BudgetWise

> This is the final project of the ***2023FA CSC-284-WB*** in BHCC. The requirements of this project are in Google Docs.
> This is a **private repository** on GitHub.

BudgetWise serves as an intuitive graphical tool designed for effective budget and expense tracking. Empowering users with the ability to effortlessly input both income and expenses, the platform offers seamless categorization and subcategorization features. Dive into insightful charts that provide a comprehensive overview of total income and expenses within specified categories or subcategories. For a more in-depth understanding of BudgetWise, explore the detailed User Story. Experience financial management with ease, precision, and visual clarity.

TimeLine

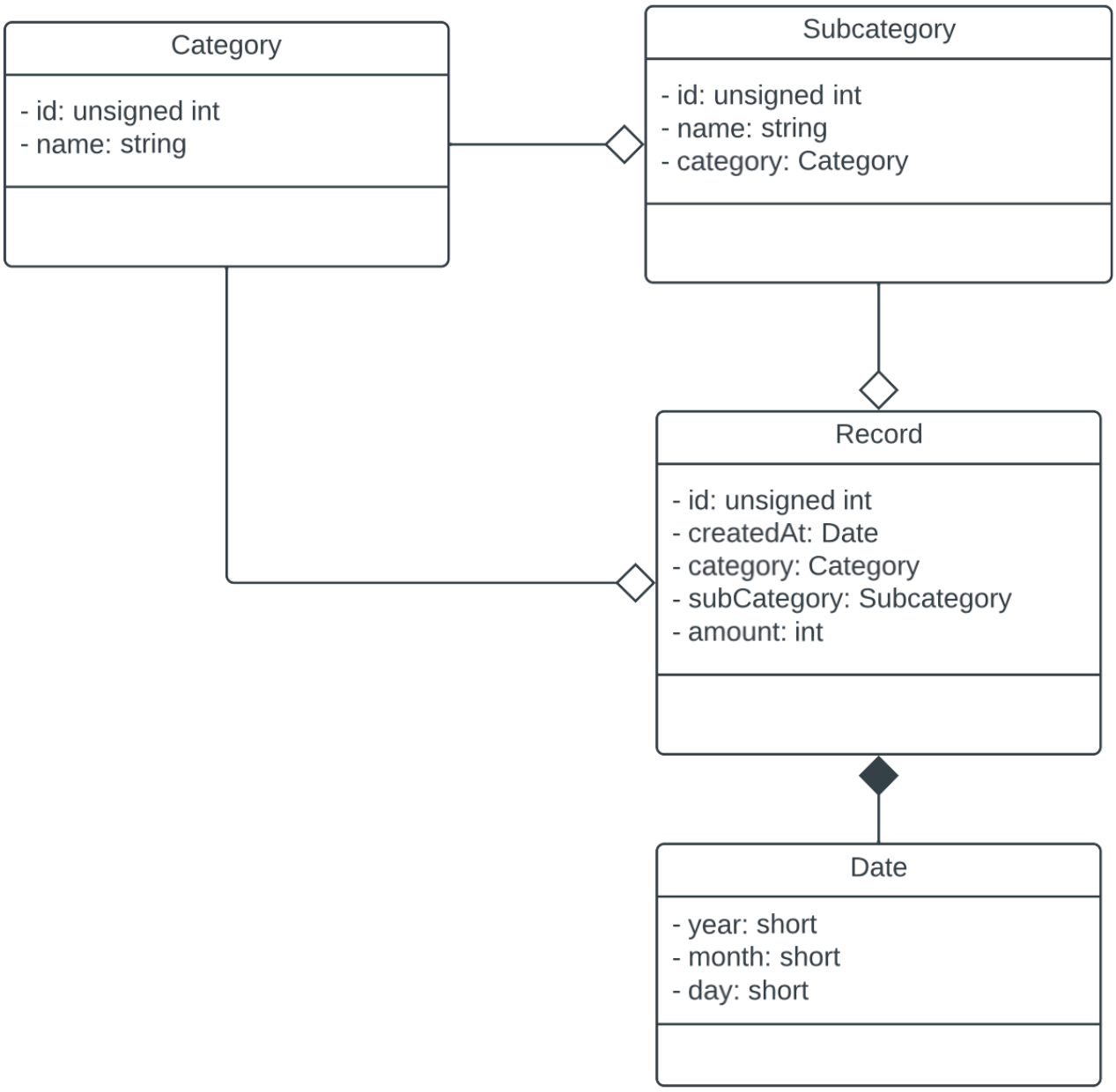
This project is supposed to be accomplished before **Dec 12, 2023**, and the timeline is as follows:

Time	Checklist
Nov 19, 2023	1. Finish the entities. 2. Build `LoginWindow` and `SignUpWindow`. 3. Create utility functions. 4. Accomplish the timeline.
Nov 26, 2023	1. Create a server (backend). 2. Test the network connection between two ends. 3. Test the server receiving HTTP requests.
Dec 3, 2023	1. Implement all APIs on the backend. 2. Test sending HTTP requests from the frontend to the backend.
Dec 10, 2023	1. Implement all windows. 2. Implement graphical charts to display user data.
Dec 12, 2023	1. Test through user stories. 2. Supplement documentation.

UML Diagrams

> The UML diagrams are edited on Lucid.

The following is the UML diagram for the four main entities in this project. Each entity has a corresponding table in the server database. Note that in the MVC architecture, entities are also known as models.



Development

Within this chapter, we will be covering crucial aspects of development, including style guides, utilities, and naming strategies. As this project is of a personal nature, consider this document as a personal aide, serving the purpose of aiding future recall and facilitating a comprehensive understanding of the elements created during subsequent follow-ups.

Style Guides

Adherence to the Google C++ Style Guide is imperative for maintaining consistency and best practices throughout the entire codebase of this project.

Utilities

Ensure the incorporation of utility functions as necessary to align with best practices, thereby enhancing code readability. Utility files are placed under the `util` directory, which includes `FileUtil` and `WindowUtil`.

Utilize the `FileUtil::open()` method to seamlessly handle file operations. This method not only verifies the file's existence but also performs the tasks of opening the file, invoking a specified callback function, and subsequently closing the file. Below are two illustrative examples demonstrating the application of this versatile method:

```
#include "util/FileUtil.h"

// Create a QFile object
QFile file("path/to/the/file");

// Read the file
FileUtil::open(file, QIODevice::ReadOnly, [](QFile& file) {
  // It's recommended using QTextStream() to read the file
  QTextStream inStream(&file);
});

// Write the file
FileUtil::open(file, QIODevice::WriteOnly, [](QFile& file) {
  // It's recommended using QTextStream() to write the file
  QTextStream outStream(&file);
});

// You can also use these functions to achieve the same effect
FileUtil::read(file, [](QTextStream& inStream) {
  // Do something
});
FileUtil::write(file, [](QTextStream& outStream) {
  // Do something
});
```

Utilize the `Window::navigate()` method to seamlessly navigate from one window to another:

```
#include "util/WindowUtil.h"

// Navigate to the main window in a Widget member method
// The current window will automatically close after the main window shows
WindowUtil::navigate(this, new MainWindow);
```