Programming Language Research Assignment 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 of the requirements for a diploma in Computer Programming

MVC Architecture

The Model-View-Controller (MVC) is a design pattern widely used in web application development. Its main objective is to separate the application into three interconnected components:

1. Model: Often seen as the backbone, the Model is ensured with data management. This involves keeping the actual data, its storage mechanisms, and the core business logic. It can be defined as the Persistence Layer or Business Layer within the MVC structure.

2. View: Acting as the interface of the application, the View is all about user interaction. It showcases data to the users and, in some scenarios collects the inputs. The Presentation Layer and the View have the same definitions in Model-View-Controller pattern.

3. Controller: The Controller adeptly manages user inputs, ensuring the Model and View are updated as required. Acts as a bridge, it takes in user commands, processes them (which may involve changing the Model), and then updates the View.

4. Entity Object: The Entity Object, which deserves particular attention, defines the data structure in the MVC pattern. It includes important model components, increasing the architecture's stability.

When dealing with CSV files in C++, using the MVC (Model-View-Controller) design pattern can be significantly helpful. This pattern makes source code simpler by clearly defining the roles and responsibilities of different parts of the CSV handling.

The "Model" is the script holds all the essential information. The "View" is similar to the stage it's what the audience (or users) sees and interacts with. The "Controller" is like the director which is making sure everything goes according to the script and managing any changes or inputs.

MVC pattern leads to cleaner and more understandable code. It's easier to work on the project, reduce errors, and make improvements. Adopting the MVC pattern makes software development more orderly and straightforward, which can be especially beneficial for tasks like handling CSV files.

N-Layer Architecture

The N-Layer architecture helps creating scalable and maintainable applications. This pattern separates an application into multiple layers, each having its own distinct set of responsibilities.

1. **Data Access Layer:** Data Access Layer is using data persistence. This involves handling all database operations and storage mechanisms. Within the N-Layer architecture, it can be likened to the actual storage of data and its retrieval methods.

2. **Business Logic Layer:** Business Logic Layer is tasked with processing data between the Data Access Layer and the Presentation Layer. It contains all the core business rules, ensuring that the data sent to and received from the database remains consistent.

3. **Presentation Layer:** Presentation Layer is concerned with user interaction. It is responsible for demonstrating data to users and might also collect inputs from the users. The Presentation Layer is similar to the "View" in the MVC pattern, focusing on the user interface and user experience.

4. **Service Layer:** Service layer responsible for communication between the Business logic layer and the Presentation Layer. It encapsulates the application's business logic, ensuring smoother interactions.

5. **Entity Layer:** Entity Layer defines the data structure in the N-Layer architecture. It includes primary model components, ensuring that data remains consistent throughout its journey between layers.

Final Pattern Decision for Practical Project 02

I am choosing the **MVC pattern** for my C++ application because I value clear organization and separation. In C++, this decision is particularly beneficial, especially for programs that involve user interactions. Additionally, I can integrate powerful libraries like Qt, which are built around the MVC concept. By selecting MVC over N-Layer, I ensure that every part (Model, View, Controller) has its distinct role. This not only simplifies building and error-checking but also effects better collaboration among developers. It becomes easier to scale and maintain the application, as changes in one section have limited impact on others. In summary, while both architectural choices have their own benefits, the choice of MVC for a C++ application reflects a clear idea on UI clarity, modularity, and full potential of C++ libraries optimized for MVC pattern.

Overview Of the Layered Design With MVC:

The design focuses on a clear separation of concerns, dividing the project into distinct layers to ensure a clean and maintainable codebase.

1. Presentation Layer:

UserInterface.h: This header file declares all the necessary interfaces and classes to facilitate user interaction.

UserInterface.cpp: The source file which contains the implementation of the user interaction methods.

2. Business Layer:

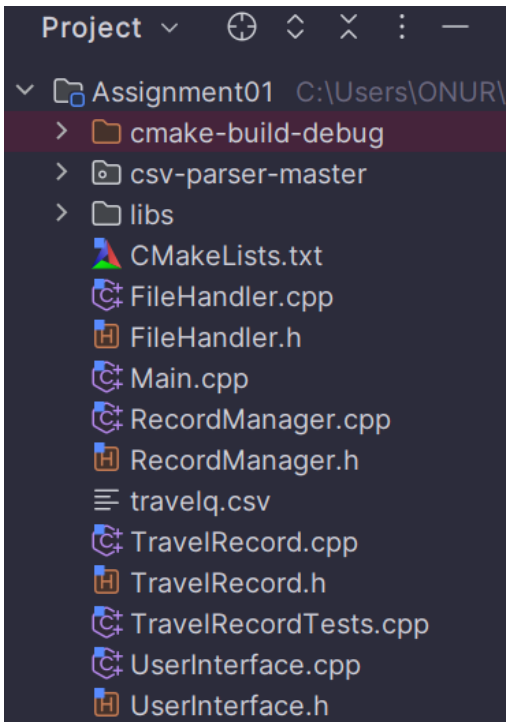
RecordManager.h: This header file acts as the blueprint for the record management system. It declares all the methods and data structures to handle travel records.

RecordManager.cpp: The basic logic controlling record management is included in the declared methods of the file RecordManager.cpp.

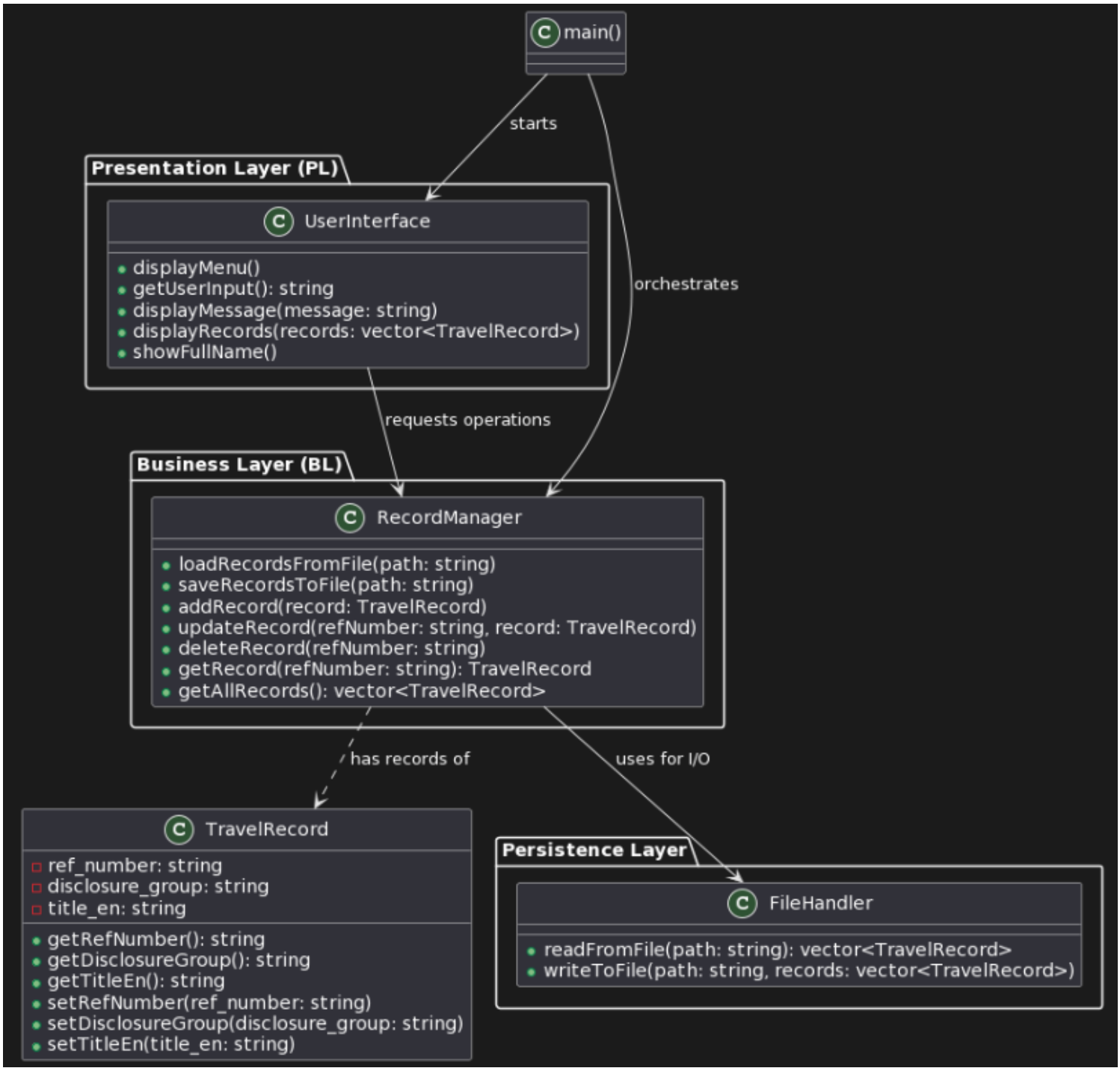
3. Persistence Layer:

FileHandler.h: This header file outlines the structure for file operations, ensuring a standardized way to interact with files.

FileHandler.cpp: It contains the actual file operations, such as reading from or writing to files, which are defined FileHandler.cpp.

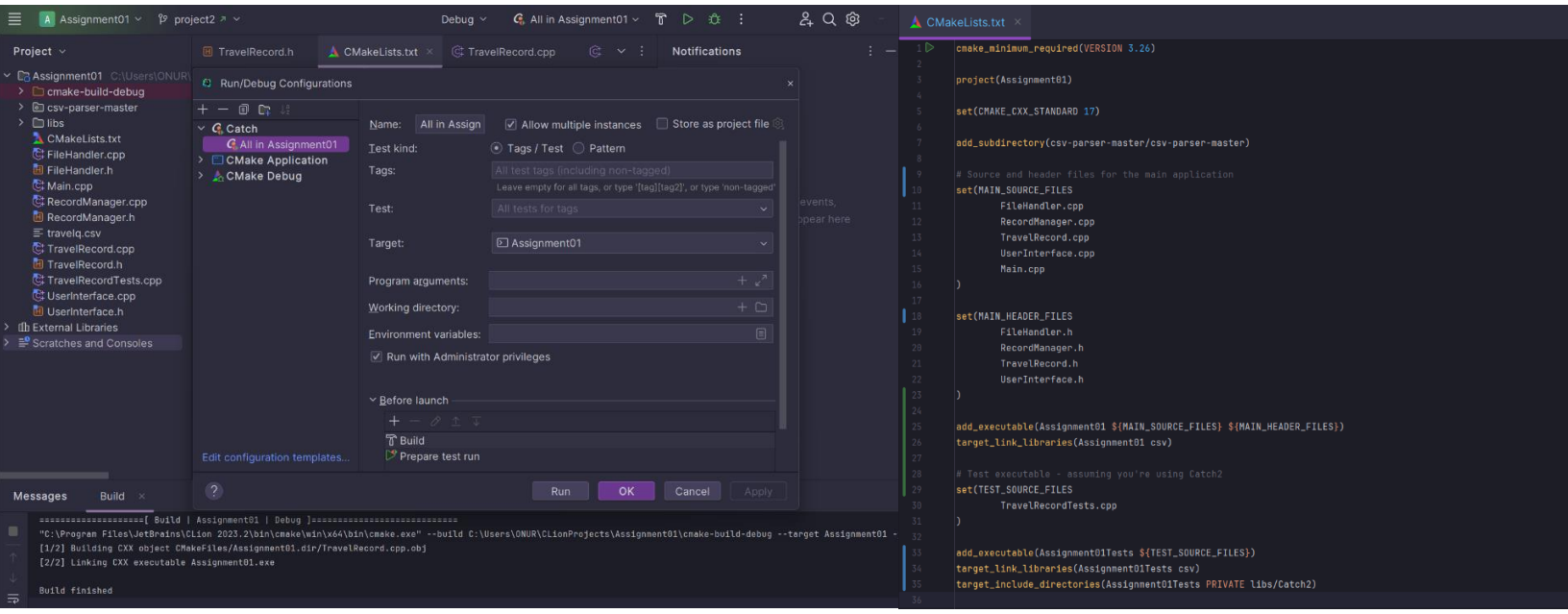


UML Class Diagram: Enhanced Design for Project Phase II



The Presentation Layer, through the UserInterface class, handles user interactions, displaying menus, messages, and records. The Business Layer has operations on travel records using the RecordManager class. This class bridges with the FileHandler in the Persistence Layer for data storage and retrieval. The TravelRecord class encapsulates individual record data which has methods for data access and modification.

Unit Testing with Catch2



CLion, JetBrains' cross-platform C++ IDE, provides native support for Catch, allowing developers to work with Catch versions 1.7.2 and higher . This support facilitates smooth test discovery, execution, and result visualization within the IDE. Integrating Catch2 with CLion primarily revolves around the usage of CMake, Catch2's primary build system. Catch2 offers two exported CMake targets*: Catch2::Catch2* and *Catch2::Catch2WithMain.* The latter includes a main() function, making it useful for projects that don't define their custom main. To summarize, the integration of Catch2 with CLion streamlines the testing process for developers, making it easier to write, execute, and review test outcomes, all within the comfort of the CLion environment.

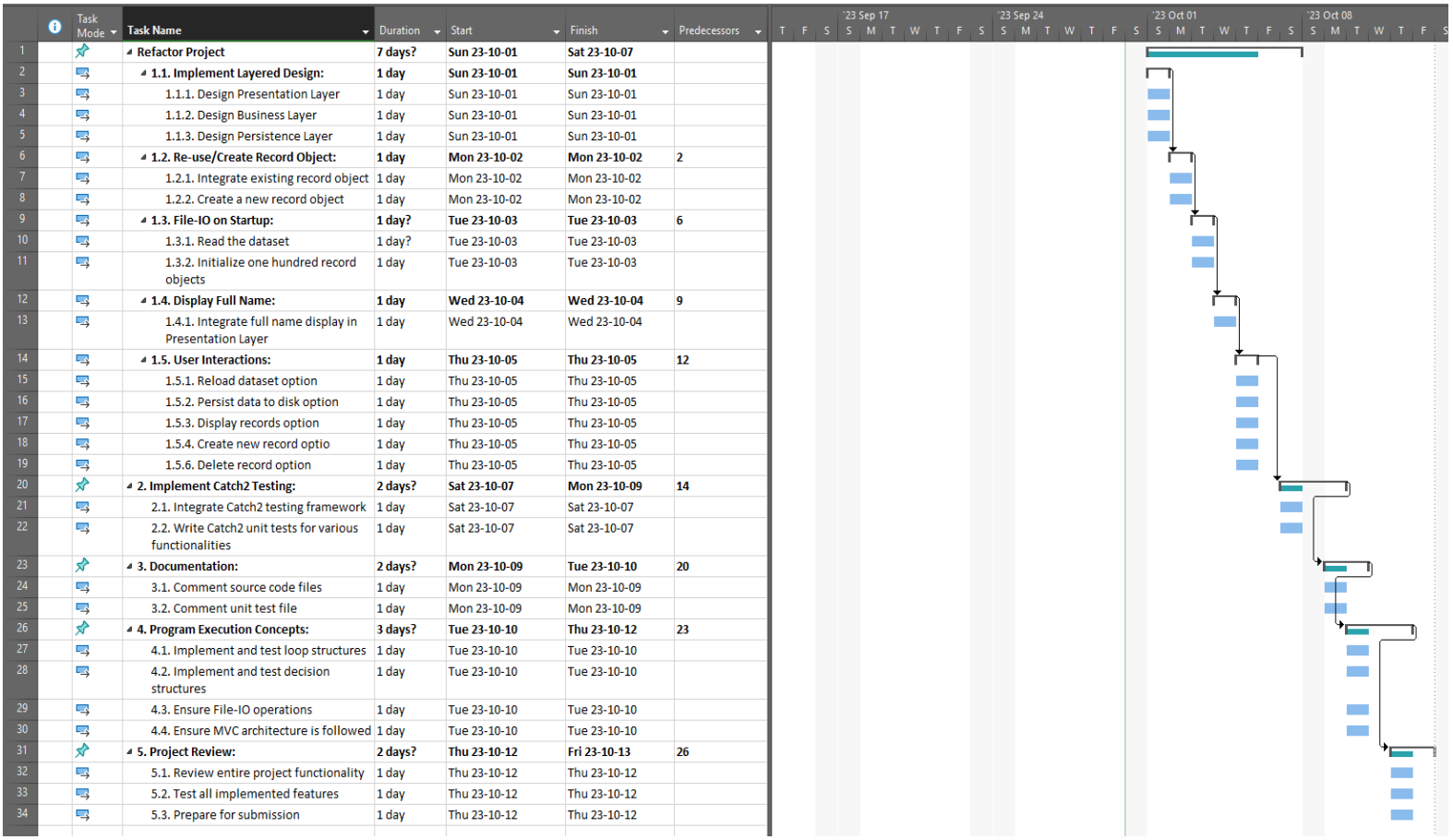
```
#define CATCH_CONFIG_MAIN // provides main(); this line is required in on
#include "catch_amalgamated.hpp"

int theAnswer() { return 6*9; } // function to be tested

TEST_CASE( "Life, the universe and everything", "[42][theAnswer]" ) {
REQUIRE(theAnswer() == 42);
}
```

In the above example, Life, the universe and everything is a free-form test name, which must be unique. The second argument of the TEST_CASE macro is a combination of two tags, [42] and [theAnswer]. Both test name and tags are regular strings that are not limited to be valid C++ identifiers. You can run collections of tests by specifying a wildcarded test name or a tag expression.

WBS and Gantt Chart for Practical Project Part 2



	Task Mode	Task Name	Duration	Start	Finish	Predecessors
1		**Refactor Project**	**7 days?**	**Sun 23-10-01**	**Sat 23-10-07**	
2		**1.1. Implement Layered Design:**	**1 day**	**Sun 23-10-01**	**Sun 23-10-01**	
3		1.1.1. Design Presentation Layer	1 day	Sun 23-10-01	Sun 23-10-01	
4		1.1.2. Design Business Layer	1 day	Sun 23-10-01	Sun 23-10-01	
5		1.1.3. Design Persistence Layer	1 day	Sun 23-10-01	Sun 23-10-01	
6		**1.2. Re-use/Create Record Object:**	**1 day**	**Mon 23-10-02**	**Mon 23-10-02**	**2**
7		1.2.1. Integrate existing record object	1 day	Mon 23-10-02	Mon 23-10-02	
8		1.2.2. Create a new record object	1 day	Mon 23-10-02	Mon 23-10-02	
9		**1.3. File-IO on Startup:**	**1 day?**	**Tue 23-10-03**	**Tue 23-10-03**	**6**
10		1.3.1. Read the dataset	1 day?	Tue 23-10-03	Tue 23-10-03	
11		1.3.2. Initialize one hundred record objects	1 day	Tue 23-10-03	Tue 23-10-03	
12		**1.4. Display Full Name:**	**1 day**	**Wed 23-10-04**	**Wed 23-10-04**	**9**
13		1.4.1. Integrate full name display in Presentation Layer	1 day	Wed 23-10-04	Wed 23-10-04	
14		**1.5. User Interactions:**	**1 day**	**Thu 23-10-05**	**Thu 23-10-05**	**12**
15		1.5.1. Reload dataset option	1 day	Thu 23-10-05	Thu 23-10-05	
16		1.5.2. Persist data to disk option	1 day	Thu 23-10-05	Thu 23-10-05	
17		1.5.3. Display records option	1 day	Thu 23-10-05	Thu 23-10-05	
18		1.5.4. Create new record optio	1 day	Thu 23-10-05	Thu 23-10-05	
19		1.5.6. Delete record option	1 day	Thu 23-10-05	Thu 23-10-05	
20		**2. Implement Catch2 Testing:**	**2 days?**	**Sat 23-10-07**	**Mon 23-10-09**	**14**
21		2.1. Integrate Catch2 testing framework	1 day	Sat 23-10-07	Sat 23-10-07	
22		2.2. Write Catch2 unit tests for various functionalities	1 day	Sat 23-10-07	Sat 23-10-07	
23		**3. Documentation:**	**2 days?**	**Mon 23-10-09**	**Tue 23-10-10**	**20**
24		3.1. Comment source code files	1 day	Mon 23-10-09	Mon 23-10-09	
25		3.2. Comment unit test file	1 day	Mon 23-10-09	Mon 23-10-09	
26		**4. Program Execution Concepts:**	**3 days?**	**Tue 23-10-10**	**Thu 23-10-12**	**23**
27		4.1. Implement and test loop structures	1 day	Tue 23-10-10	Tue 23-10-10	
28		4.2. Implement and test decision structures	1 day	Tue 23-10-10	Tue 23-10-10	
29		4.3. Ensure File-IO operations	1 day	Tue 23-10-10	Tue 23-10-10	
30		4.4. Ensure MVC architecture is followed	1 day	Tue 23-10-10	Tue 23-10-10	
31		**5. Project Review:**	**2 days?**	**Thu 23-10-12**	**Fri 23-10-13**	**26**
32		5.1. Review entire project functionality	1 day	Thu 23-10-12	Thu 23-10-12	
33		5.2. Test all implemented features	1 day	Thu 23-10-12	Thu 23-10-12	
34		5.3. Prepare for submission	1 day	Thu 23-10-12	Thu 23-10-12	

The Gantt chart offers a structured visualization of the planned tasks and milestones needed for the successful completion of the project. The primary focus is on implementing a layered design, consisting of the presentation, business, and persistence layers. As the project advances, there's a need to either integrate or develop a new record object, followed by using File-IO operations upon startup, ensuring the user can effortlessly interact with the system. The Catch2 testing framework's integration is essential to ensuring robustness. Comprehensive documentation, which aids future development, is stressed upon, and it's critical to use the right documentation syntax. The project also emphasizes core programming concepts, from loop structures to ensuring adherence to the MVC architecture. Finally, a holistic project review is essential to ensure all functionalities cohesively work together, ensuring readiness for submission.

References

[1] “MVC design pattern,” GeeksforGeeks, https://www.geeksforgeeks.org/mvc-design-pattern/ (accessed Sep. 28, 2023).

[2] B. Gorman, “Reading and writing CSV files with C++,” GormAnalysis, https://www.gormanalysis.com/blog/reading-and-writing-csv-files-with-cpp/ (accessed Sep. 28, 2023).

[3] R. Grimm, “Home,” MC++ BLOG, https://www.modernescpp.com/index.php/model-view-controller/ (accessed Sep. 28, 2023).

[4] M. Ozkaya, “Layered (N-layer) architecture,” Medium, https://medium.com/design-microservices-architecture-with-patterns/layered-n-layer-architecture-e15ffdb7fa42 (accessed Sep. 28, 2023).

[5] “Open-source tool that uses simple textual descriptions to draw beautiful UML diagrams.,” PlantUML.com, https://plantuml.com/ (accessed Sep. 28, 2023).

[6] “Tutorial,” GitHub, https://github.com/catchorg/Catch2/blob/devel/docs/tutorial.md (accessed Sep. 29, 2023).

[7] “Catch2: Clion,” CLion Help, https://www.jetbrains.com/help/clion/catch-tests-support.html (accessed Sep. 29, 2023).

[8] Fronthem, fronthemfronthem 4, Arnav BorborahArnav Borborah 11.4k88 gold badges4343 silver badges8989 bronze badges, and Guillaume RacicotGuillaume Racicot 39.8k99 gold badges7878 silver badges143143 bronze badges, “How to use CMAKE with Catch2?,” Stack Overflow, https://stackoverflow.com/questions/49413898/how-to-use-cmake-with-catch2 (accessed Sep. 29, 2023).