

Contents

[Application design 5](#_Toc93282817)

[Class hierarchy 6](#_Toc93282818)

[Class list 7](#_Toc93282819)

[AddExpense 7](#_Toc93282820)

[AddIncome 7](#_Toc93282821)

[Income 7](#_Toc93282822)

[MainWindow 7](#_Toc93282823)

[Settings 7](#_Toc93282824)

[Transaction 8](#_Toc93282825)

[Class documentation 9](#_Toc93282826)

[AddExpense Class Reference 9](#_Toc93282827)

[#include <addexpense.h> 9](#_Toc93282828)

[~AddExpense () 9](#_Toc93282829)

[Private Slots: void save () 10](#_Toc93282830)

[void clear () 10](#_Toc93282831)

[AddIncome Class Reference 11](#_Toc93282832)

[#include <addincome.h> 11](#_Toc93282833)

[Private Slots void save () 11](#_Toc93282834)

[Private 12](#_Toc93282835)

[Dbconnection Class Reference 13](#_Toc93282836)

[QString getDatabasebPath () 13](#_Toc93282837)

[bool connectDb () 13](#_Toc93282838)

[void closeDb () 13](#_Toc93282839)

[Expense Class Reference 14](#_Toc93282840)

[#include <expense.h> 14](#_Toc93282841)

[Public Slots bool insertTransaction (Transaction \*expense) 14](#_Toc93282842)

[bool updateTransaction (Transaction \*expense) 14](#_Toc93282843)

[Income Class Reference 15](#_Toc93282844)

[bool Income::insertTransaction (Transaction \* income) 15](#_Toc93282845)

[bool Income::updateTransaction (Transaction \* income) 15](#_Toc93282846)

[Login Class Reference 16](#_Toc93282847)

[#include <login.h> 16](#_Toc93282848)

[~Login () 16](#_Toc93282849)

[void on\_btnExit\_clicked(); 16](#_Toc93282850)

[signals: void loginClicked(); 17](#_Toc93282851)

[MainWindow Class Reference 18](#_Toc93282852)

[#include <mainwindow.h> 18](#_Toc93282853)

[explicit MainWindow(QWidget \*parent = 0); 18](#_Toc93282854)

[signals: void on\_actionLog\_out\_triggered(); 18](#_Toc93282855)

[void editTransaction(QString type, QSqlTableModel\* tableModel) //!< opens new tab to edit the transactions 19](#_Toc93282856)

[Settings Class Reference 20](#_Toc93282857)

[#include <settings.h> 20](#_Toc93282858)

[explicit Settings(QWidget \*parent = 0); 20](#_Toc93282859)

[void toggleButtons (bool) void deleteRecord () 20](#_Toc93282860)

[bool updateUser (int rowNumber, int id, QString username, QString password) This will update the selected user from the database 21](#_Toc93282861)

[Transaction Class Reference 22](#_Toc93282862)

[#include <transaction.h> 22](#_Toc93282863)

[void setDate (const QString &value) set the date of the transaction 22](#_Toc93282864)

[void setAmount (int value) set the amount of the transaction QString getParty () const 23](#_Toc93282865)

[void setCategory (const QString &value) set the category of the transaction 23](#_Toc93282866)

[virtual bool insertTransaction (Transaction \*transaction)=0 23](#_Toc93282867)

[virtual bool updateTransaction (Transaction \*transaction)=0 23](#_Toc93282868)

[QSqlTableModel \* getTransaction (QString type, int id) 24](#_Toc93282869)

[QStandardItemModel \* getCategoriesList (QString type) gets the categories as a list according to the type selected 24](#_Toc93282870)

[User Class Reference 25](#_Toc93282871)

[Private 25](#_Toc93282872)

[ViewExpense Class Reference 26](#_Toc93282873)

[#include <viewexpense.h> 26](#_Toc93282874)

[~ViewExpense () 26](#_Toc93282875)

[void on\_btnEdit\_clicked () 26](#_Toc93282876)

[ViewIncome Class Reference 28](#_Toc93282877)

[#include <viewincome.h> 28](#_Toc93282878)

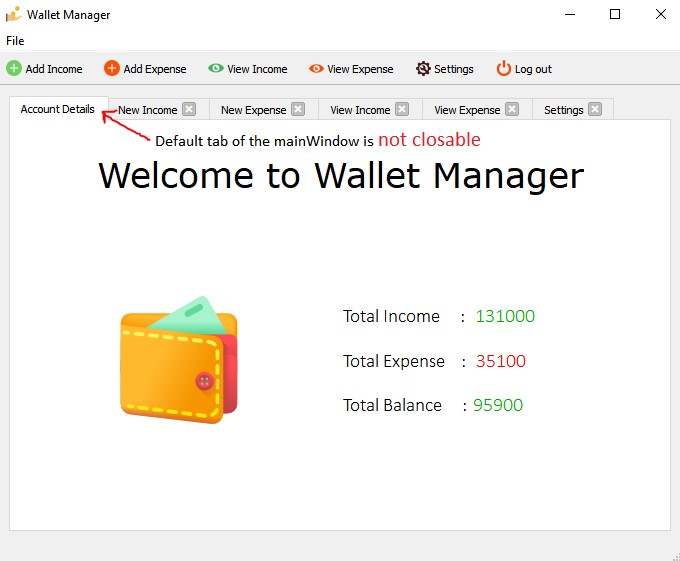
[~ViewIncome () 28](#_Toc93282879)

[Private Slots void on\_btnDelete\_clicked () 28](#_Toc93282880)

Wallet Manager

# Application design

The application starts with a login screen but if the database is not selected or not found dialog box would appear and user will have to select the database. If database connected successfully the login screen will appear and it will authenticate the user.

After entering the correct credentials, MainWindow (home screen) will show and the login dialog will be close automatically. All the tasks are carried on Wallet Manger is done within QTabWidget. Further, MainWindow’s default tab cannot be closed.

NOTE: This documentation is generated using Doxygen.

# 

# Class hierarchy

dbconnection QDialog

Login

QMainWindow

MainWindow

QWidget

AddExpense

AddIncome

Settings

ViewExpense

ViewIncome

Transaction

Expense

Income

User

## Class list

Brief description about the classes.

### AddExpense

(Child of QWidget and it is the screen needed to add a New Expense or Update an entered

Expense record)

### AddIncome

(Child of QWidget and it is the screen needed to add a New Income or Update an entered

Income record)

**Dbconnection**

(The Dbconnection class represents the connectivity of the Database.)

**Expense**

(Child of the Transaction class and models the expense object of Wallet Manager)

### Income

(Child of the Transaction class and models the income object of Wallet Manager) **Login**

(Child of the QDialog class is called when the application is compiled. This screen is called again if the logout is clicked from the MainWindow. This class describes the user)

### MainWindow

(Child of QMainWindow and it acts as the controller and contains all the QWidgets of the application)

### Settings

(Child of QWidget and it is the screen needed used to manage all Settings related to Wallet

Manager such as add users and categories)

### Transaction

(Represent the Transaction Object of Wallet Manager. Transaction class is an abstract class.

This class contains two members called Income and Expense transaction types) **User**

(Reprent the user object of the wallet manager application)

**ViewExpense**

(Child of QWidget and it is the screen needed to View all the Expense records)

**ViewIncome**

(Child of QWidget and it is the screen needed to View all the Income records)

# Class documentation

## AddExpense Class Reference

AddExpense class is a child of QWidget and it is the screen needed to add a New Expense or

Update an entered Expense record

### #include <addexpense.h>

Inheritance diagram for AddExpense:



**Public Member Function:**

**AddExpense** (QWidget \*parent=0)

Creates the UI, an instance of the expense class is created and setting up the connectors Parameters:

|  |  |
| --- | --- |
| parent | Parent describes the QWidget that calls this constructor |

### ~AddExpense ()

This is the destructor of this class and it will delete the AddExpense UI instance

void **setupEdit** (QSqlTableModel \*tableModel)

This function will create the screen to conduct an expense edit.

Parameters:

|  |  |
| --- | --- |
| tableModel | contains a single existing record which will be set to the fields of this screen |

### Private Slots: void save ()

This function will get all the values from the text fields and stores within an expense object.

After the validation, the object will be saved in the database.

### void clear ()

This function will clear all the fields and reset the date to today.

**Private:**

Ui::AddExpense \* **ui** Transaction \* expense

int tempId

## AddIncome Class Reference

The AddIncome class is a child of QWidget and it is essentially the screen used to add a New Income or Edit a previous Income record.

### #include <addincome.h>

Inheritance diagram for AddIncome:



**Public Member Functions**

**AddIncome (QWidget \*parent=0)**

Creates the UI, setting up the connectors and an instance of the income class is created.

**~AddIncome ()**

This is the destructor of this class and it will delete the AddIncome UI instance

**void setupEdit (QSqlTableModel \*tableModel)**

This function will create the screen to conduct an income edit.

### Private Slots void save ()

This function will get all the values from the text fields and stores within an income object.

After the validation, the object will be saved in the database.

**void clear ()**

This function will clear all the fields and reset the date to today.

### Private

Ui::AddIncome \* ui Transaction \* income int tempId

## Dbconnection Class Reference

The Dbconnection class represents the connectivity of the Database.

#include <dbconnection>

**Public Member Functions dbconnection ()**

This method will create an instance of the QSqlDatabase class and set path to the DB.

### QString getDatabasebPath ()

Path of the database is stored in the text file in the project folder. If the path is not correct, user will have to select the database path by himself. After selecting the database path, it will be stored in the text file.

### bool connectDb ()

This function will test to see if the database is currently open and open the database if it’s not open.

### void closeDb ()

This function will test to see if the database is currently close and close the database if it’s not closed.

**Private:**

QSqlDatabase database

## Expense Class Reference

Expense is a child of the Transaction class and models the expense object of Wallet Manager.

### #include <expense.h>

Inheritance diagram for Expense:



### Public Slots bool insertTransaction (Transaction \*expense)

This is pure virtual function in the base class **Parameters:**

|  |  |
| --- | --- |
| expense | expense is a pointer which points to an Expense object |

Returns: This will return true if the insertion is successful and false if not.

### bool updateTransaction (Transaction \*expense)

This is pure virtual function in the base class

Parameters:

|  |  |
| --- | --- |
| expense | expense is a pointer which points to an Expense object |

Returns: This will return true if the update is successful and false if not.

## Income Class Reference

Income class is a child of the Transaction class and models the income object of Wallet Manager.

#include <income.h>

Inheritance diagram for Income:



**Public Slots:**

### bool Income::insertTransaction (Transaction \* income)

This is pure virtual function in the base class Parameters:

|  |  |
| --- | --- |
| income | income is a pointer which points to an Income object |

Returns: This will return true if the insertion is successful and false if not.

### bool Income::updateTransaction (Transaction \* income)

This is pure virtual function in the base class Parameters:

|  |  |
| --- | --- |
| income | income is a pointer which points to an Income object |

Returns: This will return true if the insertion is successful and false if not.

## Login Class Reference

The Login class is child of the QDialog class is called when the application is compiled. This screen is called again if the logout is clicked from the MainWindow. This class describes the user

### #include <login.h>

Inheritance diagram for Login:



**Public:**

**Login (QWidget \*parent=0)**

Creates the UI, initiate MainWindow and the user class and matches the signal to their slots

### ~Login ()

This is the destructor of this class and it will delete the Login UI instance **private slots:**

### void on\_btnExit\_clicked();

Application will be terminated with the database connection **void login();**

checking the user credentials with the user class. This will open the main window if the credentials are validated correctly, if not message box will appear.  **void logout();**  this will hide the main screen and directed back to the login screen.

### signals: void loginClicked();

This is the signal created for the button login.

**private:**

Ui::Login \*ui;

User \*user;

MainWindow \*w;

## MainWindow Class Reference

The mainWidnow is a child of QMainWindow and it acts as the controller and contains all the QWidgets of the application.

### #include <mainwindow.h>

Inheritance diagram for MainWindow:



**Public:**

### explicit MainWindow(QWidget \*parent = 0);

The constructor creates the UI, Hides the close button of the displayed tab, calls the account details method and it will displayed the calculated values of the transaction on the main tab.

**~MainWindow();**

This is the destructor of this class and closes the database

### signals: void on\_actionLog\_out\_triggered();

**private slots:**

**void onAddIncome()** //!< new tab is opened and the UI is created for the new income  **void onAddExpense()** //!< new tab is opened and the UI is created for the new expense **void onViewIncome()** //!< new tab is opened and the UI is created for the view income  **void onViewExpense()** //!< new tab is opened and the UI is created for the view expense

**void closeTab(int index)** //!< Current tab will be closed

**bool tabExists(QString label)** //!< check if a tab is opened or not and if it is open, tab will be moved

**void account\_details(int index)** //!< Recalculated the transaction values displayed in the main tab but tab index should be Zero

**void account\_details()** //!< Settings tab is opened if it is not opened.

### void editTransaction(QString type, QSqlTableModel\* tableModel) //!< opens new tab to edit the transactions

**void quitApp()** //!< This will terminate the application, it is called only when quit application is clicked

**Private:**

Ui::MainWindow \*ui; dbconnection \*dbConn;

AddIncome \*addIncome;

AddExpense \*addExpense;

ViewIncome \*viewIncome;

ViewExpense \*viewExpense;

Settings \*settings;

## Settings Class Reference

The Settings class is a child of QWidget and it is the screen needed used to manage all Settings related to Wallet Manager such as add users and categories

### #include <settings.h>

Inheritance diagram for Settings:



**public:**

### explicit Settings(QWidget \*parent = 0);

Creates the UI, initiates an instance of the income and expense classes

**~Settings()** //!< This is the destructor of this class and closes the database

**Signals:**

### void toggleButtons (bool) void deleteRecord ()

**Private Slots:**

**void on\_btnIncomeCat\_clicked ()** This method is triggered when the income category button is clicked. **void on\_btnExpenseCat\_clicked ()** This method is triggered when the expense category button is clicked.

**void on\_btnAddCat\_clicked ()** This method is triggered when the add income category button is clicked. **void on\_btnEditCat\_clicked ()** This method is triggered when the add expense category button is clicked. **void on\_btnUser\_clicked ()** This method is triggered when the view all user button is clicked. **void on\_btnAddUser\_clicked ()** This method is triggered when the add user button is clicked. **void on\_btnEditUser\_clicked () This** method is triggered when the view update user button is clicked.

**void doToggleButtons (bool on)** This method is triggered when the table view is changed and toggle the visibility of the buttons **void doDeleteRecord ()** This method is triggered when the delete or button is clicked.

**Private Member Functions void setupViewTable (QString type)**

Creates the table view in the middle of the tab screen.

**bool insertUser (QString username, QString password)**

This will create new user and it will be stored in the database

### bool updateUser (int rowNumber, int id, QString username, QString password) This will update the selected user from the database

**Private :**

Ui::Settings \* ui

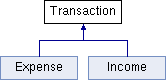
QSqlTableModel \* tableModel Transaction \* income Transaction \* expense User \* user

## Transaction Class Reference

This class represent the Transaction Object of Wallet Manager. Transaction class is an abstract class. This class contains two members called Income and Expense transaction types

### #include <transaction.h>

Inheritance diagram for Transaction:



**Public:**

**int getId () const**  get the Id of the transaction

**void setId (int value)**  set the id of the transaction

**QString getDate () const** get the date of the transaction

### void setDate (const QString &value) set the date of the transaction

**int getAmount () const**  get the amount of the transaction

### void setAmount (int value) set the amount of the transaction QString getParty () const

Get transaction party (paid to or paid by)

**void setParty (const QString &value)** set transaction party (paid to or paid by)

### void setCategory (const QString &value) set the category of the transaction

**QString getCategory () const** get the category of the transaction

**QString getDescription () const** get the description of the transaction

**void setDescription (const QString &value)** set the description of the transaction

**public slots:**

### virtual bool insertTransaction (Transaction \*transaction)=0

Different types of transaction have to be recorded in different ways, there for this is implemented as pure virtual method .

### virtual bool updateTransaction (Transaction \*transaction)=0

Different types of transaction have to be updated in different ways, there for this is implemented as pure virtual method .

**QSqlTableModel \* viewTransaction (QString type)**

Returns a pointer to QSqlTableModel object that it created after querying the data.

### QSqlTableModel \* getTransaction (QString type, int id)

Returns a pointer to QSqlTableModel object that it created after querying the data and this contain one record.

**void deleteTransaction (QModelIndexList list)**

Deletes a Transaction from the database

### QStandardItemModel \* getCategoriesList (QString type) gets the categories as a list according to the type selected

DbConn \* dbConn

QSqlTableModel \* tableModel

QSqlQueryModel \* queryModel QStandardItemModel \* itemModel

int id QString date int amount

QString party

QString category

QString description

## User Class Reference

User class represent the user object of the wallet manager application

**bool loginUser (QString, QString)**

This method will verify the user credentials that is entered with the database.

### Private

QString username

QString password

## ViewExpense Class Reference

View income is a child of QWidget and it is the screen needed to View all the expense records

### #include <viewexpense.h>

Inheritance diagram for Viewexpense:



**Signals:**

**void editTransactionClicked (QString type, QSqlTableModel \*tableModel)**

**Public Member Functions:**

**ViewExpense (QWidget \*parent=0)**

Creates the UI, an instance of the expense class is created and sets the table up.

### ~ViewExpense ()

This is the destructor of this class and it will delete the ViewExpense UI instance  **void setupTable ()**

Creates the table with expense data that is stored in the database

**Private Slots void on\_btnDelete\_clicked ()**

This method is triggered when the delete button is clicked.

### void on\_btnEdit\_clicked ()

This method is triggered when the update button is clicked.

**Private:**

Ui::ViewExpense \* ui

Transaction \* expense

QSqlTableModel \* tableModel

QSqlQueryModel \* queryModel

## ViewIncome Class Reference

View income is a child of QWidget and it is the screen needed to View all the income records

### #include <viewincome.h>

Inheritance diagram for ViewIncome:



**Signals:**

**void editTransactionClicked (QString type, QSqlTableModel \*tableModel)**

**Public Member Functions ViewIncome (QWidget \*parent=0)**

Creates the UI, an instance of the income class is created and sets the table up.

### ~ViewIncome ()

This is the destructor of this class and it will delete the ViewIncome UI instance  **void setupTable ()**

Creates the table with income data that is stored in the database

### Private Slots void on\_btnDelete\_clicked ()

This method is triggered when the delete button is clicked.  **void on\_btnEdit\_clicked ()**

This method is triggered when the update button is clicked.

**Private:**

Ui::ViewIncome \* ui

Transaction \* income

QSqlTableModel \* tableModel

QSqlQueryModel \* queryModel