**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ   
УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информатика»

Специальность 1-40 04 01 «Информатика и технологии программирования»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

по дисциплине «Объектно-ориентированное программирование»

на тему: **«Автоматизация службы транспортных перевозок»**

Исполнитель: студент гр. ИП-22

Прокопенко А.Р.

Руководитель: преподаватель

Олизарович Я.М.

Дата проверки: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата допуска к защите: ­\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата защиты: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подписи членов комиссии

по защите курсовой работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Гомель 2021

**СОДЕРЖАНИЕ**

|  |  |  |  |
| --- | --- | --- | --- |
| Введение | | | 3 |
| 1. Аналитический обзор | | | 4 |
|  | 1.1 Описание объекта | | 4 |
|  | 1.2 Обзор существующих методов | | 5 |
|  | 1.3 Требования к проектируемому программному обеспечению | | 5 |
|  | 1.4 Используемые средства…………………………………………..... | | 6 |
| 2. Архитектура программного обеспечения | | | 7 |
|  | 2.1 Информационно-логическая модель информационной системы.. | | 7 |
|  | 2.2 Физическая модель базы данных | | 15 |
|  | 2.3 Пользовательские роли…………………………………………………………. | | 16 |
| 3.Структура приложения | | | 18 |
|  | 3.1 Описание общей структуры приложения | | 18 |
|  | 3.2 Взаимодействие с базой данных | | 18 |
|  | 3.3 Реализация графического интерфейса | | 18 |
|  | 3.4 Реализация графической составляющей приложения | | 19 |
|  | 3.5 Описание интерфейса | | 19 |
| 4. Тестирование | | | 26 |
|  | | 4.1 Верификация | 26 |
|  | | 4.2 Обработка исключительных ситуаций | 26 |
| Заключение | | | 29 |
| Список использованных источников | | | 30 |
| Приложение А – Листинг программы | | | 31 |

**ВВЕДЕНИЕ**

В основе решения многих задач лежит обработка информации. Для упрощения процесса обработки информации создаются информационные системы.

Информационная система – это программный комплекс, функции которого состоят в поддержке надежного хранения информации в памяти компьютера, выполнении специфических для данного приложения преобразований информации и вычислений, предоставления для пользователя удобного и легко осваиваемого интерфейса. Обычно объемы информации, с которыми приходится иметь дело таким системам, достаточно велики, а сама информация имеет достаточно сложную структуру. Классическими примерами информационных систем являются банковские и бухгалтерские системы, системы авиационных или железнодорожных билетов, системы налоговой службы, статистические системы, системы резервирования мест в отеле и так далее.

В зависимости от специфики предметной области информационные системы могут очень сильно отличаться друг от друга по своим функциям, архитектуре, реализации, но все-таки можно выделить некоторые общие свойства характерные для информационных систем:

* информационные системы предназначены для сбора, хранения и обработки информации. Поэтому в основе любой из них лежит среда хранения и доступа к данным;
* информационные системы ориентируются на конечного пользователя, не обладающего высокой квалификацией в области применения вычислительной техники. Поэтому клиентские приложения должны обладать простым, удобным интерфейсом, который предоставляет возможность конечному пользователю выполнять все необходимые для работы функции, но в то же время не дает ему выполнять лишние действия.

Таким образом, при разработке информационной системы приходится решать следующие задачи: обработку базы данных, предназначенной для хранения информации и задачу разработки графического интерфейса пользователя.

**1 АНАЛИТИЧЕСКИЙ ОБЗОР**

* 1. **Описание объекта**

Объектом данной курсовой работы является служба транспортных перевозок.

Конкуренция среди компаний, предоставляющих услуги транспортных перевозок, растет год от года. Здесь необходима грамотная работа с транспортной логистикой. Любая компания, у которой есть потребность регулярного транспортирования, сталкивается со сложностями планирования работ и финансового учета, а значит и с трудностями документооборота. Другими словами, если логистический процесс компании не оптимизирован – она терпит убытки.

Наиболее распространенные проблемы логистики легко перечислить:

* **Нерациональное использование транспортных средств**. Порожний пробег и простой транспортных средств – частая проблема на предприятиях с собственным автопарком.
* **Сложности и потери в коммуникациях**. Как правило, без автоматизации перегружается диспетчерский отдел, и даже в этом случае может быть сложно удовлетворить все входящие запросы. Если информация не хранится централизованно, в одной системе, оператору приходится выполнять множество лишних действий чтобы ее получить и передать дальше.
* **Негативное влияние человеческого фактора**. Ввод и передача в ручном режиме отнимает много время и порождает ошибки. Также, если процессы не автоматизированы, компании приходится прибегать к интуитивным решениям и ручным вычислениям – как правило, неточным.

Все это неизбежно ведет к снижению продуктивности, а значит – оборота и доходов. Каков здесь выход? Разумеется, автоматизация. Грамотное внедрение специализированных продуктов позволяет решить все эти проблемы.

Итогом проектирования будет кроссплатформенное приложение с подключенным источником, в котором будет храниться информация о пользователях, их ролях, спектре услуг, предоставляемых организацией, а также о заказах, ресурсах организации и городах, в пределах которых предоставляются услуги.

На основе накопленных данных будет производиться автоматическое формирование отчетов, включающих построение графических отчетов по деятельности организации.

**1.2 Обзор существующих методов**

Основная задача разрабатываемого приложения – автоматизация службы транспортных перевозок.

В ходе изучения аналогов было просмотрено множество различных приложений, таких как ТРАНСПОРТИКА, АВТОДОЗОР, АТИ, Della. У всех этих приложений основными преимуществами являются: удобный и интуитивно понятный интерфейс, учет клиентов, возможность просмотра истории заказов, наличие приложений для мобильных платформ.

Основными преимуществами разрабатываемого приложения являются: его совместимость с одной из популярных операционных систем: Windows, простота использования, отсутствие подписок в любом проявлении, широкий спектр возможностей и большое пространство для будущих обновлений. Также приложение будет иметь современный и понятный любому пользователю интерфейс.

**1.3 Требования к проектируемому программному обеспечению**

В качестве предметной области рассматривается разработка автоматизированной системы для оформления заказа. Проектируемая программа должна быть понятна пользователю и проста в обращении.

Пользователь с помощью данного приложения сможет, не выходя из дома, сделать заказ, заполнив соответствующую форму своими данными, просмотреть его статус, а также произвести оплату.

Автоматизированная система должна вести учет следующих видов справочников: города, виды услуг, виды транспорта, парка автотранспорта. В функции разрабатываемого приложения входят: ведение справочников, прейскурант, прием заказов, учет оплаты и отмены заказов.

В обязательные функции проектируемого программного обеспечения входит еще ведение статистики, к которой относятся: графическое отображение прибыли за все месяцы, диаграмма частоты заказов различных услуг, а также диаграмма, отражающая частоту заказов по городам. Кроме того, программа должна разграничивать пользователей на четыре роли: клиент, оператор, курьер и директор.

Подводя итоги раздела, можно с уверенностью сказать, что вышеописанные требования к проектируемому программному обеспечению должны максимально способствовать развитию бизнеса и обслуживания, а также сводить к минимуму временные и денежные растраты клиентов и сотрудников.

**1.4 Используемые средства**

Для решения поставленной задачи в качестве СУБД используется *MySQL*. Данная СУБД обеспечивает поддержку баз данных очень большого объёма и обработку сложных запросов, а также имеет эффективные алгоритмы для работы с памятью и автоматизированным контролем размера файлов баз данных [6].

Язык, на котором написано приложение – *Python*. *Python* – мультипарадигмальный высокоуровневый язык программирования общего назначения с динамической строгой типизацией и автоматическим управлением памятью, ориентированный на повышение производительности разработчика, читаемости кода и его качества, а также на обеспечение переносимости написанных на нем программ. Язык является полностью объектно-ориентированным. Здесь имеется большое число как стандартных, так и сторонних библиотек, что, несомненно, ускоряет процесс разработки приложений.

Для обеспечения кроссплатформенности был задействован фреймворк *Kivy*, работающий на Windows, Mac, Linux, Raspberry Pi, Android. *Kivy* не задействует нативные элементы управления, или виджеты. Все его виджеты настраиваются. Это значит, что приложения *Kivy* будут выглядеть одинаково на всех платформах. Помимо этого, он поддерживает ускорение GPU своей графики.

**2 АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

**2.1 Информационно-логическая модель информационной системы**

Перед проектированием физической базы данных необходимо построить информационно-логическую модель. Данная модель представляет собой совокупность сущностей и связей между ними [3]. Каждая сущность имеет набор атрибутов, описывающих её. Для получения информационно-логической модели необходимо:

* выявить и смоделировать сущности исходя из исходных данных;
* выявить и смоделировать связи между сущностями;
* выявить и смоделировать атрибуты;
* указать уникальный идентификатор для каждой сущности;
* провести нормализацию.

В результате изучения исходных данных выделены следующие сущности:

* «Статус»;
* «Этап»;
* «Вид транспорта»;
* «Город»;
* «Роль»;
* «Пользователь»;
* «Парк автотранспорта»;
* «Услуга»;
* «Клиент»;
* «Город клиента»;
* «Сотрудник»;
* «Роль пользователя»;
* «Город парка автотранспорта»;
* «Транспорт»;
* «Заказ»;
* «Транспорт заказа»;
* «Услуга заказа»;
* «Начальный город услуги заказа»;
* «Конечный город услуги заказа».

Сущность «Статус» содержит атрибуты: «Наименование статуса», «Код статуса», «Описание». Более подробное описание каждого атрибута приведено в таблице 2.1.

Таблица 2.1 – Описание атрибутов таблицы «Статус»

|  |  |  |
| --- | --- | --- |
| Атрибуты | Описание домена | Тип данных |
| Код статуса | Уникальный инкрементируемый идентификатор. Является первичным ключом | Целочисленный |
| Наименование статуса | Содержит наименование статуса | Строка |
| Описание | Содержит информацию о статусе | Строка |

Сущность «Этап» содержит атрибуты: «Код этапа», «Наименование этапа», «Описание». Описание атрибутов более подробно приведено в таблице 2.2.

Таблица 2.2 – Описание атрибутов таблицы «Этап»

|  |  |  |
| --- | --- | --- |
| Атрибуты | Описание домена | Тип данных |
| Код этапа | Уникальный инкрементируемый идентификатор. Является первичным ключом | Целочисленный |
| Наименование этапа | Содержит наименование этапа | Строка |
| Описание | Содержит информацию об этапе | Строка |

Сущность «Вид транспорта» содержит атрибуты: «Код вида транспорта», «Наименование вида транспорта», «Описание», «Грузоподъемность», «Объем». Подробное описание атрибутов сущности приведено в таблице 2.3.

Таблица 2.3 – Описание атрибутов таблицы «Вид транспорта»

|  |  |  |
| --- | --- | --- |
| Атрибуты | Описание атрибута | Тип данных |
| Код вида транспорта | Уникальный инкрементируемый идентификатор. Является первичным ключом | Целочисленный |
| Наименование вида транспорта | Содержит наименование вида транспорта | Строка |
| Описание | Содержит информацию о виде транспорта | Строка |
| Грузоподъемность | Отражает грузоподъемность | Вещественный |
| Объем | Отражает объем | Вещественный |

Сущность «Город» содержит атрибуты: «Код города», «Наименование города». Подробное описание атрибутов сущности приведено в таблице 2.4.

Таблица 2.4 – Описание атрибутов таблицы «Город»

|  |  |  |
| --- | --- | --- |
| Атрибуты | Описание атрибута | Тип данных |
| Код города | Уникальный инкрементируемый идентификатор. Является первичным ключом | Целочисленный |
| Наименование города | Содержит наименование города | Строка |

Сущность «Роль» содержит атрибуты: «Код роли», «Наименование роли», «Описание». Описание атрибутов более подробно приведено в таблице 2.5.

Таблица 2.5 – Описание атрибутов таблицы «Роль»

|  |  |  |
| --- | --- | --- |
| Атрибуты | Описание домена | Тип данных |
| Код роли | Уникальный инкрементируемый идентификатор. Является первичным ключом. | Целочисленный |
| Наименование роли | Содержит наименование роль | Строка |
| Описание | Содержит информацию о роли | Строка |

Сущность «Пользователь» содержит атрибуты: «Код пользователя», «Логин», «Пароль», «Почта», «Телефон». Описание атрибутов более подробно приведено в таблице 2.6.

Таблица 2.6 – Описание атрибутов таблицы «Пользователь»

|  |  |  |
| --- | --- | --- |
| Атрибуты | Описание домена | Тип данных |
| Код пользователя | Уникальный инкрементируемый идентификатор. Является первичным ключом. | Целочисленный |
| Логин | Содержит логин для входа в систему | Строка |
| Пароль | Содержит пароль для входа в систему | Строка |
| Почта | Содержит почту пользователя | Строка |
| Телефон | Содержит телефон пользователя | Строка |

Сущность «Парк автотранспорта» содержит атрибуты: «Наименование», «Код парка автотранспорта», «Описание», «Адрес», «Площадь», «Количество звезд». Описание атрибутов более подробно приведено в таблице 2.7.

Таблица 2.7 – Описание атрибутов таблицы «Парк автотранспорта»

|  |  |  |
| --- | --- | --- |
| Атрибуты | Описание домена | Тип данных |
| Код парка автотранспорта | Уникальный инкрементируемый идентификатор. Является первичным ключом | Целочисленный |
| Наименование | Содержит наименование парка автотранспорта | Строка |
| Описание | Содержит информацию о парке автотранспорта | Строка |
| Адрес | Содержит адрес парка автотранспорта | Строка |
| Площадь | Содержит значение площади парка автотранспорта | Вещественный |
| Количество звезд | Относит парк автотранспорта к определенному классу по уровню разработанности | Целочисленный |

Сущность «Услуга» содержит атрибуты: «Код услуги», «Наименование услуги», «Описание», «Стоимость за вес», «Стоимость за расстояние». Описание атрибутов более подробно приведено в таблице 2.8.

Таблица 2.8 – Описание атрибутов таблицы «Услуга»

|  |  |  |
| --- | --- | --- |
| Атрибуты | Описание домена | Тип данных |
| Код услуги | Уникальный инкрементируемый идентификатор. Является первичным ключом. | Целочисленный |
| Наименование услуги | Содержит наименование услуги | Строка |
| Описание | Содержит информацию об услуге | Строка |
| Стоимость за вес | Содержит стоимость за единицу веса | Вещественный |
| Стоимость за расстояние | Содержит стоимость за единицу расстояния | Вещественный |

Сущность «Клиент» содержит атрибуты: «Код клиента», «Код пользователя», «Страна», «Улица», «Дом», «Квартира», «Фамилия», «Имя», «Отчество». Более подробное описание каждого атрибута приведено в таблице 2.9.

Таблица 2.9 – Описание атрибутов таблицы «Клиент»

|  |  |  |
| --- | --- | --- |
| Атрибуты | Описание домена | Тип данных |
| Код клиента | Уникальный инкрементируемый идентификатор. Является первичным ключом | Целочисленный |
| Код пользователя | Является внешним ключом | Целочисленный |
| Страна | Содержит страну проживания клиента | Строка |
| Улица | Содержит улицу клиента | Строка |
| Дом | Содержит номер дома клиента | Целочисленный |
| Квартира | Содержит номер квартиры клиента | Целочисленный |
| Фамилия | Содержит фамилию клиента | Строка |
| Имя | Содержит имя клиента | Строка |
| Отчество | Содержит отчество клиента | Строка |

Сущность «Город клиента» содержит атрибуты: «Код клиента», «Код города». Описание атрибутов более подробно приведено в таблице 2.10.

Таблица 2.10 – Описание атрибутов таблицы «Город клиента»

|  |  |  |
| --- | --- | --- |
| Атрибуты | Описание домена | Тип данных |
| Код клиента | Является внешним ключом | Целочисленный |
| Код города | Является внешним ключом | Целочисленный |

Сущность «Сотрудник» содержит атрибуты: «Код сотрудника», «Код пользователя», «Паспортные данные», «Оклад», «Требования», «Обязанности», «Статус». Подробное описание атрибутов сущности приведено в таблице 2.11.

Таблица 2.11 – Описание атрибутов таблицы «Сотрудник»

|  |  |  |
| --- | --- | --- |
| Атрибуты | Описание атрибута | Тип данных |
| Код сотрудника | Уникальный инкрементируемый идентификатор. Является первичным ключом | Целочисленный |
| Код пользователя | Является внешним ключом | Целочисленный |
| Паспортные данные | Содержит паспортные данные сотрудника | Строка |
| Оклад | Содержит значение оклада сотрудника | Вещественный |
| Требования | Содержит требования, предъявляемые к сотруднику | Строка |
| Обязанности | Содержит обязанности сотрудника | Строка |
| Статус | Содержит статус сотрудника | Строка |

Сущность «Роль пользователя» содержит атрибуты: «Код пользователя», «Код роли». Подробное описание атрибутов сущности приведено в таблице 2.12.

Таблица 2.12 – Описание атрибутов таблицы «Роль пользователя»

|  |  |  |
| --- | --- | --- |
| Атрибуты | Описание атрибута | Тип данных |
| Код пользователя | Является внешним ключом | Целочисленный |
| Код роли | Является внешним ключом | Целочисленный |

Сущность «Город парка автотранспорта» содержит атрибуты: «Код парка автотранспорта», «Код города». Описание атрибутов более подробно приведено в таблице 2.13.

Таблица 2.13 – Описание атрибутов таблицы «Город парка автотранспорта»

|  |  |  |
| --- | --- | --- |
| Атрибуты | Описание домена | Тип данных |
| Код парка автотранспорта | Является внешним ключом | Целочисленный |
| Код города | Является внешним ключом | Целочисленный |

Сущность «Транспорт» содержит атрибуты: «Код транспорта», «Код вида транспорта», «Номер транспорта», «Код парка автотранспорта». Описание атрибутов более подробно приведено в таблице 2.14.

Таблица 2.14 – Описание атрибутов таблицы «Транспорт»

|  |  |  |
| --- | --- | --- |
| Атрибуты | Описание домена | Тип данных |
| Код транспорта | Уникальный инкрементируемый идентификатор. Является первичным ключом. | Целочисленный |
| Код вида транспорта | Является внешним ключом | Целочисленный |
| Номер транспорта | Содержит номер транспорта | Строка |
| Код парка автотранспорта | Является внешним ключом | Целочисленный |

Сущность «Заказ» содержит атрибуты: «Код заказа», «Дата совершения», «Дата выполнения», «Код клиента», «Код оператора», «Код курьера». Описание атрибутов более подробно приведено в таблице 2.15.

Таблица 2.15 – Описание атрибутов таблицы «Заказ»

|  |  |  |
| --- | --- | --- |
| Атрибуты | Описание домена | Тип данных |
| Код заказа | Уникальный инкрементируемый идентификатор. Является первичным ключом | Целочисленный |
| Дата совершения | Содержит дату совершения заказа | Дата |
| Дата выполнения | Содержит дату выполнения заказа | Дата |
| Код клиента | Является внешним ключом | Целочисленный |
| Код оператора | Является внешним ключом | Целочисленный |
| Код курьера | Является внешним ключом | Целочисленный |

Сущность «Транспорт заказа» содержит атрибуты: «Код транспорта заказа», «Код транспорта», «Код заказа». Описание атрибутов более подробно приведено в таблице 2.16.

Таблица 2.16 – Описание атрибутов таблицы «Транспорт заказа»

|  |  |  |
| --- | --- | --- |
| Атрибуты | Описание домена | Тип данных |
| Код транспорта заказа | Уникальный инкрементируемый идентификатор. Является первичным ключом | Целочисленный |
| Код транспорта | Является внешним ключом | Целочисленный |
| Код заказа | Является внешним ключом | Целочисленный |

Сущность «Услуга заказа» содержит атрибуты: «Код услуги заказа», «Код услуги», «Код заказа», «Количество, вес», «Количество, расстояние», «Пункт назначения», «Пункт отправления», «Общая стоимость». Описание атрибутов более подробно приведено в таблице 2.17.

Таблица 2.17 – Описание атрибутов таблицы «Услуга заказа»

|  |  |  |
| --- | --- | --- |
| Атрибуты | Описание домена | Тип данных |
| Код услуги заказа | Уникальный инкрементируемый идентификатор. Является первичным ключом | Целочисленный |
| Код услуги | Является внешним ключом | Целочисленный |
| Код заказа | Является внешним ключом | Целочисленный |
| Количество, вес | Содержит число условных единиц веса | Вещественный |
| Количество, расстояние | Содержит число условных единиц расстояния | Вещественный |
| Пункт назначения | Содержит адрес пункта назначения | Строка |
| Пункт отправления | Содержит адрес пункта отправления | Строка |
| Общая стоимость | Содержит общую стоимость услуги заказа | Вещественный |

Сущность «Начальный город услуги заказа» содержит атрибуты: «Код начального города услуги заказа», «Код города», «Код услуги заказа». Описание атрибутов более подробно приведено в таблице 2.18.

Таблица 2.18 – Описание атрибутов таблицы «Начальный город услуги заказа»

|  |  |  |
| --- | --- | --- |
| Атрибуты | Описание домена | Тип данных |
| Код начального города услуги заказа | Уникальный инкрементируемый идентификатор. Является первичным ключом | Целочисленный |
| Код города | Является внешним ключом | Целочисленный |
| Код услуги заказа | Является внешним ключом | Целочисленный |

Сущность «Конечный город услуги заказа» содержит атрибуты: «Код конечного города услуги заказа», «Код города», «Код услуги заказа». Описание атрибутов более подробно приведено в таблице 2.19.

Таблица 2.19 – Описание атрибутов таблицы «Начальный город услуги заказа»

|  |  |  |
| --- | --- | --- |
| Атрибуты | Описание домена | Тип данных |
| Код конечного города услуги заказа | Уникальный инкрементируемый идентификатор. Является первичным ключом | Целочисленный |
| Код города | Является внешним ключом | Целочисленный |
| Код услуги заказа | Является внешним ключом | Целочисленный |

**2.2 Физическая модель базы данных**

После создания информационно-логической модели выполняется этап проектирования физической базы данных [6]. Для этого информационно-логическая модель переводится в набор операторов *SQL*. Поскольку *MySQL* является реляционной базой данных, то преобразование логической модели в физическую базу данных является относительно несложной операцией. Для данного процесса преобразования существуют следующие правила:

* сущности становятся таблицами в физической базе данных;
* атрибуты становятся столбцами в физической базе данных. Также для каждого столбца необходимо определить подходящий тип данных;
* уникальные идентификаторы становятся столбцами, не допускающими значение *NULL*, т.е. первичными ключами. Также значение идентификатора делается автоинкрементным для обеспечения уникальности;
* все отношения моделируются в виде внешних ключей.

Для формирования физической модели базы данных использовались средства СУБД *MySQL*. Диаграмма созданной базы данных представлена на рисунке 2.1.

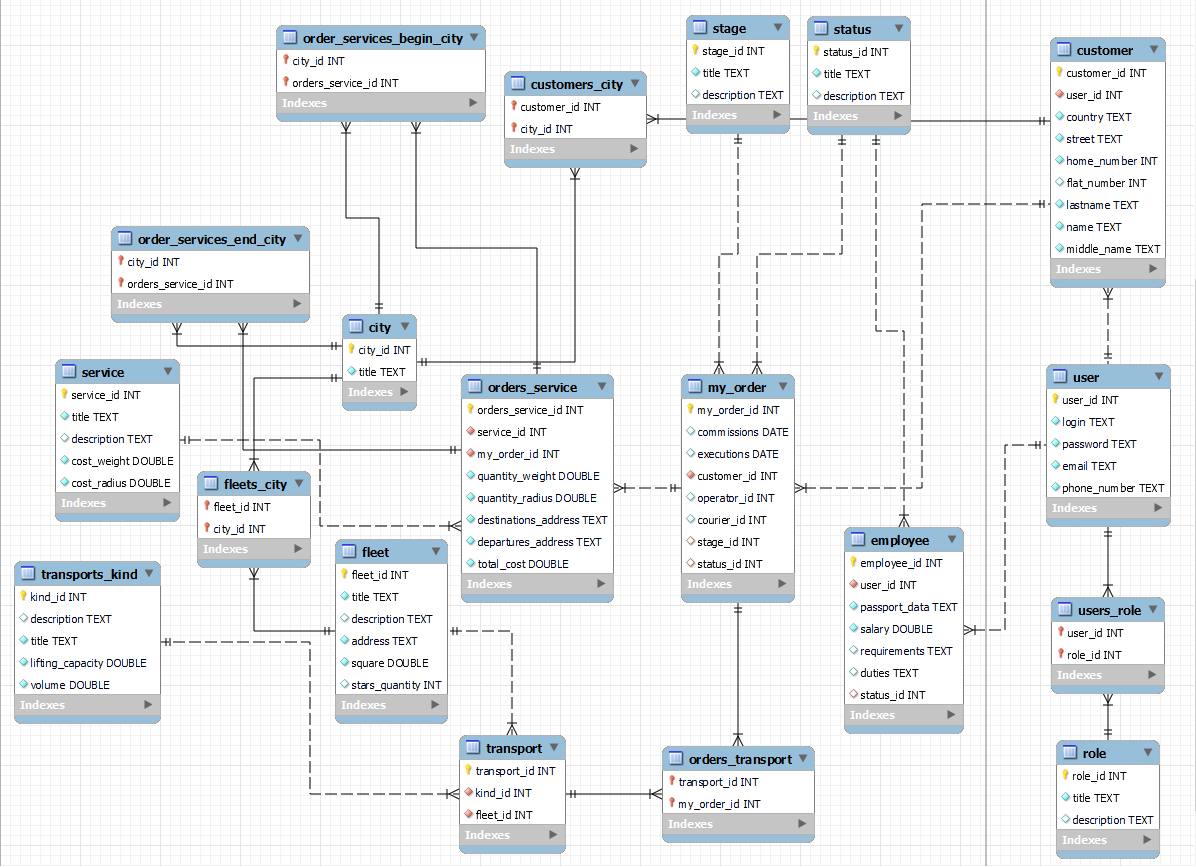


Рисунок 2.1 – Схема базы данных

**2.3 Пользовательские роли**

В данной информационной системе были реализованы следующие роли пользователей:

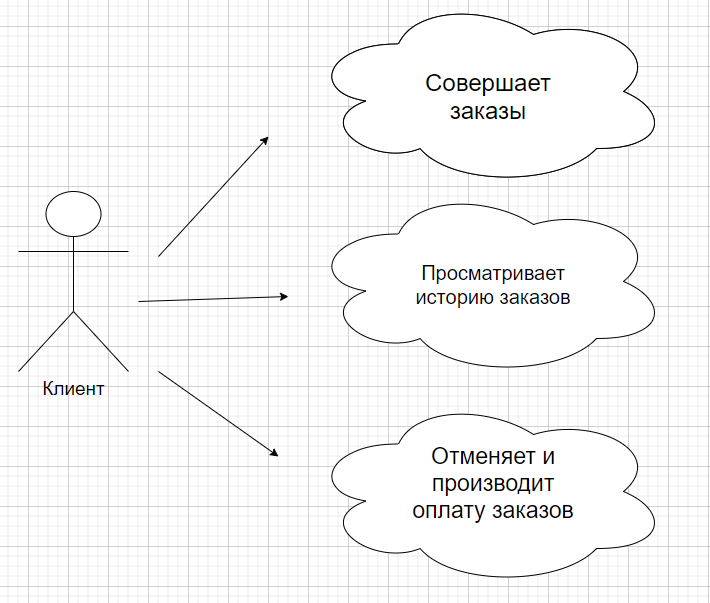


Рисунок 2.2 – Схема функций клиента

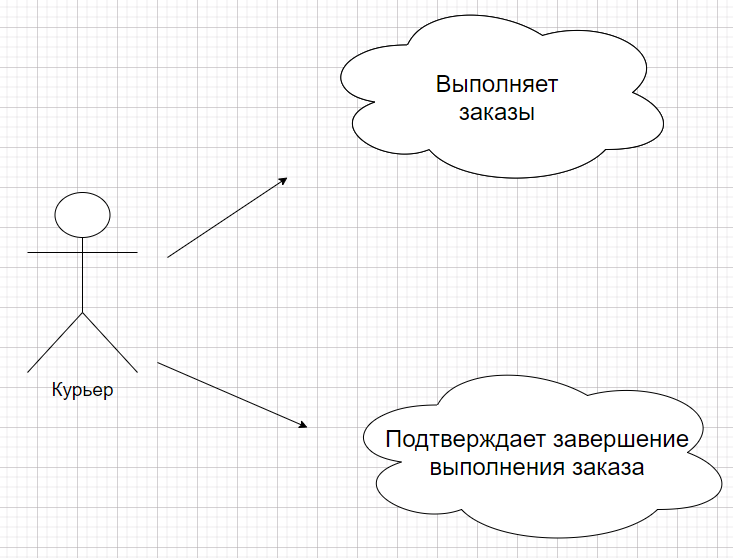


Рисунок 2.3 – Схема функций курьера

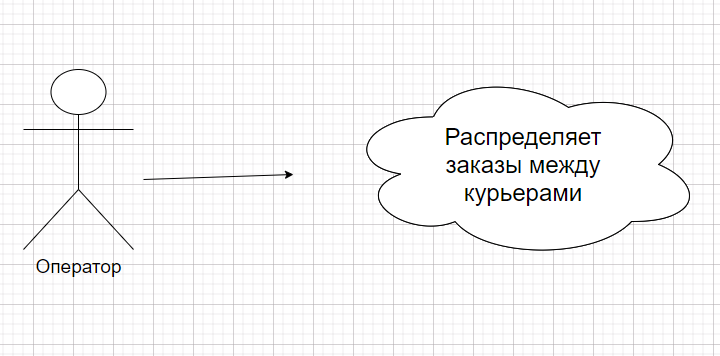


Рисунок 2.4 – Схема функций оператора

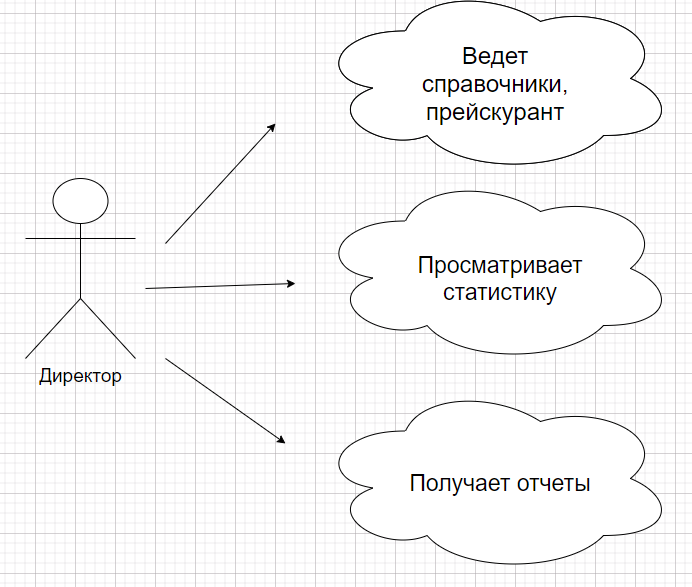


Рисунок 2.5 – Схема функций директора

**3 СТРУКТУРА ПРИЛОЖЕНИЯ**

## **3.1 Описание общей структуры приложения**

Разработанная информационная система имеет клиент-серверную архитектуру, в состав которой входят следующие компоненты:

* сервер хранения данных, на котором установлена СУБД;
* клиент, который формирует запросы к серверудля получения необходимых данных.

Для полноценного функционирования всей системы должны быть соблюдены минимальные требования к каждому её компоненту. Несоблюдение некоторых требований может привести к некорректной работе или полному отказу всей системы.

Приложение включает в себя:

* классы, описывающие взаимодействие приложения с базой данных;
* классы, реализующие графический интерфейс пользователя;
* классы, реализующие логику приложения.

## **3.2 Взаимодействие с базой данных**

Для работы с базой данных в приложении можно выделить несколько классов. Пусть первый класс содержит методы для отправления запросов к базе данных. Второй класс может в случае ошибки подключения или выполнения запроса генерировать диалоговое окно для извещения пользователя. Тогда третий класс (для него первый и второй классы являются вложенными) будет содержать словарь с запросами на языке SQL и методы, использующие возможности первых двух классов, для отправления запросов.

В данном приложении в роли первого класса (работа с базой данных) выступает класс «*MySQL*», в роли второго (генерирование диалоговых окон) – класс «*Notification*», и, наконец, в роли третьего класса – «*WithDB*».

Помимо выполнения запросов, необходимо избежать «обезличивания» пользователей. С этой целью был разработан класс *User*, содержащий необходимые для определения пользователей поля.

## **3.3 Реализация графического интерфейса**

Фреймворк *Kivy* имеет собственный язык разметки, позволяющий с относительно небольшими временными затратами создавать различные комбинации из элементов управления.

Файлы, содержащие код на данном языке разметки, имеют расширение «.*kv*»*.* Каждый такой файл привязывается к классу-наследнику от «*MDScreen*», образуя отдельный «скрин». Также имеется объект класса «*MDManagerScreen*», который, в свою очередь, связывает «скрины» и позволяет осуществлять переходы между ними.

Файлы, реализующие графический интерфейс пользователя: «*My.kv*», «*LogIn.kv*», «*SignUp.kv*», «*GetPassword.kv*», «*Director.kv*», «*Customer.kv*», «*Courier.kv*», «*Operator.kv*», «*HandBooks.kv*», «*Reports.kv*», «*CustomerOrder.kv*», «*MyOrders.kv*».

Стоит отметить тот факт, что графический интерфейс можно реализовать без использования языка разметки *Kivy*. Примером тому может служить класс «*ChooseRole*», описываемый в файле «*ChooseRole.py*».

## **3.4 Реализация логической составляющей приложения**

В данном приложении логическая составляющая заключается, в основном, в файлах с форматами «.*py*». Она представлена методами классов, на которые ссылаются некоторые пользовательские элементы управления при наступлении ключевых событий. Как и в случае с графическим интерфейсом, каждый класс описывает логическую составляющую сугубо для своего «скрина».

В основном, реакции на события подразделяются на два вида:

1. Переход от одного «скрина» к другому.
2. Изменение структурной составляющей какого-либо «скрина». Сюда можно отнести добавление/удаление элементов управления, изменение их свойств.

Реакции на события могут быть комбинированными, т.е. сочетать структурное изменение с переходом.

**3.5 Описание интерфейса**

Графический интерфейс пользователя является частью пользовательского интерфейса и определяет взаимодействие с пользователем на уровне визуализированной информации.

Интерфейс программы моделируемой предметной области был разработан в достаточно простой, и в то же время, понятной пользователю форме.

Рассмотрим интерфейс каждого окна приложения:

*Окно ошибки подключения*:

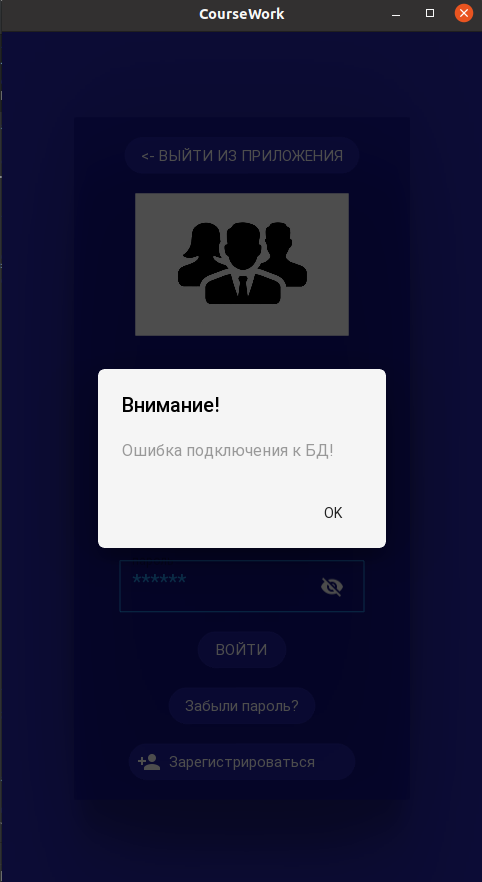


Рисунок 3.1 – Окно ошибки подключения

*Окно авторизации*:

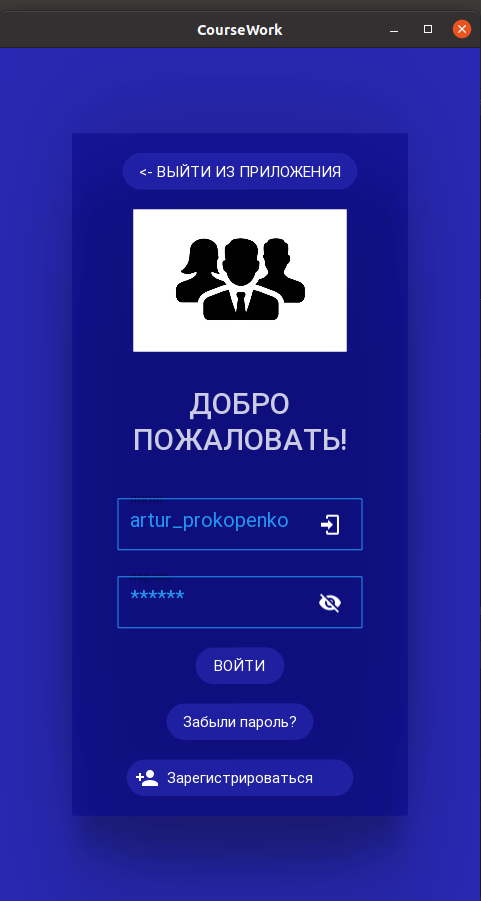


Рисунок 3.2 – Окно авторизации

Открывается сразу после запуска приложения. Предлагает авторизоваться.

*Окно выбора роли*:

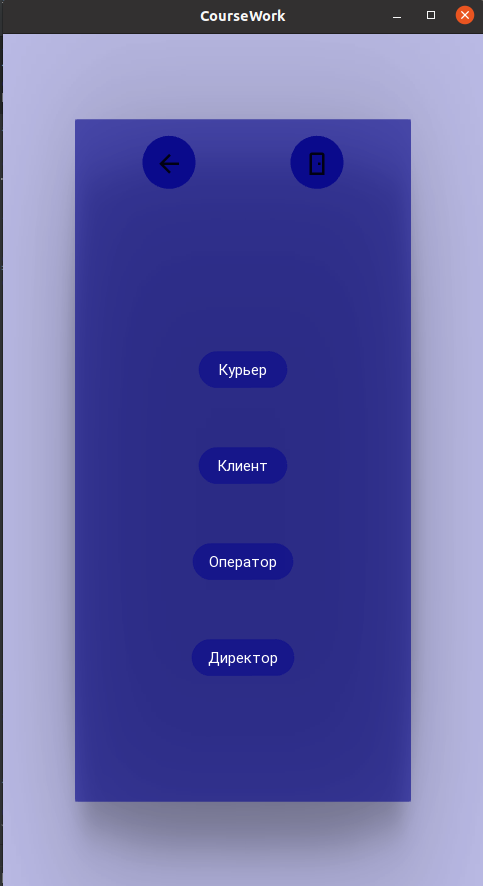


Рисунок 3.3 – Окно выбора роли

Это окно открывается после авторизации в том случае, если у пользователя имеется более одной роли. При этом приложение следит чтобы не создавалось более одного экземпляра окна одной роли.

*Окно регистрации*:

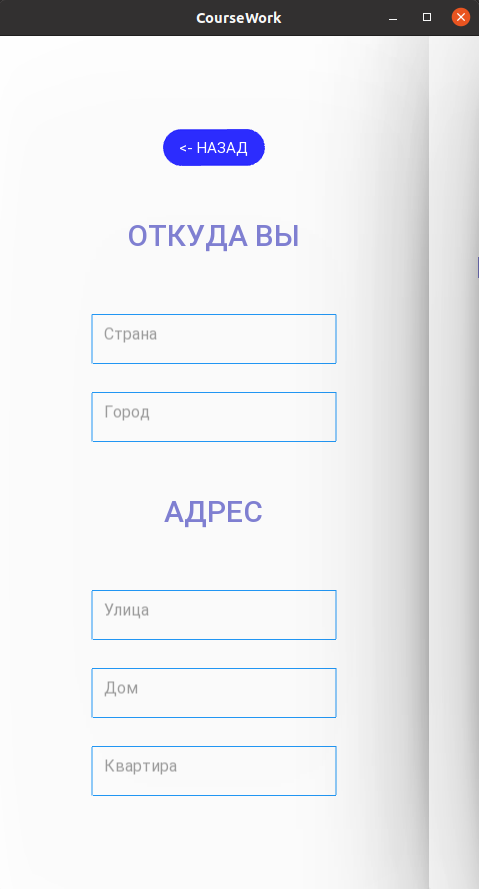
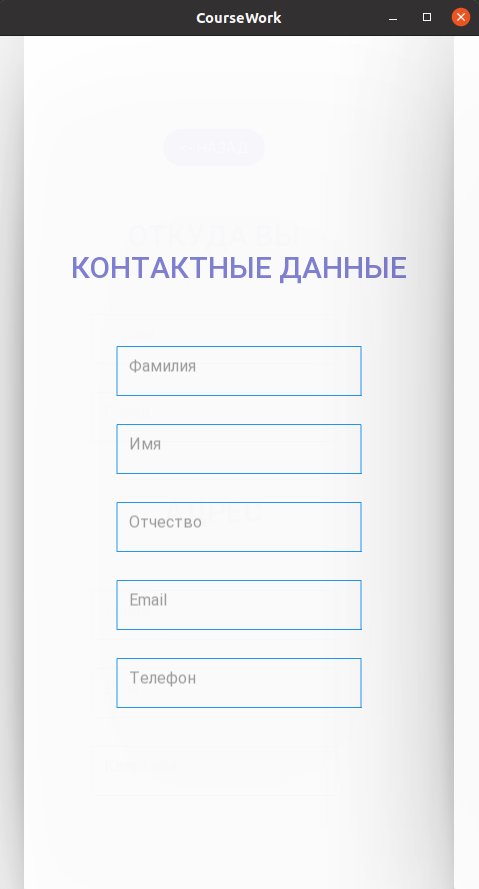
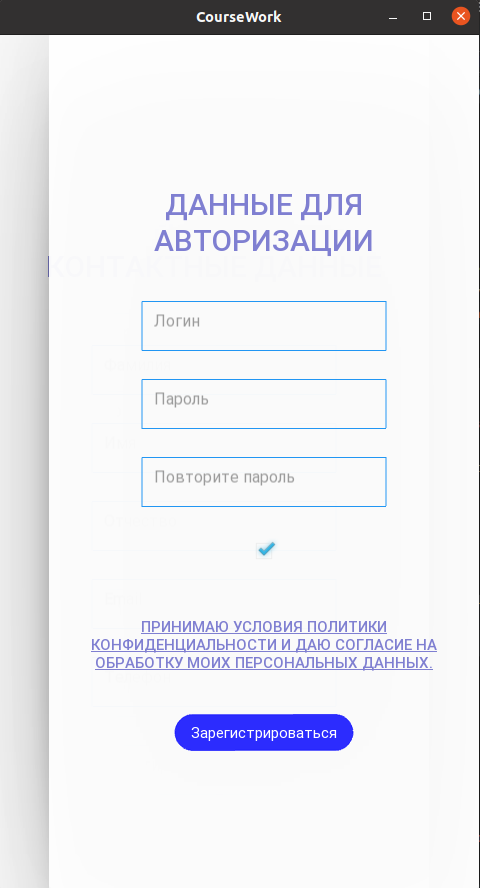
  

Рисунок 3.4 – Окно регистрации

Это окно появляется после того, как пользователь нажмет на кнопку «Зарегистрироваться» в окне «Авторизация». После ввода надежного пароля (он должен включать, как минимум, буквы 2-х регистров и хотя бы одну цифру), его подтверждения, логина, который ранее не был использован при регистрации в системе, а также при заполнении всех оставшихся полей, регистрация будет проведена успешно, в противном случае будет сгенерировано уведомление, где будет указана причина, по которой регистрация не завершилась успешно.

*Окно клиента*:

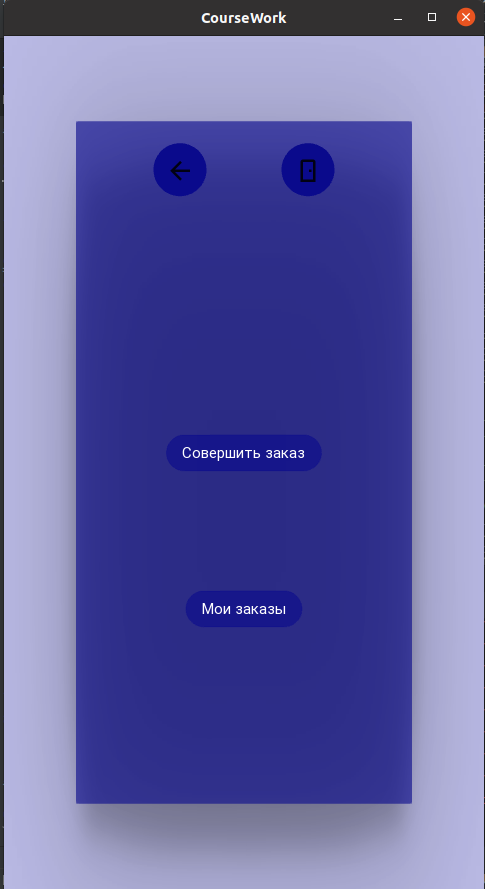


Рисунок 3.5 – Окно клиента

В этом окне пользователь может выбрать: совершить новый заказ или же просмотреть историю своих заказов.

*Окно курьера*:

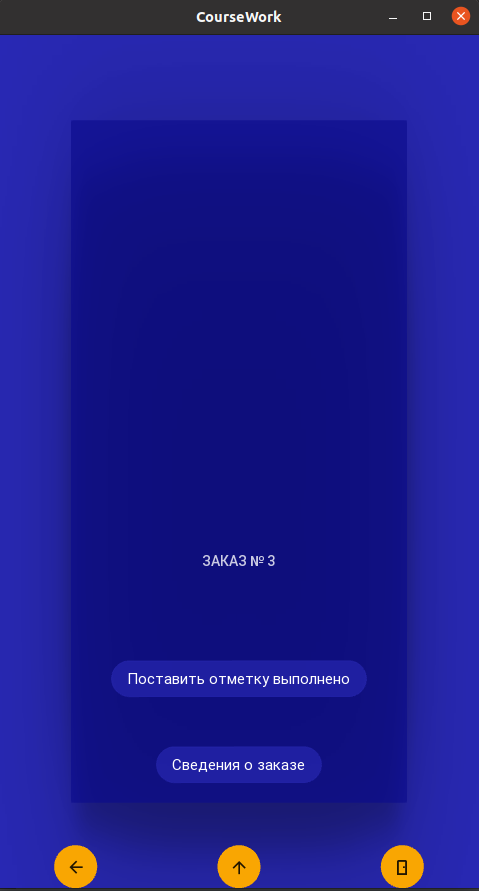


Рисунок 3.6 – Окно курьера

Содержание этого окна зависит от того, выполняет ли данный курьер заказ в настоящее время. Если да, то ему будут предложены две функции: отметить факт выполнения текущего заказа, либо просмотреть сведения о текущем заказе, в противном случае никаких функций предложено не будет.

*Окно оператора*:

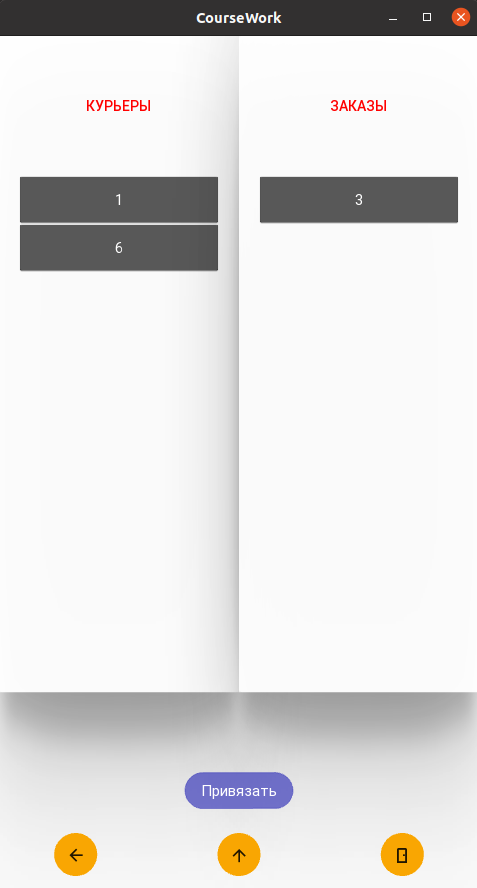


Рисунок 3.7 – Окно оператора

Окно оператора позволяет ему выбирать из одной колонки свободных курьеров, а из другой – нераспределенные оплаченные заказы.

*Окно директора*:

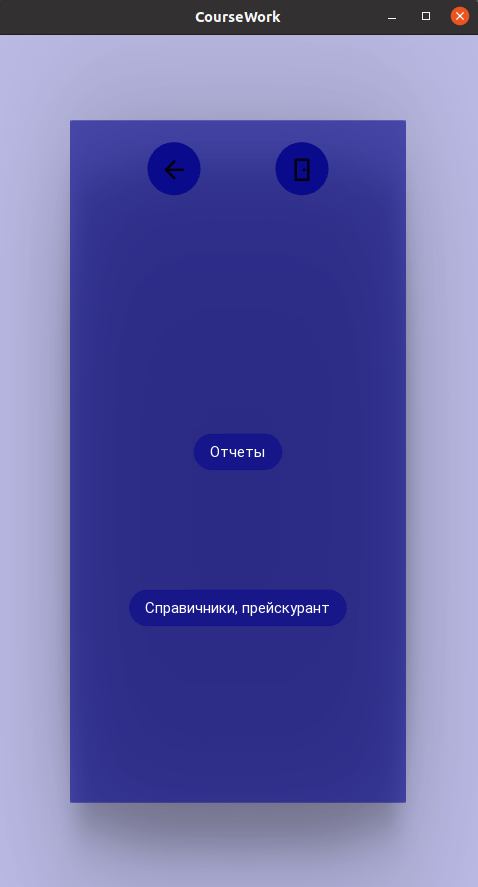


Рисунок 3.8 – Окно директора

Здесь директор может выбрать между просмотром отчета по выбранному месяцу и просмотром статистики (диаграммы частот, график прибыли).

*Окно ведения справочников*:

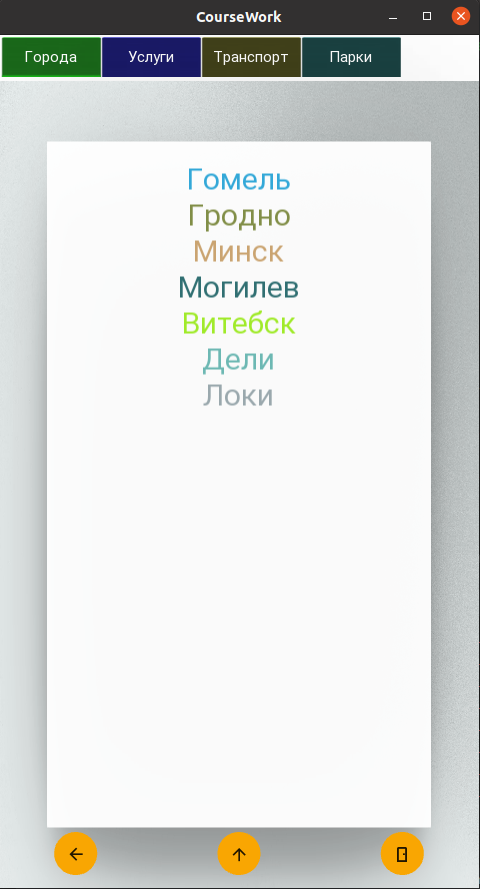


Рисунок 3.9 – Окно ведения справочников

Здесь директор может редактировать справочники, прейскурант. Окно редактирования представлено на рисунке 3.10.

*Окно редактирования*:

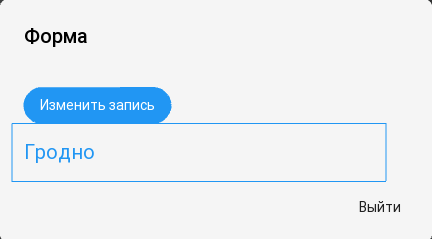


Рисунок 3.10 – Окно редактирования справочников

Данное окно содержит необходимое количество текстовых полей для редактирования записей.

*Окно заказа*:

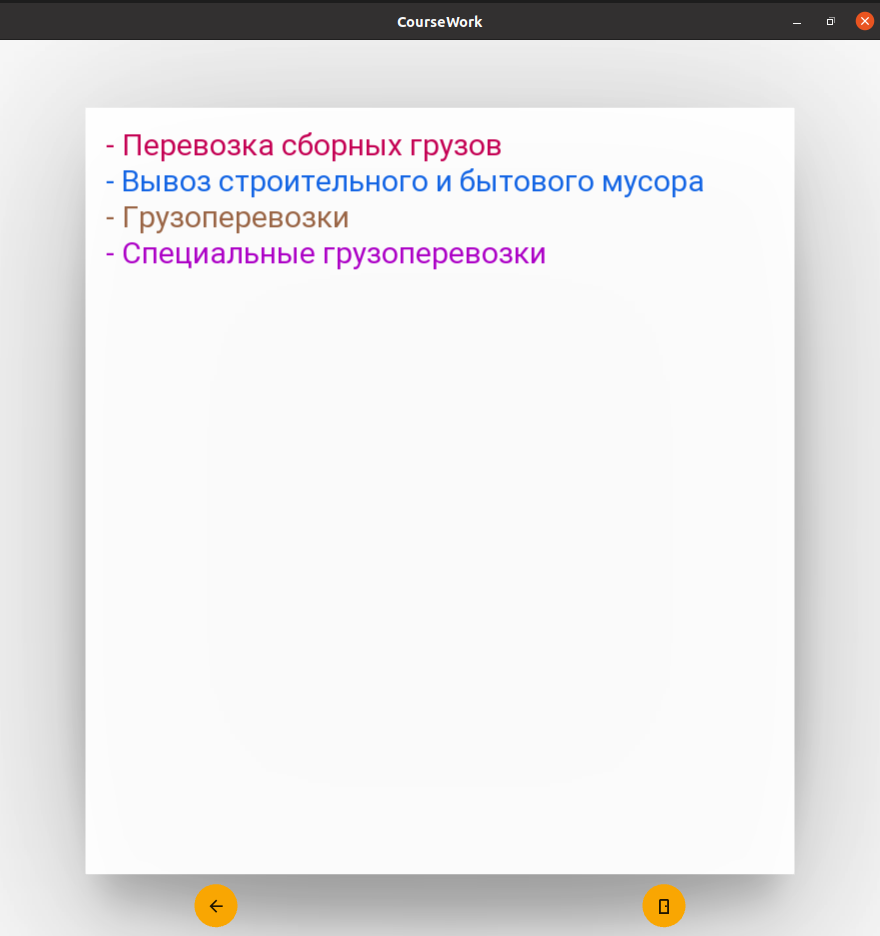


Рисунок 3.11 – Окно заказа

Здесь отображается весь перечень предоставляемых услуг для клиентов. При выборе услуги, пользователю останется уточнить заказ (см. рисунок 3.12).

*Окно уточнения заказа*:

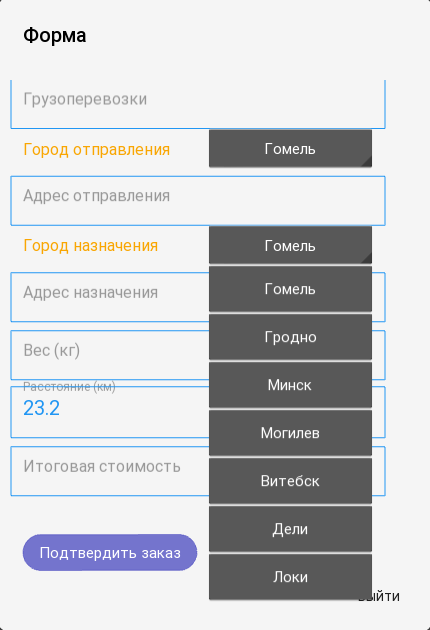


Рисунок 3.12 – Окно уточнения заказа

*Окно истории заказов*:

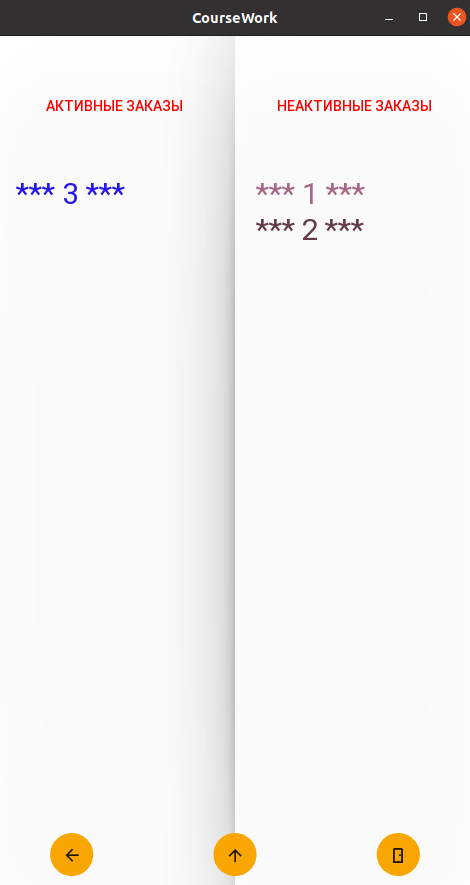


Рисунок 3.13 – Окно истории заказов

*Окно оплаты/отмены заказа*:

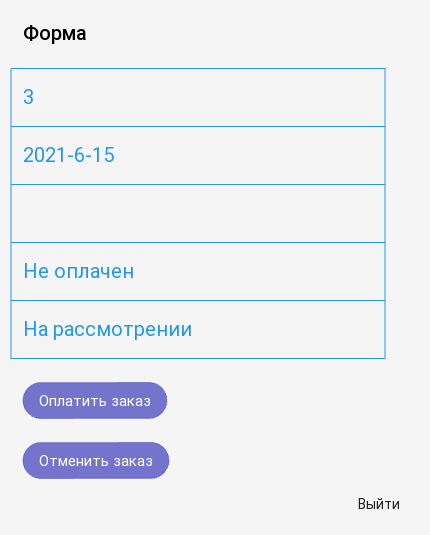


Рисунок 3.14 – Окно оплаты/отмены заказа

*Окно отчетности*:

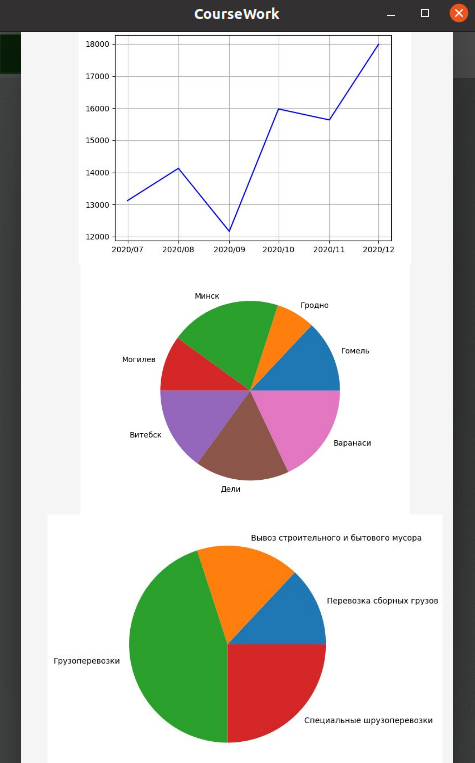


Рисунок 3.15 – Окно отчетности

**4 ТЕСТИРОВАНИЕ И ВЕРИФИКАЦИЯ**

**4.1 Верификация**

После создания программного обеспечения необходимо провести его верификацию, т.е. убедиться, что созданный функционал соответствует требуемому. Для этого создана таблица 4.1, в которой отражены требования, приведённые в подразделе 1.3.

Таблица 4.1 – Верификация

|  |  |
| --- | --- |
| **Требование к проектируемому ПО** | **Требование к проектируемому ПО** |
| Ведение справочников, прейскурант | HandBooks |
| Прием заказов, учет оплаты заказов, отмены | Customer, Operator |
| Создание отчётов, статистика | Reports |

Как видно из таблицы 4.1, программное обеспечение удовлетворяет поставленным требованиям

**4.2 Обработка исключительных ситуаций**

В грамотно спроектированном программном обеспечении количество исключительных ситуаций сравнительно мало, так как большинство из них можно контролировать на уровне доступа к элементам управления (неверный пользовательский ввод, влекущий за собой возникновение исключительных ситуаций). Тем не менее, даже в таком программном обеспечении исключительные ситуации имеют место быть, в основном они возникают по вине стороннего программного обеспечения, связанного с данным, или же по вине сбоя в компьютерном оборудовании. Поэтому такие исключительные ситуации являются достаточно сложно прогнозируемыми и тяжело воспроизводимыми.

Рассмотрим возможные исключительные ситуации:

1. Если база данных является удаленной и отсутствует интернет-соединение, либо указан неверный пароль или логин для доступа к базе данных:

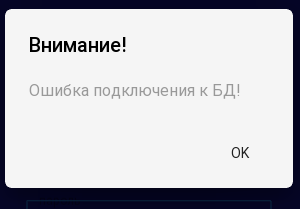


Рисунок 4.1 – Ошибка подключения к БД

1. В любом из окон, где требуется заполнить поля они не заполнены:

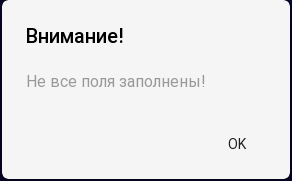


Рисунок 4.2 – Ошибка события

1. Попытка регистрации с уже существующим логином в системе:

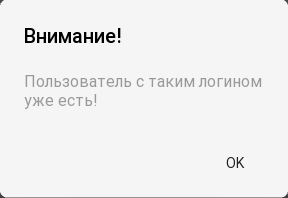


Рисунок 4.3 – Ошибка регистрации

1. Введен слишком простой пароль при попытке регистрации:

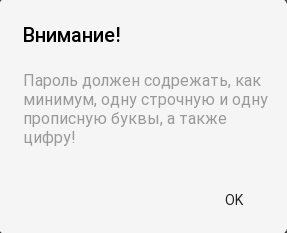


Рисунок 4.4 – Ошибка формата пароля

1. При регистрации пароли не совпадают:

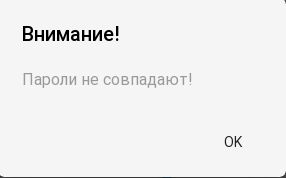


Рисунок 4.5 – Ошибка подтверждения пароля

# ЗАКЛЮЧЕНИЕ

В результате выполнения данного курсового проекта было спроектировано и разработано кроссплатформенное приложение. Для этого было проведено изучение предметной области и выявление необходимых данных. Для хранения данных спроектирована и разработана база данных, удовлетворяющая необходимым требованиям. Разработан пользовательский интерфейс с использованием фреймворка *Kivy*. Также реализована ролевая политика для установки ограничения редактирования и просмотра данных пользователям для обеспечения защиты хранимых данных. Разработанные приложения содержат необходимый функционал для выполнения поставленной задачи.

При создании проекта были изучены принципы SOLID, особенности операционной системы *Linux*, способы получения *apk*-сборки из приложения, написанного на языке программирования высокого уровня *Python*, а также особенности использования популярного фреймворка *Kivy*. Кроме этого, были закреплены навыки использования реляционной базы данных MySQL.

# Список использованных источников

1. Сам себе программист. Как научиться программировать и устроиться в Ebay? / Кори Альтхофф ; [пер. с англ. М.А. Райтмана]. – Москва : Эксмо, 2018. – 208 с.

2. Чан, Уэсли. Python: создание приложений. Библиотека профессионала, 3-е изд. : Пер. с англ. – М. : ООО «И.Д. Вильямс», 2015. – 816 с. : ил. – Парал. тит. англ.

3. Дейт, К. Дж. Введение в системы баз данных : учебник – 8-е изд., пер. с англ. – М. : Издательский дом «Вильямс», 2005. – 1328 с.

4. Программирование на Python 3. Подробное руководство. – Пер. с англ. – СПб.: Символ-Плюс, 2009. – 608 с., ил.

5. Корпоративный монитор: Документация по *MySQL* [Электронный ресурс]. – 2019. – Режим доступа: **https://dev.mysql.com/doc/relnotes/mysql-monitor/8.0/en/**. – Дата доступа: 21.04.2021.

6. Руководство для разработчиков, коннекторы: Документация по *MySQL* [Электронный ресурс]. – 2016. – Режим доступа: **https://dev.mysql.com/doc/connector-odbc/en/**. – Дата доступа: 22.04.2021.

7. Требования к оборудованию и программному обеспечению для установки *MySQL*: Документация по *MySQL* [Электронный ресурс]. – 2016. – Режим доступа: **http://www.mysql.ru/docs/man/Installing.html**. – Дата доступа: 21.04.2021.

# ПРИЛОЖЕНИЕ А – ЛИСТИНГ ПРОГРАММЫ

**LogIn.py**

from kivy.properties import ObjectProperty

from kivymd.uix.screen import MDScreen

from ChooseRole import ChooseRole

from Notification import Notification

from User import User

from WithDB import WithDB

class LogIn(MDScreen):

my\_login = ObjectProperty(None)

my\_password = ObjectProperty(None)

dialog = None

note = Notification(dialog)

#If user login successfully, static field of class User contains his user\_id

def button\_login(self):

db\_pointer = WithDB()

if self.my\_password.text == '' or self.my\_login.text == '':

self.note.universal\_note('Не все поля заполнены!', [])

return False

user\_roles = []

User.user\_id = []

if not db\_pointer.get\_smth('get\_user\_id\_with\_login\_and\_password',

[self.my\_login.text,

self.my\_password.text

],

User.user\_id

):

self.note.universal\_note('Ошибка запроса!', [])

return False

if len(User.user\_id) == 0:

self.note.universal\_note('Такого пользователя в системе нет!', [])

return False

User.user\_id = User.user\_id[0][0]

if not db\_pointer.get\_smth('get\_role\_id\_with\_user\_id',

[User.user\_id

],

user\_roles):

return False

self.\_\_choose\_role(user\_roles)

def dialog\_close(self, \*args):

self.dialog.dismiss(force=True)

#User is free to have few roles, so, here program's mechanism sudgest what to do.

#If user has only one role (CUSTOMER), he immediately get customer's screen,

#otherwise he get a choice.

def \_\_choose\_role(self, user\_roles):

if len(user\_roles) == 0:

self.my\_password.text = ''

self.note.universal\_note('Неправильно введен либо пароль, либо логин',

[self.my\_password]

)

else:

if len(user\_roles) == 1:

role\_id = []

if not WithDB().get\_smth('get\_role\_id\_with\_role\_title', ['Клиент'], role\_id):

self.note.universal\_note('Произошло обезличивание!', [])

else:

User.current\_role\_id = role\_id[0][0]

self.manager.current = 'customer'

else:

db\_pointer = WithDB()

title = []

for i in range(len(user\_roles)):

if not db\_pointer.get\_smth('get\_role\_title\_with\_role\_id', [user\_roles[i][0]], title):

title = []

return None

ChooseRole.buttons[title[-1][0]].disabled = False

self.manager.current = 'choose\_role'

**ChooseRole.py**

from kivy.app import App

from kivy.uix.anchorlayout import AnchorLayout

from kivy.uix.gridlayout import GridLayout

from kivy.uix.widget import Widget

from kivymd.uix.button import MDFillRoundFlatButton, MDIconButton

from kivymd.uix.card import MDCard

from kivymd.uix.screen import MDScreen

from Notification import Notification

from User import User

from WithDB import WithDB

class ChooseRole(MDScreen):

"""

Here there are from 2 up to 4 special buttons. It depends on user's roles quantity.

So, It would be good to create flexible screen.

"""

buttons = {"Курьер": 0,

"Клиент": 0,

"Оператор": 0,

"Директор": 0

}

dialog = None

note = Notification(dialog)

def \_\_init\_\_(self, \*\*kw):

super(ChooseRole, self).\_\_init\_\_(\*\*kw)

self.md\_bg\_color = [40 / 255, 40 / 255, 180 / 255, 80 / 255]

self.add\_buttons()

def add\_buttons(self):

main\_container = self.\_\_main\_container()

main\_container.add\_widget(self.\_\_icons())

main\_container.add\_widget(Widget())

self.buttons["Клиент"] = self.\_\_button\_customer()

self.buttons['Курьер'] = self.\_\_button\_curier()

self.buttons['Оператор'] = self.\_\_button\_operator()

self.buttons['Директор'] = self.\_\_button\_director()

for val in self.buttons.values():

main\_container.add\_widget(val)

main\_container.add\_widget(Widget())

self.add\_widget(main\_container)

@staticmethod

def \_\_main\_container():

main\_container = MDCard()

main\_container.size\_hint = (.7, .8)

main\_container.pos\_hint = {"center\_x": .5, "center\_y": .5}

main\_container.elevation = 15

main\_container.padding = 20

main\_container.spacing = 60

main\_container.orientation = 'vertical'

main\_container.md\_bg\_color = [10 / 255, 10 / 255, 140 / 255, 100 / 255]

return main\_container

def \_\_icons(self):

main\_cont = GridLayout()

main\_cont.valign = 'center'

main\_cont.cols = 2

main\_cont.rows = None

anch1 = AnchorLayout()

anch2 = AnchorLayout()

first\_icon\_button = MDIconButton()

second\_icon\_button = MDIconButton()

first\_icon\_button.user\_font\_size = '30sp'

second\_icon\_button.user\_font\_size = '30sp'

first\_icon\_button.icon\_color = [1, 1, 1, 1]

second\_icon\_button.icon\_color = [1, 1, 1, 1]

first\_icon\_button.icon = 'arrow-left'

second\_icon\_button.icon = 'door'

first\_icon\_button.md\_bg\_color = [10 / 255, 10 / 255, 140 / 255, 255 / 255]

second\_icon\_button.md\_bg\_color = [10 / 255, 10 / 255, 140 / 255, 255 / 255]

first\_icon\_button.bind(on\_press=self.go\_to\_login)

second\_icon\_button.bind(on\_press=App.get\_running\_app().stop)

anch1.add\_widget(first\_icon\_button)

anch2.add\_widget(second\_icon\_button)

main\_cont.add\_widget(anch1)

main\_cont.add\_widget(anch2)

return main\_cont

def go\_to\_login(self, \*args):

self.manager.current = 'login'

def \_\_button\_customer(self):

button = MDFillRoundFlatButton()

button.disabled = True

button.pos\_hint = {"center\_x": .5}

button.font\_size = 15

button.md\_bg\_color = [10 / 255, 10 / 255, 140 / 255, 100 / 255]

button.text\_color = [1, 1, 1, 1]

button.text = 'Клиент'

button.bind(on\_press=self.customer)

return button

def \_\_button\_curier(self):

button = MDFillRoundFlatButton()

button.disabled = True

button.pos\_hint = {"center\_x": .5}

button.font\_size = 15

button.md\_bg\_color = [10 / 255, 10 / 255, 140 / 255, 100 / 255]

button.text\_color = [1, 1, 1, 1]

button.text = 'Курьер'

button.bind(on\_press=self.curier)

return button

def \_\_button\_operator(self):

button = MDFillRoundFlatButton()

button.disabled = True

button.pos\_hint = {"center\_x": .5}

button.font\_size = 15

button.md\_bg\_color = [10 / 255, 10 / 255, 140 / 255, 100 / 255]

button.text\_color = [1, 1, 1, 1]

button.text = 'Оператор'

button.bind(on\_press=self.operator)

return button

def \_\_button\_director(self):

button = MDFillRoundFlatButton()

button.disabled = True

button.pos\_hint = {"center\_x": .5}

button.font\_size = 15

button.md\_bg\_color = [10 / 255, 10 / 255, 140 / 255, 100 / 255]

button.text\_color = [1, 1, 1, 1]

button.text = 'Директор'

button.bind(on\_press=self.director)

return button

def customer(self, \*args):

role\_id = []

if not WithDB().get\_smth('get\_role\_id\_with\_role\_title', ['Клиент'], role\_id):

self.note.universal\_note('Произошло обезличивание!', [])

else:

User.current\_role\_id = role\_id[0]

self.manager.current = 'customer'

def curier(self, \*args):

role\_id = []

if not WithDB().get\_smth('get\_role\_id\_with\_role\_title', ['Курьер'], role\_id):

self.note.universal\_note('Произошло обезличивание!', [])

else:

User.current\_role\_id = role\_id[0]

self.manager.current = 'curier'

def operator(self, \*args):

role\_id = []

if not WithDB().get\_smth('get\_role\_id\_with\_role\_title', ['Оператор'], role\_id):

self.note.universal\_note('Произошло обезличивание!', [])

else:

User.current\_role\_id = role\_id[0]

self.manager.current = 'operator'

def director(self, \*args):

role\_id = []

if not WithDB().get\_smth('get\_role\_id\_with\_role\_title', ['Директор'], role\_id):

self.note.universal\_note('Произошло обезличивание!', [])

else:

User.current\_role\_id = role\_id[0]

self.manager.current = 'director'

**Curier.py**

from datetime import datetime

from kivy.properties import ObjectProperty

from kivy.uix.boxlayout import BoxLayout

from kivy.uix.image import Image

from kivymd.uix.screen import MDScreen

from Notification import Notification

from User import User

from WithDB import WithDB

class Curier(MDScreen):

label = ObjectProperty(None)

btn1 = ObjectProperty(None)

btn2 = ObjectProperty(None)

check = False

dialog = None

note = Notification(dialog)

def \_\_init\_\_(self, \*\*kw):

super().\_\_init\_\_(\*\*kw)

self.load\_data()

def load\_data(self):

ans = []

db\_pointer = WithDB()

if not db\_pointer.get\_smth('get\_order\_with\_courier\_id', [User.user\_id], ans):

self.check = False

self.label.text = 'Заказов нет'

self.btn1.disabled = True

self.btn2.disabled = True

return None

elif len(ans) == 0:

self.check = False

self.label.text = 'Заказов нет'

self.btn1.disabled = True

self.btn2.disabled = True

return None

else:

self.check = True

ans = ans[0][0]

self.label.text = 'Заказ № ' + str(ans)

self.btn1.disabled = False

self.btn2.disabled = False

def completed(self):

ans = []

db\_pointer = WithDB()

if not db\_pointer.get\_smth('get\_order\_with\_courier\_id', [User.user\_id], ans):

self.note.universal\_note('Ошибка подключения к базе данных!', [])

return None

if len(ans) == 0:

self.note.universal\_note('Неизвестная ошибка!', [])

return None

ans = ans[0][0]

now = datetime.date(datetime.today())

stage\_id = []

if not db\_pointer.get\_smth('get\_stage\_id\_with\_stage\_title', ['Выполнен'], stage\_id):

return None

else:

stage\_id = stage\_id[0][0]

if not db\_pointer.insert\_delete\_alter\_smth('add\_orders\_executions\_and\_stage\_id\_with\_order\_id',

[now, stage\_id, ans]):

self.note.universal\_note('Операция была прервана!', [])

return None

self.note.universal\_note('Изменения были зафиксированы!', [])

self.load\_data()

def info(self):

im = Image(source='pictures/map.jpg')

im.allow\_stretch = True

im.size\_hint\_x = None

im.size\_hint\_y = None

im.height = 800

im.width = 1400

self.note.note\_with\_container([im])

**Customer.py**

from kivymd.uix.screen import MDScreen

class Customer(MDScreen):

pass

**CustomerOrder.py**

import datetime

from kivy.properties import ObjectProperty

from kivy.uix.boxlayout import BoxLayout

from kivy.uix.gridlayout import GridLayout

from kivy.uix.spinner import Spinner

from kivy.uix.widget import Widget

from kivymd.uix.boxlayout import MDBoxLayout

from kivymd.uix.button import MDTextButton, MDFillRoundFlatButton

from kivymd.uix.label import MDLabel

from kivymd.uix.screen import MDScreen

from functools import partial

from random import randrange

from kivymd.uix.textfield import MDTextField

from Notification import Notification

from User import User

from WithDB import WithDB

from kivy.uix.button import Button

class CustomerOrder(MDScreen):

cont = ObjectProperty(None)

titles = []

dialog = None

note = Notification(dialog)

container = BoxLayout()

def \_\_init\_\_(self, \*\*kw):

super().\_\_init\_\_(\*\*kw)

self.fill\_services()

self.container.add\_widget(Button(text='Hello'))

def ordering(self, title, \*args):

cont = []

cont.append(MDBoxLayout(

height='500dp',

orientation='vertical',

size\_hint\_y=None

)

)

widgets = self.\_\_widgets\_for\_ordering(title)

for i in range(len(widgets)):

cont[0].add\_widget(widgets[i])

self.note.note\_with\_container(cont)

def processing(self, collection, \*args):

for i in range(len(collection) - 1):

if collection[i].text == '':

self.note.universal\_note('Заполните все поля!', [])

return None

now = datetime.datetime.date(datetime.datetime.today())

customer\_id = []

db\_pointer = WithDB()

if not db\_pointer.get\_smth('get\_customer\_id\_with\_user\_id', [User.user\_id], customer\_id):

self.note.universal\_note('Проблемы с БД!', [])

return None

else:

customer\_id = customer\_id[0][0]

stage\_id = []

if not db\_pointer.get\_smth('get\_stage\_id\_with\_stage\_title', ['На рассмотрении'], stage\_id):

self.note.universal\_note('Проблемы с БД!', [])

return None

else:

stage\_id = stage\_id[0][0]

status\_id = []

if not db\_pointer.get\_smth('get\_status\_id\_with\_status\_title', ['Не оплачен'], status\_id):

self.note.universal\_note('Проблемы с БД!', [])

return None

else:

status\_id = status\_id[0][0]

if not db\_pointer.insert\_delete\_alter\_smth('add\_order', [now, customer\_id, stage\_id, status\_id]):

self.note.universal\_note('Заказ не был создан!', [])

return None

orders\_id = []

if not db\_pointer.get\_smth('get\_last\_orders\_id', [now, customer\_id, stage\_id, status\_id], orders\_id):

self.note.universal\_note('Проблемы с БД!', [])

return None

else:

orders\_id = orders\_id[0][0]

service\_id = []

if not db\_pointer.get\_smth('get\_service\_with\_title', [collection[-1]], service\_id):

self.note.universal\_note('Проблемы с БД!', [])

return None

else:

service\_id = service\_id[0][0]

if not db\_pointer.insert\_delete\_alter\_smth('insert\_my\_orders\_service', [service\_id, orders\_id, collection[4].text,

collection[5].text, collection[3].text,

collection[1].text, collection[6].text]):

self.note.universal\_note('Услуга заказа не была добавлена!', [])

return None

orders\_service\_id = []

if not db\_pointer.get\_smth('get\_my\_orders\_service\_id', [service\_id, orders\_id, collection[4].text, collection[5].text,

collection[3].text, collection[1].text, collection[6].text], orders\_service\_id):

self.note.universal\_note('Проблемы с БД!', [])

return None

orders\_service\_id = orders\_service\_id[0][0]

begin\_city\_id = []

end\_city\_id = []

if not db\_pointer.get\_smth('get\_city\_id\_with\_city\_title',

[collection[0].text], begin\_city\_id):

self.note.universal\_note('Проблемы с БД!', [])

return None

if not db\_pointer.get\_smth('get\_city\_id\_with\_city\_title',

[collection[2].text], end\_city\_id):

self.note.universal\_note('Проблемы с БД!', [])

return None

begin\_city\_id = begin\_city\_id[0][0]

end\_city\_id = end\_city\_id[0][0]

if not db\_pointer.insert\_delete\_alter\_smth('insert\_order\_services\_begin\_city',

[begin\_city\_id, orders\_service\_id]):

self.note.universal\_note('Начальный город услуги заказа не был добавлен!', [])

return None

if not db\_pointer.insert\_delete\_alter\_smth('insert\_order\_services\_end\_city',

[end\_city\_id, orders\_service\_id]):

self.note.universal\_note('Конечный город услуги заказа не был добавлен!', [])

return None

self.note.universal\_note('Операция прошла успешно!', [])

@staticmethod

def \_\_is\_number(value):

try:

float(value)

return True

except ValueError:

return False

def \_\_calculator(self, collection, \*args):

db\_pointer = WithDB()

costs = []

cost\_weight = 0

cost\_radius = 0

if not db\_pointer.get\_smth('get\_services\_costs\_with\_title', [collection[0]], costs):

self.note.universal\_note('Проблемы с БД!', [])

return None

else:

cost\_weight = float(costs[0][0])

cost\_radius = float(costs[0][1])

if self.\_\_is\_number(collection[1].text) and self.\_\_is\_number(collection[2].text):

collection[3].text = str(float(

collection[1].text) \* cost\_weight + float(collection[2].text) \* cost\_radius)

else:

self.note.universal\_note('Вес и расстояние - числа!', [collection[1]])

def \_\_widgets\_for\_ordering(self, title):

txt\_field1 = self.\_\_txt\_field(title)

txt\_field1.disabled = True

txt\_field2 = self.\_\_txt\_field('Адрес отправления')

txt\_field3 = self.\_\_txt\_field('Адрес назначения')

txt\_field4 = self.\_\_txt\_field('Вес (кг)')

txt\_field5 = self.\_\_txt\_field('Расстояние (км)')

txt\_field5.text = '23.2'

txt\_field5.disabled = True

txt\_field6 = self.\_\_txt\_field('Итоговая стоимость')

txt\_field6.disabled = True

txt\_field4.bind(on\_text\_validate=partial(self.\_\_calculator, [title, txt\_field4, txt\_field5, txt\_field6]))

txt\_field5.bind(on\_text\_validate=partial(self.\_\_calculator, [title, txt\_field4, txt\_field5, txt\_field6]))

#---------------------------

label1 = self.\_\_label('Город отправления')

label2 = self.\_\_label('Город назначения')

self.titles.clear()

db\_pointer = WithDB()

sp\_cities = []

sp1 = 'pass'

sp2 = 'pass'

if not db\_pointer.get\_smth('get\_city\_titles', [], sp\_cities):

sp1 = self.\_\_spinner(['Пусто'])

sp2 = self.\_\_spinner(['Пусто'])

else:

for i in range(len(sp\_cities)):

self.titles.append(sp\_cities[i][0])

sp1 = self.\_\_spinner(self.titles)

sp2 = self.\_\_spinner(self.titles)

grid1 = GridLayout(cols=2,

rows=None)

grid2 = GridLayout(cols=2,

rows=None)

grid1.add\_widget(label1)

grid1.add\_widget(sp1)

grid2.add\_widget(label2)

grid2.add\_widget(sp2)

btn1 = self.\_\_button()

btn1.bind(on\_press=partial(self.processing, [sp1,

txt\_field2,

sp2,

txt\_field3,

txt\_field4,

txt\_field5,

txt\_field6,

title

]

)

)

return [txt\_field1,

grid1,

txt\_field2,

grid2,

txt\_field3,

txt\_field4,

txt\_field5,

txt\_field6,

Widget(),

btn1

]

@staticmethod

def \_\_button():

btn = MDFillRoundFlatButton()

btn.pos\_hint = {'center': .5}

btn.font\_size = 15

btn.md\_bg\_color = [40 / 255, 40 / 255, 180 / 255, 100 / 255]

btn.text\_color = [1, 1, 1, 1]

btn.size\_hint\_x = None

btn.text = 'Подтвердить заказ'

return btn

@staticmethod

def \_\_spinner(collection):

sp = Spinner()

sp.text = collection[0]

sp.values = tuple(collection)

sp.size\_hint = (None, None)

sp.size = (165, 40)

sp.pos\_hint = {'center': .5}

return sp

@staticmethod

def \_\_label(text):

label = MDLabel(text=text)

label.pos\_hint = {'center': .5}

label.size\_hint\_x = None

label.size\_hint\_y = None

label.height = 40

label.width = 185

label.color = [249 / 255, 166 / 255, 2 / 255, 255 / 255]

return label

@staticmethod

def \_\_txt\_field(hint\_text):

txt\_field = MDTextField()

txt\_field.pos\_hint = {'center': .5}

txt\_field.color\_mode = 'primary'

txt\_field.mode = 'rectangle'

txt\_field.size\_hint\_x = None

txt\_field.size\_hint\_y = None

txt\_field.height = 40

txt\_field.font\_size = 20

txt\_field.width = 350

txt\_field.hint\_text = hint\_text

return txt\_field

def fill\_services(self, \*args):

self.cont.clear\_widgets()

self.titles.clear()

db\_pointer = WithDB()

services = []

if not db\_pointer.get\_smth('get\_service\_titles', [], services):

self.cont.add\_widget(MDLabel(text='Список пуст.'))

else:

for i in range(len(services)):

self.titles.append(services[i][0])

self.cont.add\_widget(MDTextButton(text='- ' + services[i][0],

heigh=80,

custom\_color=[randrange(256) / 255, randrange(256) / 255, randrange(256) / 255, 255 / 255],

font\_size='30sp',

on\_press=partial(self.ordering, services[i][0])

)

)

**Director.py**

from kivymd.uix.screen import MDScreen

class Director(MDScreen):

pass

**GetPassword.py**

from kivy.properties import ObjectProperty

from kivymd.uix.screen import MDScreen

from Notification import Notification

class GetPassword(MDScreen):

phone = ObjectProperty(None)

email = ObjectProperty(None)

dialog = None

note = Notification(None)

def dialog\_close(self, \*args):

self.dialog.dismiss(force=True)

def send\_password(self):

if self.phone.text == '' or self.email.text == '':

self.note.universal\_note('Не все поля заполнены!', [])

else:

self.note.universal\_note('На указанную почту и телефон \nбыли высланы необходимые данные!', [])

**HandBooks.py**

from random import randrange

from kivy.properties import ObjectProperty

from kivymd.uix.boxlayout import MDBoxLayout

from kivymd.uix.button import MDTextButton, MDFillRoundFlatButton

from kivymd.uix.label import MDLabel

from kivymd.uix.screen import MDScreen

from kivymd.uix.textfield import MDTextField

from Notification import Notification

from WithDB import WithDB

from functools import partial

class HandBooks(MDScreen):

cont\_cities = ObjectProperty(None)

cont\_fleets = ObjectProperty(None)

cont\_services = ObjectProperty(None)

cont\_kinds = ObjectProperty(None)

titles = []

dialog = None

note = Notification(dialog)

cont = []

def \_\_init\_\_(self, \*\*kw):

super().\_\_init\_\_(\*\*kw)

self.fill\_cities()

self.fill\_fleets()

self.fill\_kinds()

self.fill\_services()

@staticmethod

def \_\_txt\_field(text):

txt\_field = MDTextField()

txt\_field.pos\_hint = {'center': .5}

txt\_field.color\_mode = 'primary'

txt\_field.mode = 'rectangle'

txt\_field.size\_hint\_x = None

txt\_field.size\_hint\_y = None

txt\_field.height = 40

txt\_field.font\_size = 20

txt\_field.width = 350

txt\_field.text = text

return txt\_field

def action\_city(self, title, \*args):

city = []

db\_pointer = WithDB()

if not db\_pointer.get\_smth('get\_city\_with\_title', [title], city):

self.note.universal\_note('Выбрнного Вами города не существует...', [])

else:

self.cont.clear()

self.cont.append(MDBoxLayout(

height='110dp',

orientation='vertical',

size\_hint\_y=None)

)

title = city[0][0]

self.cont[0].add\_widget(MDFillRoundFlatButton(text='Изменить запись',

on\_press=partial(self.change\_record, 'alter\_city', title)))

for i in range(1, len(city[0])):

self.cont[0].add\_widget(self.\_\_txt\_field(str(city[0][i])))

self.note.note\_with\_container(self.cont)

def action\_fleet(self, title, \*args):

fleet = []

db\_pointer = WithDB()

if not db\_pointer.get\_smth('get\_fleet\_with\_title', [title], fleet):

self.note.universal\_note('Выбрнного Вами парка не существует...', [])

else:

self.cont.clear()

self.cont.append(MDBoxLayout(

height='370dp',

orientation='vertical',

size\_hint\_y=None)

)

title = fleet[0][0]

self.cont[0].add\_widget(MDFillRoundFlatButton(text='Изменить запись',

on\_press=partial(self.change\_record, 'alter\_fleet', title)))

for i in range(1, len(fleet[0])):

self.cont[0].add\_widget(self.\_\_txt\_field(str(fleet[0][i])))

self.note.note\_with\_container(self.cont)

def action\_kind(self, title, \*args):

kind = []

db\_pointer = WithDB()

if not db\_pointer.get\_smth('get\_kind\_with\_title', [title], kind):

self.note.universal\_note('Выбрнного Вами вида транспорта не существует...', [])

else:

self.cont.clear()

self.cont.append(MDBoxLayout(

height='310dp',

orientation='vertical',

size\_hint\_y=None)

)

title = kind[0][0]

self.cont[0].add\_widget(MDFillRoundFlatButton(text='Изменить запись',

on\_press=partial(self.change\_record, 'alter\_transports\_kind', title)))

for i in range(1, len(kind[0])):

self.cont[0].add\_widget(self.\_\_txt\_field(str(kind[0][i])))

self.note.note\_with\_container(self.cont)

def action\_service(self, title, \*args):

service = []

db\_pointer = WithDB()

if not db\_pointer.get\_smth('get\_service\_with\_title', [title], service):

self.note.universal\_note('Выбрнной Вами услуги не существует...', [])

else:

self.cont.clear()

self.cont.append(MDBoxLayout(

height='310dp',

orientation='vertical',

size\_hint\_y=None)

)

title = service[0][0]

self.cont[0].add\_widget(MDFillRoundFlatButton(text='Изменить запись',

on\_press=partial(self.change\_record, 'alter\_service', title)))

for i in range(1, len(service[0])):

self.cont[0].add\_widget(self.\_\_txt\_field(str(service[0][i])))

self.note.note\_with\_container(self.cont)

def change\_record(self, query, title, \*args):

collection = []

for child in self.cont[0].children:

if isinstance(child, MDTextField):

collection.append(child.text)

collection.reverse()

collection.append(title)

db\_pointer = WithDB()

print(collection)

if not db\_pointer.insert\_delete\_alter\_smth(query, collection):

return None

def fill\_cities(self, \*args):

self.cont\_cities.clear\_widgets()

self.titles.clear()

db\_pointer = WithDB()

cities = []

if not db\_pointer.get\_smth('get\_city\_titles', [], cities):

self.cont\_cities.add\_widget(MDLabel(text='Список пуст.'))

else:

for i in range(len(cities)):

self.titles.append(cities[i][0])

self.cont\_cities.add\_widget(MDTextButton(text=cities[i][0],

heigh=80,

custom\_color=[randrange(256) / 255, randrange(256) / 255,

randrange(256) / 255, 255 / 255],

font\_size='30sp',

on\_press=partial(self.action\_city, cities[i][0]),

pos\_hint={"center\_x": .5, "center\_y": .5}

)

)

def fill\_fleets(self, \*args):

self.cont\_fleets.clear\_widgets()

self.titles.clear()

db\_pointer = WithDB()

fleets = []

if not db\_pointer.get\_smth('get\_fleet\_titles', [], fleets):

self.cont\_fleets.add\_widget(MDLabel(text='Список пуст.'))

else:

for i in range(len(fleets)):

self.titles.append(fleets[i][0])

self.cont\_fleets.add\_widget(MDTextButton(text=fleets[i][0],

heigh=80,

custom\_color=[randrange(256) / 255, randrange(256) / 255,

randrange(256) / 255, 255 / 255],

font\_size='30sp',

on\_press=partial(self.action\_fleet, fleets[i][0]),

pos\_hint={"center\_x": .5, "center\_y": .5}

)

)

def fill\_kinds(self, \*args):

self.cont\_kinds.clear\_widgets()

self.titles.clear()

db\_pointer = WithDB()

kinds = []

if not db\_pointer.get\_smth('get\_kind\_titles', [], kinds):

self.cont\_kinds.add\_widget(MDLabel(text='Список пуст.'))

else:

for i in range(len(kinds)):

self.titles.append(kinds[i][0])

self.cont\_kinds.add\_widget(MDTextButton(text=kinds[i][0],

heigh=80,

custom\_color=[randrange(256) / 255, randrange(256) / 255,

randrange(256) / 255, 255 / 255],

font\_size='30sp',

on\_press=partial(self.action\_kind, kinds[i][0]),

pos\_hint={"center\_x": .5, "center\_y": .5}

)

)

def fill\_services(self, \*args):

self.cont\_services.clear\_widgets()

self.titles.clear()

db\_pointer = WithDB()

services = []

if not db\_pointer.get\_smth('get\_service\_titles', [], services):

self.cont\_services.add\_widget(MDLabel(text='Список пуст.'))

else:

for i in range(len(services)):

self.titles.append(services[i][0])

self.cont\_services.add\_widget(MDTextButton(text=services[i][0],

heigh=80,

custom\_color=[randrange(256) / 255, randrange(256) / 255,

randrange(256) / 255, 255 / 255],

font\_size='30sp',

on\_press=partial(self.action\_service, services[i][0]),

pos\_hint={"center\_x": .5, "center\_y": .5}

)

)

def load\_data(self, \*args):

self.fill\_services()

self.fill\_kinds()

self.fill\_cities()

self.fill\_fleets()

**main.py**

from kivymd.app import MDApp

from kivy.config import Config

from kivy.uix.screenmanager import ScreenManager, FallOutTransition

from kivy.lang import Builder

from CustomerOrder import CustomerOrder

from Director import \*

from Customer import \*

from ChooseRole import ChooseRole

from HandBooks import HandBooks

from LogIn import LogIn

from Reports import Reports

from SignUp import \*

from GetPassword import \*

from Operator import Operator

from Curier import Curier

from kivy.core.window import Window

from MyOrders import MyOrders

Config.set('kivy', 'keyboard\_mode', 'systemanddock')

Window.size = (480, 853)

Builder.load\_file('kv-files/My.kv')

Builder.load\_file('kv-files/LogIn.kv')

Builder.load\_file('kv-files/SignUp.kv')

Builder.load\_file('kv-files/GetPassword.kv')

Builder.load\_file('kv-files/Customer.kv')

Builder.load\_file('kv-files/Director.kv')

Builder.load\_file('kv-files/Operator.kv')

Builder.load\_file('kv-files/Courier.kv')

Builder.load\_file('kv-files/HandBooks.kv')

Builder.load\_file('kv-files/Reports.kv')

Builder.load\_file('kv-files/CustomerOrder.kv')

Builder.load\_file('kv-files/MyOrders.kv')

class Myapp(MDApp):

title = 'CourseWork'

def build(self):

sm = ScreenManager(transition=FallOutTransition())

sm.add\_widget(LogIn(name='login'))

sm.add\_widget(GetPassword(name='get\_password'))

sm.add\_widget(SignUp(name='sign\_up'))

sm.add\_widget(ChooseRole(name='choose\_role'))

sm.add\_widget(Director(name='director'))

sm.add\_widget(Customer(name='customer'))

sm.add\_widget(Operator(name='operator'))

sm.add\_widget(Curier(name='curier'))

sm.add\_widget(HandBooks(name='handbooks'))

sm.add\_widget(Reports(name='reports'))

sm.add\_widget(CustomerOrder(name='customer\_order'))

sm.add\_widget(MyOrders(name='my\_orders'))

return sm

if \_\_name\_\_ == '\_\_main\_\_':

Myapp().run()

**MyOrders.py**

import datetime

from kivy.properties import ObjectProperty

from kivy.uix.widget import Widget

from kivymd.uix.boxlayout import MDBoxLayout

from kivymd.uix.button import MDTextButton, MDFillRoundFlatButton

from kivymd.uix.label import MDLabel

from kivymd.uix.screen import MDScreen

from functools import partial

from random import randrange

from kivymd.uix.textfield import MDTextField

from Notification import Notification

from User import User

from WithDB import WithDB

class MyOrders(MDScreen):

cont1 = ObjectProperty(None)

cont2 = ObjectProperty(None)

titles = []

dialog = None

note = Notification(dialog)

def \_\_init\_\_(self, \*\*kw):

super().\_\_init\_\_(\*\*kw)

self.fill\_first\_cont()

self.fill\_second\_cont()

def fill\_second\_cont(self):

self.cont2.clear\_widgets()

self.titles.clear()

db\_pointer = WithDB()

customer\_id = []

if not db\_pointer.get\_smth('get\_customer\_id\_with\_user\_id', [User.user\_id], customer\_id):

return None

if len(customer\_id) == 0:

return None

customer\_id = customer\_id[0][0]

db\_pointer = WithDB()

orders = []

if not db\_pointer.get\_smth('get\_passive\_orders\_data\_for\_customer\_with\_customer\_id', [customer\_id], orders):

self.cont2.add\_widget(MDLabel(text='Список пуст.'))

else:

for i in range(len(orders)):

self.titles.append(orders[i][0])

self.cont2.add\_widget(MDTextButton(text='\*\*\* ' + str(self.titles[-1]) + ' \*\*\*',

heigh=80,

custom\_color=[randrange(256) / 255, randrange(256) / 255,

randrange(256) / 255, 255 / 255],

font\_size='30sp',

on\_press=partial(self.ordering\_second, orders[i])

)

)

def fill\_first\_cont(self):

self.cont1.clear\_widgets()

self.titles.clear()

db\_pointer = WithDB()

customer\_id = []

if not db\_pointer.get\_smth('get\_customer\_id\_with\_user\_id', [User.user\_id], customer\_id):

return None

if len(customer\_id) == 0:

return None

customer\_id = customer\_id[0][0]

orders = []

if not db\_pointer.get\_smth('get\_active\_orders\_data\_for\_customer\_with\_customer\_id', [customer\_id], orders):

self.cont1.add\_widget(MDLabel(text='Список пуст.'))

else:

for i in range(len(orders)):

self.titles.append(orders[i][0])

self.cont1.add\_widget(MDTextButton(text='\*\*\* ' + str(self.titles[-1]) + ' \*\*\*',

heigh=80,

custom\_color=[randrange(256) / 255, randrange(256) / 255,

randrange(256) / 255, 255 / 255],

font\_size='30sp',

on\_press=partial(self.ordering, orders[i])

)

)

def ordering(self, collection, \*args):

cont = []

cont.append(MDBoxLayout(

height='410dp',

orientation='vertical',

size\_hint\_y=None

)

)

widgets = self.\_\_widgets\_for\_ordering(collection)

for i in range(len(widgets)):

cont[0].add\_widget(widgets[i])

self.note.note\_with\_container(cont)

def ordering\_second(self, collection, \*args):

cont = []

cont.append(MDBoxLayout(

height='410dp',

orientation='vertical',

size\_hint\_y=None

)

)

widgets = self.\_\_widgets\_for\_ordering\_second(collection)

for i in range(len(widgets)):

cont[0].add\_widget(widgets[i])

self.note.note\_with\_container(cont)

def processing(self, order\_id, \*args):

db\_pointer = WithDB()

status\_id = []

if not db\_pointer.get\_smth('get\_status\_id\_with\_status\_title', ['Оплачен'], status\_id):

return None

else:

status\_id = status\_id[0][0]

if not db\_pointer.insert\_delete\_alter\_smth('alter\_orders\_status\_id\_with\_order\_id',

[status\_id, order\_id]):

return None

self.fill\_first\_cont()

self.fill\_second\_cont()

def refusing(self, order\_id, \*args):

db\_pointer = WithDB()

stage\_id = []

now = datetime.datetime.date(datetime.datetime.today())

if not db\_pointer.get\_smth('get\_stage\_id\_with\_stage\_title', ['Отменен'], stage\_id):

return None

else:

stage\_id = stage\_id[0][0]

if not db\_pointer.insert\_delete\_alter\_smth('add\_orders\_executions\_and\_stage\_id\_with\_order\_id',

[now, stage\_id, order\_id]):

return None

self.fill\_first\_cont()

self.fill\_second\_cont()

def \_\_widgets\_for\_ordering(self, collection):

txt\_field1 = self.\_\_txt\_field('Код заказа')

txt\_field1.disabled = True

txt\_field1.text = str(collection[0])

txt\_field2 = self.\_\_txt\_field('Дата начала')

txt\_field2.disabled = True

if collection[1] is not None:

txt\_field2.text = str(collection[1].year) + '-' + str(collection[1].month) + '-' + str(collection[1].day)

txt\_field3 = self.\_\_txt\_field('Дата выполнения')

txt\_field3.disabled = True

if collection[2] is not None:

txt\_field3.text = str(collection[2].year) + '-' + str(collection[2].month) + '-' + str(collection[2].day)

txt\_field4 = self.\_\_txt\_field('Статус')

txt\_field4.disabled = True

txt\_field4.text = collection[3]

txt\_field5 = self.\_\_txt\_field('Этап')

txt\_field5.disabled = True

txt\_field5.text = collection[4]

btn1 = self.\_\_button()

btn1.bind(on\_press=partial(self.processing, str(collection[0])))

btn2 = self.\_\_button()

btn2.text = 'Отменить заказ'

btn2.bind(on\_press=partial(self.refusing, str(collection[0])))

if txt\_field4.text == 'Оплачен':

btn2.disabled = True

return [txt\_field1,

txt\_field2,

txt\_field3,

txt\_field4,

txt\_field5,

Widget(),

btn1,

Widget(),

btn2]

def \_\_widgets\_for\_ordering\_second(self, collection):

txt\_field1 = self.\_\_txt\_field('Код заказа')

txt\_field1.disabled = True

txt\_field1.text = str(collection[0])

txt\_field2 = self.\_\_txt\_field('Дата начала')

txt\_field2.disabled = True

if collection[1] is not None:

txt\_field2.text = str(collection[1].year)+'-'+str(collection[1].month)+'-'+str(collection[1].day)

txt\_field3 = self.\_\_txt\_field('Дата выполнения')

txt\_field3.disabled = True

if collection[2] is not None:

txt\_field3.text = str(collection[2].year)+'-'+str(collection[2].month)+'-'+str(collection[2].day)

txt\_field4 = self.\_\_txt\_field('Статус')

txt\_field4.disabled = True

txt\_field4.text = collection[3]

txt\_field5 = self.\_\_txt\_field('Этап')

txt\_field5.disabled = True

txt\_field5.text = collection[4]

return [txt\_field1,

txt\_field2,

txt\_field3,

txt\_field4,

txt\_field5,

Widget()]

def load\_data(self, \*args):

self.fill\_first\_cont()

self.fill\_second\_cont()

@staticmethod

def \_\_txt\_field(text\_hint):

txt\_field = MDTextField()

txt\_field.pos\_hint = {'center': .5}

txt\_field.color\_mode = 'primary'

txt\_field.mode = 'rectangle'

txt\_field.size\_hint\_x = None

txt\_field.size\_hint\_y = None

txt\_field.height = 40

txt\_field.font\_size = 20

txt\_field.width = 350

txt\_field.text\_hint = text\_hint

return txt\_field

@staticmethod

def \_\_button():

btn = MDFillRoundFlatButton()

btn.pos\_hint = {'center': .5}

btn.font\_size = 15

btn.md\_bg\_color = [40 / 255, 40 / 255, 180 / 255, 100 / 255]

btn.text\_color = [1, 1, 1, 1]

btn.size\_hint\_x = None

btn.text = 'Оплатить заказ'

return btn

**MySQL.py**

import mysql.connector

from mysql.connector import Error

class MySQL:

def \_\_init\_\_(self, host, user, password):

self.\_\_host = host

self.\_\_user = user

self.\_\_password = password

def connect\_host(self):

connection = None

try:

connection = mysql.connector.connect(

host=self.\_\_host,

user=self.\_\_user,

passwd=self.\_\_password

)

except Error as e:

pass

return connection

def connect\_database(self, database):

connection = None

try:

connection = mysql.connector.connect(

host=self.\_\_host,

user=self.\_\_user,

passwd=self.\_\_password,

database=database

)

except Error as e:

pass

return connection

@staticmethod

def create\_database(connection, query):

try:

cursor = connection.cursor()

cursor.execute(query)

cursor.close()

connection.close()

return True

except Error as e:

return False

@staticmethod

def execute\_query(connection, query):

try:

cursor = connection.cursor()

cursor.execute(query)

connection.commit()

cursor.close()

connection.close()

return True

except Error as e:

return False

@staticmethod

def execute\_query\_data(connection, query, data):

try:

cursor = connection.cursor()

cursor.execute(query, data)

connection.commit()

cursor.close()

connection.close()

return True

except Error as e:

return False

@staticmethod

def inserting\_many\_records(connection, insert, values):

try:

cursor = connection.cursor()

cursor.executemany(insert, values)

connection.commit()

cursor.close()

connection.close()

return True

except Error as e:

return False

@staticmethod

def execute\_read\_query(connection, query):

result = None

try:

cursor = connection.cursor()

cursor.execute(query)

result = cursor.fetchall()

cursor.close()

connection.close()

return result

except Error as e:

return None

@staticmethod

def dalete\_database(connection, query):

try:

cursor = connection.cursor()

cursor.execute(query)

cursor.close()

connection.close()

return True

except Error as e:

return False

**Notification.py**

from kivymd.uix.button import MDFlatButton

from kivymd.uix.dialog import MDDialog

class Notification:

def \_\_init\_\_(self, dialog):

self.dialog = dialog

def dialog\_close(self, \*args):

self.dialog.dismiss(force=True)

def universal\_note(self, title, params, \*args):

for i in range(len(params)):

params[i].text = ''

self.dialog = MDDialog(

title='Внимание!',

text=title,

buttons=[

MDFlatButton(

text='OK',

on\_release=self.dialog\_close

)

],

size\_hint=(0.6, 0.6)

)

self.dialog.open()

def note\_with\_container(self, cont, \*args):

self.dialog = MDDialog(

title='Форма',

type="custom",

content\_cls=cont[0],

buttons=[

MDFlatButton(

text='Выйти',

on\_release=self.dialog\_close

)

],

size\_hint=(0.9, 0.6)

)

self.dialog.open()

**Operator.py**

from random import randrange

from kivy.properties import ObjectProperty

from kivy.uix.togglebutton import ToggleButton

from kivy.uix.widget import Widget

from kivymd.uix.label import MDLabel

from kivymd.uix.screen import MDScreen

import User

from Notification import Notification

from WithDB import WithDB

class Operator(MDScreen):

cont1 = ObjectProperty(None)

cont2 = ObjectProperty(None)

titles = []

dialog = None

note = Notification(dialog)

def \_\_init\_\_(self, \*\*kw):

super().\_\_init\_\_(\*\*kw)

self.fill\_first\_cont()

self.fill\_second\_cont()

def fill\_first\_cont(self):

self.cont1.clear\_widgets()

self.titles.clear()

db\_pointer = WithDB()

couriers = []

if not db\_pointer.get\_smth('get\_free\_couriers', [], couriers):

return None

if len(couriers) == 0:

return None

self.cont1.add\_widget(Widget())

for i in range(len(couriers)):

self.titles.append(couriers[i][0])

self.cont1.add\_widget(ToggleButton(text=str(self.titles[-1]),

size\_hint\_y=None,

height='48dp',

group='g1')

)

def link(self):

order\_id = 0

courier\_id = 0

for i in self.cont1.children:

if isinstance(i, ToggleButton) and i.state == 'down':

courier\_id = i.text

for i in self.cont2.children:

if isinstance(i, ToggleButton) and i.state == 'down':

order\_id = i.text

db\_pointer = WithDB()

stage\_id = []

if not db\_pointer.get\_smth('get\_stage\_id\_with\_stage\_title', ['Выполняется'], stage\_id):

self.note.universal\_note('Операция была прервана!', [])

return None

stage\_id = stage\_id[0][0]

if not db\_pointer.insert\_delete\_alter\_smth('add\_courier\_id\_and\_operator\_id\_into\_order\_with\_order\_id',

[str(courier\_id), str(User.User.user\_id), str(stage\_id), str(order\_id)]):

self.note.universal\_note('Операция была прервана!', [])

return None

self.note.universal\_note('Заказ был передан курьеру!', [])

def fill\_second\_cont(self):

self.cont2.clear\_widgets()

self.titles.clear()

db\_pointer = WithDB()

orders = []

if not db\_pointer.get\_smth('get\_paid\_orders', [], orders):

return None

if len(orders) == 0:

return None

self.cont2.add\_widget(Widget())

for i in range(len(orders)):

self.titles.append(orders[i][0])

self.cont2.add\_widget(ToggleButton(text=str(self.titles[-1]),

size\_hint\_y=None,

height='48dp',

group='g2')

)

def load\_data(self, \*args):

self.fill\_first\_cont()

self.fill\_second\_cont()

**Reports.py**

from kivy.properties import ObjectProperty

from kivymd.uix.screen import MDScreen

import matplotlib.dates as mdates

import matplotlib

from matplotlib import pylab

from WithDB import WithDB

class Reports(MDScreen):

statistic = ObjectProperty(None)

volume\_per\_month = ObjectProperty(None)

def \_\_init\_\_(self, \*\*kw):

super().\_\_init\_\_(\*\*kw)

self.fill\_volume\_per\_month()

self.fill\_statistic()

def fill\_statistic(self):

self.\_\_profit\_per\_month()

self.\_\_services\_diagramm()

self.\_\_customers\_cities\_giagramm()

def fill\_volume\_per\_month(self):

pass

def \_\_profit\_per\_month(self):

db\_pointer = WithDB()

data = []

if not db\_pointer.get\_smth('get\_sum\_profit\_for\_each\_month', [], data):

return None

if len(data) == 0:

return None

dates = []

y = []

for i in range(len(data)):

dates.append(data[i][0])

y.append(data[i][1])

xdata\_float = matplotlib.dates.date2num(dates)

axes = pylab.subplot(1, 1, 1)

axes.xaxis.set\_major\_formatter(matplotlib.dates.DateFormatter('%Y'))

pylab.plot\_date(xdata\_float, y, fmt='b-')

pylab.grid()

pylab.savefig('pictures/profit\_per\_month.png')

def \_\_services\_diagramm(self):

db\_pointer = WithDB()

data = []

if not db\_pointer.get\_smth('get\_quantity\_of\_each\_services\_item', [], data):

return None

if len(data) == 0:

return None

print(data)

def \_\_customers\_cities\_giagramm(self):

pass

def load\_data(self):

self.fill\_volume\_per\_month()

self.fill\_statistic()

**SignUp.py**

from kivy.properties import ObjectProperty

from kivymd.uix.screen import MDScreen

from Notification import Notification

from WithDB import WithDB

class SignUp(MDScreen):

field\_country = ObjectProperty(None)

field\_city = ObjectProperty(None)

field\_street = ObjectProperty(None)

filed\_house = ObjectProperty(None)

field\_flat = ObjectProperty(None)

surname = ObjectProperty(None)

user\_name = ObjectProperty(None)

middle\_name = ObjectProperty(None)

email = ObjectProperty(None)

phone = ObjectProperty(None)

login = ObjectProperty(None)

password1 = ObjectProperty(None)

password2 = ObjectProperty(None)

check\_box = ObjectProperty(None)

last\_btn = ObjectProperty(None)

dialog = None

note = Notification(dialog)

def dialog\_close(self, \*args):

self.dialog.dismiss(force=True)

def on\_checkbox\_active(self):

if self.check\_box.active:

self.last\_btn.disabled = False

else:

self.last\_btn.disabled = True

def \_\_check\_data(self):

up\_symbol = False

low\_symbol = False

number = False

for i in self.password1.text:

if i.islower():

low\_symbol = True

for i in self.password1.text:

if i.isupper():

up\_symbol = True

for i in self.password1.text:

if i.isdigit():

number = True

return up\_symbol and low\_symbol and number

def \_\_check\_city(self, title):

db\_pointer = WithDB()

city\_id = []

if db\_pointer.get\_smth('get\_city\_id\_with\_city\_title', [title.text], city\_id):

if len(city\_id) == 0:

self.note.universal\_note('Извините, но указанный город не обслуживается нашей компанией!', [title])

return False

return True

def \_\_check\_not\_login(self, title):

db\_pointer = WithDB()

user\_id = []

if db\_pointer.get\_smth('get\_user\_id\_with\_login', [title.text], user\_id):

if len(user\_id) == 1:

self.note.universal\_note('Пользователь с указанным логином уже есть!', [title])

return False

return True

def button\_sign\_up(self):

db\_pointer = WithDB()

if (self.field\_country.text == ''

or self.field\_city.text == ''

or self.field\_street.text == ''

or self.filed\_house.text == ''

or self.field\_flat.text == ''

or self.surname.text == ''

or self.user\_name.text == ''

or self.middle\_name.text == ''

or self.email.text == ''

or self.phone.text == ''

or self.login.text == ''

or self.password1.text == ''

or self.password2.text == ''):

self.note.universal\_note('Не все поля заполнены!', [])

return None

elif self.password1.text != self.password2.text:

self.note.universal\_note('Пароли не совпадают!',

[self.password1, self.password2]

)

return None

elif not self.\_\_check\_city(self.field\_city):

return None

elif not self.\_\_check\_not\_login(self.login):

return None

elif not self.\_\_check\_data():

self.note.universal\_note("""Пароль должен содрежать, как минимум, одну строчную и одну прописную буквы, а также цифру!""",

[self.password1, self.password2]

)

return None

else:

if self.\_\_sign\_up(self.login.text,

self.password1.text,

self.email.text,

self.phone.text,

self.field\_country.text,

self.field\_city.text,

self.field\_street.text,

self.filed\_house.text,

self.field\_flat.text,

self.surname.text,

self.user\_name.text,

self.middle\_name.text

):

self.note.universal\_note('Регистрация прошла успешно!', [])

self.manager.current = 'login'

else:

self.note.universal\_note('Ошибка. Проверьте корректность введенных данных и проверьте состояние соединения с базой данных!', [])

return None

def \_\_sign\_up(self, login, password, email, phone\_number, country, city,

street, home\_number, flat\_number, lastname, name, middle\_name):

if not self.\_\_sign\_up\_check(login, password, email, phone\_number, country, city,

street, home\_number, flat\_number, lastname, name, middle\_name):

db\_pointer = WithDB()

db\_pointer.insert\_delete\_alter\_smth('delete\_user\_with\_login', [login])

return False

return True

@staticmethod

def \_\_sign\_up\_check(login, password, email, phone\_number, country, city,

street, home\_number, flat\_number, lastname, name, middle\_name):

db\_pointer = WithDB()

check = db\_pointer.insert\_delete\_alter\_smth('insert\_user',

[login,

password,

email,

phone\_number

]

)

if check:

user\_id = []

if not db\_pointer.get\_smth('get\_user\_id\_with\_login', [login], user\_id):

return False

check = db\_pointer.insert\_delete\_alter\_smth('insert\_customer',

[user\_id[0][0],

country,

street,

home\_number,

flat\_number,

lastname,

name,

middle\_name

]

)

if check:

try:

customer\_id = []

customers\_city = []

if not db\_pointer.get\_smth('get\_customer\_id\_with\_user\_id', [user\_id[0][0]], customer\_id):

return False

if not db\_pointer.get\_smth('get\_city\_id\_with\_city\_title', [city], customers\_city):

return False

check = db\_pointer.insert\_delete\_alter\_smth('insert\_customers\_city', [customer\_id[0][0],

customers\_city[0][0]

]

)

except:

print('MISTAKEEE!!!')

return False

if check:

role\_id = []

if not db\_pointer.get\_smth('get\_role\_id\_with\_role\_title', ['Клиент'], role\_id):

return False

check = db\_pointer.insert\_delete\_alter\_smth('insert\_users\_role',

[user\_id[0][0],

role\_id[0][0],

]

)

return check

return False

**User.py**

class User:

user\_id = None

current\_role\_id = None

**WithDB.py**

from MySQL import MySQL

class WithDB:

def \_\_init\_\_(self):

self.db\_title = 'coursework'

self.sql\_pointer = MySQL('localhost', 'root', 'root')

self.connection = None

self.queries = {'get\_user\_id\_with\_login\_and\_password':

"""

SELECT

user\_id

FROM

user

WHERE

login = \"%s\" AND password = \"%s\"

""",

'get\_role\_id\_with\_user\_id':

"""

SELECT

role\_id

FROM

users\_role

WHERE user\_id = %s

""",

'get\_role\_title\_with\_role\_id':

"""

SELECT title

FROM

role

WHERE role\_id = %s

""",

'get\_user\_id\_with\_phone\_and\_email':

"""

SELECT user\_id

FROM

user

WHERE phone = \"%s\" AND email = \"%s\"

""",

'get\_role\_id\_with\_role\_title':

"""

SELECT role\_id

FROM role

WHERE title = \"%s\"

""",

'insert\_stage':

"""INSERT INTO stage (

title,

description

)

VALUES ( \"%s\", \"%s\" )

""",

'insert\_status':

"""INSERT INTO status (

title,

description

)

VALUES ( \"%s\", \"%s\" )

""",

'insert\_transports\_kind':

"""INSERT INTO transports\_kind (

title,

description,

lifting\_capacity,

volume

)

VALUES ( \"%s\", \"%s\", %s, %s )

""",

'alter\_transports\_kind':

"""

UPDATE transports\_kind

SET title = \"%s\",

description = \"%s\",

lifting\_capacity = %s,

volume = %s

WHERE kind\_id = %s

""",

'insert\_city':

"""INSERT INTO city (

title

)

VALUES ( \"%s\" )

""",

'insert\_role':

"""INSERT INTO role (

title,

description

)

VALUES ( \"%s\", \"%s\" )

""",

'insert\_user':

"""INSERT INTO user (

login,

password,

email,

phone\_number

)

VALUES ( \"%s\", \"%s\", \"%s\", \"%s\" )

""",

'insert\_fleet':

"""INSERT INTO fleet (

title,

description,

address,

square,

stars\_quantity

)

VALUES ( \"%s\", \"%s\", \"%s\", %s, %s )

""",

'alter\_fleet':

"""

UPDATE fleet

SET title = \"%s\",

description = \"%s\",

address = \"%s\",

square = %s,

stars\_quantity = %s

WHERE fleet\_id = %s

""",

'insert\_service':

"""INSERT INTO service (

title,

description,

cost\_weight,

cost\_radius

)

VALUES ( \"%s\", \"%s\", %s, %s )

""",

'alter\_service':

"""

UPDATE service

SET title = \"%s\",

description = \"%s\",

cost\_weight = %s,

cost\_radius = %s,

WHERE service\_id = %s

""",

'insert\_customer':

"""INSERT INTO customer (

user\_id,

country,

street,

home\_number,

flat\_number,

lastname,

name,

middle\_name

)

VALUES ( %s, \"%s\", \"%s\", %s, %s, \"%s\", \"%s\", \"%s\" )

""",

'insert\_customers\_city':

"""INSERT INTO customers\_city (

customer\_id,

city\_id

)

VALUES ( %s, %s )

""",

'insert\_employee':

"""INSERT INTO employee (

user\_id,

passport\_data,

salary,

requirements,

duties,

status\_id

)

VALUES ( %s, \"%s\", %s, \"%s\", \"%s\", %s )

""",

'insert\_users\_role':

"""INSERT INTO users\_role (

user\_id,

role\_id

)

VALUES ( %s, %s )

""",

'insert\_fleets\_city':

"""INSERT INTO fleets\_city (

fleet\_id,

city\_id

)

VALUES ( %s, %s )

""",

'insert\_transport':

"""INSERT INTO transport (

kind\_id,

fleet\_id

)

VALUES ( %s, %s)

""",

'insert\_my\_order':

"""INSERT INTO my\_order (

commissions,

executions,

customer\_id,

operator\_id,

courier\_id,

stage\_id,

status\_id

)

VALUES ( %s, %s, %s, %s, %s, %s, %s )

""",

'insert\_my\_orders\_transport':

"""INSERT INTO orders\_transport (

transport\_id,

my\_order\_id

)

VALUES ( %s, %s )

""",

'insert\_my\_orders\_service':

"""INSERT INTO orders\_service (

service\_id,

my\_order\_id,

quantity\_weight,

quantity\_radius,

destinations\_address,

departures\_address,

total\_cost

)

VALUES ( %s, %s, %s, %s, \"%s\", \"%s\", %s )

""",

'get\_my\_orders\_service\_id':

"""

SELECT orders\_service\_id

FROM orders\_service

WHERE

service\_id = %s AND

my\_order\_id = %s AND

quantity\_weight = %s AND

quantity\_radius = %s AND

destinations\_address = \"%s\" AND

departures\_address = \"%s\" AND

total\_cost = %s

""",

'insert\_order\_services\_begin\_city':

"""INSERT INTO order\_services\_begin\_city (

city\_id,

orders\_service\_id

)

VALUES ( %s, %s )

""",

'insert\_order\_services\_end\_city':

"""INSERT INTO order\_services\_end\_city (

city\_id,

orders\_service\_id

)

VALUES ( %s, %s )

""",

'get\_city\_id\_with\_city\_title':

"""

SELECT city\_id FROM city WHERE title = \"%s\"

""",

'get\_user\_id\_with\_login':

"""

SELECT user\_id FROM user WHERE login = \"%s\"

""",

'get\_customer\_id\_with\_user\_id':

"""

SELECT customer\_id FROM customer WHERE user\_id = %s

""",

'delete\_user\_with\_login':

"""

DELETE FROM user WHERE login = \"%s\"

""",

'get\_city\_titles':

"""

SELECT title FROM city

""",

'get\_fleet\_titles':

"""

SELECT title FROM fleet

""",

'get\_service\_titles':

"""

SELECT title FROM service

""",

'get\_kind\_titles':

"""

SELECT title FROM transports\_kind

""",

'delete\_city\_with\_title':

"""

DELETE FROM city WHERE title = \"%s\"

""",

'alter\_city':

"""

UPDATE city

SET title = \"%s\"

WHERE city\_id = %s

""",

'get\_city\_with\_title':

"""

SELECT \* FROM city WHERE title = \"%s\"

""",

'get\_kind\_with\_title':

"""

SELECT \* FROM transports\_kind WHERE title = \"%s\"

""",

'get\_fleet\_with\_title':

"""

SELECT \* FROM fleet WHERE title = \"%s\"

""",

'get\_service\_with\_title':

"""

SELECT \* FROM service WHERE title = \"%s\"

""",

'get\_free\_couriers':

"""

SELECT

user.user\_id

FROM

user

INNER JOIN users\_role ON users\_role.user\_id = user.user\_id

INNER JOIN role ON users\_role.role\_id = role.role\_id

WHERE

role.title = \"Курьер\"

""",

'get\_services\_costs\_with\_title':

"""

SELECT

cost\_weight, cost\_radius

FROM

service

WHERE

title = \"%s\"

""",

'add\_order':

"""INSERT INTO my\_order (

commissions,

customer\_id,

stage\_id,

status\_id

)

VALUES ( \"%s\", %s, %s, %s )

""",

'get\_last\_orders\_id':

"""

SELECT

my\_order\_id

FROM

my\_order

WHERE

commissions = \"%s\" AND

executions IS NULL AND

customer\_id = %s AND

operator\_id IS NULL AND

courier\_id IS NULL AND

stage\_id = %s AND

status\_id = %s

""",

'add\_courier\_id\_and\_operator\_id\_into\_order\_with\_order\_id':

"""

UPDATE my\_order

SET

courier\_id = %s, operator\_id = %s, stage\_id = %s

WHERE my\_order\_id = %s

""",

'alter\_orders\_status\_id\_with\_order\_id':

"""

UPDATE my\_order

SET status\_id = %s

WHERE my\_order\_id = %s

""",

'add\_orders\_executions\_and\_stage\_id\_with\_order\_id':

"""

UPDATE my\_order

SET

executions = \"%s\", stage\_id = %s

WHERE my\_order\_id = %s

""",

'alter\_orders\_stage\_id\_with\_order\_id':

"""

UPDATE my\_order

SET stage\_id = %s

WHERE my\_order\_id = %s

""",

'get\_stage\_id\_with\_stage\_title':

"""

SELECT

stage\_id

FROM

stage

WHERE title = \"%s\"

""",

'get\_status\_id\_with\_status\_title':

"""

SELECT

status\_id

FROM

status

WHERE title = \"%s\"

""",

'get\_active\_orders\_data\_for\_customer\_with\_customer\_id':

"""

SELECT

my\_order.my\_order\_id,

my\_order.commissions,

my\_order.executions,

status.title,

stage.title

FROM

my\_order

INNER JOIN status ON status.status\_id = my\_order.status\_id

INNER JOIN stage ON stage.stage\_id = my\_order.stage\_id

WHERE

(stage.title = \"На рассмотрении\" OR

stage.title = \"Выполняется\") AND

my\_order.customer\_id = %s

""",

'get\_passive\_orders\_data\_for\_customer\_with\_customer\_id':

"""

SELECT

my\_order.my\_order\_id,

my\_order.commissions,

my\_order.executions,

status.title,

stage.title

FROM

my\_order

INNER JOIN status ON status.status\_id = my\_order.status\_id

INNER JOIN stage ON stage.stage\_id = my\_order.stage\_id

WHERE

(stage.title <> \"На рассмотрении\" AND

stage.title <> \"Выполняется\") AND

my\_order.customer\_id = %s

""",

'get\_paid\_orders':

"""

SELECT my\_order.my\_order\_id

FROM

my\_order

INNER JOIN stage ON stage.stage\_id = my\_order.stage\_id

INNER JOIN status ON status.status\_id = my\_order.status\_id

WHERE

stage.title = \"На рассмотрении\" AND status.title = \"Оплачен\"

""",

'get\_order\_with\_courier\_id':

"""

SELECT my\_order.my\_order\_id

FROM

my\_order

INNER JOIN stage ON stage.stage\_id = my\_order.stage\_id

WHERE

courier\_id = %s AND

stage.title <> \"Выполнен\"

""",

'get\_sum\_profit\_for\_each\_month':

"""

SELECT

my\_order.executions, Sum(orders\_service.total\_cost)

FROM

my\_order

INNER JOIN orders\_service ON orders\_service.my\_order\_id = my\_order.my\_order\_id

GROUP BY my\_order.executions

""",

'get\_quantity\_of\_each\_services\_item':

"""

SELECT Count(service.service\_id)

FROM

service INNER JOIN

orders\_service ON orders\_service.service\_id = service.service\_id

ORDER BY service.service\_id

"""

}

def \_\_connect\_db(self):

self.connection = self.sql\_pointer.connect\_database(self.db\_title)

if self.connection is None:

return False

return True

def get\_smth(self, query\_title, arguments, outcome):

if not self.\_\_connect\_db():

return False

try:

res = self.sql\_pointer.execute\_read\_query(self.connection,

self.queries[query\_title] % tuple(arguments)

)

if res is not None:

for i in range(len(res)):

outcome.append(res[i])

return True

except BaseException as e:

return False

def insert\_delete\_alter\_smth(self, query\_title, arguments):

if not self.\_\_connect\_db():

return False

try:

self.sql\_pointer.execute\_query(self.connection,

self.queries[query\_title] % tuple(arguments)

)

return True

except BaseException as e:

return False

**Courier.kv**

<Curier> :

label: label

btn1: btn1

btn2: btn2

md\_bg\_color: [40/255, 40/255, 180/255, 1]

Card :

spacing: 50

Widget:

height: 10

MyLabel:

id: label

text: 'Заказ №...'

MyButton:

id: btn1

text: 'Поставить отметку выполнено'

on\_press: root.completed()

MyButton:

id: btn2

text: 'Сведения о заказе'

on\_press: root.info()

GridLayout:

size\_hint: (1, .05)

valign: 'top'

cols: 3

rows: None

spacing: 10

AnchorLayout:

MDIconButton :

user\_font\_size: "20sp"

icon\_color: [1, 1, 1, 1]

icon: 'arrow-left'

md\_bg\_color: [249 / 255, 166 / 255, 2 / 255, 255 / 255]

on\_press: root.manager.current = 'choose\_role'

AnchorLayout:

MDIconButton :

user\_font\_size: "20sp"

icon\_color: [1, 1, 1, 1]

icon: 'arrow-up'

md\_bg\_color: [249 / 255, 166 / 255, 2 / 255, 255 / 255]

on\_press: root.load\_data()

AnchorLayout:

MDIconButton :

user\_font\_size: "20sp"

icon\_color: [1, 1, 1, 1]

icon: 'door'

md\_bg\_color: [249 / 255, 166 / 255, 2 / 255, 255 / 255]

on\_press: app.stop()

**Customer.kv**

<Customer> :

md\_bg\_color: [40 / 255, 40 / 255, 180 / 255, 80 / 255]

Card :

spacing: 120

padding: 40

GridLayout:

valign: 'center'

cols: 2

rows: None

AnchorLayout:

MDIconButton :

user\_font\_size: "30sp"

icon\_color: [1, 1, 1, 1]

icon: 'arrow-left'

md\_bg\_color: [10 / 255, 10 / 255, 140 / 255, 255 / 255]

on\_press: root.manager.current = 'login'

AnchorLayout:

MDIconButton :

user\_font\_size: "30sp"

icon\_color: [1, 1, 1, 1]

icon: 'door'

md\_bg\_color: [10 / 255, 10 / 255, 140 / 255, 255 / 255]

on\_press: app.stop()

Widget:

height: 100

MyButton :

text : 'Совершить заказ'

md\_bg\_color: [10 / 255, 10 / 255, 140 / 255, 100 / 255]

on\_press: root.manager.current = 'customer\_order'

MyButton :

text : 'Мои заказы'

md\_bg\_color: [10 / 255, 10 / 255, 140 / 255, 100 / 255]

on\_press: root.manager.current = 'my\_orders'

Widget :

height : 100

**CustomerOrder.kv**

<CustomerOrder>:

cont: cont

BoxLayout:

AnchorLayout:

Card:

size\_hint: .8, .85

pos\_hint: {'center\_x': .5, 'center\_y': .5}

md\_bg\_color: [255/255, 255/255, 255/255, 0.8]

ScrollView:

BoxLayout:

id: cont

orientation: "vertical"

size\_hint\_y: None

height: self.minimum\_height

MDTextButton:

text: 'game 1\_'

heigh: dp(60)

GridLayout:

size\_hint: (1, .05)

valign: 'top'

cols: 2

rows: None

spacing: 10

AnchorLayout:

MDIconButton :

user\_font\_size: "20sp"

icon\_color: [1, 1, 1, 1]

icon: 'arrow-left'

md\_bg\_color: [249 / 255, 166 / 255, 2 / 255, 255 / 255]

on\_press: root.manager.current = 'customer'

AnchorLayout:

MDIconButton :

user\_font\_size: "20sp"

icon\_color: [1, 1, 1, 1]

icon: 'door'

md\_bg\_color: [249 / 255, 166 / 255, 2 / 255, 255 / 255]

on\_press: app.stop()

Widget:

Widget:

**Director.kv**

<Director> :

md\_bg\_color: [40 / 255, 40 / 255, 180 / 255, 80 / 255]

Card :

spacing: 120

padding: 40

GridLayout:

valign: 'center'

cols: 2

rows: None

AnchorLayout:

MDIconButton :

user\_font\_size: "30sp"

icon\_color: [1, 1, 1, 1]

icon: 'arrow-left'

md\_bg\_color: [10 / 255, 10 / 255, 140 / 255, 255 / 255]

on\_press: root.manager.current = 'login'

AnchorLayout:

MDIconButton :

user\_font\_size: "30sp"

icon\_color: [1, 1, 1, 1]

icon: 'door'

md\_bg\_color: [10 / 255, 10 / 255, 140 / 255, 255 / 255]

on\_press: app.stop()

Widget:

height: 100

MyButton :

text : 'Отчеты'

md\_bg\_color: [10 / 255, 10 / 255, 140 / 255, 100 / 255]

on\_release: root.manager.current = 'reports'

MyButton :

text : 'Справичники, прейскурант'

md\_bg\_color: [10 / 255, 10 / 255, 140 / 255, 100 / 255]

on\_release: root.manager.current = 'handbooks'

Widget:

height: 100

**GetPassword.kv**

<GetPassword>:

phone: phone

email: email

md\_bg\_color: [40/255, 40/255, 180/255, 1]

Card :

spacing: 50

Widget:

height: 10

MyButton :

text: '<- НАЗАД'

on\_press: root.manager.current = 'login'

MyLabel :

underline: True

font\_size: 15

multiline: True

text : 'Для восстановления пароля введите необходимые данные.\nНа указанный номер телефона и почту будет отправлено сообщение с паролем от Вашего аккаунта.'

MyTextField :

id: phone

hint\_text : "телефон"

MyTextField:

id: email

hint\_text : "email"

MyIconButton :

text : 'Получить пароль'

icon: 'email'

on\_press: root.send\_password()

Widget:

height: 100

**HandBooks.kv**

<HandBooks> :

cont\_cities: cont\_cities

cont\_fleets: cont\_fleets

cont\_services: cont\_services

cont\_kinds: cont\_kinds

md\_bg\_color: [1, 1, 1, 1]

BoxLayout:

TabbedPanel:

border: [0, 0, 0, 0]

background\_image: 'pictures/white.jpg'

do\_default\_tab: False

TabbedPanelItem:

text: 'Города'

background\_color: [0, 1, 0, .9]

md\_bg\_color: [10 / 255, 10 / 255, 140 / 255, 100 / 255]

AnchorLayout:

Card:

size\_hint: .8, .85

pos\_hint: {'center\_x': .5, 'center\_y': .5}

md\_bg\_color: [255/255, 255/255, 255/255, 0.8]

ScrollView:

BoxLayout:

id: cont\_cities

orientation: "vertical"

size\_hint\_y: None

height: self.minimum\_height

MDTextButton:

text: 'game 1\_'

heigh: dp(60)

TabbedPanelItem:

text: 'Услуги'

background\_color: [0, 0, 1, .9]

md\_bg\_color: [10 / 255, 10 / 255, 140 / 255, 100 / 255]

AnchorLayout:

Card:

size\_hint: .8, .85

pos\_hint: {'center\_x': .5, 'center\_y': .5}

md\_bg\_color: [255/255, 255/255, 255/255, 0.8]

ScrollView:

BoxLayout:

id: cont\_services

orientation: "vertical"

size\_hint\_y: None

height: self.minimum\_height

MDTextButton:

text: 'game 1\_'

heigh: dp(60)

TabbedPanelItem:

text: 'Транспорт'

background\_color: [128/255, 128/255, 0 , .9]

md\_bg\_color: [10 / 255, 10 / 255, 140 / 255, 100 / 255]

AnchorLayout:

Card:

size\_hint: .8, .85

pos\_hint: {'center\_x': .5, 'center\_y': .5}

md\_bg\_color: [255/255, 255/255, 255/255, 0.8]

ScrollView:

BoxLayout:

id: cont\_kinds

orientation: "vertical"

size\_hint\_y: None

height: self.minimum\_height

MDTextButton:

text: 'game 1\_'

heigh: dp(60)

TabbedPanelItem:

text: 'Парки'

background\_color: [0, 128/255, 128/255, .9]

md\_bg\_color: [10 / 255, 10 / 255, 140 / 255, 100 / 255]

AnchorLayout:

Card:

size\_hint: .8, .85

pos\_hint: {'center\_x': .5, 'center\_y': .5}

md\_bg\_color: [255/255, 255/255, 255/255, 0.8]

ScrollView:

BoxLayout:

id: cont\_fleets

orientation: "vertical"

size\_hint\_y: None

height: self.minimum\_height

MDTextButton:

text: 'game 1\_'

heigh: dp(60)

GridLayout:

size\_hint: (1, .05)

valign: 'top'

cols: 3

rows: None

spacing: 10

AnchorLayout:

MDIconButton :

user\_font\_size: "20sp"

icon\_color: [1, 1, 1, 1]

icon: 'arrow-left'

md\_bg\_color: [249 / 255, 166 / 255, 2 / 255, 255 / 255]

on\_press: root.manager.current = 'director'

AnchorLayout:

MDIconButton :

user\_font\_size: "20sp"

icon\_color: [1, 1, 1, 1]

icon: 'arrow-up'

md\_bg\_color: [249 / 255, 166 / 255, 2 / 255, 255 / 255]

on\_press: root.load\_data()

AnchorLayout:

MDIconButton :

user\_font\_size: "20sp"

icon\_color: [1, 1, 1, 1]

icon: 'door'

md\_bg\_color: [249 / 255, 166 / 255, 2 / 255, 255 / 255]

on\_press: app.stop()

Widget:

Widget:

**LogIn.kv**

<LogIn>:

my\_login: my\_login

my\_password: my\_password

md\_bg\_color: [40/255, 40/255, 180/255, 255 /255]

Card:

ButtonExit:

on\_press: app.stop()

MyImage:

source: 'pictures/1.jpg'

MyLabel:

text: 'Добро пожаловать!'

font\_size: 30

MyTextField:

id: my\_login

hint\_text : "логин"

icon\_right: 'login'

MyTextField:

id: my\_password

hint\_text : "пароль"

icon\_right: 'eye-off'

password: True

MyButton:

id: lets\_go

text : 'ВОЙТИ'

on\_press : root.button\_login()

MyButton:

text: 'Забыли пароль?'

on\_press: root.manager.current = 'get\_password'

MyIconButton:

icon: "account-plus"

text: 'Зарегистрироваться'

on\_press: root.manager.current = 'sign\_up'

**My.kv**

<Card@MDCard>:

size\_hint: (.7, .8)

pos\_hint: {"center\_x": .5, "center\_y": .5}

elevation: 15

padding: 20

spacing: 20

orientation: 'vertical'

md\_bg\_color: [10 / 255, 10 / 255, 140 / 255, 100 / 255]

<ButtonExit@MDFillRoundFlatButton>:

pos\_hint: {"center\_x": .5}

font\_size: 15

md\_bg\_color: [40 / 255, 40 / 255, 180 / 255, 100 / 255]

text\_color: [1, 1, 1, 1]

text: '<- ВЫЙТИ ИЗ ПРИЛОЖЕНИЯ'

<MyImage@Image>:

pos\_hint: {"center\_x": .5, "center\_y": .5}

allow\_stretch: True

size\_hint: (1, 1)

<MyLabel@MDLabel>:

font\_style: 'Button'

color: [1, 1, 1, 200 / 255]

halign: 'center'

size\_hint\_y: None

<MyTextField@MDTextField>:

pos\_hint: {"center\_x": .5}

icon\_right\_color: [1, 1, 1, 1]

helper\_text: 'Check the field!'

helper\_text\_mode: 'on\_error'

color\_mode: 'primary'

mode: 'rectangle'

size\_hint\_y: None

size\_hint\_x: None

height: 40

font\_size: 20

width: 220

<MyButton@MDFillRoundFlatButton>:

pos\_hint: {"center\_x": .5}

font\_size: 15

md\_bg\_color: [40 / 255, 40 / 255, 180 / 255, 100 / 255]

text\_color: [1, 1, 1, 1]

size\_hint\_x: None

<MyIconButton@MDFillRoundFlatIconButton>:

pos\_hint: {"center\_x": .5}

font\_size: 15

md\_bg\_color: [40 / 255, 40 / 255, 180 / 255, 100 / 255]

text\_color: [1, 1, 1, 1]

size\_hint\_x: None

<MyCheckBox@CheckBox>:

size\_hint: None, None

md\_bg\_color: [1 / 255, 1 / 255, 1 / 255, 1]

size: "48dp", "48dp"

pos\_hint: {'center\_x': .5, 'center\_y': .5}

active: False

**MyOrders.kv**

<MyOrders>:

cont1: cont1

cont2: cont2

BoxLayout:

AnchorLayout:

GridLayout:

cols: 2

rows: None

Card:

size\_hint: .8, .85

pos\_hint: {'center\_x': .5, 'center\_y': .5}

md\_bg\_color: [255/255, 255/255, 255/255, 0.8]

MyLabel:

color: [1, 0, 0, 1]

text: 'Активные заказы'

ScrollView:

BoxLayout:

id: cont1

orientation: "vertical"

size\_hint\_y: None

height: self.minimum\_height

MDTextButton:

text: 'game 1\_'

heigh: dp(60)

Card:

size\_hint: .8, .85

pos\_hint: {'center\_x': .5, 'center\_y': .5}

md\_bg\_color: [255/255, 255/255, 255/255, 0.8]

MyLabel:

color: [1, 0, 0, 1]

text: 'Неактивные заказы'

ScrollView:

BoxLayout:

id: cont2

orientation: "vertical"

size\_hint\_y: None

height: self.minimum\_height

MDTextButton:

text: 'game 1\_'

heigh: dp(60)

GridLayout:

size\_hint: (1, .05)

valign: 'top'

cols: 3

rows: None

spacing: 10

AnchorLayout:

MDIconButton :

user\_font\_size: "20sp"

icon\_color: [1, 1, 1, 1]

icon: 'arrow-left'

md\_bg\_color: [249 / 255, 166 / 255, 2 / 255, 255 / 255]

on\_press: root.manager.current = 'customer'

AnchorLayout:

MDIconButton :

user\_font\_size: "20sp"

icon\_color: [1, 1, 1, 1]

icon: 'arrow-up'

md\_bg\_color: [249 / 255, 166 / 255, 2 / 255, 255 / 255]

on\_press: root.load\_data()

AnchorLayout:

MDIconButton :

user\_font\_size: "20sp"

icon\_color: [1, 1, 1, 1]

icon: 'door'

md\_bg\_color: [249 / 255, 166 / 255, 2 / 255, 255 / 255]

on\_press: app.stop()

Widget:

Widget:

**Operator.kv**

<Operator> :

cont1: cont1

cont2: cont2

BoxLayout:

orientation: 'vertical'

AnchorLayout:

GridLayout:

cols: 2

rows: None

Card:

size\_hint: .8, .85

pos\_hint: {'center\_x': .5, 'center\_y': .5}

md\_bg\_color: [255/255, 255/255, 255/255, 0.8]

MyLabel:

color: [1, 0, 0, 1]

text: 'Курьеры'

ScrollView:

BoxLayout:

id: cont1

orientation: "vertical"

size\_hint\_y: None

height: self.minimum\_height

MDTextButton:

text: 'game 1\_'

heigh: dp(60)

Card:

size\_hint: .8, .85

pos\_hint: {'center\_x': .5, 'center\_y': .5}

md\_bg\_color: [255/255, 255/255, 255/255, 0.8]

MyLabel:

color: [1, 0, 0, 1]

text: 'Заказы'

ScrollView:

BoxLayout:

id: cont2

orientation: "vertical"

size\_hint\_y: None

height: self.minimum\_height

MDTextButton:

text: 'game 1\_'

heigh: dp(60)

AnchorLayout:

size\_hint: (1, .3)

MyButton:

text: 'Привязать'

on\_press: root.link()

GridLayout:

size\_hint: (1, .05)

valign: 'top'

cols: 3

rows: None

spacing: 10

AnchorLayout:

MDIconButton :

user\_font\_size: "20sp"

icon\_color: [1, 1, 1, 1]

icon: 'arrow-left'

md\_bg\_color: [249 / 255, 166 / 255, 2 / 255, 255 / 255]

on\_press: root.manager.current = 'choose\_role'

AnchorLayout:

MDIconButton :

user\_font\_size: "20sp"

icon\_color: [1, 1, 1, 1]

icon: 'arrow-up'

md\_bg\_color: [249 / 255, 166 / 255, 2 / 255, 255 / 255]

on\_press: root.load\_data()

AnchorLayout:

MDIconButton :

user\_font\_size: "20sp"

icon\_color: [1, 1, 1, 1]

icon: 'door'

md\_bg\_color: [249 / 255, 166 / 255, 2 / 255, 255 / 255]

on\_press: app.stop()

Widget:

Widget:

**Reports.py**

<Reports> :

statistic: statistic

volume\_per\_month: volume\_per\_month

Accordion:

orientation: 'vertical'

AccordionItem:

title: 'Статистика'

GridLayout:

padding: 80

spacing: 20

pos\_hint: {"center\_x": .5, "center\_y": .5}

cols: 2

rows: None

spacing: '8dp'

size\_hint\_y: None

height: self.minimum\_height

id: statistic

Label:

text: '1'

AccordionItem:

title: 'Объемы услуг за месяц'

GridLayout:

padding: 80

spacing: 20

pos\_hint: {"center\_x": .5, "center\_y": .5}

cols: 2

rows: None

spacing: '8dp'

size\_hint\_y: None

height: self.minimum\_height

id: volume\_per\_month

Label:

text: '5'

GridLayout:

size\_hint: (1, .05)

valign: 'top'

cols: 3

rows: None

spacing: 10

AnchorLayout:

MDIconButton :

user\_font\_size: "20sp"

icon\_color: [1, 1, 1, 1]

icon: 'arrow-left'

md\_bg\_color: [249 / 255, 166 / 255, 2 / 255, 255 / 255]

on\_press: root.manager.current = 'director'

AnchorLayout:

MDIconButton :

user\_font\_size: "20sp"

icon\_color: [1, 1, 1, 1]

icon: 'arrow-up'

md\_bg\_color: [249 / 255, 166 / 255, 2 / 255, 255 / 255]

on\_press: root.load\_data()

AnchorLayout:

MDIconButton :

user\_font\_size: "20sp"

icon\_color: [1, 1, 1, 1]

icon: 'door'

md\_bg\_color: [249 / 255, 166 / 255, 2 / 255, 255 / 255]

on\_press: app.stop()

Widget:

Widget:

**SignUp.kv**

<SignUp>:

last\_btn: last\_btn

field\_country: field\_country

field\_city: field\_city

field\_street: field\_street

filed\_house: filed\_house

field\_flat: field\_flat

surname: surname

user\_name: user\_name

middle\_name: middle\_name

email: email

phone: phone

login: login

password1: password1

password2: password2

check\_box: check\_box

PageLayout:

Card:

md\_bg\_color: [255/255, 255/255, 255/255, 0.8]

Widget:

height: 20

MyButton:

md\_bg\_color: [1/255, 1/255, 255/255, 150 / 255]

text: '<- НАЗАД'

on\_press: root.manager.current = 'login'

MyLabel:

text: "Откуда Вы"

font\_size: 30

color: [40 / 255, 40 / 255, 180 / 255, 150 / 255]

MyTextField:

id: field\_country

hint\_text: "Страна"

MyTextField:

id: field\_city

hint\_text: "Город"

MyLabel:

font\_size: 30

color: [40 / 255, 40 / 255, 180 / 255, 150 / 255]

MyTextField:

id: field\_street

hint\_text: "Улица"

MyTextField:

id: filed\_house

hint\_text: "Дом"

MyTextField:

id: field\_flat

hint\_text: "Квартира"

Widget:

height: 20

Card:

md\_bg\_color: [255/255, 255/255, 255/255, 0.8]

Widget:

height: 20

MyLabel:

text: "Контактные данные"

font\_size: 30

color: [40 / 255, 40 / 255, 180 / 255, 150 / 255]

MyTextField:

id: surname

hint\_text: "Фамилия"

MyTextField:

id: user\_name

hint\_text : "Имя"

MyTextField :

id: middle\_name

hint\_text : "Отчество"

MyTextField:

id: email

hint\_text: "Email"

MyTextField:

id: phone

hint\_text: "Телефон"

Widget:

height: 20

Card:

md\_bg\_color: [255/255, 255/255, 255/255, 0.8]

Widget:

height: 20

MyLabel:

text: "Данные для авторизации"

font\_size: 30

color: [40 / 255, 40 / 255, 180 / 255, 150 / 255]

MyTextField:

id: login

hint\_text: "Логин"

MyTextField:

id: password1

hint\_text: "Пароль"

password: True

MyTextField:

id: password2

hint\_text: "Повторите пароль"

password: True

MyCheckBox:

id: check\_box

on\_active: root.on\_checkbox\_active()

MyLabel:

text: "Принимаю условия Политики конфиденциальности и даю согласие на обработку моих персональных данных."

font\_size: 15

underline: True

color: [40 / 255, 40 / 255, 180 / 255, 150 / 255]

MyButton:

id: last\_btn

md\_bg\_color: [1/255, 1/255, 255/255, 150 / 255]

text : 'Зарегистрироваться'

disabled: True

on\_press: root.button\_sign\_up()

Widget:

height: 20