Министерство образования Республики Беларусь

Учреждение образования

БелорусскиЙ государственный университет

информатики и радиоэлектроники

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

**Программное средство «Менеджер доходов и расходов»**

|  |  |  |
| --- | --- | --- |
| Студент |  | А.А. Шукан |
| Руководитель |  | Д.С. Шулицкий |

Минск 2019

СОДЕРЖАНИЕ

[Введение](#_Toc513665878) 5

[1 Анализ предметной области](#_Toc513665879) 7

[1.1 Обзор аналогов](#_Toc513665880) 7

[1.2 Постановка задачи](#_Toc513665881) 12

[2 Разработка программного средства](#_Toc513665882) 13

[2.1 Структура программы](#_Toc513665883) 13

[2.2 Интерфейс программного средства](#_Toc513665884) 13

[2.3 Работа со звуковыми ресурсами](#_Toc513665885) 14

[2.4 Сетевая логика](#_Toc513665888) 16

[2.4.1 Серверная часть](#_Toc513665886) 17

[2.4.2 Клиентская часть](#_Toc513665887) 20

[3 Тестирование программного средства](#_Toc513665890) 23

[4 Руководство пользователя](#_Toc513665891) 25

[4.1 Интерфейс программы](#_Toc513665892) 25

[4.2 Порядок настройки](#_Toc513665893) 27

[Заключение](#_Toc513665894) 29

[Список использованных источников](#_Toc513665895) 30

[Приложение А. Исходный код программы](#_Toc513665896) 31

ВВЕДЕНИЕ

Трудно представить современный мир без такого явления как деньги. Деньги являются одной из важнейших частей человеческого общества и отношений в нём. С их помощью мы приобретаем всё необходимое для нашей жизнедеятельности, получаем же их путём обмена на какие-либо вещи либо в результате труда.

С появлением денег пришла и необходимость в развитии различных отраслей науки и даже создания новых. Например, обеспечение безопасности хранения и использования денежных средств стало возможным благодаря исследованиям в области криптографии. Также появились и отдельные отрасли знания такие, как менеджмент, маркетинг и огромное множество других.

Можно считать, что сама идея денег появилась далеко до нашей эры, когда на смену бартеру и обмену пришло использование жемчуга, ракушек, камней и металлов в качестве эквивалентов, служащих мерами стоимости товаров и услуг. Со временем роль денег всё больше стали играть металлы: медь, бронза, железо, серебро и золото. Однако большим недостатком такого подхода стала необходимость во взвешивании слитков и определении их подлинности. Идейным продолжением стали чеканные монеты, которые, по сравнению с использованием слитков, отличались удобством хранения, дробления и соединения, относительной большой стоимостью при небольшом весе и объёме, а также большей надёжностью. К началу второй половины второго тысячелетия стали появляться бумажные варианты денежных средств. Их особенностью стало отсутствие какой-либо реальной самостоятельной ценности, по сравнению с монетами. Однако именно бумажные деньги, или банкноты, являются основными в большинстве современных денежных систем. Итого исторически можно выделить четыре основных вида денег:

– товарные;

– обеспеченные;

– фиатные;

– кредитные.

Появление денег и денежных отношений вызвало острую необходимость в учёте результатов операций с ними. Так, например, в древности использовали бумагу или папирус, на которых фиксировали информацию о совершенном обмене. Позже с течением времени стали появляться более практичные методы ведения учёта, например, специальные журналы, дневники, которые заполнялись вручную или же с помощью печатных машинок. В качестве отдельной эпохи развития можно взять двадцатый век, который принято считать прорывным в области компьютерной техники и программировании. С течением времени активно развивались различные области программирования, модернизировались техническая и технологическая базы, что в итоге привело к созданию большого количества полезных программ различного назначения. В том числе появился и такой класс приложений, как табличные редакторы, среди которых можно выделить и известный Microsoft Excel. Это значительно упростило работу с денежными средствами и операциями. Однако и это не стало пределом, так как после стремительного развития техники и технологий программирования появилась возможность создавать более мобильные, удобные и простые в использовании приложения.

Целью данного курсового проекта является разработка приложения «Менеджер доходов и расходов».

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ
   1. Обзор аналогов

На сегодняшний день существует большое количество приложений, связанных с учётом доходов и расходов, хранением информации о транзакциях и счетах и сбором необходимой статистики. И каждое из них имеет свои особенности.

Одним из наиболее удобных и популярных является приложение «Cost Track», разработанное компанией Artezio. В первую очередь приложение выделяется своим удобным и простым интерфейсом, который представлен на рисунке 1.1. В приложении присутствуют опции создания счетов, в масштабе которых можно вести учёт доходов и расходов с использованием удобной системы создания транзакции. Присутствует возможность выбрать категорию, дату, периодичность, иконку и добавление комментария к операции.



Рисунок 1.1 – Главное меню приложения «Cost Track»

Также стоит отметить, что данное приложение позволяет проводить контроль денежных средств и с помощью удобных круговых диаграмм, представленных на рисунке 1.2, которые отображают статистику по различным критериям и фильтрам:

– по временным отрезкам;

– по категориям;

– по типам операций.

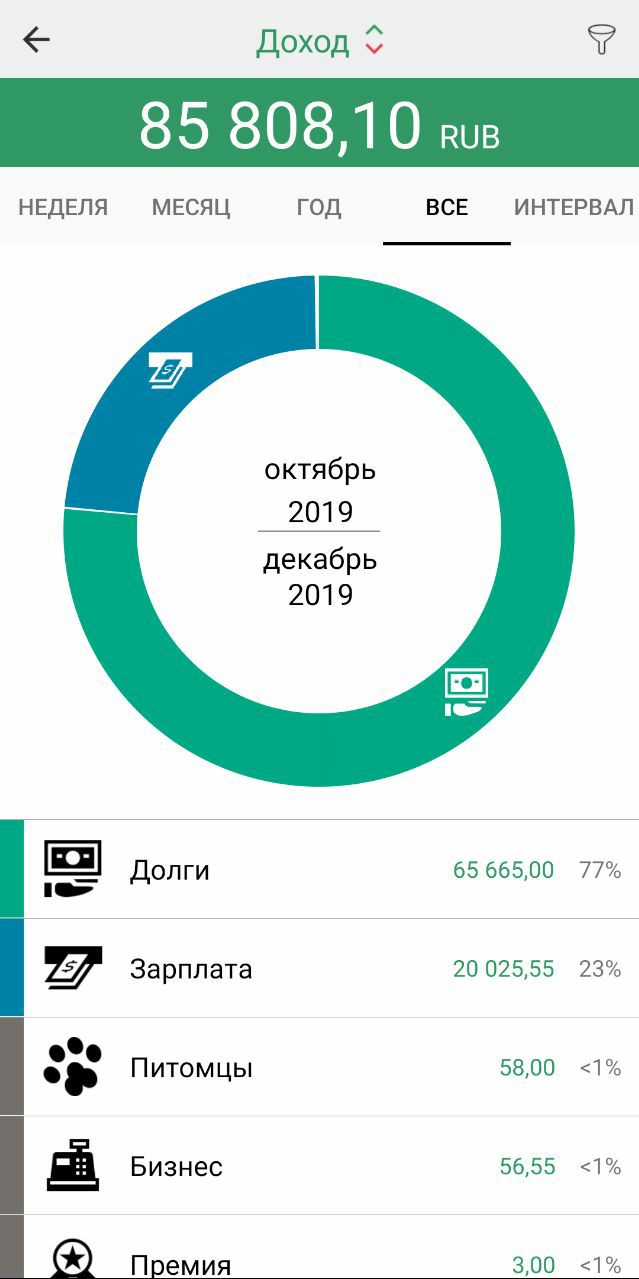


Рисунок 1.2 – Внешний вид окна со статистикой в приложении «Cost Track»

Одним из главных плюсов данного программного средства является постоянная поддержка со стороны разработчиков, которая выражена в наличии свежих обновлений, а также гибкая настройка всего приложения под конкретные нужды пользователя.

Единственным недостатком является наличие платных возможностей и ресурсов, что в определённой мере ограничивает функционал программы.

Вторым аналогом послужило мобильное приложение «Кошельки», которое по концепции схоже с предыдущим, однако имеет ряд существенных преимуществ. Внешний вид программного средства показан на рисунке 1.3. На нём видно, что для отображения транзакций точно так же выбран список с возможностью прокрутки. Важным отличием является то, что приложение нацелено на работу в рамках одного кошелька, хотя опция добавление других так же присутствует, но имеет не такой ярко-выраженный акцент. Из основных достоинств продукта можно выделить:

– более гибкую систему транзакций;

– наличие тегов для упрощённой работы с фильтрами;

– более продвинутую и детальную статистику, вкладка с которой представлена на рисунке 1.4;

– возможность управления данными на уровне таблиц;

– возможность экспорта таблиц с данными в случае необходимости переноса информации на другое устройство;

– многоуровневую иерархическую систему категорий доходов и расходов.

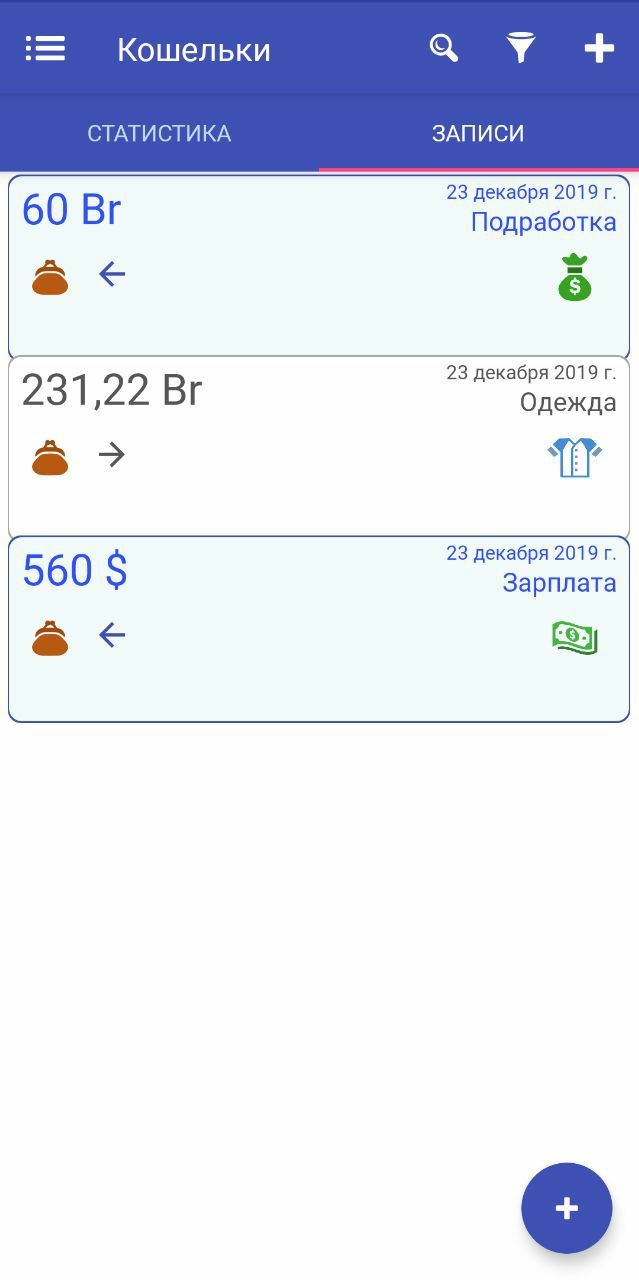


Рисунок 1.3 – Внешний вид главного меню приложения «Кошельки»

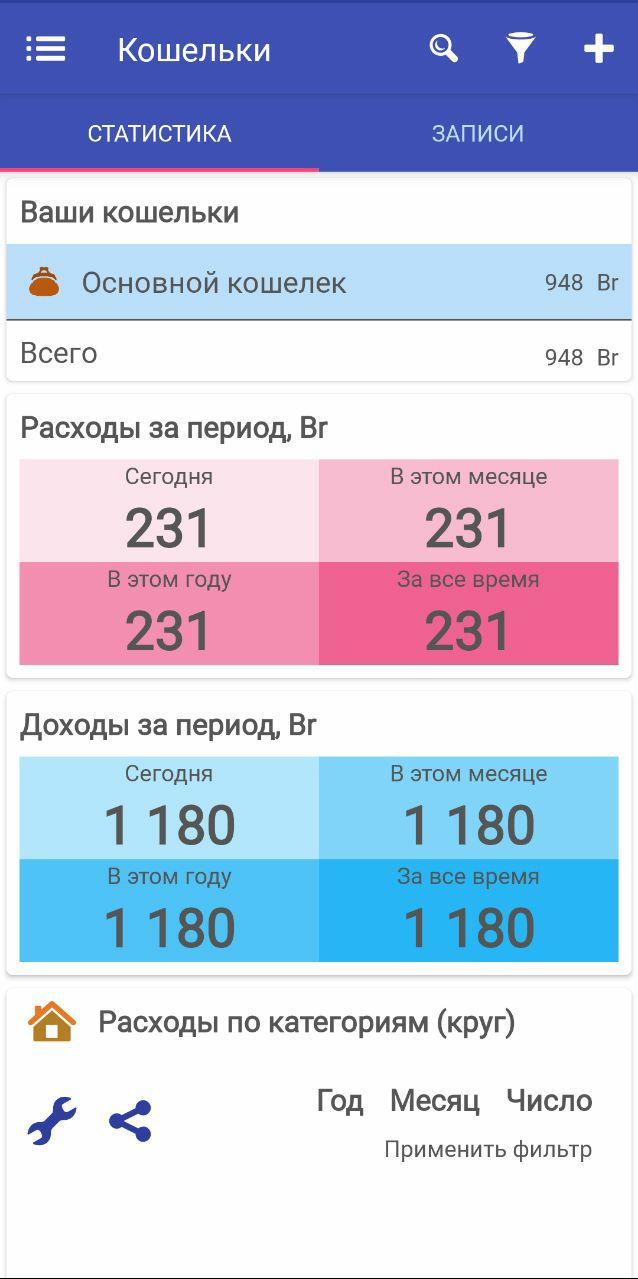


Рисунок 1.4 – Внешний вид вкладки со статистикой

При наличии большого количества преимуществ у данного программного продукта имеется не менее серьёзный недостаток: приложение «Кошельки» находится на стадии разработки и тестирования, а потому не исключаются многочисленные ошибки и не предоставляются гарантии к безопасному и полноценному использованию.

Третьим программным средством является приложение «myMoneyTracker», внешний вид которого проиллюстрирован на рисунке 1.5. Данный программный продукт собрал в себе лучшие идеи из двух предыдущих аналогов, развил их, а также предложил новые.

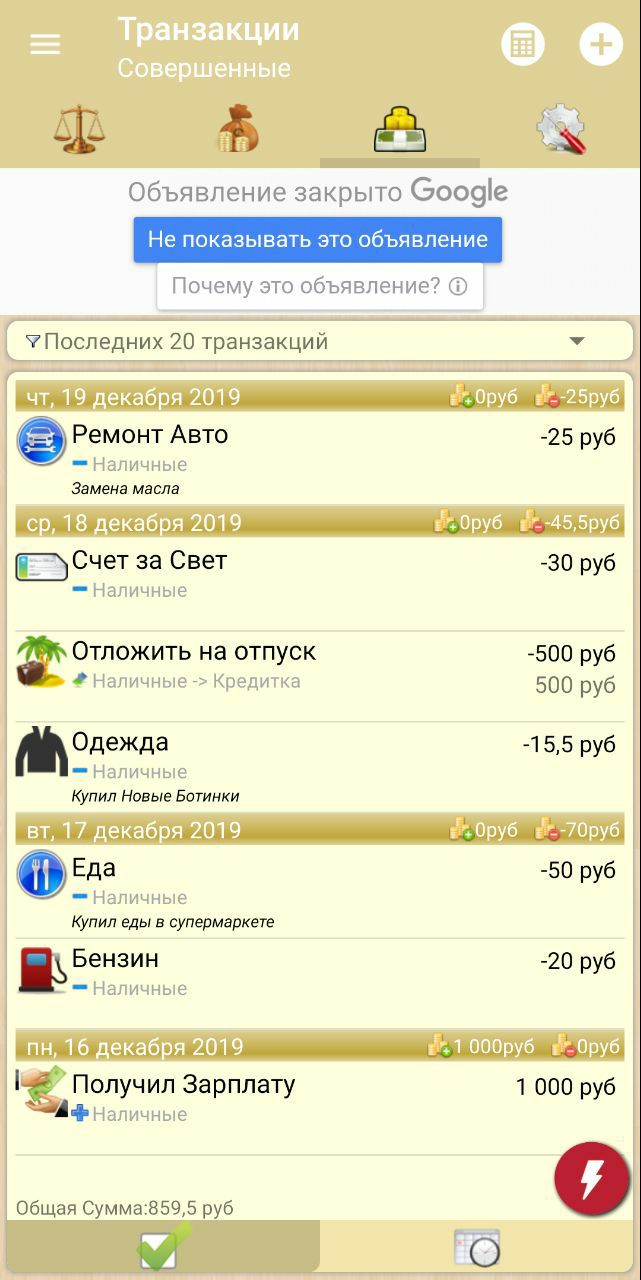


Рисунок 1.5 – Внешний вид главного меню приложения «myMoneyTracker»

В качестве основных отличительных достоинств приложения «myMoneyTracker» можно выделить:

– детальную инструкцию эксплуатации;

– систему бюджетов и сбережений;

– наличие групп и подгрупп;

– синхронизацию с облачным хранилищем;

– парсинг мобильных сообщений об операциях с банковскими картами с целью автоматизации;

– менеджер долгов;

– менеджер сбережений.

Также отдельным пунктом можно выделить то, что разработчики периодически выпускают обновления, дорабатывают уже имеющиеся части продукта и добавляют новые особенности и возможности.

Данное приложение также имеет и ряд недостатков, среди которых:

– наличие рекламы в главном меню;

– достаточно небольшой выбор категорий расходов и доходов;

– неудобный интерфейс.

* 1. Постановка задачи

В рамках данного курсового проекта планируется разработка программного средства «Менеджер доходов и расходов».

Будут разработаны алгоритмы создания баз данных, получения доступа к ним и записи и чтения из них.

В программном средстве планируется реализовать следующие функции:

– создание отдельных кошельков и счетов;

– добавление расходов и доходов;

– преобразование валют;

– настройка кошельков;

– просмотр статистики с помощью круговых диаграмм.

Логическая составляющая приложения будет включать в себя:

* взаимодействие структурных блоков приложения;
* работу с базой данных;
* алгоритмы преобразования данных.

Для разработки программного средства будет использоваться язык программирования C++, операционная система Windows 10, среда разработки Qt Creator 5.4.2, а также графический редактор Adobe XD.

1. РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА
2. 1. Структура программы

Структура программы представляет собой модули, разделённые на две группы:

– модули моделей;

– модули отображения.

Группа модулей отображения выглядит следующим образом:

– categoriesWindow – окно редактирования списка категорий;

– currenciesWindow – окно редактирования списка валют;

– incomeWindow – окно добавления транзакции;

– main – модуль создания главного окна;

– mainWindow – главное окно;

– newWalletWindow – окно создания нового кошелька;

– settingsWindow – окно с общими настройками приложения;

– statWindow – окно получения статистики;

– walletSettingsWindow – окно с настройками отдельного кошелька;

– walletWindow – окно кошелька.

Модули моделей представлены ниже:

– categoriesModel;

– dbConnection;

– incomeWinModel;

– mainWinModel;

– newWalletWinModel;

– settingsWinModel;

– statWinModel;

– walletSettingsWinModel;

– walletWinModel;

Подобная структура позволила разнести методы и функции с логикой от методов и функций представления данных в пользовательском интерфейсе, что придало исходному коду определённый вид, благодаря которому он стал более читабельным. С наличием большого количества окон такой метод позволил сократить многократное повторение больших участков кода. Также подобный подход облегчил отладку и тестирование программы на этапе разработки.

Стоит также отметить, что большую роль в написании программы с данной структурой сыграла среда программирования Qt Creator, в которой поддерживается система сигналов и слотов, которые позволили приблизиться к архитектуре модель – контроллер – представление.

* 1. Интерфейс программного средства

Внешний вид и удобность в использовании являются одними из главных критериев качества программного средства. Поэтому взаимодействие приложения с пользователем необходимо организовать максимально интуитивно и просто.

Для создания главного окна были выбраны стандартные компоненты среды Qt Creator такие, как QPushButton, QLabel и QListView. Последний был использован для отображения модели списка, в котором записаны названия кошельков. Также для каждого из компонентов отдельно были разработаны варианты внешнего вида с помощью графического редактора Adobe XD. Изменение внешнего вида стандартных компонентов было возможно благодаря наличию свойства styleSheet в описании их классов, с помощью которого можно было настроить поведение при наведении мыши или нажатии.

На рисунке 2.1 представлено главное окно программы.

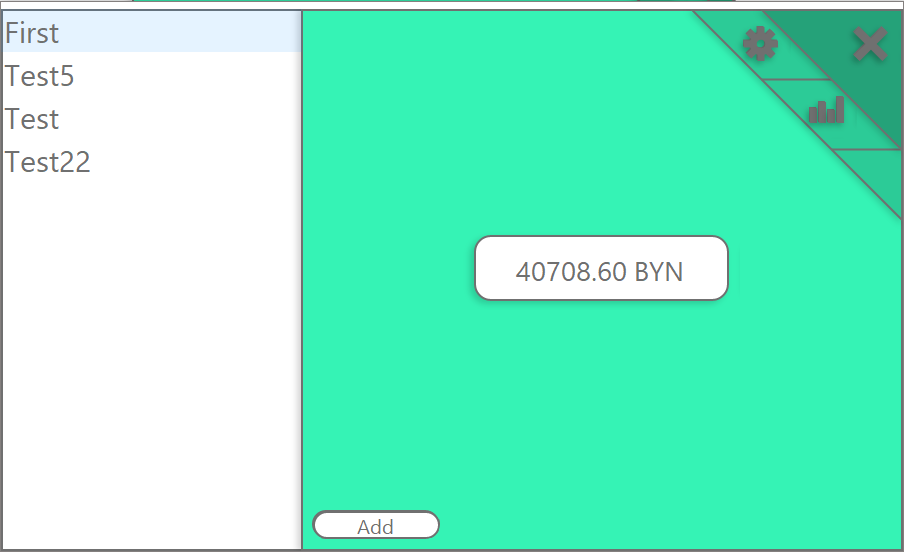


Рисунок 2.1 – Внешний вид главного окна программы

Также в остальных окнах, помимо вышеперечисленных, были использованы такие компоненты, как QRadioButton, QLineEdit, QComboBox, QTableView, QDoubleSpinBox. QTableView предназначен для отображения содержимого модели, основанной на запросе к базе данных. Компонент QDoubleSpinBox нашёл применение в окнах, где необходимо вводить числовые значения с определённым количеством знаков после запятой. Пример его применения представлен на рисунке 2.2.

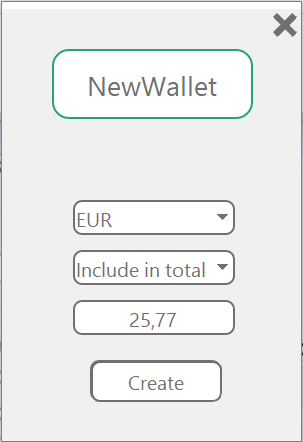


Рисунок 2.2 – Внешний вид окна создания нового кошелька

* 1. Работа с графическими ресурсами

Программное средство взаимодействует с определённым набором графических ресурсов, обеспечивающих работу с круговыми диаграммами.

Создание круговых диаграмм и управление ими реализовано при помощи подключаемой библиотеки QtCharts. Она позволяет рисовать и настраивать диаграммы и графики различной сложности и различного уровня детализации. Данная библиотека имеет обширный список предоставляемых возможностей, среди которых гибкая настройка отображения самой диаграммы, легенды, подписей и совместимость со стандартными структурами данных, благодаря чему процесс написания кода был упрощён и ускорен.

Для отображения диаграммы, пример которой показан на рисунке 2.3, создаётся объект класса QChartView на основе объекта класса QChart, возвращаемого как значение метода модели getChart.

QChartView \*view = new QChartView(model->getChart(

ui->cbPeriod->currentText(),

ui->cbList->currentText(),

ui->cbType->currentText() == "Incomes"? "Income":

"Consumption", currency));

view->setRenderHint(QPainter::Antialiasing);

view->setGeometry(300, 300, 500, 500);

view->show();

Параметрами метода getChart являются класс временного промежутка, сам временной промежуток, основная валюта кошелька и тип транзакций: доход, расход.

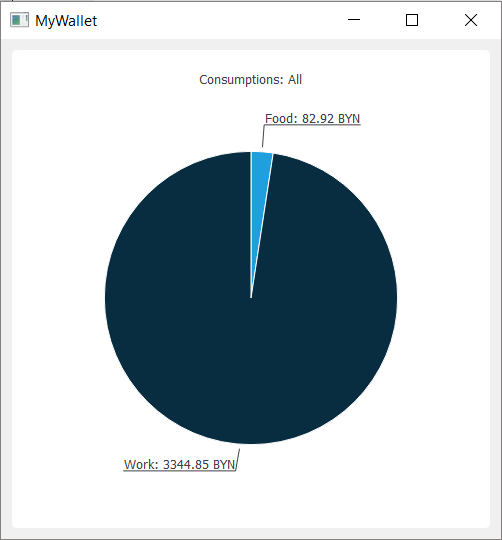


Рисунок 2.3 – Внешний вид окна с диаграммой расходов за всё время

В самом начале метода getChart определяется класс временного промежутка вместе с конкретным промежутком, на основании которых строятся два списка:

– values – хранит числовые значения для соответствующих категорий;

– names – хранит перечень категорий.

А далее на основании этих двух списков формируется и объект класса QChart. Код формирования объекта представлен ниже:

QPieSeries \*series = new QPieSeries;

for(int i = 0; i < names.count(); i++)

{

series->append(names.value(i), values.value(i));

series->slices().at(i)->setLabelVisible();

series->slices().at(i)->setLabel(names.value(i) + ": " +

QString::number(values.value(i), 'f', 2) + " " +

currency);

}

QChart \*chart = new QChart();

chart->addSeries(series);

chart->setTitle(type + "s: " + currPeriod);

chart->legend()->hide();

Также стоит отметить, что валюты, в которых осуществлялись транзакции, могли отличаться, поэтому при формировании диаграммы необходимо было привести всё к одной главной валюте кошелька. Это было реализовано при помощи метода convertCurrency, на вход которого подавалось значение, текущая валюта и валюта, в которую необходимо преобразовать значение. Блок-схема метода convertCurrency представлена на рисунке 2.4.



Рисунок 2.4 – Блок-схема метода convertCurrency

* 1. Логика работы с базой данных

Основная логика программного средства состоит в работе с базой данных. Для того, чтобы приложение могло использовать базы данных и их преимущества, была выбрана подключаемая библиотека QtSql. Она предоставляет возможности управления базами данных при помощи запросов, написанных на языке SQL.

В качестве хранилища было решено использовать локальную базу данных SQLite из-за широких возможностей взаимодействия с ней, простотой в использовании и читаемости кода, осуществляющего управление базой данных.

На этапе проектирования программного средства было решено разделить базу данных на несколько таблиц, различающихся по назначению:

– wallets – хранение информации о кошельках;

– transactions – хранение информации о всех транзакциях;

– currencies – хранение информации обо всех используемых валютах;

– categories – хранение информации о категориях расходов или доходов.

Данные о кошельках в таблице wallets распределены по следующим полям:

– id – название кошелька;

– date – дата создания кошелька;

– inclusion – опция включения кошелька в общий счёт;

– currency – валюта кошелька;

– value – значение кошелька.

QSqlQuery query(QSqlDatabase::database("wallets\_connection"));

query.exec("CREATE table wallets"

"(id text,"

"date text,"

"inclusion text,"

"currency text,"

"value text)");

Данные о транзакциях в таблице transactions распределены по полям, представленным ниже:

– id – идентификатор кошелька, которому принадлежит данная транзакция;

– type – тип транзакции: расход или доход;

– value – значение;

– currency – используемая валюта;

– date – дата транзакции;

– category – категория транзакции.

QSqlQuery secondQuery(QSqlDatabase::database(

"transactions\_connection"));

secondQuery.exec("CREATE TABLE transactions"

"(id text,"

"type text,"

"value text,"

"currency text,"

"date text,"

"category text)");

Таблица currencies хранит название валюты и два коэффициента, характеризующие данную валюту относительно некоторой базовой. В данном приложении в качестве базовой валюты был выбран американский доллар.

При запуске программы начинается процесс связывания приложения с таблицами, который вынесен в отдельную функцию createConnection, блок-схема которой показана на рисунке 2.5. Эта функция вызывается для каждой из таблиц, после чего все таблицы конфигурируются на случай их ручного изменения или преднамеренного удаления.



Рисунок 2.5 – Блок-схема функции createConnection

После того, как связь между приложением и базой данных успешно настроена, работа с ней заключается в отправке запросов.

Добавление нового кошелька в таблицу wallets происходит после нажатия на объект bCreate класса QPushButton в окне создания кошелька. При этом происходит вызов метода addWallet, сокращённая блок-схема которого представлена на рисунке 2.6. Непосредственно добавление происходит после отправки запроса, код которого представлен ниже:

inclusion = inclusion == "Include in total"? "yes": "no";

QSqlQuery query(QSqlDatabase::database("wallets\_connection"));

query.prepare("INSERT INTO wallets (id, date, inclusion,

currency, value)"

"VALUES (:id, :date, :inclusion, :currency,

:value)");

query.bindValue(":id", name);

QString dateTime = QLocale{QLocale::English}.toString(

QDateTime::currentDateTime(), DATE\_FORMAT);

query.bindValue(":date", dateTime);

query.bindValue(":inclusion", inclusion);

query.bindValue(":currency", currency);

QString sign = value < 0? "": "+";

query.bindValue(":value", sign + QString::number(

value, 'f', 2));

query.exec();

Дата записывается в хранилище в формате, описанном в DATE\_FORMAT, и выглядит он так: год-месяц-день, часы:минуты:секунды.



Рисунок 2.6 – Сокращённая блок-схема метода addWallet

Добавление транзакции реализовано похожим образом, только отличие состоит в полях, в которых хранятся данные, и в том, что вместе с добавлением новой строки в таблице transactions обновляется строка соответствующего кошелька, а в частности поле value.

В приложении реализовано два типа настроек:

– частные настройки кошелька;

– общие настройки.

Последние включают в себя опцию изменения текущей главной валюты приложения, которая будет отображаться в главном меню и в которую будут конвертироваться все остальные валюты при выводе статистики, а также опции редактирования таблиц валют и категорий.

Частные настройки кошелька позволяют отредактировать:

– название кошелька;

– валюту кошелька;

– опцию включения в общий счёт.

Окно с данным типом настроек показано на рисунке 2.7. При смене названия кошелька происходит вызов метода setName, сокращённая блок-схема которого представлена на рисунке 2.8. При вызове этого метода реализуются проверки правильности введённого идентификатора: новое имя должно быть уникальным, и оно не должно быть пустым.

QSqlQuery query(QSqlDatabase::database("wallets\_connection"));

QSqlQuery queryTransactions(QSqlDatabase::database(

"transactions\_connection"));

query.prepare("UPDATE wallets SET id = ? WHERE id = ?");

query.addBindValue(newName);

query.addBindValue(name);

query.exec();

queryTransactions.prepare("UPDATE transactions SET id = ? WHERE

id = ?");

queryTransactions.addBindValue(newName);

queryTransactions.addBindValue(name);

queryTransactions.exec();

Особенность запроса для данного случая состоит в том, что вместе с обновлением строки в таблице wallets обновляются строки в таблице транзакций, чтобы после изменения названия кошелька сохранялось верное соответствие.

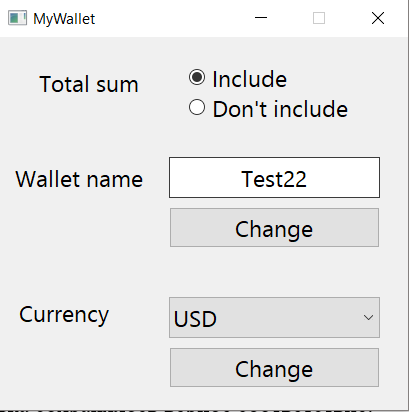


Рисунок 2.7 – Внешний вид окна с частными настройками кошелька



Рисунок 2.8 – Сокращённая блок-схема метода setName

1. ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

В ходе тестирования приложения были выявлены некоторые недостатки интерфейса и логики программного средства. Была составлена таблица 3.1, показывающая ожидаемые и реальные результаты, полученные при заданных условиях, она представлена ниже.

Таблица 3.1 – Ожидаемые и реальные результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № | Тестовые случаи | Ожидаемый результат | Полученный результат |
| 1. | Смена главной валюты | Успешная конвертация общего значения | Успешная конвертация общего значения |
| 2. | Изменение имени кошелька | Успешное изменение имени | Завершение программы с ошибкой |
| 3. | Удаление таблицы кошельков | Создание новой пустой таблицы | Создание новой пустой таблицы |
| 4. | Изменение валюты кошелька | Успешная конвертация значения | Успешная конвертация значения |
| 5. | Удаление кошелька | Успешное удаление кошелька | Успешное удаление кошелька |
| 6. | Ручное удаление таблицы валют | Создание пустой таблицы | Создание пустой таблицы |
| 7. | Создание 13 кошелька | Появление предупреждения | Появление предупреждения |
| 8. | Ввод идентификатора размером более 12 символов | Появление предупреждения | Появление предупреждения |
| 9. | Создание кошелька с пустым полем названия | Вывод предупреждающего сообщения | Вывод предупреждающего сообщения |
| 10. | Добавление новой валюты | Успешное добавление валюты в таблицу | Успешное добавление валюты в таблицу |

Разработка приложения велась с использованием системы контроля версий GitHub, позволившая сохранять состояние программы на каждом отдельном этапе по ходу добавления нового функционала или изменения уже существующего. Появление новых точек возврата происходит посредством группировки изменённых файлов, затем они объединяются под общим именем «коммита», в котором кратко изложена суть изменений. Также можно добавлять к каждому этапу новые файлы, или удалять устаревшие варианты. После накопления определённого количества групп изменений, их следует отправить на удалённый репозиторий, где видна вся история приложения и разница между каждым новым «коммитом».

Одной из проблем во время тестирования, с которой пришлось столкнуться, стала проблема завершения работы программы с ошибкой после изменения имени кошелька в настройках. Связана эта ошибка была с неправильностью передачи нового имени модели окна кошелька. Получение непредсказуемых результатов прекратилось после изменения способа передачи. Передача нового идентификатора была реализована с помощью сигналов и слотов, как показано ниже:

connect(walletSettingsWindow, SIGNAL(walletWindow(QString)),

this, SLOT(setName(QString)));

Окончательный вид процедуры closeEvent окна настроек:

void WalletSettingsWindow::***closeEvent***(QCloseEvent \*event)

{

emit walletWindow(name);

event->accept();

}

Второй проблемой стала неправильная интерпретация данных во время добавления транзакций. Например, вводя значение с двумя знаками после запятой, пользователь мог получить в реальной таблице значение, округлённое в меньшую сторону как результат выполнения следующей строки кода:

query.bindValue(":value", sign + QString::number(incomeValue);

Решением стало использование перегруженного метода QString::number, где последний параметр отвечает за количество знаков после запятой. Итоговой вид строки кода представлен ниже:

query.bindValue(":value", sign + QString::number(incomeValue,

'f', 2));

1. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ
   1. Интерфейс программы

Интерфейс программы разрабатывался таким образом, чтобы он был как можно более простым, понятным и удобным. А также, чтобы пользователь мог с первых секунд понять, на что способно данное программное средство.

При запуске программы в первую очередь происходит подключение базы данных, и если на каком-то из этапов происходит ошибка, то пользователь будет об этом уведомлён. Точно так же пользователь узнает об ошибке, которая будет вызвана намеренным изменением данных в таблицах вручную.

При успешном подключении базы данных открывается главное окно, подробная схема которого представлена на рисунке 4.1.

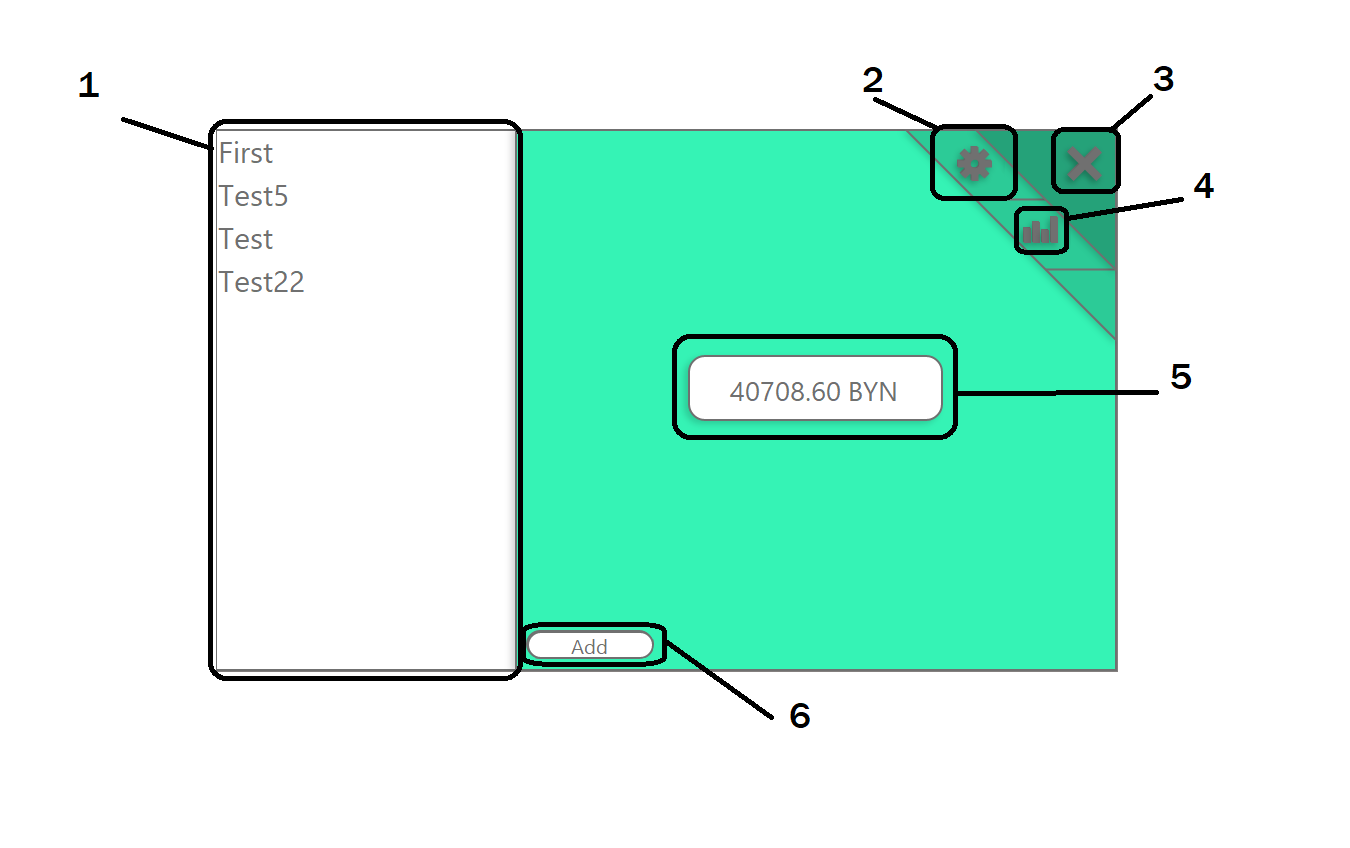


Рисунок 4.1 – Схема главного окна

Главное окно программы содержит следующие элементы:

1. Список кошельков и счетов;
2. Кнопка открытия окна с общими настройками;
3. Кнопка закрытия приложения;
4. Кнопка открытия окна со статистикой;
5. Поле для отображения общей суммы значений всех кошельков;
6. Кнопка открытия окна создания нового кошелька.

При нажатии на один из идентификаторов, находящихся в списке кошельков происходит открытие окна данного счёта.

Окно кошелька, показанное на рисунке 4.2, состоит из следующих элементов:

1. Поле для отображения значения кошелька;
2. Поле, отображающее дату создания;
3. Поле имени;
4. Кнопка открытия окна частных настроек;
5. Кнопка добавления транзакции;
6. Кнопка удаления кошелька;
7. Таблица со всеми транзакциями данного счёта.

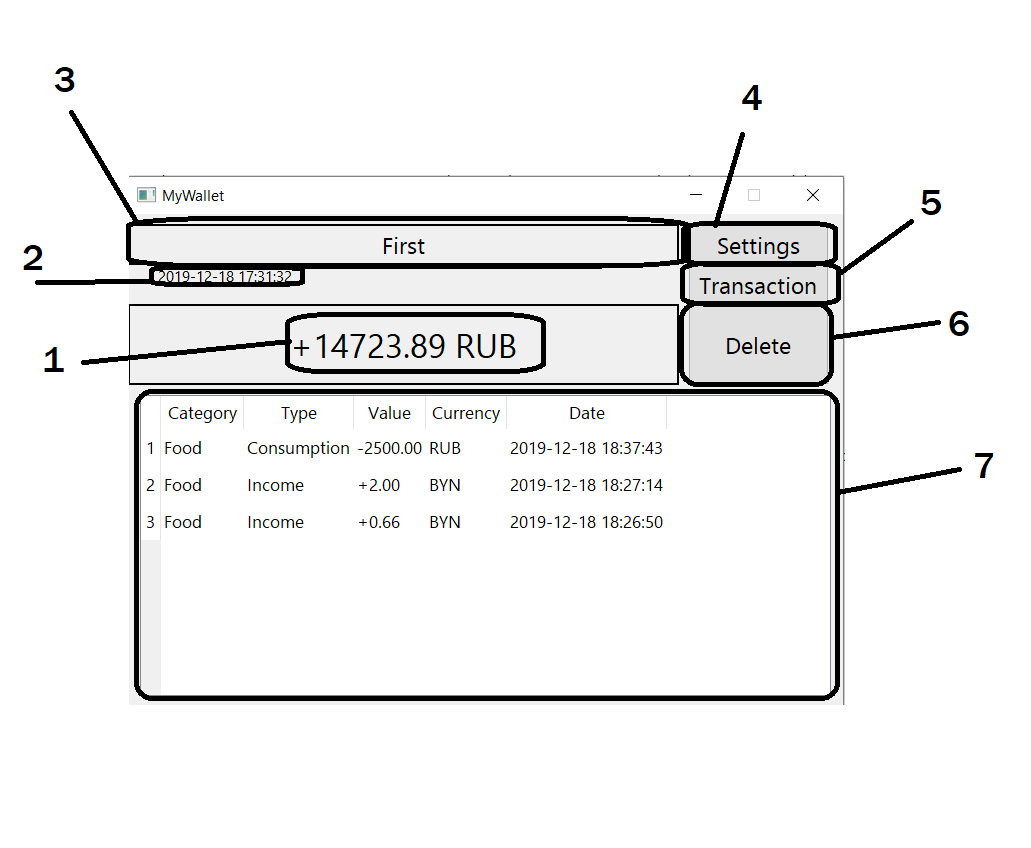


Рисунок 4.2 – Внешний вид окна кошелька

После нажатия на кнопку добавления транзакции открывается соответствующее окно, представленное на рисунке 4.3. Данное окно состоит из следующих элементов:

1. Кнопка добавления транзакции;
2. Поле ввода значения;
3. Поле выбора категории.

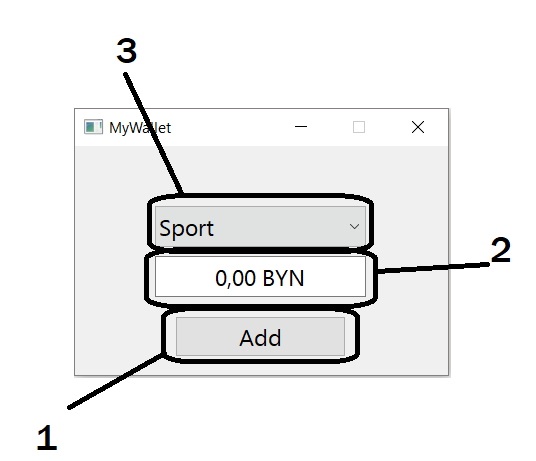


Рисунок 4.3 – Внешний вид окна добавления транзакции

Окно просмотра статистики, проиллюстрированное на рисунке 4.4, состоит из нижеприведённых элементов:

1. Поле выбора типа транзакции;
2. Поле выбора типа временного промежутка;
3. Поле выбора конкретного временного промежутка;
4. Кнопка отображения круговой диаграммы.

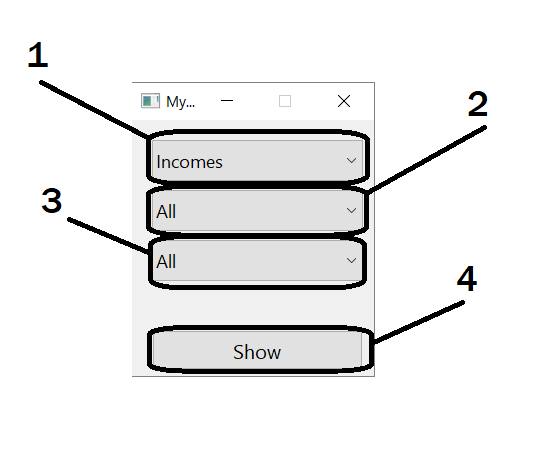


Рисунок 4.4 – Внешний вид окна просмотра статистики

* 1. Порядок работы

Для того, чтобы добавить новый кошелёк, необходимо нажать кнопку «Add» внизу на главном окне, после чего откроется окно создания кошелька, где будет необходимо заполнить обязательные поля и нажать кнопку «Create».

Для того, чтобы удалить счёт, необходимо нажать кнопку «Delete» в меню кошелька.

Для добавления транзакции нужно выбрать нужный кошелёк или счёт, в меню нажать на кнопку «Transaction», в открывшемся окне заполнить поля и нажать кнопку «Add».

Для открытия статистики необходимо нажать кнопку с изображением графиков, расположенную на главном окне.

Для редактирования списков валют и категорий нужно нажать кнопку с изображением шестерёнки, расположенную на главном окне, после чего в открывшемся меню нажать на одну из кнопок с названием «Edit…».

ЗАКЛЮЧЕНИЕ

В настоящее время актуальность контроля бюджета и учёта доходов с расходами только возрастает, согласно современным тенденциям. Деньги и операции с ними окружают нас каждый день, поэтому очень важным является умение правильно ими распоряжаться и с умом подходить к их подсчёту.

В рамках данного курсового проекта было разработано программное средство «Менеджер доходов и расходов», которое позволит вести денежный учёт и контроль баланса. Согласно поставленным задачам, в данном приложении были реализованы следующие функции:

– создание отдельных кошельков и счетов;

– добавление расходов и доходов;

– преобразование валют;

– настройка кошельков;

– просмотр статистики с помощью круговых диаграмм.

Для успешного выполнения всех поставленных целей потребовалось изучить библиотеки QtSql и QtCharts, основы работы с базами данных, объектно-ориентированные возможности языка C++, изучить основные принципы данной парадигмы, а также освоить взаимодействия существующих компонентов.

Существует много возможностей для дальнейшего улучшения приложения. Одним из самых очевидных является модернизация системы хранения данных и перенос её в облачное хранилище или в сеть. Также возможно улучшение приложения за счёт добавления гибкой настройки и переноса на мобильные платформы, что позволит приложению стать более удобным.

Использование данного приложения сможет развить полезную привычку умело считать свои сбережения и распоряжаться ими, а также оно способно развить в пользователе чувство бережливости.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Программирование на языке C++ / Гирберд Шилдт. Учебный курс: / Гирберд Шилдт. – СПб: изд. С. В. Малгачёва, 2001. – 231 с;
2. Qt Documentation [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://doc.qt.io/qt-5>;
3. Орлов, С. А. Технологии разработки программного обеспечения: учеб. Пособие. – СПб, 2003;
4. Уилсон, С. Принципы проектирования и разработки программного обеспечения, yчебн. курс. – СПб, 2003;
5. SQLite Documentation [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://www.sqlite.org/docs.html>;
6. Культин, Н. А. Основы программирования в C++ 2-е издание – учеб. пособие. – Москва: изд. «НТ Пресс», 2008. – 182с;
7. Культин, Н. А. Основы программирования в C++ – учеб. пособие. – Москва: изд. «НТ Пресс», 2008. – 167с;

ПРИЛОЖЕНИЕ А

Исходный код программы

**mainwindow.h**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QtGui>

#include "newwalletwindow.h"

#include "walletwindow.h"

#include "settingswindow.h"

#include "statwindow.h"

#include "mainwinmodel.h"

#define WINDOW\_SIZE\_WIDTH 902

#define WINDOW\_SIZE\_HEIGHT 542

namespace **Ui** {

class **MainWindow**;

}

class **MainWindow** : public QMainWindow

{

Q\_OBJECT

public:

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

~***MainWindow***();

private slots:

void **on\_bClose\_clicked**();

void **on\_bAdd\_clicked**();

void **on\_list\_clicked**(const QModelIndex &index);

void **on\_bSettings\_clicked**();

void **on\_bStats\_clicked**();

private:

Ui::MainWindow \*ui;

NewWalletWindow \*newWalletWindow;

WalletWindow \*walletWindow;

SettingsWindow \*settingsWindow;

StatWindow \*statWindow;

MainWinModel \*model;

void **prepareWindow**();

};

#endif // MAINWINDOW\_H

**mainwinmodel.h**

#ifndef MAINWINMODEL\_H

#define MAINWINMODEL\_H

#include <QString>

#define TOTAL\_MAX 9999999999

#define TOTAL\_MIN -9999999999

#define APP\_NAME "MyWallet"

class **MainWinModel**

{

public:

**MainWinModel**();

bool **isReadyForAdding**();

void **setCurrency**();

QString **getCurrency**();

double **convertToMainCurrency**(double value, QString walletCurrency);

double **getSum**();

QStringList **getWalletsList**();

private:

QString currency;

};

#endif // MAINWINMODEL\_H

**dbconnection.h**

#ifndef DBCONNECTION\_H

#define DBCONNECTION\_H

#include <QString>

#include <QSqlDatabase>

class **DBConnection**

{

public:

**DBConnection**();

bool **createConnection**(QString id);

void **configureTables**();

private:

QSqlDatabase db;

};

#endif // DBCONNECTION\_H

**newwalletwindow.h**

#ifndef NEWWALLETWINDOW\_H

#define NEWWALLETWINDOW\_H

#include <QWidget>

#include "newwalletwinmodel.h"

#include <QShowEvent>

namespace **Ui** {

class **NewWalletWindow**;

}

class **NewWalletWindow** : public QWidget

{

Q\_OBJECT

public:

explicit **NewWalletWindow**(QWidget \*parent = 0);

~***NewWalletWindow***();

signals:

void **firstWindow**();

private slots:

void **on\_bCreate\_clicked**();

void **on\_bClose\_clicked**();

private:

Ui::NewWalletWindow \*ui;

NewWalletWinModel \*model;

void ***closeEvent***(QCloseEvent \*event);

void ***showEvent***(QShowEvent \*event);

};

#endif // NEWWALLETWINDOW\_H

**newwalletwinmodel.h**

#ifndef NEWWALLETWINMODEL\_H

#define NEWWALLETWINMODEL\_H

#include <QStringList>

#define APP\_NAME "MyWallet"

#define DATE\_FORMAT "yyyy-MM-dd hh:mm:ss"

class **NewWalletWinModel**

{

public:

**NewWalletWinModel**();

QStringList **getCurrenciesList**();

bool **isCorrectName**(QString name);

bool **addWallet**(QString name, QString currency, QString inclusion, double value);

};

#endif // NEWWALLETWINMODEL\_H

**settingswindow.h**

#ifndef SETTINGSWINDOW\_H

#define SETTINGSWINDOW\_H

#include <QWidget>

#include <QShowEvent>

#include <QCloseEvent>

#include "settingswinmodel.h"

#include "currencieswindow.h"

#include "categorieswindow.h"

#define APP\_NAME "MyWallet"

namespace **Ui** {

class **SettingsWindow**;

}

class **SettingsWindow** : public QWidget

{

Q\_OBJECT

public:

explicit **SettingsWindow**(QWidget \*parent = 0);

~***SettingsWindow***();

signals:

void **firstWindow**();

private slots:

void **on\_bChangeCurrency\_clicked**();

void **on\_bAbout\_clicked**();

void **on\_bEditCurr\_clicked**();

void **on\_bEditCat\_clicked**();

private:

Ui::SettingsWindow \*ui;

SettingsWinModel \*model;

CategoriesWindow \*categoriesWindow;

CurrenciesWindow \*currenciesWindow;

void ***closeEvent***(QCloseEvent \*event);

void ***showEvent***(QShowEvent \*event);

};

#endif // SETTINGSWINDOW\_H

**settingswinmodel.h**

#ifndef SETTINGSWINMODEL\_H

#define SETTINGSWINMODEL\_H

#include <QStringList>

class **SettingsWinModel**

{

public:

**SettingsWinModel**();

QStringList **getCurrencies**();

void **setMainCurrency**(QString currency);

};

#endif // SETTINGSWINMODEL\_H

**dbconnection.cpp**

#include "dbconnection.h"

#include "QtSql/QSqlDatabase"

#include <QMessageBox>

#include <QSqlQuery>

DBConnection::**DBConnection**()

{

}

bool DBConnection::**createConnection**(QString id)

{

if(!(db.contains("qt\_sql\_default\_connection")))

db = QSqlDatabase::addDatabase("QSQLITE", id + "\_connection");

db.setDatabaseName(id + ".sqlite");

if(!db.open())

{

QMessageBox::warning(0, "Error", "Database connecting error!");

return false;

}

else return true;

}

void DBConnection::**configureTables**()

{

QSqlQuery query(QSqlDatabase::database("wallets\_connection"));

query.exec("CREATE table wallets"

"(id text,"

"date text,"

"inclusion text,"

"currency text,"

"value text)");

QSqlQuery currenciesQuery(QSqlDatabase::database(

"currencies\_connection"));

currenciesQuery.exec("CREATE table currencies"

"(id text,"

"toUSD text,"

"fromUSD text,"

"isMain text)");

QSqlQuery secondQuery(QSqlDatabase::database("transactions\_connection"));

secondQuery.exec("CREATE TABLE transactions"

"(id text,"

"type text,"

"value text,"

"currency text,"

"date text,"

"category text)");

QSqlQuery categoriesQuery(QSqlDatabase::database(

"categories\_connection"));

categoriesQuery.exec("CREATE TABLE categories"

"(id text)");

}

**main.cpp**

#include "mainwindow.h"

#include <QApplication>

#include "dbconnection.h"

int main(int argc, char \*argv[])

{

QApplication a(*argc*, argv);

DBConnection \*model = new DBConnection;

model->createConnection("wallets");

model->createConnection("transactions");

model->createConnection("currencies");

model->createConnection("categories");

model->configureTables();

MainWindow w;

w.show();

return a.exec();

}

**mainwindow.cpp**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <QMessageBox>

#include <QStringListModel>

#include "QtSql/QSqlQuery"

MainWindow::**MainWindow**(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow)

{

this->setWindowFlags(Qt::CustomizeWindowHint);

setFixedSize(WINDOW\_SIZE\_WIDTH, WINDOW\_SIZE\_HEIGHT);

ui->setupUi(this);

newWalletWindow = new(NewWalletWindow);

walletWindow = new(WalletWindow);

settingsWindow = new(SettingsWindow);

statWindow = new(StatWindow);

model = new(MainWinModel);

connect(newWalletWindow, &NewWalletWindow::firstWindow,

this, &MainWindow::show);

connect(newWalletWindow, &NewWalletWindow::firstWindow,

this, &MainWindow::prepareWindow);

connect(settingsWindow, &SettingsWindow::firstWindow,

this, &MainWindow::show);

connect(settingsWindow, &SettingsWindow::firstWindow,

this, &MainWindow::prepareWindow);

connect(statWindow, &StatWindow::firstWindow, this, &MainWindow::show);

connect(statWindow, &StatWindow::firstWindow,

this, &MainWindow::prepareWindow);

connect(walletWindow, &WalletWindow::firstWindow,

this, &MainWindow::show);

connect(walletWindow, &WalletWindow::firstWindow,

this, &MainWindow::prepareWindow);

prepareWindow();

}

void MainWindow::**prepareWindow**()

{

model->setCurrency();

double sum = model->getSum();

QStringList sList = model->getWalletsList();

QStringListModel \*sListModel = new QStringListModel;

if((sum > TOTAL\_MAX) || (sum < TOTAL\_MIN))

{

QMessageBox::warning(0, APP\_NAME, "Total sum is out of range!\n

App can work incorrectly!"

"\nTo continue repair .sqlite file");

ui->bAdd->setEnabled(false);

ui->bSettings->setEnabled(false);

ui->bStats->setEnabled(false);

ui->list->setEnabled(false);

ui->laTotal->setEnabled(false);

ui->laTotal->setText("0 " + model->getCurrency());

}

else

{

QString strSum = QString::number(sum, 'f', 2) + " " +

model->getCurrency();

ui->laTotal->setText(strSum);

sListModel->setStringList(sList);

ui->list->*setModel*(sListModel);

}

}

MainWindow::~***MainWindow***()

{

delete ui;

}

void MainWindow::**on\_bClose\_clicked**()

{

this->close();

}

void MainWindow::**on\_bAdd\_clicked**()

{

if(model->isReadyForAdding())

{

newWalletWindow->show();

this->close();

}

}

void MainWindow::**on\_list\_clicked**(const QModelIndex &index)

{

QString walletName = ui->list->model()->*data*(index).toString();

walletWindow->name = walletName;

walletWindow->show();

this->close();

}

void MainWindow::**on\_bSettings\_clicked**()

{

settingsWindow->show();

this->close();

}

void MainWindow::**on\_bStats\_clicked**()

{

statWindow->currency = model->getCurrency();

statWindow->show();

this->close();

}

**mainwinmodel.cpp**

#include "mainwinmodel.h"

#include "QtSql/QSqlDatabase"

#include "QtSql/QSqlQuery"

#include <QMessageBox>

#include <sstream>

#include <QStringListModel>

#include <cmath>

MainWinModel::**MainWinModel**()

{

}

double MainWinModel::**convertToMainCurrency**(double value, QString walletCurrency)

{

QSqlQuery query(QSqlDatabase::database("currencies\_connection"));

double toCoeff, fromCoeff;

query.prepare("SELECT toUSD FROM currencies WHERE id = ?");

query.addBindValue(walletCurrency);

query.exec();

while(query.next())

{

toCoeff = query.value(0).toDouble();

query.prepare("SELECT fromUSD FROM currencies WHERE id = ?");

query.addBindValue(currency);

query.exec();

while(query.next())

{

fromCoeff = query.value(0).toDouble();

return round(value \* toCoeff \* fromCoeff \* 100) / 100;

}

}

}

double MainWinModel::**getSum**()

{

double sum = 0;

QSqlQuery query(QSqlDatabase::database("wallets\_connection"));

query.exec("SELECT id, date, inclusion, currency, value FROM wallets");

while(query.next())

if(query.value(2) == "yes")

sum += convertToMainCurrency(query.value(4).toDouble(),

query.value(3).toString());

return sum;

}

QStringList MainWinModel::**getWalletsList**()

{

QStringList sList;

QSqlQuery query(QSqlDatabase::database("wallets\_connection"));

query.exec("SELECT id, date, inclusion, currency, value FROM wallets");

while(query.next())

sList.append(query.value(0).toString());

return sList;

}

bool MainWinModel::**isReadyForAdding**()

{

QSqlQuery query(QSqlDatabase::database("wallets\_connection"));

query.exec("SELECT COUNT(\*) FROM wallets");

int rows = 0;

if(query.next())

rows = query.value(0).toInt();

if(rows < 12)

{

return true;

}

else

{

QMessageBox::warning(0, APP\_NAME, "Maximal amount of

wallets is 12!");

return false;

}

}

void MainWinModel::**setCurrency**()

{

QString currency;

QSqlQuery query(QSqlDatabase::database("currencies\_connection"));

query.prepare("SELECT id FROM currencies WHERE isMain = ?");

query.addBindValue("yes");

query.exec();

while(query.next())

currency = query.value(0).toString();

this->currency = currency;

}

QString MainWinModel::**getCurrency**()

{

return this->currency;

}

**newwalletwindow.cpp**

#include "newwalletwindow.h"

#include "ui\_newwalletwindow.h"

#include "QtSql/QSqlDatabase"

#include "QtSql/QSqlQuery"

#include <QCloseEvent>

#include <QMessageBox>

NewWalletWindow::**NewWalletWindow**(QWidget \*parent) :

QWidget(parent),

ui(new Ui::NewWalletWindow)

{

this->setWindowFlags(Qt::CustomizeWindowHint);

setFixedSize(299, 432);

ui->setupUi(this);

model = new NewWalletWinModel;

ui->cbInclusion->addItem("Include in total");

ui->cbInclusion->addItem("Don't include in total");

}

NewWalletWindow::~***NewWalletWindow***()

{

delete ui;

}

void NewWalletWindow::***showEvent***(QShowEvent \*event)

{

event->accept();

ui->cbCurrency->clear();

ui->cbCurrency->addItems(model->getCurrenciesList());

}

void NewWalletWindow::**on\_bCreate\_clicked**()

{

QString name = ui->eName->text();

QString currency = ui->cbCurrency->currentText();

QString inclusion = ui->cbInclusion->currentText();

double value = ui->value->value();

if(model->addWallet(name, currency, inclusion, value))

this->close();

}

void NewWalletWindow::***closeEvent***(QCloseEvent \*event)

{

ui->eName->clear();

ui->value->setValue(0);

emit firstWindow();

event->accept();

}

void NewWalletWindow::**on\_bClose\_clicked**()

{

this->close();

}

**newwalletwinmodel.cpp**

#include "newwalletwinmodel.h"

#include "QtSql/QSqlDatabase"

#include "QtSql/QSqlQuery"

#include <QMessageBox>

#include <QDateTime>

#include <QLocale>

NewWalletWinModel::**NewWalletWinModel**()

{

}

QStringList NewWalletWinModel::**getCurrenciesList**()

{

QStringList list;

QSqlQuery query(QSqlDatabase::database("currencies\_connection"));

query.exec("SELECT id FROM currencies");

while(query.next())

{

list.append(query.value(0).toString());

}

return list;

}

bool NewWalletWinModel::**isCorrectName**(QString name)

{

QSqlQuery query(QSqlDatabase::database("wallets\_connection"));

query.exec("SELECT id FROM wallets");

while(query.next())

if(query.value(0) == name)

return false;

return true;

}

bool NewWalletWinModel::**addWallet**(QString name, QString currency, QString inclusion, double value)

{

if(name.count() != 0)

{

if(isCorrectName(name))

{

if(currency.count() != 0)

{

if(inclusion.count() != 0)

{

inclusion = inclusion == "Include in total"? "yes": "no";

QSqlQuery query(QSqlDatabase::database(

"wallets\_connection"));

query.prepare("INSERT INTO wallets (id, date,

inclusion, currency, value)"

"VALUES (:id, :date, :inclusion,

:currency, :value)");

query.bindValue(":id", name);

QString dateTime = QLocale{QLocale::English}.toString(

QDateTime::currentDateTime(),

DATE\_FORMAT);

query.bindValue(":date", dateTime);

query.bindValue(":inclusion", inclusion);

query.bindValue(":currency", currency);

QString sign = value < 0? "": "+";

query.bindValue(":value", sign + QString::number(value,

'f', 2));

query.exec();

return true;

}

else

QMessageBox::warning(0, APP\_NAME, "Inclusion option must

be selected!");

}

else

QMessageBox::warning(0, APP\_NAME, "Currency must be

selected!");

}

else

QMessageBox::warning(0, APP\_NAME, "Such identifier is already

exists!");

}

else

QMessageBox::warning(0, APP\_NAME, "Name field must be filled!");

return false;

}

**settingswindow.cpp**

#include "settingswindow.h"

#include "ui\_settingswindow.h"

#include <QMessageBox>

SettingsWindow::**SettingsWindow**(QWidget \*parent) :

QWidget(parent),

ui(new Ui::SettingsWindow)

{

ui->setupUi(this);

setFixedSize(443, 282);

model = new SettingsWinModel;

currenciesWindow = new CurrenciesWindow;

categoriesWindow = new CategoriesWindow;

connect(categoriesWindow, &CategoriesWindow::settingsWindow,

this, &SettingsWindow::show);

connect(currenciesWindow, &CurrenciesWindow::settingsWindow,

this, &SettingsWindow::show);

}

SettingsWindow::~***SettingsWindow***()

{

delete ui;

}

void SettingsWindow::***showEvent***(QShowEvent \*event)

{

event->accept();

ui->cbCurrency->clear();

ui->cbCurrency->addItems(model->getCurrencies());

}

void SettingsWindow::***closeEvent***(QCloseEvent \*event)

{

emit firstWindow();

event->accept();

}

void SettingsWindow::**on\_bChangeCurrency\_clicked**()

{

model->setMainCurrency(ui->cbCurrency->currentText());

QMessageBox::information(0, APP\_NAME, "Main currency was successfully

changed on " + ui->cbCurrency->currentText());

}

void SettingsWindow::**on\_bAbout\_clicked**()

{

QMessageBox::information(0, APP\_NAME, "This program was designed and

created by Aleksey Shukan - \n"

"student of BSUIR, group № 851005. \n"

"Belarus, Minsk 2019. \n\n"

"MyWallet©");

}

void SettingsWindow::**on\_bEditCurr\_clicked**()

{

currenciesWindow->show();

this->hide();

}

void SettingsWindow::**on\_bEditCat\_clicked**()

{

categoriesWindow->show();

this->hide();

}

**settingswinmodel.cpp**

#include "settingswinmodel.h"

#include "QtSql/QSqlDatabase"

#include "QtSql/QSqlQuery"

#include <QVariant>

SettingsWinModel::**SettingsWinModel**()

{

}

QStringList SettingsWinModel::**getCurrencies**()

{

QStringList list;

QSqlQuery query(QSqlDatabase::database("currencies\_connection"));

query.exec("SELECT id FROM currencies");

while(query.next())

list.append(query.value(0).toString());

return list;

}

void SettingsWinModel::**setMainCurrency**(QString currency)

{

QSqlQuery query(QSqlDatabase::database("currencies\_connection"));

query.prepare("UPDATE currencies SET isMain = ?");

query.addBindValue("no");

query.exec();

query.prepare("UPDATE currencies SET isMain = ? WHERE id = ?");

query.addBindValue("yes");

query.addBindValue(currency);

query.exec();

}

**walletwindow.h**

#ifndef WALLETWINDOW\_H

#define WALLETWINDOW\_H

#include <QWidget>

#include "incomewindow.h"

#include "walletsettingswindow.h"

#include "walletwinmodel.h"

#define APP\_NAME "MyWallet"

namespace **Ui** {

class **WalletWindow**;

}

class **WalletWindow** : public QWidget

{

Q\_OBJECT

public:

explicit **WalletWindow**(QWidget \*parent = 0);

~***WalletWindow***();

QString name;

QString date;

QString currency;

QString value;

signals:

void **firstWindow**();

public slots:

void **setName**(QString newName);

private slots:

void **on\_bDelete\_clicked**();

void **on\_bSettings\_clicked**();

void **on\_bIncome\_clicked**();

private:

Ui::WalletWindow \*ui;

IncomeWindow \*incomeWindow;

WalletSettingsWindow \*walletSettingsWindow;

WalletWinModel \*model;

void ***closeEvent***(QCloseEvent \*event);

void ***showEvent***(QShowEvent \*event);

void **prepareTable**();

};

#endif // WALLETWINDOW\_H

**walletwindow.cpp**

#include "walletwindow.h"

#include "ui\_walletwindow.h"

#include <QSqlQuery>

#include <QMessageBox>

#include <QCloseEvent>

#include <QShowEvent>

#include <QSqlQueryModel>

WalletWindow::**WalletWindow**(QWidget \*parent) :

QWidget(parent),

ui(new Ui::WalletWindow)

{

ui->setupUi(this);

setFixedSize(715, 491);

incomeWindow = new IncomeWindow;

walletSettingsWindow = new WalletSettingsWindow;

model = new WalletWinModel;

connect(incomeWindow, &IncomeWindow::walletWindow,

this, &WalletWindow::show);

connect(walletSettingsWindow, SIGNAL(walletWindow(QString)),

this, SLOT(setName(QString)));

}

void WalletWindow::**setName**(QString newName)

{

name = newName;

this->show();

}

void WalletWindow::**prepareTable**()

{

ui->transactions->*setModel*(model->getQueryModel());

ui->transactions->resizeColumnsToContents();

}

void WalletWindow::***showEvent***(QShowEvent \*event)

{

event->accept();

model->setName(name);

date = model->getParameter("date");

currency = model->getParameter("currency");

value = model->getParameter("value");

ui->laDate->setText(date);

ui->laName->setText(name);

ui->laValue->setText(value + " " + currency);

prepareTable();

}

WalletWindow::~***WalletWindow***()

{

delete ui;

}

void WalletWindow::***closeEvent***(QCloseEvent \*event)

{

ui->laDate->clear();

ui->laName->clear();

ui->laValue->clear();

emit firstWindow();

event->accept();

}

void WalletWindow::**on\_bDelete\_clicked**()

{

model->deleteWallet();

this->close();

}

void WalletWindow::**on\_bSettings\_clicked**()

{

walletSettingsWindow->currency = currency;

walletSettingsWindow->name = name;

walletSettingsWindow->inclusion = model->getParameter("inclusion");

walletSettingsWindow->show();

this->hide();

}

void WalletWindow::**on\_bIncome\_clicked**()

{

incomeWindow->currency = currency;

incomeWindow->id = name;

incomeWindow->value = value;

incomeWindow->show();

this->hide();

}