

Министерство образования Республики Беларусь  
Учреждение образования «Белорусский государственный университет  
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Конструирование программ и языки программирования

К ЗАЩИТЕ ДОПУСТИТЬ

\_\_\_\_\_ Б. В. Никульшин

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА  
к курсовому проекту  
на тему

«Программное средство для аренды электровелосипедов и  
электросамокатов»

БГУИР КП 1-40 02 01 514 ПЗ

Студент:

Чеботарёв В.С.

Руководитель:

Ассистент кафедры ЭВМ  
Юревич А.С.

Минск 2021

## СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	3
1 ОБЗОР ЛИТЕРАТУРЫ.....	4
2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ.....	6
3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ.....	8
4 ПРОЕКТИРОВАНИЕ ПРОГРАММНЫХ МОДУЛЕЙ.....	16
5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ.....	18
СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ.....	21
ЗАКЛЮЧЕНИЕ.....	22
ПРИЛОЖЕНИЕ А.....	23
ПРИЛОЖЕНИЕ Б.....	24
ПРИЛОЖЕНИЕ В.....	25
ПРИЛОЖЕНИЕ Г.....	26
ПРИЛОЖЕНИЕ Д.....	27

## ВВЕДЕНИЕ

В наше время большой популярностью стали пользоваться электросамокаты и электровелосипеды. Часто возникает желание разовой поездки, без последующей покупки. В данном случае было бы хорошо иметь под рукой приложение с возможностью взять на прокат один из видов электротранспорта, покататься и вернуть на место откуда его брали. Гуляя по парку я увидел такое место, но оно не было снабжено каким-либо приложением, и вся аренда производилась посредством общения с персоналом данной аренды. Для управляющих данным заведением было проблемой постоянный контроль заряда аккумуляторов и общение с людьми. Возможным решением ситуации виделось создание удобного приложения, которое позволило бы контролировать весь процесс по аренде транспорта и производить большинство действий благодаря приложению.

В плане выбора языка программирования, на котором будет писаться приложение, сомнений не было. С++ является одним из лучших вариантов. Открыв любой тест производительности языков программирования в таблице лидеров, вы обязательно увидите его. В противовес этому часто ставится скорость написания кода, которая, например, у интерпретируемых языков на порядок выше. В этом есть доля истины — С#, Java и конечно же Python даже визуально занимают меньше места, с их помощью можно создавать сложные программы, затратив минимум времени. Однако, что лучше для конечного пользователя: время разработки приложения или его медленная работа? Ответ очевиден.

Еще одной причиной, стала универсальность данного языка, компиляторы С++ есть на каждой операционной системе, большинство программ легко переносится с платформы на платформу, со средой разработки и библиотеками точно не возникнет проблем. Язык имеет богатую классическую библиотеку, которая включает в себя разные контейнеры и алгоритмы, регулярные выражения, разные фреймворки и библиотеки которые позволяют создавать графическую часть приложения.

Исходя из этого можно с уверенностью сказать, что данный язык достаточно удобен для написания выбранной курсовой работы.

Для разработки курсового проекта и реализации графической составляющей приложения был выбран фреймворк для разработки программного обеспечения на языке программирования С++, который носит название Qt.

# 1 ОБЗОР ЛИТЕРАТУРЫ

## 1.1 Анализ аналогов программного средства

Часто пользователи задаются вопросом — какой же сервис по прокату электросамокатов и электровелосипедов лучше? На самом деле ответить на этот вопрос так просто нельзя, кому-то более нравится подписка в одном сервисе, другому человеку цена на разовую поездку, третьему наличие транспортного средства в любом месте города. Сейчас уже появилось достаточное количество приложений проката и каждый найдёт для себя то которое подойдёт именно ему. Разберём наиболее популярные из них:

Eleven. Приложение, которое имеет прокат только электросамокатов. Оно нашло своих пользователей в разных странах. Есть возможность оставить транспортное средство в разных местах города. Фиксированный небольшой ценник и достаточно большое количество точек в городе, где можно арендовать электросамокат. Удобный и понятный каждому интерфейс приложения.

Колоbike. В прокате имеются электросамокаты, электровелосипеды и велосипеды. Первое приложение, которое появилось в Беларуси и тем самым получило большую популярность. Удобен в использовании. Не удобен в качестве оплаты — нужно заранее пополнять баланс перед поездкой. На данный момент имеет большое количество точек с арендой находящиеся в разных точках города и за его пределами. Прокат уже появился в Бресте, Гомеле, Гродно и Пинске.

Urent. Одно из самых новых в Беларуси приложений. Является одним из самых удобных по использованию среди остальных приложений в сфере проката. Имеет возможность ежемесячной подписки благодаря чему брать транспорт на прокат становится дешевле. Часто появляются разные акции и скидки на прокат.

Так же в Беларуси большое количество мест для аренды электросамокатов и электровелосипедов которые не имеют приложений. В основном они являются заведениями, где есть возможность оплаты аренды наличным расчётом и электросамокат и велосипед возвращается на место откуда произошла аренда.

## **1.2 Постановка задачи**

Созданное приложение должно иметь простое меню с возможностью просмотра информации, редактирование информации пользователя, аренде выбранного вида транспорта. Администратор сможет иметь доступ к зарядке электросамокатов и велосипедов, редактировании информации о них, добавление и удаление. Так же он имеет возможность заблокировать или удалить пользователя.

Приложение должно быть интуитивно понятно и удобно в использовании.

Для реализации программы используется объектно-ориентированный язык программирования C++, среда разработки Qt Creator 5.0.3. Приложение написано для ОС Windows 10.

## **2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ**

### **2.1 Сторонние программные компоненты**

Для разработки курсового проекта был выбран фреймворк для разработки программного обеспечения на языке программирования C++, который носит название Qt. Непосредственно разработка в Qt Creator. Отличительная особенность — использование мета объектного компилятора — предварительной системы обработки исходного кода. Расширение возможностей обеспечивается системой плагинов, которые возможно размещать непосредственно в панели визуального редактора Qt Designer. Также существует возможность расширения привычной функциональности виджетов, связанной с размещением их на экране, отображением, перерисовкой при изменении размеров окна.

Qt Designer — кроссплатформенная свободная среда для разработки графических интерфейсов (GUI) программ использующих библиотеку Qt. Он позволяет создавать графические интерфейсы пользователя при помощи ряда инструментов. Существует панель инструментов «Панель виджетов», в которой доступны для использования элементы интерфейса — виджеты, такие как, например, «выпадающий список» ComboBox, «поле ввода» LineEdit, «кнопка» PushButton и многие другие. Каждый виджет имеет свой набор свойств, определяемый соответствующим ему классом библиотеки Qt.

Для хранения информации к приложению подключена база данных MySQL. MySQL – это свободная реляционная система управления базами данных (СУБД). Данная система управления базами данных позволяет хранить данные в обособленных таблицах в виде записей и связывать различные таблицы между собой при помощи ключей.

### **2.2 Структура приложения**

В приложении можно выделить несколько основных элементов: блок авторизации, блок основного меню, блок управления, блок подсчёта времени.

Блок авторизации отвечает за регистрацию и авторизацию пользователя, подключение базы данных к приложению.

Блок основного меню отвечает за доступ ко всем основным действиям в программе.

Блок управления отвечает за редактирование, изменение, удаление данных из базы данных.

Блок подсчёта времени отвечает за отслеживание времени аренды транспортного средства и расчёта стоимости поездки.

Структурная схема представлена в приложении А.

### 3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

Диаграмма классов представлена в приложении Б

#### 3.1 Описание хранения данных программы

Данные программы хранятся в базе данных. База данных имеет название *rent*. В базе данных находятся 2 таблицы: *personal\_data.sql*, *vehicle.sql*. При отсутствии базы данных программа выдаст ошибку в консоли.

Для работы с базой данных, развёрнутой в MySQL, используется декларативный язык программирования «structured query language», или сокращённо SQL, применяющийся для создания, управления и модификации данных реляционных БД (в том числе и в MySQL). Данный язык программирования определяет ряд операторов и команд, при помощи которых организуется работа с данными и самой базой.

Для работы с базой данных используются команды:

`SELECT column1, column2, ... FROM table_name WHERE condition;` — Выполняет выборку данных из таблицы по заданному условию.

`INSERT INTO table_name (column1, column2, column3, ...) VALUES (value1, value2, value3, ...);` — Используется для добавления новых данных в существующую таблицу.

`UPDATE table_name SET column1 = value1, column2 = value2, ... WHERE condition;` — Используется для изменения уже существующих данных в таблице.

Далее рассмотрим более подробно таблицы базы данных блока работы сервера.

##### 3.1.1 Таблица *personal\_data*

Таблица хранит персональные данные пользователей приложения, которые используют. Первый запрос к этой таблице выполняется в самом начале программы, на моменте авторизации пользователя. Более подробно рассмотрим поля, которые хранятся в данной таблице:

- поле *id*. Уникальный номер, является индивидуальным у каждого пользователя.

- поле *login*. Содержит уникальное имя пользователя для входа в приложение.

- поле *pass*. Содержит пароль, который пользователь выбирает самостоятельно для входа в приложение.

- поле *access\_rights*. Содержит информацию о том какие права пользователь имеет в приложении.



- поле `name`. Содержит настоящее имя пользователя.
- поле `surname`. Содержит настоящую фамилию пользователя.
- поле `email`. Содержит адрес электронной почты пользователя.
- поле `post`. Содержит информацию о том какую должность занимает администратор приложения.
- поле `cardnumber`. Содержит информацию о номере карты пользователя.

### 3.1.2 Таблица `vehicle`

Таблица хранит основную информацию о электросамокатах и электровелосипедах.

Рассмотрим подробнее поля данной таблицы:

- поле `id`. Уникальный номер, является индивидуальным у каждого транспортного средства.
- поле `vehicle_type`. Содержит уникальное имя пользователя для входа в приложение.
- поле `model`. Содержит информацию о том какой модели данное транспортное средство.
- поле `is_active_status`. Содержит информацию о том является ли данное транспортное средство активным в данный момент.
- поле `is_on_ride_status`. Содержит информацию о том арендовано ли данное транспортное средство.
- поле `charge`. Содержит информацию о заряде транспортного средства.

## 3.2 Описание работы классов

### 3.2.1 Класс `Human`

Представляет основной класс информации о пользователе:

- `protected QString name` – поле, которое хранит имя пользователя.
- `protected QString login` – поле, которое хранит логин пользователя.
- `protected QString password` – поле, которое хранит пароль пользователя.
- `protected int access_rights` – поле, которое хранит значение прав доступа пользователя.

Для работы с данными полями используется классический набор методов доступа (так называемые «геттеры» и «сеттеры», позволяющие получать и устанавливать значения для полей соответственно):

- `public void set_name (QString)` – метод необходим для установки значения в поле `name`.
- `public QString get_name()` – метод нужен для получения имени из объекта.
- `public int get_access_rights()` – метод нужен для получения прав доступа из объекта.
- `public QString get_login()` – метод нужен для получения логина из объекта.
- `public QString get_password()` – метод нужен для получения пароля напитков из объекта.

### 3.2.2 Класс User

Данный класс наследуется от класса `Human` и предоставляет основную информацию о пользователе не являющимся администратором:

- `private QString surname` – поле, которое хранит фамилию пользователя.
- `private QString email` – поле, которое хранит адрес электронной почты пользователя.
- `private QString cardnumber` – поле, которое хранит номер карты пользователя.

Данный класс реализует стандартный набор методов доступа к полям класса (`get` и `set`).

### 3.2.3 Класс Admin

Данный класс наследуется от класса `Human` и предоставляет основную информацию о администраторе:

- `private QString post` – поле, которое хранит данные о том какой пост занимает администратор.

Данный класс реализует стандартный набор методов доступа к полям класса (`get` и `set`).

### 3.2.4 Класс MainWindow

Данный класс реализует интерфейс `Qt Designer` начального окна авторизации в приложение. Содержит следующие поля:

- поле `private QMainWindow *rent` – класс, который реализует главное окно.

- поле `private MainWindow *ui` – объект, который вызывает интерфейс главного окна.
- поле `private QString login` – строка, содержащая логин.
- поле `private QString password` – строка, содержащая пароль.
- поле `private QSqlDatabase data_base_rent` – объект, который предоставляет подключение к базе данных.

Сигналы кнопок, реализуемые в классе:

- поле `private void on_exit_clicked ()` – нажатие на кнопку производит выход из программы.
- поле `private void on_sign_up_clicked()` – нажатие на кнопку производит регистрацию аккаунта в системе.
- поле `private void on_sign_in_clicked()` – нажатие на кнопку производит вход в приложение .

### 3.2.5 Класс Rent

Представляет собой основное меню приложения. Содержит следующие поля:

- поле `private QString login` – хранит логин пользователя.
- поле `private Rent *ui` – объект, который вызывает интерфейс окна.
- поле `private Human person` – объект класса Human который хранит информацию о пользователе.

Сигналы кнопок, реализуемые в классе:

- поле `private void on_account_clicked()` – нажатие на кнопку открывает окно редактирования и просмотра информации о пользователе.
- поле `private void on_take_scooter_clicked()` – открывает окно аренды транспортного средства и включает таймер проката.
- поле `private void on_take_bike_clicked()` – открывает окно аренды транспортного средства и включает таймер проката.
- поле `private void on_exit_clicked ()` – нажатие на кнопку производит выход из программы.
- поле `private void on_admin_menu_button_clicked()` – нажатие на кнопку открывает меню администратора, кнопка доступна только с правами доступа 2.

### 3.2.6 Класс Vehicle

Хранит информацию о транспортном средстве. Содержит следующие поля:

- поле `private int id` – которое хранит уникальный `id` транспортного средства.
- поле `private int charge` – поле, которое хранит информацию о заряде транспортного средства.
- поле `private QString model` – поле, которое хранит модель транспортного средства.
- поле `private int type` – поле, которое тип транспортного средства.

Данный класс реализует стандартный набор методов доступа к полям класса (`get` и `set`).

### 3.2.7 Класс Timer

Представляет собой класс, который подсчитывает время аренды самоката и высчитывает стоимость поездки. Содержит следующие поля:

- поле `private int charge` – поле, которое хранит информацию о заряде транспортного средства.
- поле `private int time` – поле, которое хранит общее время проката.
- поле `private int sec` – поле для работы таймера и вывода его на экран, хранит в себе количество секунд.
- поле `private int min` – поле для работы таймера и вывода его на экран, хранит в себе количество минут.
- поле `private int hours` – поле для работы таймера и вывода его на экран, хранит в себе количество часов.
- поле `private int vehicle_id` – которое хранит уникальный `id` транспортного средства, которое взято в аренду.
- поле `private Vehicle inf` – объект класса `Vehicle`, которой хранит информацию о транспортном средстве.
- поле `private Timer *ui` – объект, который вызывает интерфейс окна.
- поле `private QTimer *timer` – объект, управляет работой таймера.

Сигналы, реализуемые в классе:

- поле `private void TimerSlot()` – метод который осуществляет работу таймера.

- поле `private void on_end_button_clicked()` – нажатие на кнопку производит завершение поездки и выход в основное меню.

### 3.2.8 Класс Information

Представляет собой класс, который выводит на экран информационное сообщение. Содержит следующие поля:

- поле `private Information *ui` – объект, который вызывает интерфейс окна.
- поле `private void on_close_button_clicked()` – нажатие на кнопку производит закрытие и выход в основное меню.

### 3.2.9 Класс Account

Представляет собой класс, который содержит в себе поля который позволяет просматривать и изменять информацию об аккаунте. Содержит следующие поля:

- поле `private Account *ui` – объект, который вызывает интерфейс окна

Методы:

- поле `public void set_information()` – метод, который достаёт из базы данных информацию об аккаунте и выводит её на экран.
- поле `public bool check_email_on_valid()` – метод, с помощью регулярного выражения проверяет правильность заполнения формы email.
- поле `public bool check_cardnumber_on_valid()` – метод, с помощью регулярного выражения проверяет правильность заполнения формы cardnumber.

Сигналы, реализуемые в классе:

- поле `private void on_changeinf_button_clicked()` – нажатие на кнопку открывает доступ к редактированию информации.
- поле `private void on_back_button_clicked()` – нажатие на кнопку производит выход в основное меню.
- поле `private void on_confirm_button_clicked()` – считывает нажатие на кнопку которая сохраняет изменённые данные.

### 3.2.10 Класс VehiclesShow

Представляет собой таблицу, который содержит в себе поля который позволяет просматривать и изменять информацию об всех транспортах. Содержит следующие поля:

- поле `private VehiclesShow *ui` – объект, который вызывает интерфейс окна
- поле `private QSqlTableModel *model` – объект, который вызывает таблицу получая данные из базы данных

Сигналы, реализуемые в классе:

- поле `private void on_exit_button_clicked()` – нажатие на кнопку производит выход в меню администратора.
- поле `private void on_add_button_clicked()` – нажатие на кнопку добавляет строку в таблицу.
- поле `private void on_delete_button_clicked()` – нажатие на кнопку удаляет выбранную строку в таблице.
- поле `private void on_revertall_button_clicked()` – нажатие на кнопку отменяет все изменения в таблице.
- поле `private void on_submit_button_clicked()` – считывает нажатие на кнопку которая сохраняет все изменения в таблице.

### 3.2.11 Класс UsersShow

Представляет собой таблицу, который содержит в себе поля который позволяет просматривать и изменять информацию об всех пользователях. Содержит следующие поля:

- поле `private UsersShow *ui` – объект, который вызывает интерфейс окна
- поле `private QSqlTableModel *model` – объект, который вызывает таблицу получая данные из базы данных

Сигналы, реализуемые в классе:

- поле `private void on_exit_clicked()` – нажатие на кнопку производит выход в меню администратора.
- поле `private void on_add_clicked()` – нажатие на кнопку добавляет строку в таблицу.
- поле `private void on_del_clicked()` – нажатие на кнопку удаляет выбранную строку в таблице.
- поле `private void on_cancel_clicked()` – нажатие на кнопку отменяет все изменения в таблице.

- поле `private void on_save_clicked()` – считывает нажатие на кнопку которая сохраняет все изменения в таблице.

### **3.2.11 Класс Admin\_menu**

Представляет собой таблицу, который содержит в себе поля который позволяет просматривать и изменять информацию об всех пользователях. Содержит следующие поля:

- поле `private Admin_menu *ui` – объект, который вызывает интерфейс окна

Сигналы, реализуемые в классе:

- поле `private void on_exit_clicked()` – нажатие на кнопку производит выход в меню администратора.

- поле `private on_charge_button_clicked()` – нажатие на кнопку заряжает все транспортные средства у которых заряд  $< 30$ .

- поле `private void on_show_button_clicked()` – нажатие на кнопку открывает окно VehicleShow.

## 4 ПРОЕКТИРОВАНИЕ ПРОГРАММНЫХ МОДУЛЕЙ

### 4.1 Схема алгоритма функции `void MainWindow::onSingInClicked()`

Схема алгоритма метода представлена в приложении В.

Данная функция высчитывает стоимость поездки пользователя и в зависимости от заряда аккумулятора изменяет значения в базе данных после поездки.

### 4.2 Схема алгоритма функции `void Account::onConfirmButtonClicked()`

Схема алгоритма метода представлена в приложении Г.

Данный метод делает проверку правильности ввода всех новых значений аккаунта, добавляет их в базу данных и сохраняет изменения.

### 4.3 Алгоритм по шагам функции `void MainWindow::onSingUpClicked()`

Метод регистрации нового пользователя в программе

1. Начало.
2. Создание переменной `flag` которая отвечает за нахождение пользователя.
3. Сохранение данных с введенного в окне поля в переменную `login`.
4. Сохранение данных с введенного в окне поля в переменную `password`.
5. Проверка `login` и `password` на пустоту.
6. В случае пустоты вывести сообщение об ошибке и завершить метод.
7. Запустить базу данных.
8. Считать с базы данных все логины для проверки на уникальность.
9. Цикл по всем считанным с базы данных значениям пока не дойдет до конца.
10. Если логин совпадает с каким-либо который есть в базе данных вывести на экран ошибку и завершить метод.
11. Установить значение `flag = 0`
12. Завершить цикл 1.
13. Если значение `flag` не равно нулю.
14. Подготовить данные `login` и `password` для занесения в базу данных.
15. Добавить значения в базу данных.
16. Вывести на экран сообщение об успешной регистрации.
17. Конец.

### 4.4 Взаимодействие с базой данных

Qt дает возможность создания приложений для работы с базами данных, используя стандартные СУБД. Qt включает «родные» драйвера для Oracle,



Microsoft SQL Server, Sybase Adaptive Server, IBM DB2, PostgreSQL, MySQL и ODBC-совместимых баз данных. Qt включает специфичные для баз данных виджеты, а также поддерживает расширение для работы с базами данных любых встроенных или отдельно написанных виджетов.

Чтобы получить доступ к базе данных с помощью QSqlQuery и QSqlQueryModel, необходимо создать и открыть одно или более соединений с базой данных. Соединиться с базой данных можно вот так:

```
data_base_rent = QSqlDatabase::addDatabase("QMYSQL",  
"mydb");  
data_base_rent.setHostName("localhost");  
data_base_rent.setDatabaseName("rent");  
data_base_rent.setUserName("root");  
data_base_rent.setPassword("root");  
data_base_rent.open();
```

Первая строка создает объект соединения, а последняя открывает его. В промежутке инициализируется некоторая информация о соединении, включая имя соединения, имя базы данных, имя узла, имя пользователя, пароль. В этом примере происходит соединение с базой данных MySQL rent на узле localhost. Аргумент «QMYSQL» в addDatabase() указывает тип драйвера базы данных, чтобы использовать для соединения, а «mydb» — имя соединения. Как только соединение установлено, можно вызвать статическую функцию QSqlDatabase::database() из любого места программы с указанием имени соединения, чтобы получить указатель на это соединение. Если не передать имя соединения, она вернет соединение по умолчанию. Если open() потерпит неудачу, он вернет false. В этом случае, можно получить информацию об ошибке, вызвав QSqlDatabase::lastError().

Класс QSqlQuery обеспечивает интерфейс для выполнения SQL запросов и навигации по запросу. Для выполнения SQL запросов, просто создают объект QSqlQuery и вызывают QSqlQuery::exec(). Например, вот так:

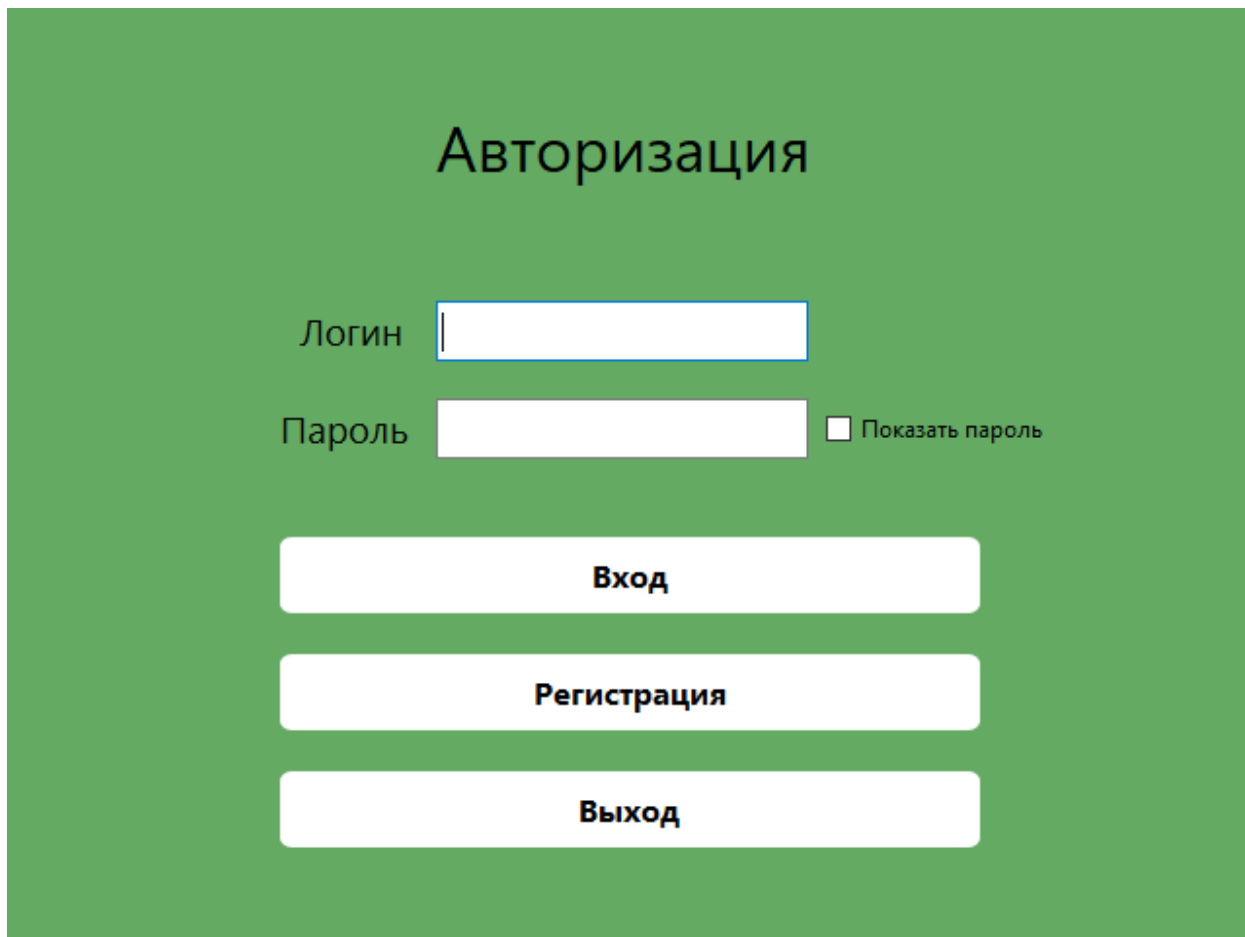
```
QSqlQuery query;  
query.exec("SELECT login FROM personal_data");  
while (query.next()) {  
    if(login == query.value(0).toString()) {  
        QMessageBox::warning(this, "Ошибка", "Такой  
пользователь уже существует!");  
        break;  
    }  
}
```

Так же класс QSqlTableModel может так же являться источником данных для классов представлений, таких как QTableView и QListView. В следующем примере создаётся модель QTableView:

```
QSqlTableModel model = new QSqlTableModel;  
model->setTable("personal_data");  
model->select();  
ui->tableView->setModel(model);
```

## 5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Зайдя в приложение, пользователь увидит окно авторизации пользователю надо будет ввести данные для регистрации или авторизации в приложении (Рисунок 5.1).



Авторизация

Логин

Пароль  ☐ Показать пароль

**Вход**

**Регистрация**

**Выход**

Рисунок 5.1

После окна авторизации появится основное окно, где можно посмотреть информацию о прокате, информацию профиля и кнопка управления, которая видна только администратору. Так же в этом окне можно арендовать транспортное средство. Но если вы только зарегистрировались вам необходимо будет зайти в профиль и заполнить основные данные о себе (Рисунок 5.2).

В окне профиль вы сможете посмотреть или заполнить информацию о себе, нажав на кнопку изменить информацию (Рисунок 5.3).



Рисунок 5.2

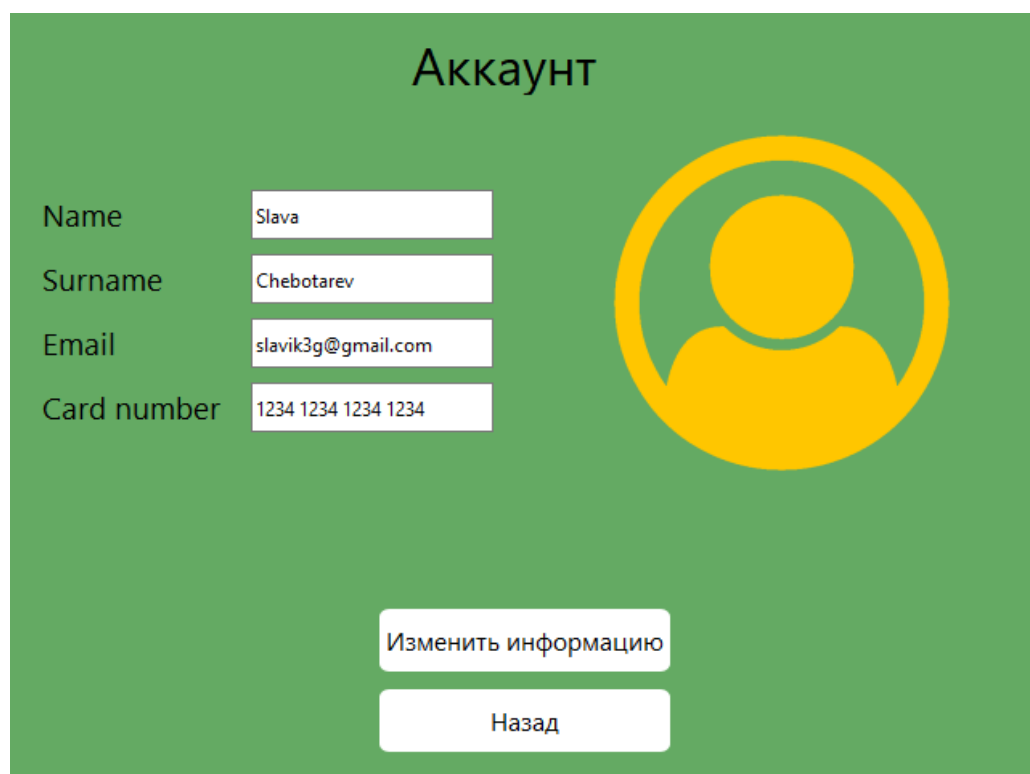


Рисунок 5.3

Нажав на кнопку “Взять в аренду” у вас на экране появится таймер со временем поездки и кнопкой которая позволяем закончить поездку (Рисунок 5.3).

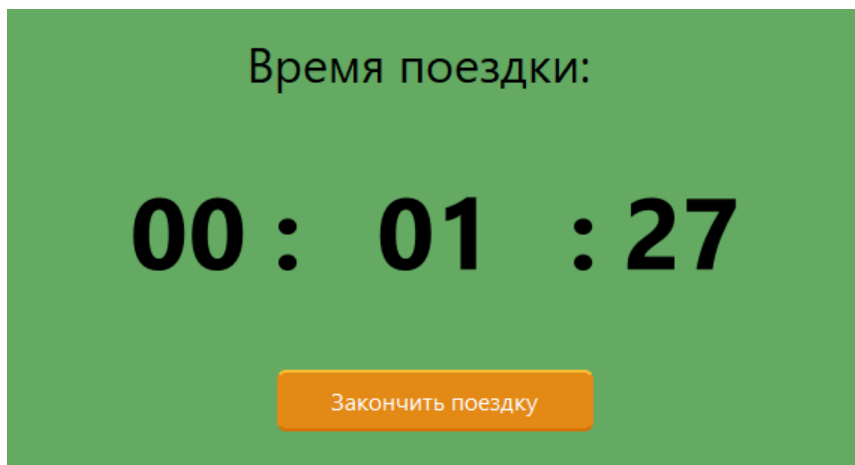


Рисунок 5.3

Нажав на кнопку “Управление” вас переместит на меню администратора, где вы сможете редактировать информацию о пользователях и транспорте, а также зарядить все электросамокаты и электровелосипеды (Рисунок 5.4).

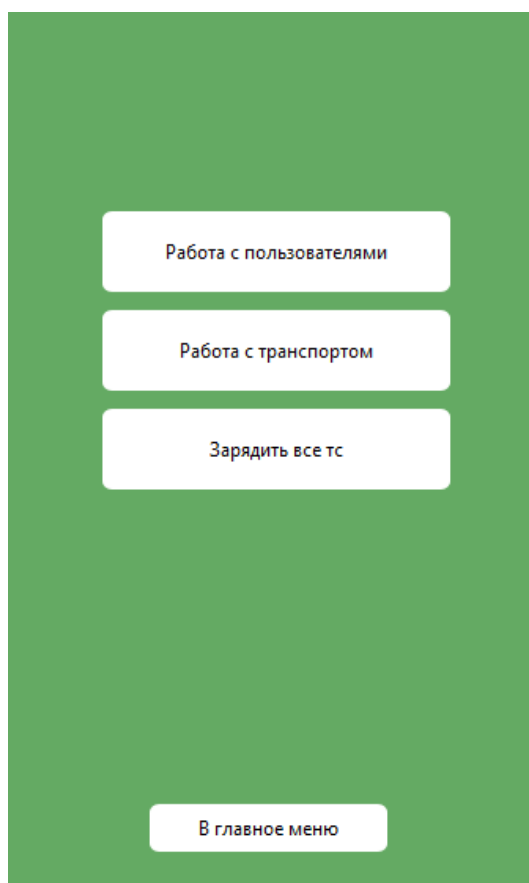


Рисунок 5.4

## СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

- [1] Qt Documentation – информационный ресурс для фреймворка Qt [Электронный ресурс]. – Режим доступа: <https://doc.qt.io/>
- [2] METANIT – информационный ресурс для разработчиков [Электронный ресурс]. – Режим доступа: <https://metanit.com/>
- [3] RAVESLI – информационный обучающие для фреймворка Qt [Электронный ресурс]. – Режим доступа: <https://ravesli.com/uroki-po-qt5/>
- [4] Шилд, Герберт. Полный справочник по C++, 4-е издание. : Пер. с англ. – Издательский дом “Вильямс”, 2010. – 800 с.

## **ЗАКЛЮЧЕНИЕ**

В результате выполнения курсовой работы было разработано удобное приложение по аренде, которое имеет весь базовый, а также дополнительный функционал. В ходе разработки данного программного средства были получены знания по программированию пользовательских интерфейсов в Qt — кроссплатформенная библиотека разработки GUI на C++.

Разработанное программное средство представляет собой законченный продукт, готовый к использованию. Однако при желании функционал программы можно расширить, добавив новые функции, например снятие денежных средств с карты, дисконтная программа, ежемесячная подписка.

**ПРИЛОЖЕНИЕ А**  
(обязательное)

Структурная схема.

**ПРИЛОЖЕНИЕ Б**  
(обязательное)

Диаграмма классов.



## **ПРИЛОЖЕНИЕ В**

**(обязательное)**

Схема алгоритма функции `void Timer::on_end_button_clicked()`

## **ПРИЛОЖЕНИЕ Г**

(обязательное)

Схема алгоритма функции

```
void MainWindow::on_sing_in_clicked()
```

**ПРИЛОЖЕНИЕ Е**  
(обязательное)

Код программы

### Файл mainwindow.h:

```
#include <QMainWindow>
#include "libraries.h"
#include "user.h"

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:

    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private slots:
    void on_exit_clicked();
    void onSingUp_clicked();
    void onSingIn_clicked();
    void on_checkBox_stateChanged(int arg1);
private:
    QMainWindow *rent;
    Ui::MainWindow *ui;
    QString login;
    QString password;
    QSqlDatabase data_base_rent;
};
```

### Файл mainwindow.cpp:

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "rent.h"

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    this->setFixedSize(640, 480);
    data_base_rent = QSqlDatabase::addDatabase("QMYSQL");
    data_base_rent.setHostName("localhost");
    data_base_rent.setDatabaseName("rent");
    data_base_rent.setUserName("root");
    data_base_rent.setPassword("slava2002slava");
```

```

data_base_rent.setPort(3306);
if(!data_base_rent.open()) {
    qDebug() << "Error";
    qDebug() << data_base_rent.lastError().text();
}
else {
    qDebug() << "Succes";
}
ui->password->setEchoMode(QLineEdit::Password);
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::on_exit_clicked()
{
    QMessageBox::StandardButton reply =
    QMessageBox::question(this, "Предупреждение", "Вы уверены что
хотите выйти", QMessageBox::Yes | QMessageBox::No);
    if (reply == QMessageBox::Yes){
        QApplication::quit();
    }
}

void MainWindow::on_sign_up_clicked()
{
    int flag = 1;
    login = ui->login->text();
    password = ui->password->text();
    if(login == "" || password == ""){
        QMessageBox::warning(this, "Ошибка", "Заполните поля!");
        return;
    }
    QSqlQuery query;
    query.exec("SELECT login FROM personal_data");
    if (!query.isActive() ){
        QMessageBox::warning(this, tr("Database Error"),
query.lastError().text());
    }
    while (query.next()) {
        if(login == query.value(0).toString()) {
            QMessageBox::warning(this, "Ошибка", "Такой
пользователь уже существует!");
            flag = 0;
            break;
        }
    }
    if(flag) {

```

```

        query.prepare("INSERT INTO personal_data (login, pass) "
                      "VALUES (:login, :pass)");
        query.bindValue(":login", login);
        query.bindValue(":pass", password);
        query.exec();
        QMessageBox::information(this, "Ура!", "Вы успешно
зарегистрировались!");
    }
}

void MainWindow::on_sign_in_clicked()
{
    int flag = 1;
    QString login = ui->login->text();
    QString password = ui->password->text();
    if(login == "" || password == ""){
        QMessageBox::warning(this, "Ошибка", "Заполните поля!");
        return;
    }
    QSqlQuery query;
    query.exec("SELECT login, pass, access_rights, name FROM
personal_data");
    if (!query.isActive()){
        QMessageBox::warning(this, tr("Database Error"),
query.lastError().text());
    }
    while (query.next()) {
        if(login == query.value(0).toString() && password ==
query.value(1).toString()) {
            if(query.value(2).toInt() == 0){
                QMessageBox::information(this, "Ошибка", "Ваш
аккаунт заблокирован в приложении проката!");
                return;
            }
            QMessageBox::information(this, "Поздравляем", "Вы
успешно авторизовались!");
            flag = 0;
            Human _human(query.value(0).toString(),
query.value(1).toString(),
query.value(2).toInt(),
query.value(3).toString());
            Rent window(nullptr, &_human);
            hide();
            window.setModal(true);
            window.exec();
            break;
        }
    }
    if(flag){
        QMessageBox::information(this, "Ошибка", "Такой
пользователь не найден!");
    }
}

```

```

void MainWindow::on_checkBox_stateChanged(int arg1)
{
    if(arg1 == 2){
        ui->password->setEchoMode(QLineEdit::Normal);
    }
    else{
        ui->password->setEchoMode(QLineEdit::Password);
    }
}

```

### Файл Human.h

```

#include "libraries.h"

class Human
{
protected:
    QString name;
    QString login;
    QString password;
    int access_rights; //права доступа
public:
    Human();
    Human(QString, QString, int);
    Human(QString, QString, int, QString);
    void set_name(QString);
    QString get_name();
    int get_access_rights();
    QString get_login();
    QString get_password();
};

```

### Файл Human.cpp

```

#include "human.h"

Human::Human()
{

}

Human::Human(QString log, QString pass, int rights) : login(log),
password(pass), access_rights(rights)
{

}

Human::Human(QString log, QString pass, int rights, QString n) :
name(n), login(log), password(pass), access_rights(rights)
{

```

```

}

void Human::set_name(QString n)
{
    name = n;
    return;
}

QString Human::get_name()
{
    return name;
}

int Human::get_access_rights()
{
    return access_rights;
}

QString Human::get_login()
{
    return login;
}

QString Human::get_password()
{
    return password;
}

```

### **Файл User.h:**

```

#include "libraries.h"
#include "human.h"

class User: public Human
{
    QString surname;
    QString email;
    QString cardnumber;
public:
    User();
    User(QString, QString, int);
    QString get_surname();
    QString get_email();
    QString get_cardnumber();
};

```



### Файл User.cpp:

```
#include "user.h"

User::User()
{

}

User::User(QString log, QString pass, int rights) : Human(log,
pass, rights)
{
    QSqlQuery query;
    query.exec("SELECT login, name, surname, cardnumber, email
FROM personal_data");
    if (!query.isActive()) qDebug() << "Database Error: " <<
query.lastError().text();
    while (query.next()) {
        if(log == query.value(0).toString()) {
            set_name(query.value(1).toString());
            surname = query.value(2).toString();
            cardnumber = query.value(3).toString();
            email = query.value(4).toString();
            break;
        }
    }
}

QString User::get_surname()
{
    return surname;
}

QString User::get_cardnumber()
{
    return cardnumber;
}

QString User::get_email()
{
    return email;
}
```

### Файл Admin.h:

```
#include "human.h"
#include "libraries.h"

class Admin: public Human
{
private:
    QString post;
public:
    Admin();
    Admin(QString, QString, int);
    QString get_post();
    void set_post(QString );
};
```

### Файл Admin.cpp:

```
Admin::Admin()
{

}

Admin::Admin(QString login_, QString password_, int acces_rights_)
: Human(login_, password_, acces_rights_)
{
    QSqlQuery query;
    query.prepare("SELECT  name,post  FROM  personal_data  WHERE
login=:l");
    query.bindValue(":l", login_);
    query.exec();
    if(query.next()) {
        set_name(query.value(0).toString());
        set_post(query.value(1).toString());
    }
}

void Admin::set_post(QString post_) {
    this->post = post_;
}

QString Admin::get_post() {
    return this->post;
}
```

### Файл rent.h:

```
#include <QDialog>
#include "libraries.h"
#include "user.h"
#include "admin.h"

namespace Ui {
class Rent;
}

class Rent : public QDialog
{
    Q_OBJECT

public:
    explicit Rent(QWidget *parent = nullptr, Human *_human =
    nullptr);
    ~Rent();

private slots:
    void on_account_clicked();
    void on_exit_clicked();
    void on_take_bike_clicked();
    void on_take_scooter_clicked();
    void on_admin_menu_button_clicked();

    void on_informarion_button_clicked();

private:
    Ui::Rent *ui;
    QString login;
    Human *person;
};
```

### Файл rent.cpp:

```
#include "rent.h"
#include "ui_rent.h"
#include "account.h"
#include "timer.h"
#include "admin_menu.h"
#include "information.h"
Rent::Rent(QWidget *parent, Human *_human) :
    QDialog(parent),
    ui(new Ui::Rent)
{
    ui->setupUi(this);
    this->setFixedSize(640, 480);
    QPixmap BikePix(":/resorsuce/img/bikeeeeeee.png");
    QPixmap ScooterPix(":/resorsuce/img/scooooter.png");
    QPixmap Logo(":/resorsuce/img/logon3.png");
```

```

        ui->logo->setScaledContents(true);
        ui->bike->setScaledContents(true);
        ui->scooter->setScaledContents(true);
        ui->scooter->setPixmap(ScooterPix);
        ui->bike->setPixmap(BikePix);
        ui->logo->setPixmap(Logo);
        person = _human;
        if(_human->get_access_rights() == 1){
            ui->admin_menu_button->setVisible(false);
        }
    }

Rent::~Rent()
{
    delete ui;
}

void Rent::on_account_clicked()
{
    Account accountForm(nullptr, person);
    accountForm.setModal(true);
    accountForm.exec();
}

void Rent::on_exit_clicked()
{
    QMessageBox::StandardButton reply =
    QMessageBox::question(this, "Предупреждение", "Вы уверены что
хотите выйти", QMessageBox::Yes | QMessageBox::No);
    if (reply == QMessageBox::Yes){
        this->close();
    }
}

void Rent::on_take_bike_clicked()
{
    if(person->get_name() == ""){
        QMessageBox::information(this, "Предупреждение",
"Заполните данные в профиле!");
        return;
    }
    QSqlQuery query;
    query.exec("SELECT id FROM vehicle WHERE vehicle_type = 1 AND
is_on_ride_status = 0 AND is_active_status = 1 LIMIT 1");
    query.first();
    if(query.value(0).toInt() == NULL){
        QMessageBox::information(this, "Ошибка", "Просим прощения
на данный момент все велосипеды заняты!");
        return;
    }
    int id = query.value(0).toInt();
    Timer timer(nullptr, &id);

```

```

        timer.setModal(true);
        timer.exec();
    }

void Rent::on_take_scooter_clicked()
{
    if(person->get_name() == ""){
        QMessageBox::information(this, "Предупреждение",
        "Заполните данные в профиле!");
        return;
    }
    QSqlQuery query;
    query.exec("SELECT id FROM vehicle WHERE vehicle_type = 2 AND
is_on_ride_status = 0 AND is_active_status = 1 LIMIT 1");
    query.first();
    if(query.value(0).toInt() == NULL){
        QMessageBox::information(this, "Ошибка", "Просим прощения
на данный момент все самокаты заняты!");
        return;
    }
    int id = query.value(0).toInt();
    Timer timer(nullptr, &id);
    timer.setModal(true);
    timer.exec();
}

void Rent::on_admin_menu_button_clicked()
{
    Admin_menu Menu(nullptr);
    Menu.setModal(true);
    Menu.exec();
}

void Rent::on_informarion_button_clicked()
{
    Information infowin(nullptr);
    infowin.setModal(true);
    infowin.exec();
}

```