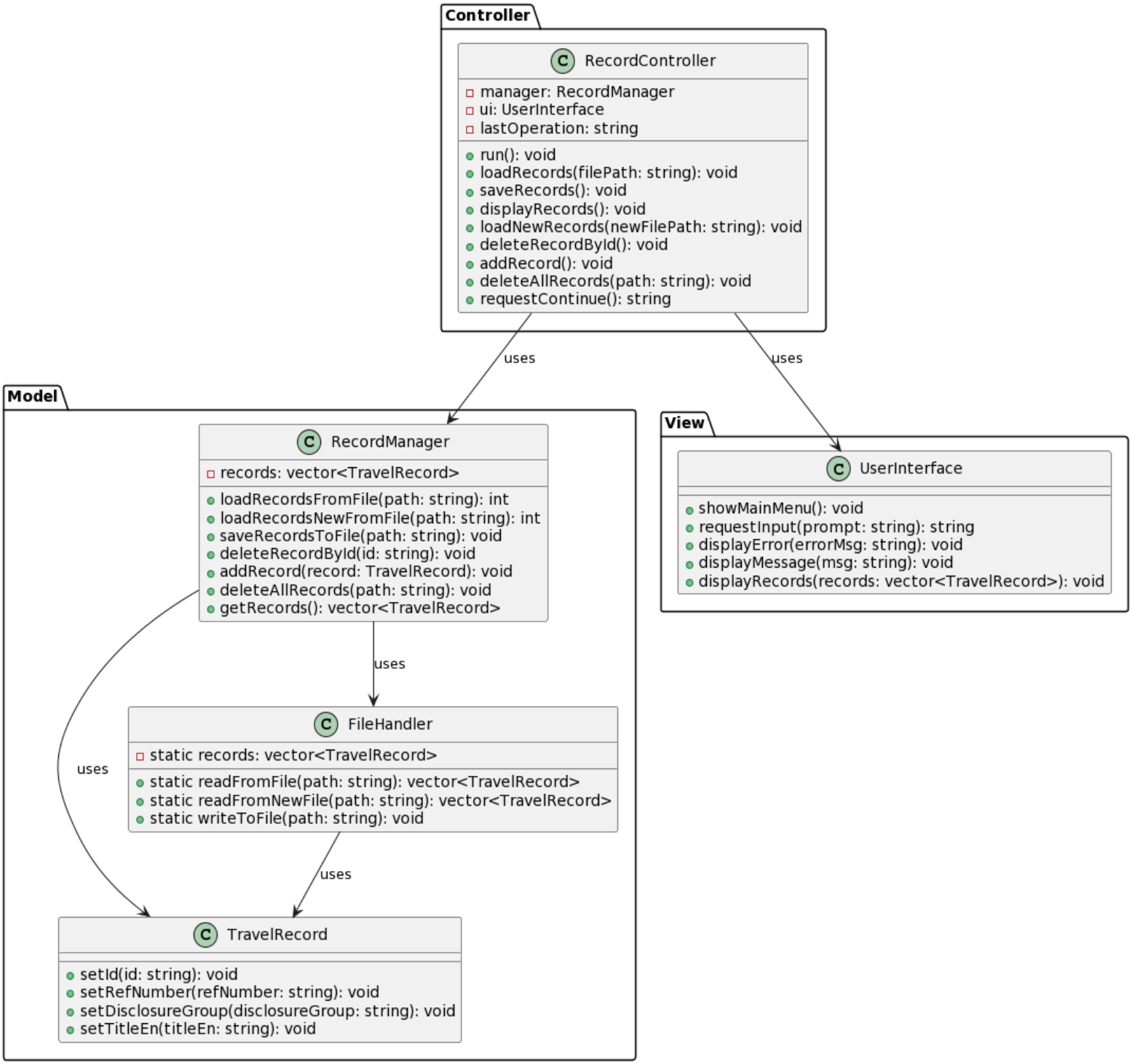
Another significant takeaway was my introduction to Catch2, a unit testing framework. Properly configuring this framework with c++ compiler is essential for the successful implementation of tests, ensuring that software functions as intended.

Lastly, I am learning the of package management through Conan. The process of integrating Conan with CMake presented its own set of challenges. However, it underscored the significance of effective resource management and the need for streamlined software distribution techniques.

Source

1. https://jacobgalam.medium.com/mvc-in-c-66497e5d7011
2. https://www.jetbrains.com/help/clion/quick-cmake-tutorial.html
3. https://github.com/catchorg/Catch2/blob/devel/docs/cmake-integration.md#top
4. https://www.jetbrains.com/help/clion/catch-tests-support.html#test-runner
5. https://www.youtube.com/watch?v=w2CzYK5ZJys
6. https://docs.conan.io/2/installation.html
7. https://docs.conan.io/1/integrations/ide/clion.html
8. https://blog.conan.io/introducing-new-conan-clion-plugin/

Program Architecture



The Advanced CSV Reader project employs the Model-View-Controller (MVC) design pattern, offering modular development, easier maintenance, and separation of concerns.

In MVC architecture, data is encapsulated in the Model, the user interface is handled by the View, and the main logic operations are managed by the Controller. This separation ensures clean, organized code that can be easily scaled and maintained.

1. **Model**

- TravelRecord: Represents the data structure for a travel record. Attributes include id, refNumber, disclosureGroup, and titleEn.
- FileHandler: Manages file operations, specifically reading and writing, ensuring data persistence for travel records.

The Model, especially FileHandler, might be designed to accommodate various storage mechanisms in the future, such as cloud storage or databases.

2. **View**

- TravelRecordView: Presents the user interface, displaying information and accepting user input. It is likely to contain methods for showing records or error messages.

The View should be adaptive and scalable to different user interfaces and devices. The use of responsive design principles will be essential for a consistent user experience.

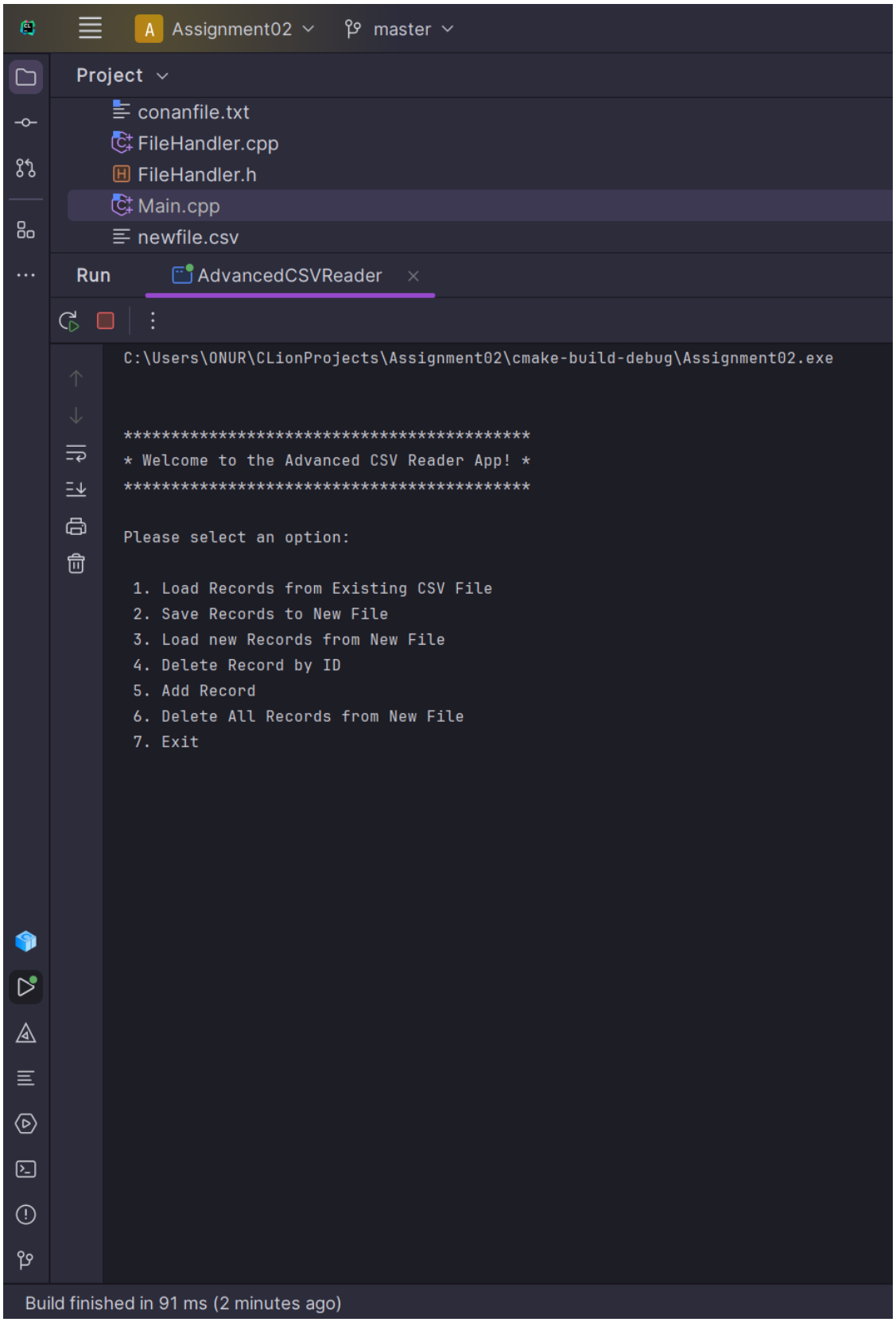
3. **Controller**

- TravelRecordController: Serves as a bridge between the Model and View. It processes user input from the View, interacts with the Model, and returns output displays to the View.

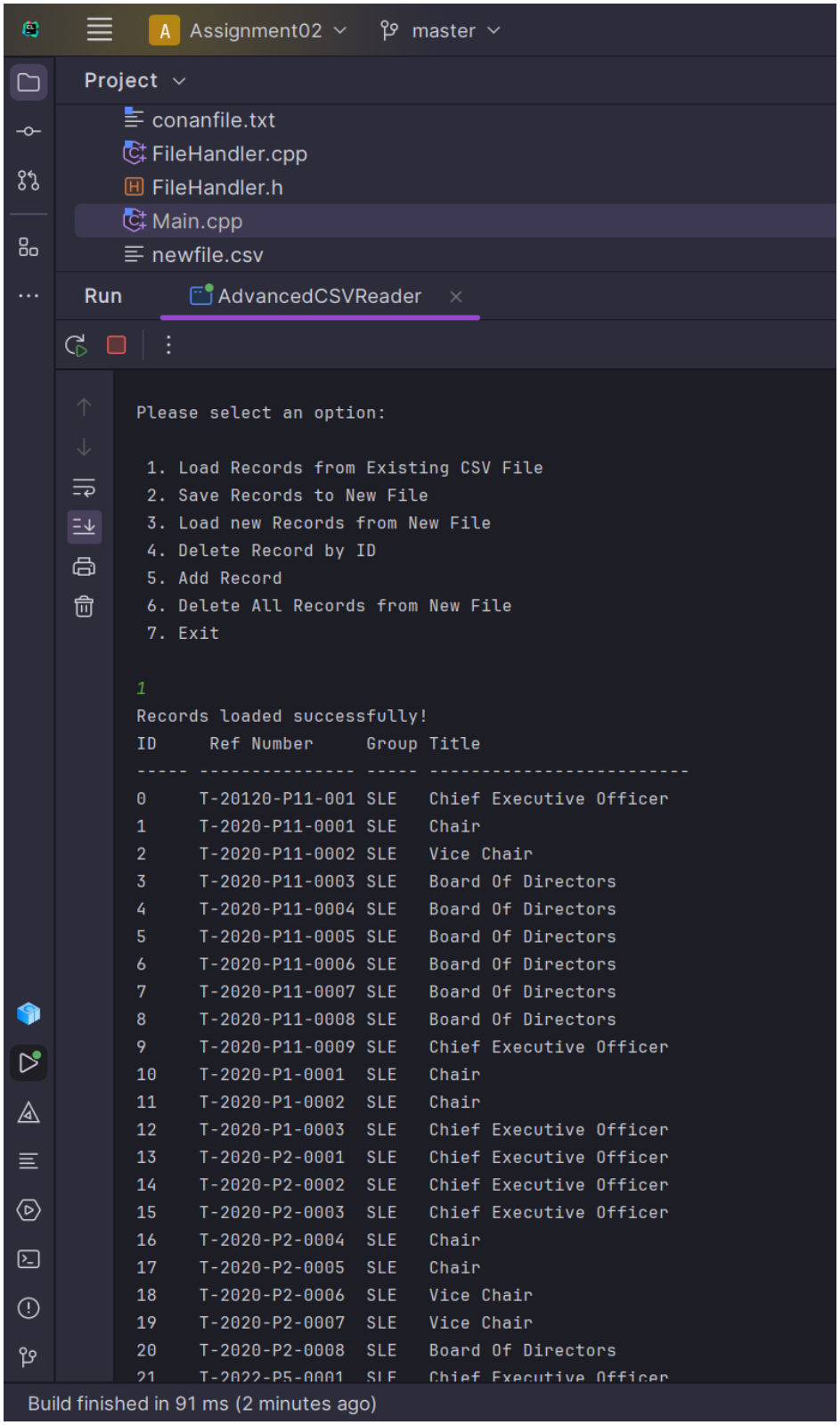
For scalability, the controller might be structured to support batch processing of records, or integrate with external APIs or services for enhanced functionalities.

While the data is encapsulated in the Model, the user interface remains the domain of the View, and logic operations reside within the Controller. Such division fortifies the codebase against convoluted overlaps, making it organized and amenable to future expansions.

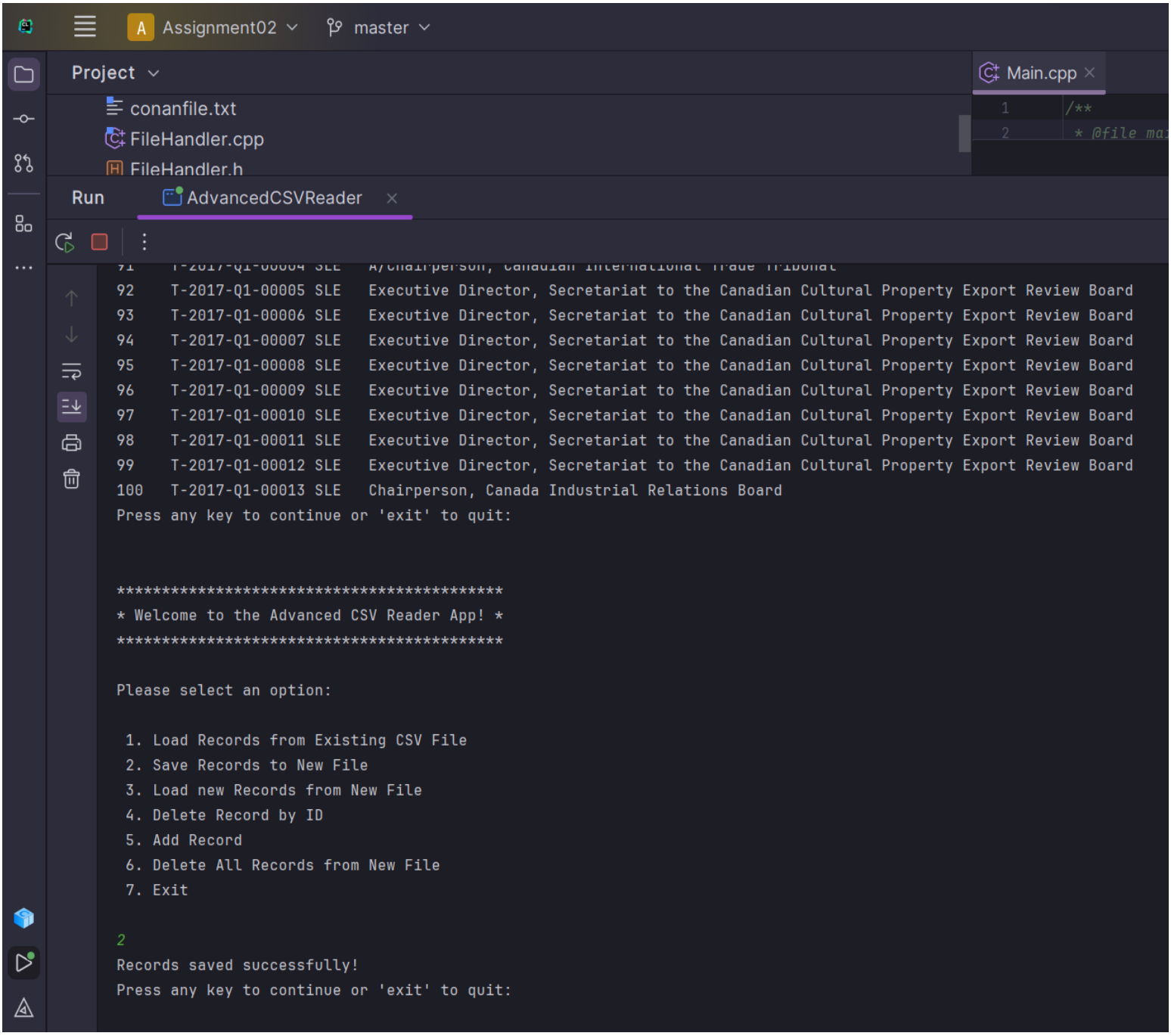
Program Demonstration via Screen Shots



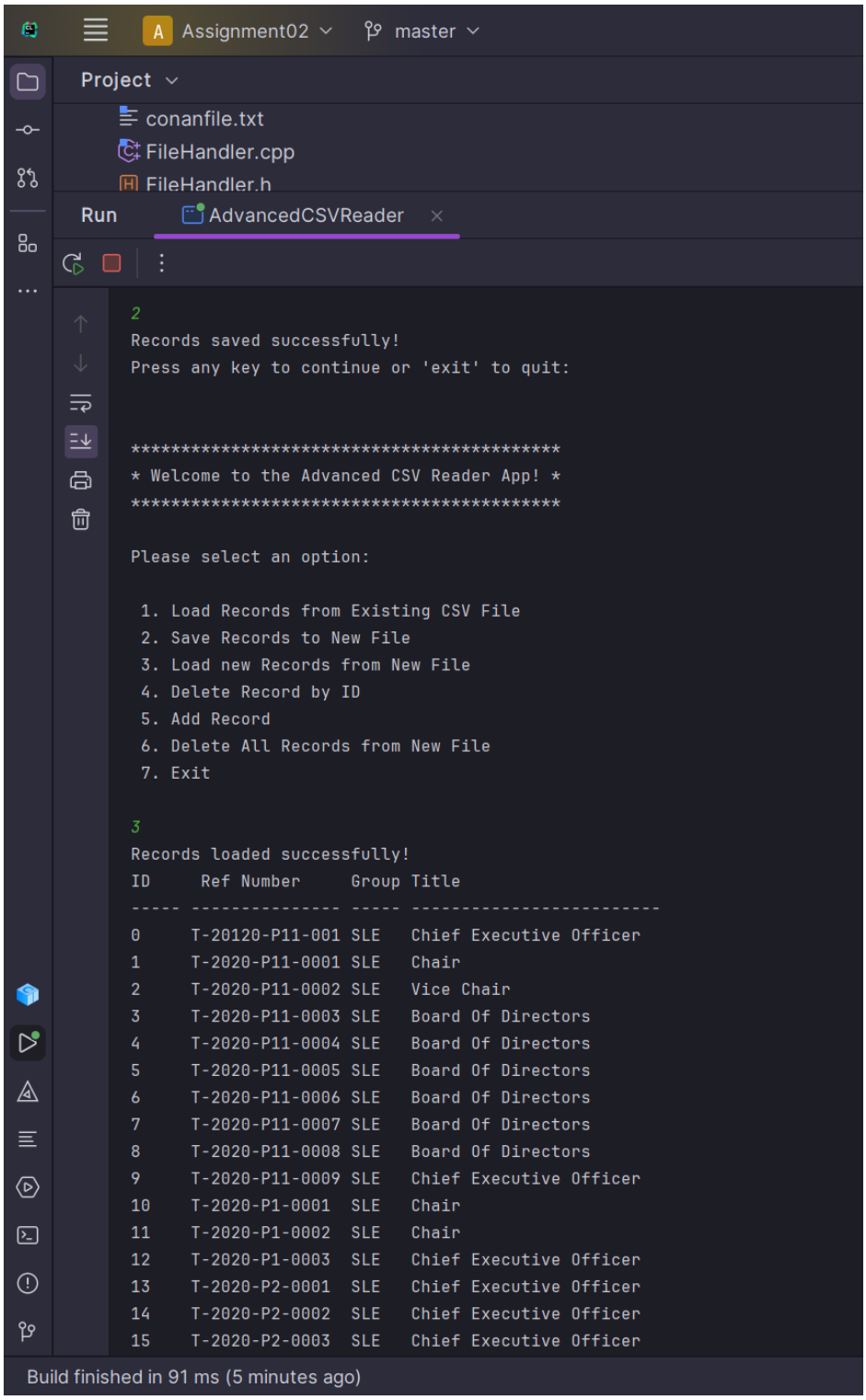
User Interface Through Terminal



Loading existing CSV file – Dataset (Read Functionality)



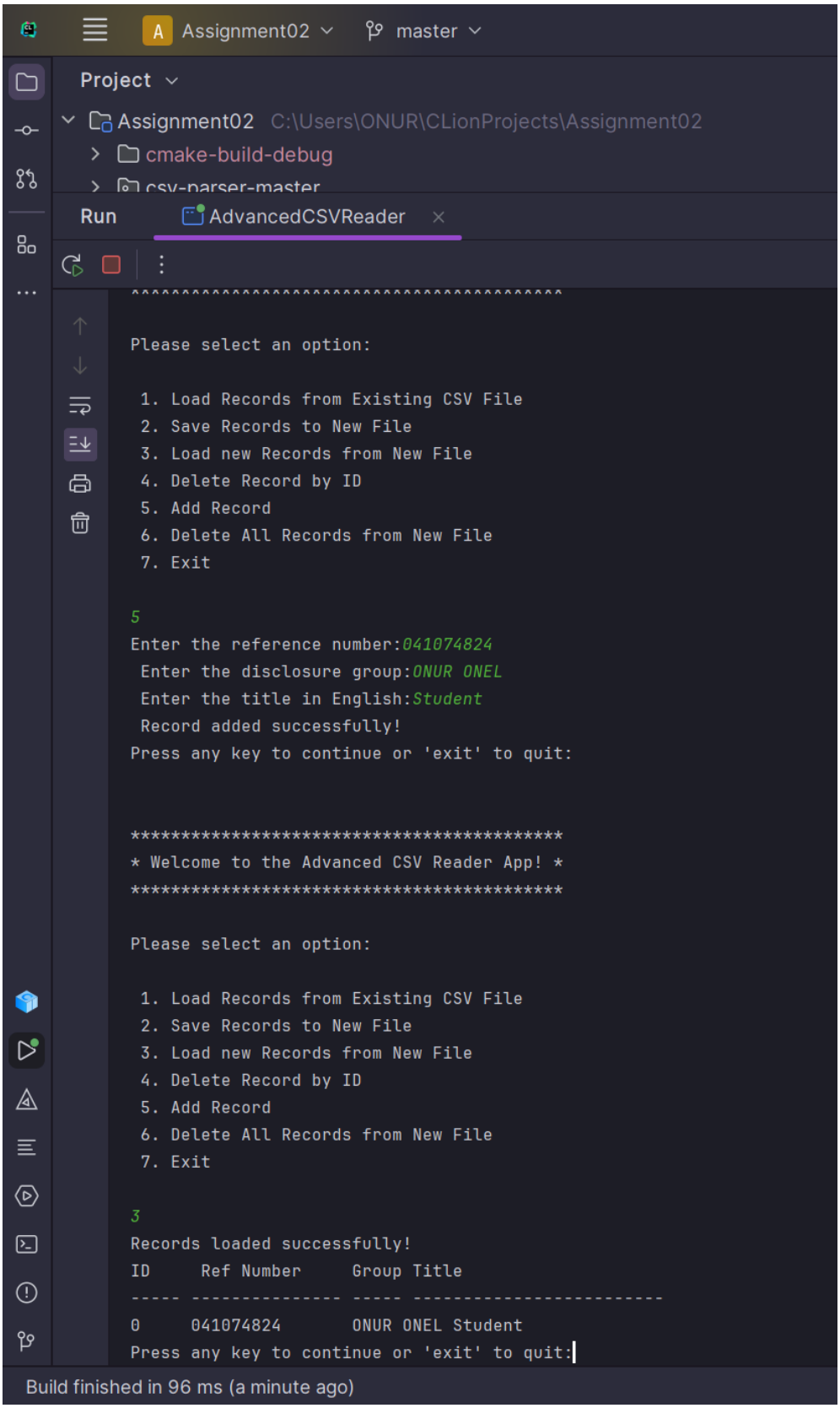
Saving Into Newly Created CSV File (Save Functionality)



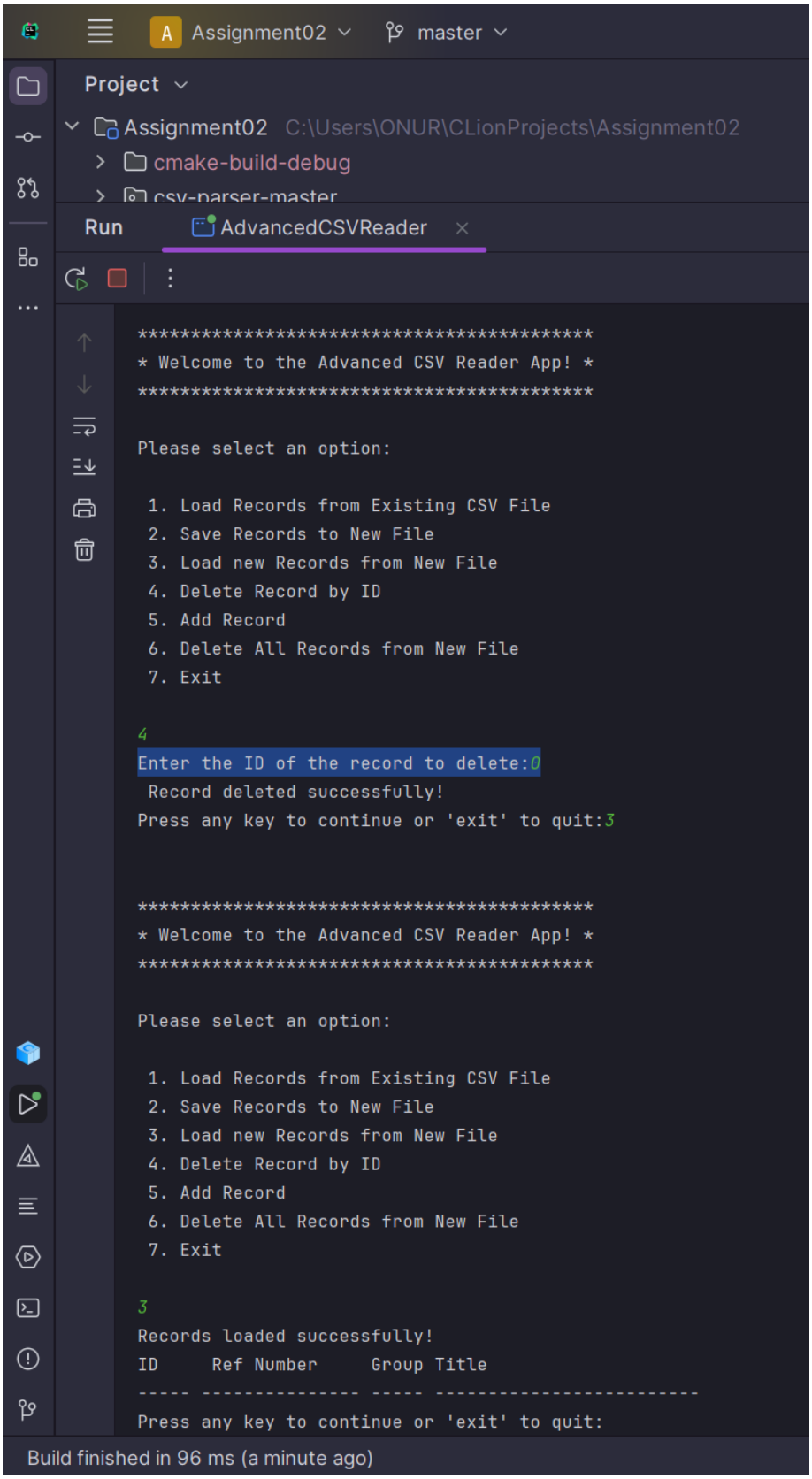
Loading From New CSV File (Read Functionality)

```
Assignment02   master
Project
conanfile.txt
FileHandler.cpp
FileHandler.h
Run   AdvancedCSVReader

*****************************************
* Welcome to the Advanced CSV Reader App! *
*****************************************

Please select an option:

 1. Load Records from Existing CSV File
 2. Save Records to New File
 3. Load new Records from New File
 4. Delete Record by ID
 5. Add Record
 6. Delete All Records from New File
 7. Exit

6
All records deleted successfully!
Press any key to continue or 'exit' to quit:


*****************************************
* Welcome to the Advanced CSV Reader App! *
*****************************************

Please select an option:

 1. Load Records from Existing CSV File
 2. Save Records to New File
 3. Load new Records from New File
 4. Delete Record by ID
 5. Add Record
 6. Delete All Records from New File
 7. Exit

3
Records loaded successfully!
ID     Ref Number     Group Title
----- --------------- ----- ------------------------
Press any key to continue or 'exit' to quit:
Build finished in 91 ms (3 minutes ago)
```

Deleting All Records (Delete Functionality)



Adding New Row into CSV File (Write Functionality)



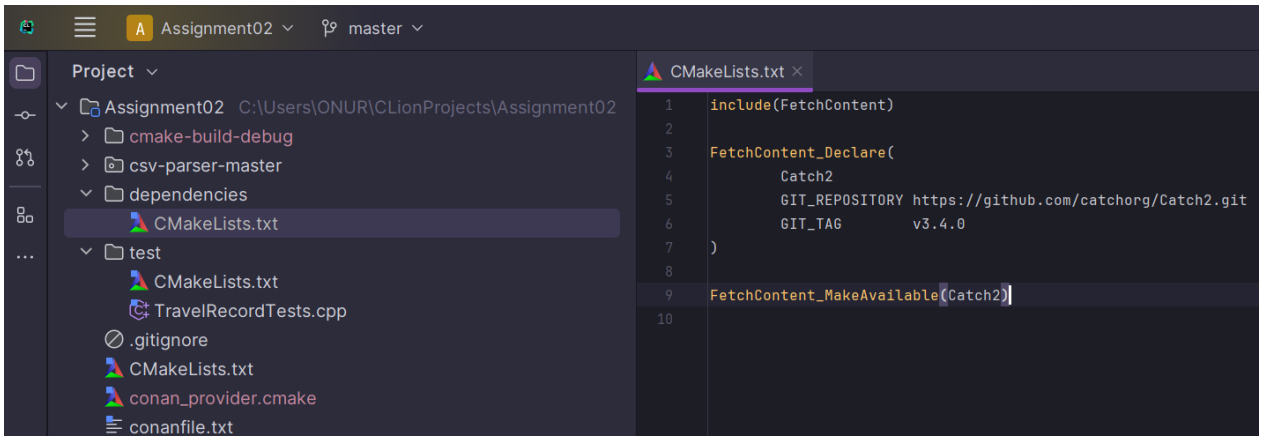
Deleting Record by ID (Read and Delete Functionality)

Unit Testing Demonstration via Screen Shots

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.26)
project(Assignment02)

set(CMAKE_CXX_STANDARD 17)

add_subdirectory(dependencies)
add_subdirectory(csv-parser-master)

set(MAIN_SOURCE_FILES
        FileHandler.cpp
        RecordManager.cpp
        UserInterface.cpp
        RecordController.cpp
)

add_library(AssignmentLib STATIC ${MAIN_SOURCE_FILES})

add_executable(Assignment02 Main.cpp)

target_link_libraries(Assignment02 AssignmentLib csv)

add_subdirectory(test)
```

1. I've set the project to require at least version 3.26 of CMake.
2. I chose to compile the project using the C++17 standard.
3. I have additional CMake configuration files in two directories: "dependencies" and "csv-parser-master". I've included them using add_subdirectory.
4. Using these source files, I created a static library named "AssignmentLib" with add_library.
5. I defined the main application executable, "Assignment02", to be compiled from Main.cpp using add_executable.
6. I linked the "Assignment02" executable against the "AssignmentLib" static "csv" library with target_link_libraries.
7. For Catch2 framework I added a "test" directory, which contains configurations related to Catch 2-unit testing, with add_subdirectory(test).



I utilized the FetchContent module in CMake to seamlessly integrate external libraries during the build process. Specifically, the Catch2 testing framework into my project.

- **Integrating FetchContent:**

Firstly, I included the module with include (FetchContent) to access its functions.

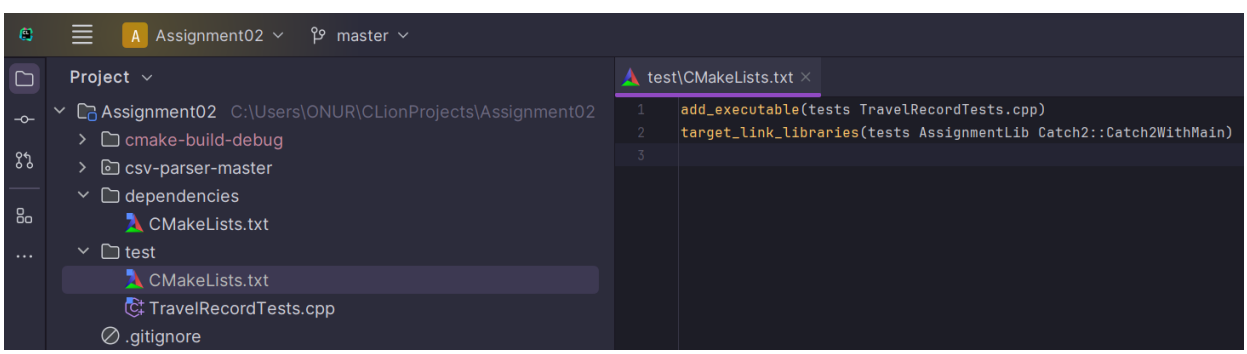
- **Specifying the Library:**

Through FetchContent_Declare, I defined the details of the Catch2 library. I pointed it to the GitHub repository using GIT_REPOSITORY and specified the version I wanted with GIT_TAG. For this project, I settled on version v3.4.0.

- **Activating the Download:**

By executing FetchContent_MakeAvailable(Catch2), I initiated the download and setup of Catch2, making it readily accessible in my project's build environment.

https://cmake.org/cmake/help/latest/module/FetchContent.html



In the test directory, I set up testing for my project I created a test executable named tests from the source file TravelRecordTests.cpp. I linked this test to the main project library, I also linked it to Catch2::Catch2WithMain, a module from Catch2, which provides the necessary foundation to run the tests.

Travel RecordTests.cpp

TravelRecordTests.cpp

```cpp
/**
 * @file TravelRecordTests.cpp
 * @brief Contains Catch2 tests for Travel Record functionalities.
 * @details This file encompasses several test cases that ensure the proper functioning
 *          of Travel Record management, including loading, saving, displaying, and
 *          modifying records.
 * @author Onur Onel
 **/

#include <catch2/catch_test_macros.hpp>
#include "../RecordManager.h"
#include "../UserInterface.h"

/**
 * @test Tests the loading of records from an existing file.
 * @note This test is currently failing unexpectedly.
 *       Expected number of records loaded: 100, Actual: 101.
 */
TEST_CASE("Testing loading records from existing file", "[RecordManager]") {
    RecordManager manager;
    REQUIRE(manager.loadRecordsFromFile(R"(C:\Users\ONUR\CLionProjects\Assignment02\travelq.csv)") == 100);
    REQUIRE_THROWS(manager.loadRecordsFromFile("invalid_path.csv"));
}

/**
 * @test Tests the loading of records from a new file.
 * @note This test is currently failing unexpectedly.
 *       Expected number of records loaded: 100, Actual: 101.
 */
TEST_CASE("Testing loading records from new file", "[RecordManager]") {
    RecordManager manager;
    REQUIRE(manager.loadRecordsFromFile(R"(C:\Users\ONUR\CLionProjects\Assignment02\newfile.csv)") == 100);
    REQUIRE_THROWS(manager.loadRecordsFromFile("invalid_path.csv"));
}

/**
 * @test Tests the saving of records to a file.
 */
TEST_CASE("Testing saving records to file", "[RecordManager]") {
    RecordManager manager;
    REQUIRE_NOTHROW(manager.saveRecordsToFile(R"(C:\Users\ONUR\CLionProjects\Assignment02\newfile.csv)"));
}
```

TravelRecordTests.cpp

```cpp
}

/**
 * @test Tests the display functionality of records using UserInterface.
 */
TEST_CASE("Testing display of records", "[UserInterface]") {
    RecordManager manager;
    UserInterface ui;
    std::vector<TravelRecord> test_records = manager.getRecords();
    REQUIRE_NOTHROW(ui.displayRecords(test_records));
}

/**
 * @test Tests the deletion of a specific record by ID.
 */
TEST_CASE("Testing Delete Record by ID", "[RecordManager]") {
    RecordManager manager;
    REQUIRE_NOTHROW(manager.deleteRecordById("1"));
}

/**
 * @test Tests the deletion of all records.
 */
TEST_CASE("Testing Delete All Records", "[RecordManager]") {
    RecordManager manager;
    REQUIRE_NOTHROW(manager.deleteAllRecords(R"(C:\Users\ONUR\CLionProjects\Assignment02\newfile.csv)"));
}

/**
* @test Tests the getters and setters of the TravelRecord class.
*/
TEST_CASE("Testing TravelRecord getters and setters", "[TravelRecord]") {
    TravelRecord record;
    record.setRefNumber("12345");
    REQUIRE(record.getRefNumber() == "12345");

    record.setDisclosureGroup("GroupA");
    REQUIRE(record.getDisclosureGroup() == "GroupA");

    record.setTitleEn("Sample Title");
    REQUIRE(record.getTitleEn() == "Sample Title");

    record.setId("001");
    REQUIRE(record.getId() == "001");
}
```

Assignment02 master Debug Testing loading

Project

- Assignment02 C:\Users\ONUR\
  - cmake-build-debug
  - csv-parser-master
  - dependencies
    - CMakeLists.txt
  - test
    - CMakeLists.txt
    - TravelRecordTests.cpp
  - .gitignore
  - CMakeLists.txt
  - conan_provider.cmake
  - conanfile.txt
  - FileHandler.cpp
  - FileHandler.h
  - Main.cpp
  - newfile.csv
  - RecordController.cpp
  - RecordController.h
  - RecordManager.cpp
  - RecordManager.h

TravelRecordTests.cpp

```cpp

/**
 * @test Tests the loading of records from an existing file.
 * @note This test is currently failing unexpectedly.
 *       Expected number of records loaded: 100, Actual: 101.
 */
TEST_CASE("Testing loading records from existing file", "[RecordManager]") {
    RecordManager manager;
    REQUIRE(manager.loadRecordsFromFile(R"(C:\Users\ONUR\CLionProjects\Assignment02\travelq.csv)") == 100);
    REQUIRE_THROWS(manager.loadRecordsFromFile("invalid_path.csv"));
}

/**
 * @test Tests the loading of records from a new file.
 * @note This test is currently failing unexpectedly.
 *       Expected number of records loaded: 100, Actual: 101.
 */
TEST_CASE("Testing loading records from new file", "[RecordManager]") {
    RecordManager manager;
    REQUIRE(manager.loadRecordsFromFile(R"(C:\Users\ONUR\CLionProjects\Assignment02\newfile.csv)") == 100);
    REQUIRE_THROWS(manager.loadRecordsFromFile("invalid_path.csv"));
}

```

Run — Testing loading records from existing file

- Test Results 144 ms
  - Testing loading records from existing file 144 ms

Tests failed: 1 of 1 test – 144 ms

```
C:\Users\ONUR\CLionProjects\Assignment02\cmake-build-debug\test\tests.exe -r xml -d yes --order lex "Testing
Testing started at 9:18 PM ...

C:/Users/ONUR/CLionProjects/Assignment02/test/TravelRecordTests.cpp:21: Failure:
  REQUIRE(manager.loadRecordsFromFile(R"(C:\Users\ONUR\CLionProjects\Assignment02\travelq.csv)") == 100)
with expansion:
  101 == 100

Process finished with exit code 1
```

```cpp
/**
 * @test Tests the loading of records from an existing file.
 * @note This test is currently failing unexpectedly.
 *       Expected number of records loaded: 100, Actual: 101.
 */
```

```
C:/Users/ONUR/CLionProjects/Assignment02/test/TravelRecordTests.cpp:21: Failure:
  REQUIRE(manager.loadRecordsFromFile(R"(C:\Users\ONUR\CLionProjects\Assignment02\travelq.csv)") == 100)
with expansion:
  101 == 100

Process finished with exit code 1
```

Source Code and Commenting

```
/**
 * @file main.cpp
 * @brief Entry point for the Advanced CSV Reader application.
 * @details This application allows users to read, write, and manipulate CSV files.
 * @author Onur Onel
 */

#include "RecordController.h"

/**
 * @brief Entry point for the program.
 *
 * Initializes the RecordController and handles the main loop
 * where users interact with the application. Users can
 * continue running the program or choose to exit.
 *
 * @return Returns 0 upon successful completion.
 */
int main() {
    RecordController controller; ///< Controller instance for managing the CSV records.

    bool continueRunning = true; ///< Flag to determine whether to keep the program
running.

    // Main program loop
    while (continueRunning) {
        controller.run();

        // Request user's choice on continuing or exiting the application.
        std::string choice = controller.requestContinue();

        // Check user's choice
        if (choice == "exit" || choice == "quit" || choice == "q") {
            continueRunning = false;
        }
    }

    return 0;
}
/**
 * @file UserInterface.h
 * @brief Contains methods for displaying information to the user.
 * @details This file contains the implementation of the UserInterface class, which is
responsible for  displaying information to the user and taking input from the user.
 * @author Onur Onel
 */
#ifndef ASSIGNMENT02_USERINTERFACE_H
#define ASSIGNMENT02_USERINTERFACE_H

#include "RecordManager.h"
#include "TravelRecord.h"
#include <iostream>

/**
 * @brief Represents the user interface for the application.
 *
 * This class is responsible for handling user interactions such as displaying records,
 * showing messages, and taking user input.
 */
class UserInterface {
public:
```

```
    /**
     * @brief Displays a list of Travel Records.
     *
     * @param records The list of travel records to be displayed.
     */
    void displayRecords(const std::vector<TravelRecord> &records);

    /**
     * @brief Displays a given message to the user.
     *
     * @param message The message to display.
     */
    void displayMessage(const std::string &message);

    /**
     * @brief Displays an error message to the user.
     *
     * @param error The error message to display.
     */
    static void displayError(const std::string &error);

    /**
     * @brief Displays the main menu of the application.
     */
    void showMainMenu();

    /**
     * @brief Requests user input with a given prompt.
     *
     * @param prompt The prompt to display before taking input.
     * @return The string entered by the user.
     */
    std::string requestInput(const std::string &prompt);
};

#endif //ASSIGNMENT02_USERINTERFACE_H
/**
 * @file UserInterface.cpp
 * @brief Contains methods for displaying information to the user.
 * @details This file contains the implementation of the UserInterface class, which is
responsible for  displaying information to the user and taking input from the user.
 * @author Onur Onel
 */
#include "UserInterface.h"
#include <iomanip>

using namespace std;

/**
 * @brief Displays a list of Travel Records.
 *
 * @param records The list of travel records to be displayed.
 */
void UserInterface::displayRecords(const std::vector<TravelRecord> &records) {
    int widthId = 5;
    int widthRefNum = 15;
    int widthDiscGroup = 5;
    int widthTitle = 70;

    cout << left
         << setw(widthId) << "ID" << "   "
         << setw(widthRefNum) << "Ref Number "
         << setw(widthDiscGroup) << "Group "
```

```
         << setw(widthTitle) << "Title " << endl;
    cout << string(widthId, '-') << " "
         << string(widthRefNum, '-') << " "
         << string(widthDiscGroup, '-') << " "s
         << string(25, '-') << endl;

    int counter = 0;
    for (const TravelRecord &record: records) {
        if (counter > 100) {
            break;
        }
        cout << left
             << setw(widthId) << record.getId() << " "
             << setw(widthRefNum) << record.getRefNumber() << " "
             << setw(widthDiscGroup) << record.getDisclosureGroup() << " "
             << setw(widthTitle) << record.getTitleEn() << endl;
        counter++;
    }
}

/**
 * @brief Displays a given message to the user.
 *
 * @param message The message to display.
 */
void UserInterface::displayMessage(const string &message) {
    cout << message << endl;
}

/**
 * @brief Displays an error message to the user.
 *
 * @param error The error message to display.
 */
void UserInterface::displayError(const string &error) {
    cerr << "Error: " << error << endl;
}

/**
 * @brief Displays the main menu of the application.
 */
void UserInterface::showMainMenu() {
    cout << "\n\n";
    cout << "*****************************************" << endl;
    cout << "* Welcome to the Advanced CSV Reader App! *" << endl;
    cout << "*****************************************" << endl;
    cout << "\nPlease select an option:\n" << endl;
    cout << " 1. Load Records from Existing CSV File" << endl;
    cout << " 2. Save Records to New File" << endl;
    cout << " 3. Load new Records from New File" << endl;
    cout << " 4. Delete Record by ID" << endl;
    cout << " 5. Add Record" << endl;
    cout << " 6. Delete All Records from New File" << endl;
    cout << " 7. Exit\n" << endl;
}

/**
 * @brief Requests user input with a given prompt.
 *
 * @param prompt The prompt to display before taking input.
 * @return The string entered by the user.
 */
string UserInterface::requestInput(const string &prompt) {
```

```
    cout << prompt;
    std::string input;
    getline(cin, input);
    return input;
}
/**
 * @file RecordController.h
 * @brief Main controller class that handles user interaction and manages the record
operations.
 * @details This class is responsible for handling user interactions and managing the
record operations.
 * @author Onur Onel
 */

#ifndef ASSIGNMENT02_RECORDCONTROLLER_H
#define ASSIGNMENT02_RECORDCONTROLLER_H

#include "RecordManager.h"
#include "UserInterface.h"

/**
 * @class RecordController
 * @brief Manages the main operations and user interactions for the record system.
 */
class RecordController {
private:
    RecordManager manager;       ///< Manager to handle CRUD operations on records.
    UserInterface ui;            ///< Interface for user interactions.
    std::string lastOperation;   ///< Keeps track of the last performed operation.

public:
    /**
     * @brief Default constructor.
     */
    RecordController();

    /**
     * @brief Displays main menu and processes user choice.
     */
    void run();

    /**
     * @brief Saves the current records to a file.
     */
    void saveRecords();

    /**
     * @brief Displays the loaded records.
     */
    void displayRecords();

    /**
     * @brief Deletes a specific record identified by its ID.
     */
    void deleteRecordById();

    /**
     * @brief Allows the user to input and add a new record.
     */
    void addRecord();

    /**
     * @brief Asks the user if they wish to continue or exit the application.
```

```
     *
     * @return User's choice as a string.
     */
    std::string requestContinue();

    /**
     * @brief Loads records from a given file path.
     *
     * @param filePath Path to the file to load records from.
     */
    void loadRecords(const std::string &filePath);

    /**
     * @brief Loads new records from a given file path.
     *
     * @param newFilePath Path to the new file to load records from.
     */
    void loadNewRecords(const std::string &newFilePath);

    /**
     * @brief Deletes all records from a given file path.
     *
     * @param path Path to the file to delete all records from.
     */
    void deleteAllRecords(const std::string &path);
};

#endif //ASSIGNMENT02_RECORDCONTROLLER_H
/**
 * @file RecordController.cpp
 * @brief Implements the main logic and user interactions for the CSV Reader
application.
 * @details This class is responsible for handling user interactions such as displaying
records,
 * @author Onur Onel
 */
#include "RecordController.h"

using namespace std;

/**
 * @brief Default constructor.
 */
RecordController::RecordController() : manager(), ui() {}

/**
 * @brief Handles the main menu display and user's choice for various operations.
 */
void RecordController::run() {
    ui.showMainMenu();
    string choice = ui.requestInput("");
    if (choice == "1") {
        loadRecords(R"(C:\Users\ONUR\CLionProjects\Assignment02\travelq.csv)");
        displayRecords();
    } else if (choice == "2") {
        saveRecords();
    } else if (choice == "3") {
        loadNewRecords(R"(C:\Users\ONUR\CLionProjects\Assignment02\newfile.csv)");
        displayRecords();
    } else if (choice == "4") {
        deleteRecordById();
    } else if (choice == "5") {
        addRecord();
```

```
    } else if (choice == "6") {
        deleteAllRecords(R"(C:\Users\ONUR\CLionProjects\Assignment02\newfile.csv)");
    } else if (choice == "7") {
    } else {
        ui.displayError("Invalid choice. Please try again.");
    }
}

/**
 * @brief Loads records from a specified file.
 *
 * @param filePath Path to the CSV file to load records from.
 */
void RecordController::loadRecords(const string &filePath) {
    try {
        manager.loadRecordsFromFile(filePath);
        ui.displayMessage("Records loaded successfully!");
    } catch (const exception &e) {
        ui.displayError(string("An error occurred: ") + e.what());
    }
    lastOperation = "load";
}

/**
 * @file RecordController.cpp
 * @brief Saves the current records.
 */
void RecordController::saveRecords() {
    if (manager.getRecords().empty()) {
        ui.displayError("No records to save!");
    } else if (lastOperation != "addRecord") {

manager.saveRecordsToFile(R"(C:\Users\ONUR\CLionProjects\Assignment02\newfile.csv)");
        ui.displayMessage("Records saved successfully!");
    }
    lastOperation = "save";
}

/**
 * @brief Displays the current set of records.
 */
void RecordController::displayRecords() {
    auto records = manager.getRecords();
    ui.displayRecords(records);
    lastOperation = "display";
}

/**
 * @brief Loads records from a new file.
 *
 * @param newFilePath Path to the new CSV file to load records from.
 */
void RecordController::loadNewRecords(const string &newFilePath) {
    try {
        manager.loadRecordsNewFromFile(newFilePath);
        ui.displayMessage("Records loaded successfully!");
    } catch (const exception &e) {
        ui.displayError(string("An error occurred: ") + e.what());
    }
    lastOperation = "loadNew";
}

/**
```

```
 * @brief Deletes a record specified by its ID.
 */
void RecordController::deleteRecordById() {
    string id = ui.requestInput("Enter the ID of the record to delete: ");
    manager.deleteRecordById(id);
    ui.displayMessage("Record deleted successfully!");
    lastOperation = "deleteRecord";
}

/**
 * @brief Allows the user to add a new record.
 */
void RecordController::addRecord() {
    string refNumber = ui.requestInput("Enter the reference number: ");
    string disclosureGroup = ui.requestInput("Enter the disclosure group: ");
    string titleEn = ui.requestInput("Enter the title in English: ");

    TravelRecord newRecord;
    newRecord.setRefNumber(refNumber);
    newRecord.setDisclosureGroup(disclosureGroup);
    newRecord.setTitleEn(titleEn);

    manager.addRecord(newRecord);
    ui.displayMessage("Record added successfully!");
    lastOperation = "addRecord";
}

/**
 * @brief Deletes all records.
 *
 * @param path Path to the CSV file to delete records from.
 */
void RecordController::deleteAllRecords(const string &path) {
    manager.deleteAllRecords(path);
    ui.displayMessage("All records deleted successfully!");
    lastOperation = "deleteAll";
}

/**
 * @brief Asks the user whether they want to continue or exit.
 *
 * @return User's choice as a string.
 */
string RecordController::requestContinue() {
    return ui.requestInput("Press any key to continue or 'exit' to quit: ");
}
/**
 * @file RecordManager.h
 * @brief Provides utilities for manipulating a collection of TravelRecord objects.
 * @details This class is responsible for handling user interactions such as displaying
records
 * @author Onur Onel
 */
#ifndef ASSIGNMENT02_RECORDMANAGER_H
#define ASSIGNMENT02_RECORDMANAGER_H

#include "FileHandler.h"
#include <vector>

/**
 * @class RecordManager
 * @brief Provides utilities for manipulating a collection of TravelRecord objects.
 */
```

```
class RecordManager {
private:
    /** @brief Stores all the travel records managed by this instance. */
    std::vector<TravelRecord> records;

public:
    /**
     * Load travel records from a given file.
     * @param path Path to the file.
     * @return Number of records loaded.
     */
    int loadRecordsFromFile(const std::string &path);

    /**
     * Save the current in-memory records to a given file.
     * @param path Path to the file.
     */
    void saveRecordsToFile(const std::string &path);

    /**
     * Delete a specific record identified by its ID.
     * @param id ID of the record to delete.
     */
    void deleteRecordById(const std::string &id);

    /**
     * Add a new record to the in-memory collection.
     * @param record The record to add.
     */
    void addRecord(const TravelRecord& record);

    /**
     * Get all the in-memory records.
     * @return Constant reference to the records vector.
     */
    const std::vector<TravelRecord>& getRecords() const;

    /**
     * Delete all records in a file and clear in-memory records.
     * @param path Path to the file.
     */
    void deleteAllRecords(const std::string &path);

    /**
     * Load records from a new file format or structure.
     * @param path Path to the file.
     * @return Number of records loaded.
     */
    int loadRecordsNewFromFile(const std::string &path);
};

#endif //ASSIGNMENT02_RECORDMANAGER_H
/**
 * @file RecordManager.h
 * @brief Provides utilities for manipulating a collection of TravelRecord objects.
 * @details This class is responsible for handling user interactions such as displaying
records
 * @author Onur Onel
 */
#ifndef ASSIGNMENT02_RECORDMANAGER_H
#define ASSIGNMENT02_RECORDMANAGER_H

#include "FileHandler.h"
```

```cpp
#include <vector>

/**
 * @class RecordManager
 * @brief Provides utilities for manipulating a collection of TravelRecord objects.
 */
class RecordManager {
private:
    /** @brief Stores all the travel records managed by this instance. */
    std::vector<TravelRecord> records;

public:
    /**
     * Load travel records from a given file.
     * @param path Path to the file.
     * @return Number of records loaded.
     */
    int loadRecordsFromFile(const std::string &path);

    /**
     * Save the current in-memory records to a given file.
     * @param path Path to the file.
     */
    void saveRecordsToFile(const std::string &path);

    /**
     * Delete a specific record identified by its ID.
     * @param id ID of the record to delete.
     */
    void deleteRecordById(const std::string &id);

    /**
     * Add a new record to the in-memory collection.
     * @param record The record to add.
     */
    void addRecord(const TravelRecord& record);

    /**
     * Get all the in-memory records.
     * @return Constant reference to the records vector.
     */
    const std::vector<TravelRecord>& getRecords() const;

    /**
     * Delete all records in a file and clear in-memory records.
     * @param path Path to the file.
     */
    void deleteAllRecords(const std::string &path);

    /**
     * Load records from a new file format or structure.
     * @param path Path to the file.
     * @return Number of records loaded.
     */
    int loadRecordsNewFromFile(const std::string &path);
};

#endif //ASSIGNMENT02_RECORDMANAGER_H
/**
 * @file FileHandler.h
 * @brief FileHandler class declaration.
 * @details Utility methods for reading from and writing to CSV files.
 * @author Onur Onel
```

```
 */

#ifndef ASSIGNMENT02_FILEHANDLER_H
#define ASSIGNMENT02_FILEHANDLER_H

#include <vector>
#include "TravelRecord.h"

/**
 * @class FileHandler
 * @brief Utility class for file operations related to TravelRecord objects.
 */
class FileHandler {
public:
    /**
     * @brief Reads records from a specified CSV file.
     * @param path Path to the CSV file.
     * @return Vector of TravelRecord objects.
     */
    static std::vector<TravelRecord> readFromFile(const std::string &path);

    /// Static container for storing records.
    static std::vector<TravelRecord> records;

    /**
     * @brief Writes records to a specified CSV file.
     * @param path Path to the destination CSV file.
     */
    static void writeToFile(const std::string &path);

    /**
     * @brief Reads records from a new specified CSV file.
     * @param path Path to the CSV file.
     * @return Vector of TravelRecord objects.
     */
    static std::vector<TravelRecord> readFromNewFile(const std::string &path);
};

#endif //ASSIGNMENT02_FILEHANDLER_H
/**
 * @file FileHandler.cpp
 * @brief Contains methods for reading from and writing to CSV files.
 * @details This file contains the implementation of the FileHandler class.
 * @author Onur Onel
 */

#include "TravelRecord.h"
#include "FileHandler.h"
#include "csv-parser-master/single_include/csv.hpp"
#include "UserInterface.h"
#include <iostream>
#include <vector>
#include <exception>

using namespace std;
using namespace csv;

/// Static container for storing records.
vector<TravelRecord> FileHandler::records;

/**
 * @brief Reads records from a specified CSV file.
 * @param path Path to the CSV file.
```

```
 * @return Vector of TravelRecord objects.
 */
vector<TravelRecord> FileHandler::readFromFile(const string &path) {
    FileHandler::records.clear();

    try {
        CSVReader reader(path);
        int count = 0;
        for (CSVRow &row: reader) {
            if (count > 100) break;

            TravelRecord record;
            record.setId(to_string(count));
            record.setRefNumber(row["ref_number"].get<string>());
            record.setDisclosureGroup(row["disclosure_group"].get<string>());
            record.setTitleEn(row["title_en"].get<string>());

            FileHandler::records.push_back(record);
            count++;
        }
    } catch (const exception &e) {
        cout << "Error reading CSV: " << e.what() << endl;
    }

    return FileHandler::records;
}

/**
 * @brief Reads records from a new specified CSV file.
 * @param path Path to the CSV file.
 * @return Vector of TravelRecord objects.
 */
vector<TravelRecord> FileHandler::readFromNewFile(const string &path) {

    try {
        CSVReader reader(path);
        int count = 0;
        for (CSVRow &row: reader) {
            if (count > 100) break;

            TravelRecord record;
            record.setId(to_string(count));
            record.setRefNumber(row["ref_number"].get<string>());
            record.setDisclosureGroup(row["disclosure_group"].get<string>());
            record.setTitleEn(row["title_en"].get<string>());

            FileHandler::records.push_back(record);
            count++;
        }
    } catch (const exception &e) {
        cout << "Error reading CSV: " << e.what() << endl;
    }

    return FileHandler::records;
}

/**
 * @brief Writes records to a specified CSV file.
 * @param path Path to the destination CSV file.
 */
void FileHandler::writeToFile(const string &path) {
    ofstream outFile;
    bool appendMode = false;
```

```
    ifstream checkFile(path);
    if (checkFile.peek() == ifstream::traits_type::eof()) {
        outFile.open(path);
        outFile << "ID,ref_number,disclosure_group,title_en\n";
    } else {
        outFile.open(path, ios::app);
        appendMode = true;
    }
    checkFile.close();

    if (!outFile.is_open()) {
        UserInterface::displayError("Could not open file " + path + " for writing.");
        return;
    }

    size_t count = 0;

    try {
        if (!appendMode) {
            outFile << "ID,ref_number,disclosure_group,title_en\n";
        }

        for (const TravelRecord &record: FileHandler::records) {
            if (count >= 200) break;

            outFile << "\"" << record.getId() << "\","
                    << "\"" << record.getRefNumber() << "\","
                    << "\"" << record.getDisclosureGroup() << "\","
                    << "\"" << record.getTitleEn() << "\"\n";
            count++;
        }
    } catch (const exception &) {
        UserInterface::displayError("Error writing to CSV.");
    }

    outFile.close();
}
```