My GitHub Project
C++ & wxWidgets

Marvus in C++

November 29, 2025 **Riyufuchi**

Contents

Glossary

wxWidgets A cross-platform GUI toolkit for C++. 1, 3

Acronyms

GUI Graphical User Interface.

1 Introduction

This application originally began as a purely CLI-based project (which is also the reason for its name, *ConsoleArt*). As the project evolved, a Graphical User Interface (GUI) was later added. Development started at a time when C++17 was the newest standard, so the use of modern features such as smart pointers was adopted gradually throughout the project.

1.1 Technologies Used

1.1.1 wxWidgets

This project is written mainly in C++ and utilize wxWidgets for the GUI. wxWidgets is cross-platform and utilize naive GUI libraries for each supported platform.

1.1.2 SQLite

This project use SQLite for simplicity as main focus is on C++ and GUI. So this application will not produce accurate statistics as SQLite do not support accurate decimal data type and use only double. I will compare results from this application with results from my Java application that uses BigDecimal for accurate decimal calculations.

1.1.3 ConsoleLib

`ConsoleLib` is my custom library developed to simplify the creation of CLI applications. It provides an abstract `IConsole` interface and several platform-specific implementations:

- **DefaultConsole** – A basic implementation using standard output (`std::cout`) without any additional formatting features.
- **UnixConsole** – Adds support for text coloring and formatting through ANSI escape codes commonly available on Unix-like systems.
- **WindowsConsole** – Inherits from `UnixConsole` and enables UTF-8 output on Windows while ensuring compatibility with ANSI escape codes.

The library also includes additional utility modules such as an argument parser and other helpers commonly required in CLI tools, making it reusable across multiple projects.

1.1.4 MiniZ

MiniZ is simple C library for working with zip files which are used to import data from my Java application to this one.

1.2 Commiting a git commit

Commit type cheat table	
Type	**Repository change**
Feat:	A new feature is added
Fixed:	A bug is fixed
Docs:	A documentation is updated
Refactor:	A code change that is not affecting functionality
Test:	A code change in unit tests
Chore:	A repository maintenance action
Version x.y	When new version is released for clearer commit history

Table 1.1: Types of commit headers

2 Core

2.1 UML

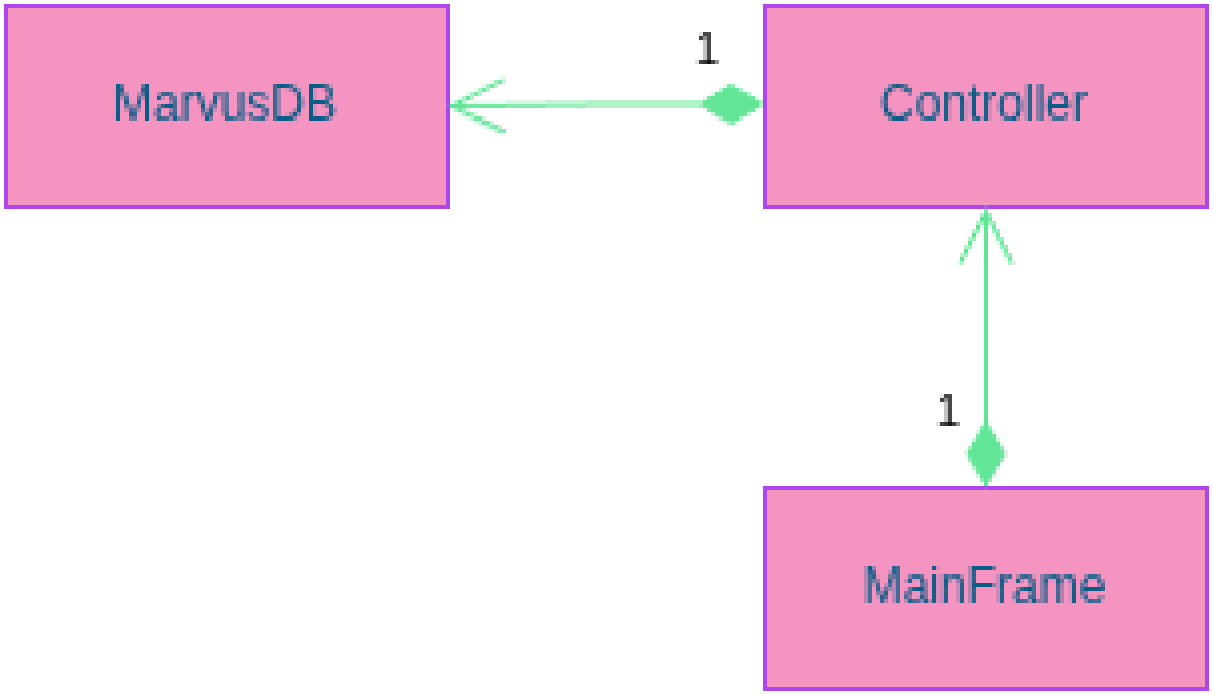

Figure 2.1: Controller relations UML

2.2 Controller

The Controller class is core of this application, it encapsulate the database controller. It is a separate unit from wxWidgets so it could be reused for Qt GUI based application for example.

2.3 Database

This is a class that works on it own and it encapsulates the SQLite C API. It is not thread safe and MarvusDB inherits from it and that is why it is not included in the UML Figure 2.1.

2.3.1 Public API

This subsection describes some functions that needs to be explained.

2.3.1.1 bool initializeDatabase();

This function loads scripts from provided path and executes *"initialize_database.sql"* whish is your script that creates tables, triggers and etc..

2.3.1.2 bool executeFileSQL(...);

This function can execute multiple SQL statements. It is mainly used for executing entire SQL files, and it expects the file's contents to be provided to it, as it does not load any files on its own.

2.3.1.3 bool initializeViews();

This functions looks for loaded files and executes those, that contains "view" in their name.

2.3.1.4 bool reconnect(const std::string& databaseFile);

This function allow to load different database file and thus switch database files at runtime.

3 GUI

3.1 Foreword

3.2 Window

4 Database

4.1 Foreword

Even thought SQLite have wxWidgets wrapper I didn't use it as I wants my application logic to be independent from the GUI framework.

4.2 ERD Diagram

Figure 4.1: Database ERD

Each **PAYMENT** have one **CATEGORY** for simplification and this will give us rough statistics per **CATEGORY** as purpose of this database/application is to track income and spending's and not fully accurate accountant statistics.

4.2.1 PAYMENTS

Payment table fields		
Data type	**Name**	**Stores**
INTEGER	payment_id	Primary key (row identifier).
INTEGER	establishment_id_key	Foreign key referencing `ESTABLISHMENTS`
INTEGER	category_id_key	Foreign key referencing `CATEGORIES`.
TEXT	payment_value	Monetary value. Stored as text to avoid loss of decimal precision.
DATE	payment_date	Date of the payment, stored as string in ISO-8601 format.*
TEXT	payment_note	Optional free-text note.

Table 4.1: PAYMENTS fields

**SQLite does not provide a dedicated `DATE` type. Dates are stored as normalized text in ISO-8601 format (`YYYY-MM-DD`).*

Summary

Listings

List of Figures

List of Tables