

Programming Language Research Assignment 1

Practical Project 01

Algonquin College
School of Advanced Technology
Computer Programming

Onur Önel
Mazin Abou-Seido

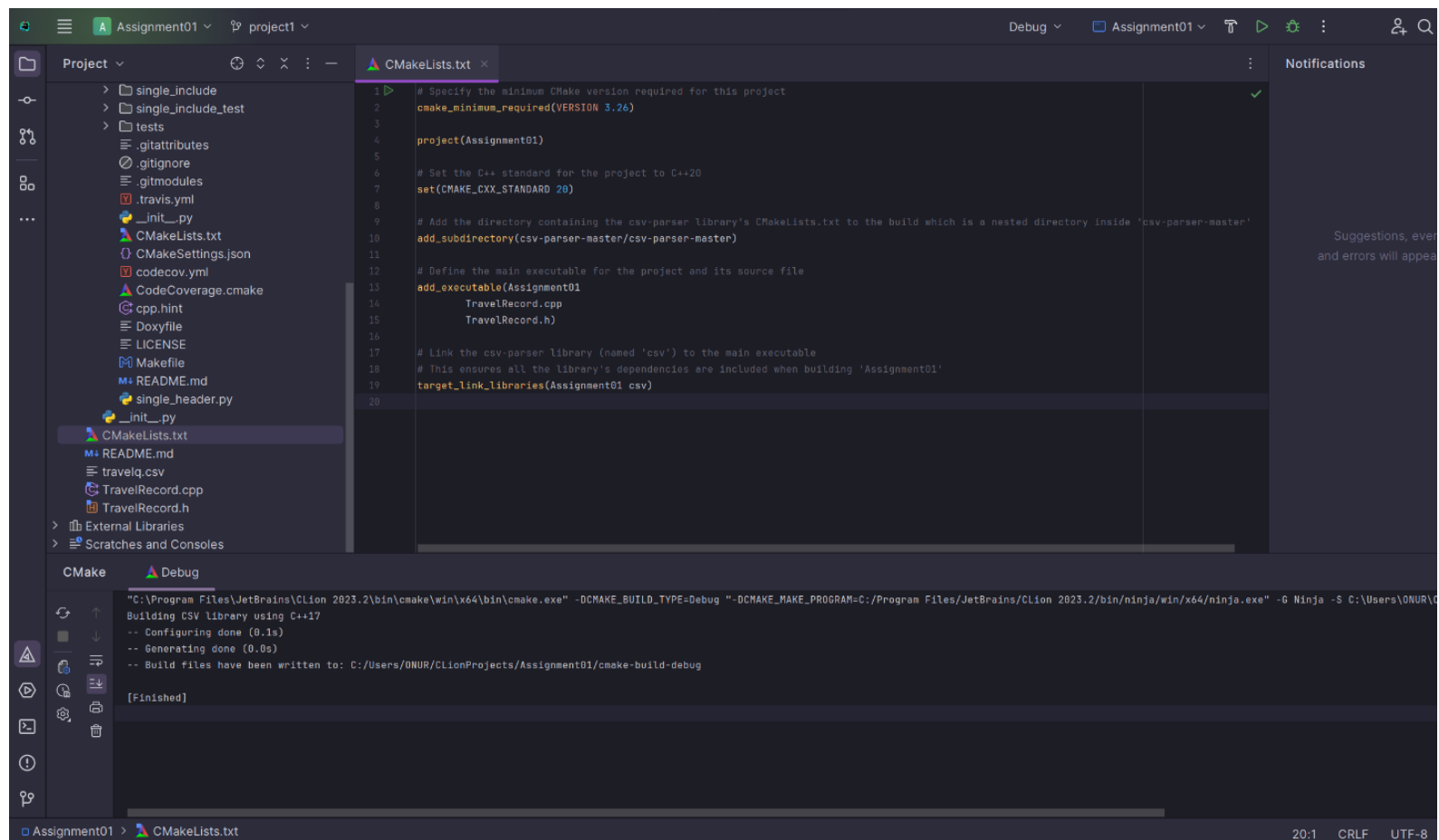
23F_CST8333_360

Submitted September 23, 2023

A technical report submitted to Algonquin College in partial fulfillment of the requirements for a diploma in Computer Programming

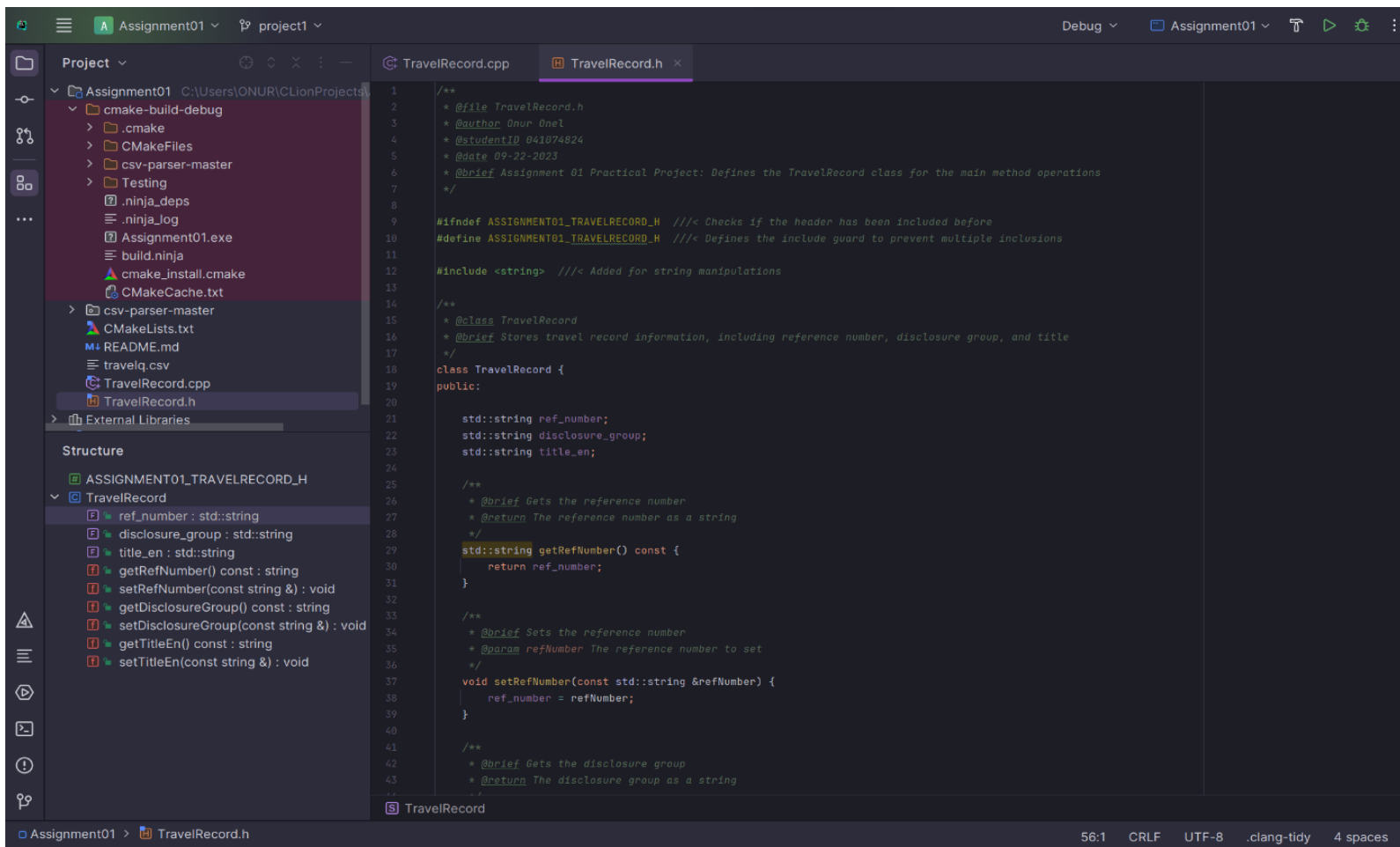
Program Demonstration via Screen Shots

I've added the CSV Parser API from GitHub “ <https://github.com/vincentlaucsb/csv-parser> ” into the directory where my project is located. The build is managed by a CMake script that starts by defining a minimum CMake version of 3.26. The script then sets the project name to "Assignment01" and specifies that the C++ standard being used is C++20. To include the third-party library, csv-parser, the script integrates it into the build process via `add_subdirectory(csv-parser-master/csv-parser-master)`. The primary executable for this project, which is called "Assignment01," is established by specifying its source files, `TravelRecord.cpp` and `TravelRecord.h`. This is done using the command `add_executable`. As a final step, the csv-parser library is linked to the main executable through `target_link_libraries(Assignment01 csv)` command. This ensures that all necessary dependencies are included when compiling the "Assignment01" project.



Linking the csv-parser library to the main executable ensures that all required dependencies are automatically included during the build, minimizing potential errors and enhancing portability. By making these adjustments to the CMakeLists.txt file, I've made the build process more robust and maintainable.

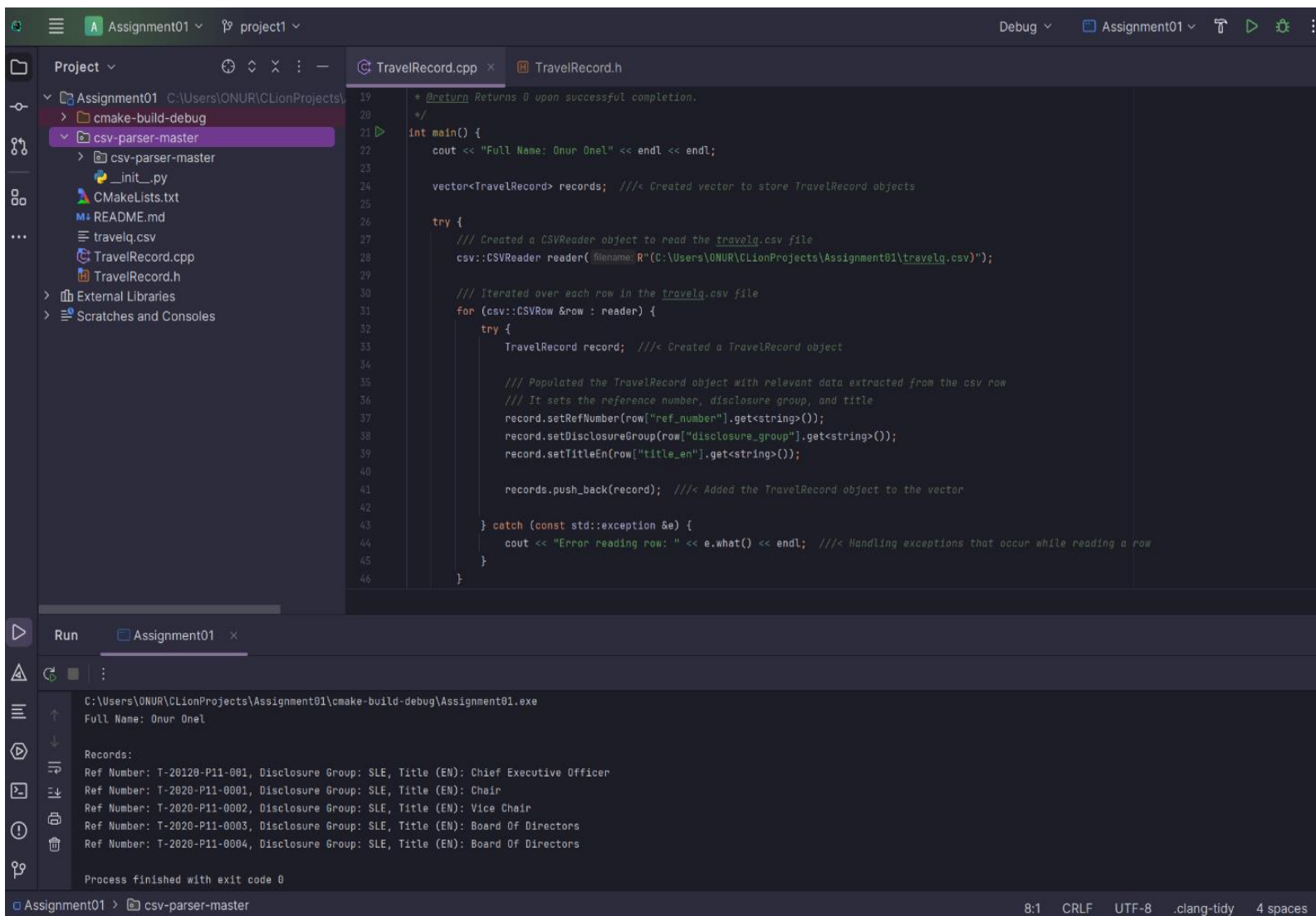
Through the TravelRecord header file, I defined the getter and setter methods for the columns that we are interested in reading. Specifically, we focus on the columns "ref_number," "disclosure_group," and "title_en." By defining these methods, we create a structured way to access or modify the data in these columns. The getter methods enable us to retrieve the values of these specific fields, while the setter methods allow us to update or change them as needed. This encapsulation of data fields ensures that we can manipulate and access the data in a consistent and controlled manner



The screenshot displays a C++ development environment with a project named "Assignment01". The left sidebar shows the project structure, including folders like "cmake-build-debug", "CMakeFiles", and "Testing", and files like "Assignment01.exe", "build.ninja", "cmake_install.cmake", "CMakeCache.txt", "csv-parser-master", "CMakeLists.txt", "README.md", "travelq.csv", "TravelRecord.cpp", and "TravelRecord.h". The "Structure" panel on the left lists the members of the "TravelRecord" class: "ref_number : std::string", "disclosure_group : std::string", "title_en : std::string", "getRefNumber() const : string", "setRefNumber(const string &) : void", "getDisclosureGroup() const : string", "setDisclosureGroup(const string &) : void", "getTitleEn() const : string", and "setTitleEn(const string &) : void". The main editor window shows the "TravelRecord.h" header file, which includes a comment block with author, student ID, date, and a brief description. It also contains an include guard, an include for <string>, and the class definition for "TravelRecord". The class has three public attributes: "ref_number", "disclosure_group", and "title_en", all of type "std::string". It also has three public methods: "getRefNumber() const", "setRefNumber(const std::string &refNumber)", and "getTitleEn() const". The "setRefNumber" method is implemented in the header file, setting "ref_number" to "refNumber".

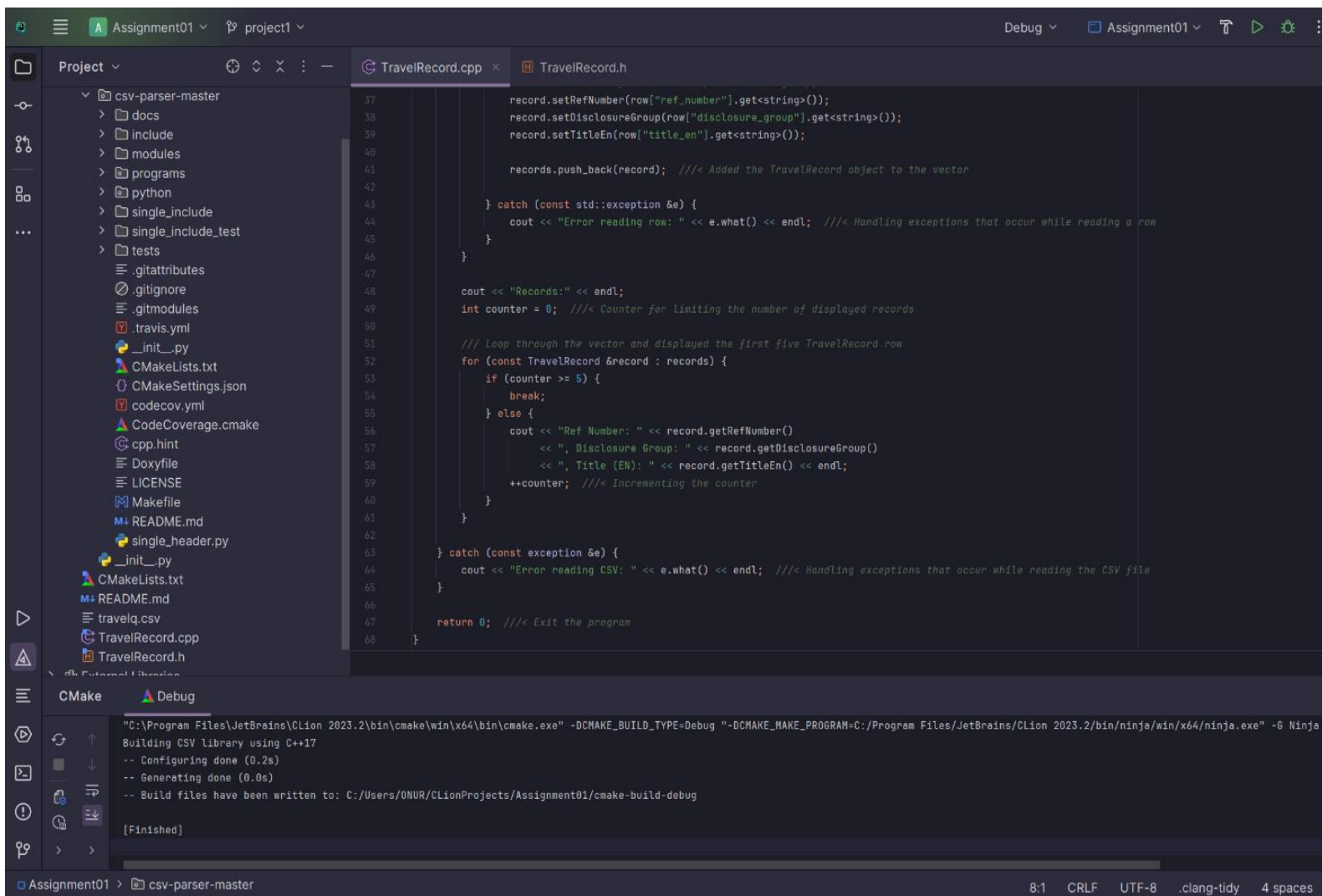
```
1  /**
2   * @file TravelRecord.h
3   * @author Onur Onel
4   * @studentID 041074824
5   * @date 09-22-2023
6   * @brief Assignment 01 Practical Project: Defines the TravelRecord class for the main method operations
7   */
8
9  #ifndef ASSIGNMENT01_TRAVELRECORD_H ///< Checks if the header has been included before
10 #define ASSIGNMENT01_TRAVELRECORD_H ///< Defines the include guard to prevent multiple inclusions
11
12 #include <string> ///< Added for string manipulations
13
14 /**
15  * @class TravelRecord
16  * @brief Stores travel record information, including reference number, disclosure group, and title
17  */
18 class TravelRecord {
19 public:
20
21     std::string ref_number;
22     std::string disclosure_group;
23     std::string title_en;
24
25     /**
26      * @brief Gets the reference number
27      * @return The reference number as a string
28      */
29     std::string getRefNumber() const {
30         return ref_number;
31     }
32
33     /**
34      * @brief Sets the reference number
35      * @param refNumber The reference number to set
36      */
37     void setRefNumber(const std::string &refNumber) {
38         ref_number = refNumber;
39     }
40
41     /**
42      * @brief Gets the disclosure group
43      * @return The disclosure group as a string
44      */
45     std::string getDisclosureGroup() const {
46         return disclosure_group;
47     }
48
49     /**
50      * @brief Sets the disclosure group
51      * @param disclosureGroup The disclosure group to set
52      */
53     void setDisclosureGroup(const std::string &disclosureGroup) {
54         disclosure_group = disclosureGroup;
55     }
56
57     /**
58      * @brief Gets the title
59      * @return The title as a string
60      */
61     std::string getTitleEn() const {
62         return title_en;
63     }
64
65     /**
66      * @brief Sets the title
67      * @param titleEn The title to set
68      */
69     void setTitleEn(const std::string &titleEn) {
70         title_en = titleEn;
71     }
72 }
```

In C++ development, it's a common best practice to separate code into header (.h) and source (.cpp) files, and the TravelRecord class follows this pattern. The header file acts as an interface, showing the structure of the TravelRecord class and its public methods. This makes it easier for other parts of the code, or even other developers, to understand what the TravelRecord class does without having to dig into the implementation details. The source (.cpp) file, on the other hand, contains the actual implementation of the methods declared in the header file. This separation makes the code more organized and easier to manage. If we want to change how a method works, we only need to update the source file without affecting the header file, which might be included in multiple other parts.



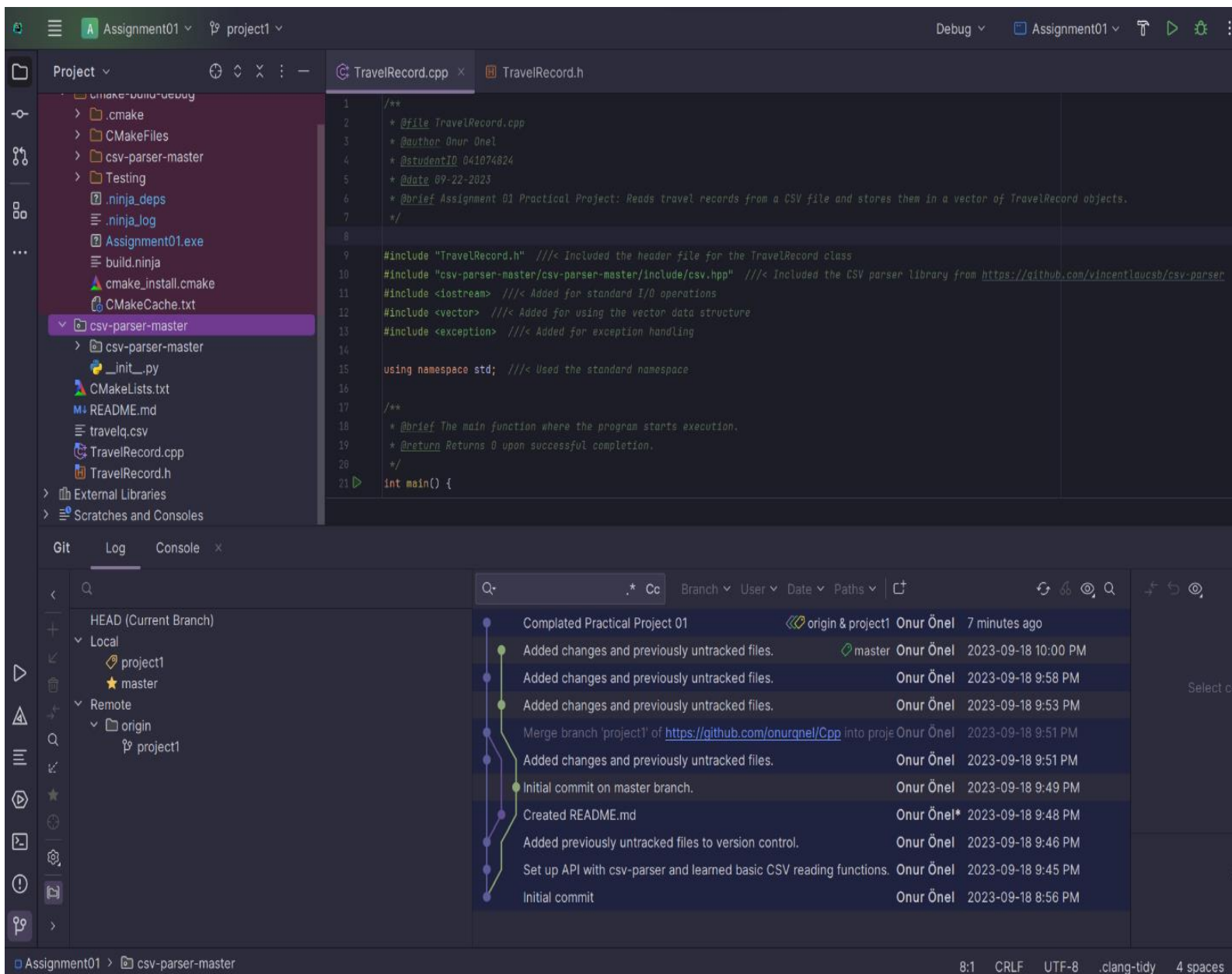
My program starts by showing my name, Onur Onel, on the console output. Next, I create a vector named 'records,' which is essentially a structured container designed to hold instances of the TravelRecord class. This list will hold all the travel data that I want to demonstrate on the output. I used a try block to read a CSV file safely. Inside the blocks, I created a 'reader' object to read data from a file named 'travelq.csv.'

Then, I used a for loop to go through each row of the file. I use another try block inside to catch any errors while reading each row. I made a new TravelRecord object called 'record' which is filled in travel record details using data from the current row. I use specialized methods like 'setRefNumber()' in the process. After filling each 'record,' I added it to my 'records' list. If any error happens while reading a row, the try block catches it then shows an error message on the screen. Additionally I learned namespaces. For example, I used the std library for printing, so I include 'using namespace std;' in my code. This lets me use 'CSVReader' as 'csv::CSVReader' when I include 'using namespace csv;'



After I've populated the 'records' vector with travel data, it's time to display some of that information. The program prints "Records:" to the console to indicate that we're about to see a sample of the collected data. I introduce a counter variable set to zero. This counter helps limit the number of records displayed, ensuring that only the first five records from the 'records' vector are shown. This way, I avoid overwhelming the console with too much data at once.

Then I initiate a loop to iterate through the 'records' vector. For each 'TravelRecord' instance, referred to as 'record,' I first check the counter's current value. If the counter has reached or exceeded five, the loop breaks, ending this part of the display. Otherwise, the program proceeds to output the 'Ref Number,' 'Disclosure Group,' and 'Title (EN)' attributes of the current 'record.' After each display, the counter is incremented by one to keep track of the number of records shown. In case any exceptions are encountered while reading the CSV file, these are caught and an error message is displayed using 'cout.' The message specifies what went wrong, aiding in troubleshooting. Finally, the program returns zero, signaling a successful execution and exiting the program.



In addition to writing the code, I also managed my project using a private GitHub repository named "Cpp." I used this repo to track changes in my code and to document the development process. Whenever I added new features like the try blocks or specialized methods, or when I made changes to existing parts of the code, I made commits to the GitHub repository.

These commits serve as a history log, showing what changes were made, when, and why. They help me stay organized and make it easier to fix mistakes or go back to older versions if needed. The private setting also ensures that the data and code are secure and only accessible to those I choose to share it with.

Source Code Commenting Example

I have improved the clarity and usefulness of code comments by adhering to the *Doxygen* style with the '///< and '/*comment*/' format. This approach allows for the easy generation of documentation directly from the code, which I have done to enhance code comprehension and readability.

TravelRecord.cpp

```
/**
 * @file TravelRecord.cpp
 * @author Onur Onel
 * @studentID 041074824
 * @date 09-22-2023
 * @brief Assignment 01 Practical Project: Reads travel records from a CSV file and stores
them in a vector of TravelRecord objects.
 */

#include "TravelRecord.h" ///< Included the header file for the TravelRecord class
#include "csv-parser-master/csv-parser-master/include/csv.hpp" ///< Included the CSV parser
library from https://github.com/vincentlaucsb/csv-parser
#include <iostream> ///< Added for standard I/O operations
#include <vector> ///< Added for using the vector data structure
#include <exception> ///< Added for exception handling

using namespace std; ///< Used the standard namespace

/**
 * @brief The main function where the program starts execution.
 * @return Returns 0 upon successful completion.
 */
int main() {
    cout << "Full Name: Onur Onel" << endl << endl;

    vector<TravelRecord> records; ///< Created vector to store TravelRecord objects

    try {
        /// Created a CSVReader object to read the 7travel.csv file
        csv::CSVReader reader(R"(C:\Users\ONUR\ClionProjects\Assignment01\travelq.csv)");

        /// Iterated over each row in the 7travel.csv file
        for (csv::CSVRow &row : reader) {
            try {
                TravelRecord record; ///< Created a TravelRecord object

                /// Populated the TravelRecord object with relevant data extracted from the
csv row

                /// It sets the reference number, disclosure group, and title
                record.setRefNumber(row["ref_number"].get<string>());
                record.setDisclosureGroup(row["disclosure_group"].get<string>());
                record.setTitleEn(row["title_en"].get<string>());

                records.push_back(record); ///< Added the TravelRecord object to the vector
            }
        }
    }
}
```

```

        } catch (const std::exception &e) {
            cout << "Error reading row: " << e.what() << endl;    ///< Handling exceptions
that occur while reading a row
        }
    }

    cout << "Records:" << endl;
    int counter = 0;    ///< Counter for limiting the number of displayed records

    ///< Loop through the vector and displayed the first five TravelRecord row
    for (const TravelRecord &record : records) {
        if (counter >= 5) {
            break;
        } else {
            cout << "Ref Number: " << record.getRefNumber()
                << ", Disclosure Group: " << record.getDisclosureGroup()
                << ", Title (EN): " << record.getTitleEn() << endl;
            ++counter;    ///< Incrementing the counter
        }
    }

    } catch (const exception &e) {
        cout << "Error reading CSV: " << e.what() << endl;    ///< Handling exceptions that
occur while reading the CSV file
    }

    return 0;    ///< Exit the program
}

```

TravelRecord.h

```

/**
 * @file TravelRecord.h
 * @author Onur Onel
 * @studentID 041074824
 * @date 09-22-2023
 * @brief Assignment 01 Practical Project: Defines the TravelRecord class for the main
method operations
 */

#ifndef ASSIGNMENT01_TRAVELRECORD_H    ///< Checks if the header has been included before
#define ASSIGNMENT01_TRAVELRECORD_H    ///< Defines the include guard to prevent multiple
inclusions

#include <string>    ///< Added for string manipulations

/**
 * @class TravelRecord
 * @brief Stores travel record information, including reference number, disclosure group,
and title
 */
class TravelRecord {
public:

```



```

std::string ref_number;
std::string disclosure_group;
std::string title_en;

/**
 * @brief Gets the reference number
 * @return The reference number as a string
 */
std::string getRefNumber() const {
    return ref_number;
}

/**
 * @brief Sets the reference number
 * @param refNumber The reference number to set
 */
void setRefNumber(const std::string &refNumber) {
    ref_number = refNumber;
}

/**
 * @brief Gets the disclosure group
 * @return The disclosure group as a string
 */
std::string getDisclosureGroup() const {
    return disclosure_group;
}

/**
 * @brief Sets the disclosure group
 * @param disclosureGroup The disclosure group to set
 */
void setDisclosureGroup(const std::string &disclosureGroup) {
    disclosure_group = disclosureGroup;
}

/**
 * @brief Gets the title in English
 * @return The title in English as a string
 */
std::string getTitleEn() const {
    return title_en;
}

/**
 * @brief Sets the title in English
 * @param titleEn The title in English to set
 */
void setTitleEn(const std::string &titleEn) {
    title_en = titleEn;
}
};

#endif //ASSIGNMENT01_TRAVELRECORD_H

```

CMakeLists.txt

```
# Specify the minimum Cmake version required for this project
cmake_minimum_required(VERSION 3.26)

project(Assignment01)

# Set the C++ standard for the project to C++20
set(CMAKE_CXX_STANDARD 20)

# Add the directory containing the csv-parser library's CMakeLists.txt to the build which is
a nested directory inside 'csv-parser-master'
add_subdirectory(csv-parser-master/csv-parser-master)

# Define the main executable for the project and its source file
add_executable(Assignment01
    TravelRecord.cpp
    TravelRecord.h)

# Link the csv-parser library (named 'csv') to the main executable
# This ensures all the library's dependencies are included when building 'Assignment01'
target_link_libraries(Assignment01 csv)
```