Personal Finance Manager

Generated by Doxygen 1.10.0

Chapter 1

Personal Finance Manager

Team name: Kolegos.

Members: Ridas Kožukauskas, Gustas Griežė.

The Personal Finance Manager is an interactive application designed for managing and monitoring one's financial situation. This application allows users to manage their income and expenses, create and track budgets, and monitor financial goals.

1.1 Project Architecture

1.1.1 Primary Components:

- **User Interface (UI):**
  - Serves as the front-end for user interactions.
  - Handles input such as transaction entries, budget creation, and goal setting.
- **Transaction Manager:**
  - Manages adding, viewing, editing, and deleting of transactions.
- **Budget Manager:**
  - Handles the creation, viewing, editing, and deletion of budgets.
  - Tracks and compares budget against actual spending.
- **Goal Manager:**
  - Facilitates the setting, viewing, editing, and deletion of financial goals.
  - Monitors progress towards financial goals.
- **Alert System:**
  - Generates and displays alerts for budget limits or goal achievements.
- **Data Processing Engine:**
  - Processes input data, computes balances, and updates financial goals.

1.1.2 Data Flow:

- Data is entered by users through the UI and directed to the appropriate managers.
- Managers update the system's state with the new data.
- Outputs such as updated balances, budgets, and goals are sent to the UI for display.
- Alerts are triggered and displayed based on specific system state changes.

1.1.3 Input/Output:

- **Input:**
  - Transactions data, budget details, and goal parameters entered by the user.
- **Output:**
  - Transaction history, budget summaries, goal progress, and alerts displayed to the user.

1.1.4 Storage:

- Utilizes a robust database system for secure data storage.
- Ensures data persistence for transaction histories, budget records, and financial goals.
- Allows historical data analysis and feature expansions such as reporting.

1.2 Activity diagram

1.3 Use-case diagram

1.4 Technologies & architecture

1.4.1 User Interface (UI) Framework

- **Qt**: It's a full-fledged application framework with a rich set of tools for building cross-platform GUI applications.

1.4.2 Database Library

- **SQLite**: A C-language library that implements a small, fast, self-contained, high-reliability, full-featured SQL database engine. It's well-suited for local/client storage in application software.

1.4.3 Testing Framework

- **Google Test**: Provides an excellent suite of tools for writing and running unit tests.

1.4.4 Build System

- **CMake**: Widely used for C++ projects, handles cross-platform builds, and is well-supported by most IDEs.

1.4.5 Version Control

- **Git**: Essential for source code management. Host your repository on platforms like GitHub, GitLab, or Bitbucket.

1.4.6 Documentation Tool

- **Doxygen**: For generating documentation from annotated C++ sources.

1.5 Workload report

Name Surname	Grade assesment amount
Ridas Kožukauskas	50%
Gustas Griežė	50%

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 Budget Class Reference

Public Member Functions

- void **addTransaction** (Transaction *transaction)
- void **removeTransaction** (int id)
- std::vector< Transaction * > **getTransactions** () const
- std::string **getName** () const
- double **getTotalAmount** () const
- double **getTotalSpent** () const
- void **setTotalAmount** (double amount)
- TransactionManager * **getTransactionManager** ()
- **Budget** (std::string name, double amount)

The documentation for this class was generated from the following files:

- Budget.h
- Budget.cpp

4.2 BudgetManager Class Reference

Public Member Functions

- **BudgetManager** (double initialBudget, TransactionManager transManager)
- Budget & **findBudget** (const std::string &name)
- void **createBudget** (const Budget &budget)
- void **deleteBudget** (const std::string &name)
- void **editBudget** (const Budget &budget)
- double **calculateRemainingBudget** () const
- double **calculateRemainingBudget** (const Budget &budget) const
- void **displaySummary** () const
- double **getTotalBudget** () const
- void **setTotalBudget** (double newBudget)
- std::unordered_map< std::string, Budget > **getAllBudgets** () const
- TransactionManager * **getTransactionManager** ()

The documentation for this class was generated from the following files:

- BudgetManager.h
- BudgetManager.cpp

4.3 ReportGenerator Class Reference

Public Member Functions

- **ReportGenerator** (TransactionManager &manager)
- void **generateSummaryReport** () const

The documentation for this class was generated from the following files:

- report_generator.h
- report_generator.cpp

4.4 Transaction Class Reference

Public Member Functions

- **Transaction** (const std::string &description, double amount, const std::string &date, Budget *budget)
- int **getId** () const
- std::string **getDescription** () const
- double **getAmount** () const
- std::string **getDate** () const
- Budget & **getBudget** ()
- void **setDescription** (const std::string &description)
- void **setAmount** (double amount)
- void **setDate** (const std::string &date)
- void **setBudget** (Budget *budget)

The documentation for this class was generated from the following files:

- Transaction.h
- Transaction.cpp

4.5 TransactionManager Class Reference

Public Member Functions

- void **addTransaction** (Transaction &transaction)
- bool **editTransaction** (int id, const std::string &description, double amount, const std::string &date)
- bool **deleteTransaction** (int id)
- void **viewTransactions** () const
- std::vector< Transaction > **getTransactions** () const
- std::vector< Transaction > **getExpenses** () const
- std::vector< Transaction > **getIncomes** () const

Public Attributes

- BudgetManager * **budgetManager**

The documentation for this class was generated from the following files:

- Transaction.h
- Transaction.cpp

Chapter 5

File Documentation

5.1 Budget.h

```
00001 #ifndef BUDGET_H
00002 #define BUDGET_H
00003
00004 #include <iostream>
00005 #include <unordered_map>
00006 #include <vector>
00007 #include <string>
00008 #include "Transaction.h"
00009
00010 class Budget {
00011 private:
00012     std::string budgetName;
00013     double totalAmount;
00014     double presentAmount;
00015     std::vector<Transaction*> transactions;
00016
00017
00018 public:
00019     void addTransaction(Transaction* transaction);
00020     void removeTransaction(int id);
00021     std::vector<Transaction*> getTransactions() const;
00022     std::string getName() const;
00023     double getTotalAmount() const;
00024     double getTotalSpent() const;
00025     void setTotalAmount(double amount);  // Added this method
00026     TransactionManager* getTransactionManager();
00027
00028     Budget();
00029     Budget(std::string name, double amount);
00030     ~Budget();
00031 };
00032
00033 #endif // BUDGET_H
```

5.2 BudgetManager.h

```
00001 #ifndef BUDGETMANAGER_H
00002 #define BUDGETMANAGER_H
00003
00004 #include <iostream>
00005 #include <string>
00006 #include <unordered_map>
00007 #include "Transaction.h"
00008 #include "Budget.h"
00009
00010 class BudgetManager {
00011 private:
00012     double totalBudget;
00013     std::unordered_map<std::string, Budget> budgets;
00014     TransactionManager transactionManager;
00015
00016 public:
00017     BudgetManager(double initialBudget, TransactionManager transManager);
00018
```

```
00019     Budget& findBudget(const std::string& name);
00020     void createBudget(const Budget& budget);
00021     void deleteBudget(const std::string& name);
00022     void editBudget(const Budget& budget);
00023 
00024     double calculateRemainingBudget() const;
00025     double calculateRemainingBudget(const Budget& budget) const;
00026     void displaySummary() const;
00027 
00028     double getTotalBudget() const;  // Keep only one declaration
00029     void setTotalBudget(double newBudget);
00030     std::unordered_map<std::string, Budget> getAllBudgets() const;
00031     TransactionManager* getTransactionManager();
00032 };
00033 
00034 #endif // BUDGETMANAGER_H
```

5.3 report_generator.h

```
00001 #ifndef REPORT_GENERATOR_H
00002 #define REPORT_GENERATOR_H
00003 
00004 #include "Transaction.h"
00005 #include <vector>
00006 #include <iostream>
00007 
00008 class ReportGenerator {
00009 public:
00010     explicit ReportGenerator(TransactionManager& manager);
00011     void generateSummaryReport() const;
00012 
00013 private:
00014     const TransactionManager& manager;
00015 };
00016 
00017 #endif // REPORT_GENERATOR_H
```

5.4 Transaction.h

```
00001 #ifndef TRANSACTION_H
00002 #define TRANSACTION_H
00003 
00004 #include <vector>
00005 #include <string>
00006 
00007 class Budget;
00008 class BudgetManager;
00009 
00010 class Transaction {
00011 private:
00012     static int nextId;
00013     int id;
00014     std::string description;
00015     double amount;
00016     std::string date;
00017     Budget* budget;
00018 
00019 public:
00020     Transaction(const std::string& description, double amount, const std::string& date, Budget*
      budget);
00021     int getId() const;
00022     std::string getDescription() const;
00023     double getAmount() const;
00024     std::string getDate() const;
00025     Budget& getBudget();
00026     void setDescription(const std::string& description);
00027     void setAmount(double amount);
00028     void setDate(const std::string& date);
00029     void setBudget(Budget* budget);
00030 };
00031 
00032 class TransactionManager {
00033 public:
00034     void addTransaction(Transaction& transaction);
00035     bool editTransaction(int id, const std::string& description, double amount, const std::string&
      date);
00036     bool deleteTransaction(int id);
00037     void viewTransactions() const;
00038     std::vector<Transaction> getTransactions() const;
```

```
00039     std::vector<Transaction> getExpenses() const;
00040     std::vector<Transaction> getIncomes() const;
00041     BudgetManager* budgetManager;
00042
00043 private:
00044     std::vector<Transaction> transactions;
00045 };
00046
00047 #endif // TRANSACTIONS_H
```

Index