

Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет  
информатики и радиоэлектроники»

Факультет компьютерного проектирования  
Кафедра проектирования информационно-компьютерных систем  
Дисциплина «Структуры и базы данных»

«К ЗАЩИТЕ ДОПУСТИТЬ»

Руководитель курсового проекта  
Магистр технических наук,  
ассистент

\_\_\_\_\_.\_\_\_\_\_.2021 А.Д. Сыс

### **ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проекту

на тему:

**«БАЗА ДАННЫХ ДЛЯ ПОДДЕРЖКИ ПРОИЗВОДСТВА ПК»**

БГУИР КП 1-39 03 02 017 ПЗ

Выполнил студент группы 913802  
Марцев Артем Сергеевич

\_\_\_\_\_  
(подпись студента)

Курсовой проект представлен на  
проверку \_\_\_\_\_.\_\_\_\_\_.2021

\_\_\_\_\_  
(подпись студента)

Минск 2021

## РЕФЕРАТ

БГУИР КП 1-39 03 02 017 ПЗ

**Марцев А.С.** База данных для поддержки работы производства ПК: пояснительная записка к курсовому проекту / А. С. Марцев. – Минск: БГУИР, 2021. – 44 с.

Пояснительная записка 44 с., 26 рис., 6 источников, 4 приложения.

АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ И ЕЁ ФОРМАЛИЗАЦИИ ДЛЯ ПРОЕКТИРОВАНИЯ БАЗЫ ДАННЫХ, ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ ДЛЯ ОСНОВНОГО ВИДА ДЕЯТЕЛЬНОСТИ РАССМАТРИВАЕМОЙ ПРЕДМЕТНОЙ ОБЛАСТИ, ПРИМЕНЕНИЕ РАЗРАБОТАННОЙ БАЗЫ ДАННЫХ

*Цель проектирования:* разработка *Android*-приложения, проектирование эффективной и безопасной базы данных для поддержки работы производства ПК.

*Методология проведения работы:* в процессе решения поставленных задач использованы принципы системного подхода, теория проектирования базы данных, методы проектирования *Android*-приложений.

*Результаты работы:* изучены способы хранения информации в базе данных, проведен анализ реляционной модели данных, разработано *Android* - приложение для производства ПК

*Область применения результатов:* разработанная база данных и *Android*-приложение может использоваться в производствах ПК для организации эффективной и комфортной работы сотрудников.

## СОДЕРЖАНИЕ

|  |    |
|--|----|
| Перечень условных обозначений, символов и терминов .....                                       | 6  |
| Введение .....   | 7  |
| 1 Анализ предметной области и её формализация для проектирования базы данных .....             | 8  |
| 1.1 Описание предметной области .....  | 8  |
| 1.2 Анализ информационных потребностей пользователей и предварительное описание запросов ..... | 10 |
| 1.3 Определение требований и ограничений к базе данных с точки зрения предметной области ..... | 10 |
| 1.4 Постановка решаемой задачи .....   | 11 |
| 2 Проектирование базы для основного вида деятельности рассматриваемой предметной области ..... | 12 |
| 2.1 Разработка инфологической модели предметной области базы данных ....                       | 12 |
| 2.2 Выбор и обоснование используемых типов данных и ограничений (доменов) .....                | 15 |
| 2.3 Проектирование запросов к базе данных .....  | 16 |
| 2.4 Программная реализация и документирование базы данных .....                                | 18 |
| 3 Применение разработанной базы данных .....   | 20 |
| 3.1 Руководство пользователя.....  | 20 |
| 3.2 Администрирование базы данных .....  | 29 |
| 3.3 Реализация клиентских запросов .....   | 30 |
| 3.4 Обоснование и реализация механизма обеспечения безопасности и сохранности данных .....     | 30 |
| Заключение .....   | 31 |
| Список используемых источников.....  | 32 |
| Приложение А (обязательное) Отчёт о проверке на уникальность в системе «Антиплагиат» .....     | 33 |
| Приложение Б (обязательное) Скрипт генерации БД .....  | 34 |
| Приложение В (обязательное) Листинг программного кода.....                                     | 40 |
| Приложение Г (обязательное) Ведомость курсового проекта .....                                  | 43 |

## ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ, СИМВОЛОВ И ТЕРМИНОВ

БД – База данных

СУБД – Система управления базами данных

*API – Application Programming Interface*

*REST – Representational state transfer*

*JSON – JavaScript Object Notation*

*HTTPS – Hypertext Transport Protocol Secure*

*Retrofit* – Библиотека для работы с сетью

## ВВЕДЕНИЕ

По данным на 2020 год 74 процента людей по всему миру имеют доступ к персональным компьютерам, а в развитых странах эта цифра достигает 92 процентов. В двадцать первом веке, который принято называть веком информационных технологий, жизнь практически каждого человека из развитых стран напрямую связано с постоянным использованием компьютерной техники, как и персональных компьютеров в частности.

Под компьютерными технологиями скрывается большое множество различных понятий, таких как:

- мобильные гаджеты;
- интернет-технологии;
- персональные компьютеры;
- различные умные гаджеты и так далее.

Говоря про отрасль производства персональных компьютерных, нельзя не отметить, что в последние годы она стала одной из самых востребованных, так как спрос на всевозможные компьютерные комплектующие значительно превышает предложение, что можно с легкостью отследить по росту цен на них, а также благодаря заявлениям самих компаний-производителей.

Никакая современная компания не обходится без баз данных, и компании-производители компьютерных комплектующих, конечно, не являются исключением, так как базы данных выполняют крайне важные функции, а именно хранение всевозможной информации, администрированию и управлению данными.

Целью данного курсового проекта является проектирование базы данных для поддержки производства персональных компьютеров.

При выполнении курсового проекта были учтены потребности пользователей, связанные с различными взаимодействиями с созданной базой данных: регистрация клиента, создание заказа, просмотр сотрудников, состояний заказов, добавление новых комплектующих, готовых сборок, и другие.

# **1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ И ЕЁ ФОРМАЛИЗАЦИЯ ДЛЯ ПРОЕКТИРОВАНИЯ БАЗЫ ДАННЫХ**

Деятельность, направленная на выявление реальных потребностей заказчика, а также на выяснения смысла высказанных требований, называется анализом предметной области. Анализ предметной области – это первый шаг этапа системного анализа, с которого начинается разработка программной системы.

Формализация предметной области заключается в построении ее модели и разработке методов преобразования модели предметной области в модель исполнителя.

## **1.1 Описание предметной области**

Предметная область – часть реального мира, подлежащая изучению с целью организации управления и в конечном итоге автоматизации. Она содержит в себе информацию, которую необходимо формализовать для внесения в базу данных.

По причине того, что, данная БД предназначена для поддержки производства персональных компьютеров, то прежде всего нужно узнать, какие потребности есть к данной базе данных. При чем надо учитывать, что база данных реализуется как для клиентов, так и для администраторов, исполнителей заказов, что вносит некоторые коррективы и дополнительные требования к базе данных.

Для полноценного функционирования производства персональных компьютеров в базе данных был создан список сущностей, полностью обеспечивающих функционирование производства персональных компьютеров.

Сущность «Клиенты» необходима для хранения данных о клиентах производства персональных компьютеров. Данная сущность обладает следующими атрибутами:

- уникальный идентификатор;
- имя;
- фамилия;
- электронная почта;
- номер телефона;
- пароль для входа.

Для хранения данных о заказе конкретного клиента была создана сущность «Заказы», обладающая следующими атрибутами:

- уникальный идентификатор;
- адрес;

Для отслеживания выбранного способа доставки была создана сущность «Способы доставки», содержащая следующие атрибуты:

- уникальный идентификатор;
- тип доставки.

Кроме отслеживания выбранного способа доставки, нужно отслеживать выбранные способ оплаты. Для этого была создана сущность «Способы оплаты», содержащая следующие атрибуты:

- уникальный идентификатор;
- тип оплаты.

Сущность «Компьютер» предназначена для хранения информации о компьютере, в неё входят следующие атрибуты:

- уникальный идентификатор;
- название;
- стоимость.

Сущность «Способы сборки» создана для возможности выбора пользователем разных вариантов конфигурации компьютера, и содержит следующие атрибуты:

- уникальный идентификатор;
- тип сборки.

Сущность «Работники» предназначена для хранения информации о работниках производства. Данная сущность содержит следующие атрибуты:

- уникальный идентификатор;
- имя;
- фамилия;
- отчество;
- адрес проживания;
- номер телефона;
- электронная почта;
- пароль для входа.

Так как на любом предприятии есть множество должностей, то их хранения была создана сущность «Должности», в которой содержатся атрибуты:

- уникальный идентификатор;
- название.

Для хранения информации о комплектующих, содержащихся на производстве, была создана сущность «Комплектующие» с следующими атрибутами:

- уникальный идентификатор;
- название;
- характеристики;
- стоимость.

С целью хранения информации о типах комплектующих была создана сущность «Типы комплектующих», в которой содержатся следующие атрибуты:

- уникальный идентификатор;
- тип комплектующей.

## **1.2 Анализ информационных потребностей пользователей и предварительное описание запросов**

Так как разрабатываемое приложение предназначено, как и для сотрудников, так и для клиентов компании, то и функционал будет значительно отличаться.

Клиент имеет доступ к регистрации и просмотру списка заказов.

Для сотрудников с правами доступа уровня администратора функционал будет значительно больше, в него также входят такие функции, как создание, редактирование и удаление всех таблиц в базе данных.

## **1.3 Определение требований и ограничений к базе данных с точки зрения предметной области**

База данных представляет собой совокупность структурированных взаимосвязанных данных, относящихся к определенной предметной области и организованных для решения определенных задач разными пользователями.

Особенности каждой предметной области могут накладывать разнообразные требования и ограничения на хранимые в базе данные.

На основе выбранной предметной области на созданную базу данных накладываются следующие ограничения:



- все электронные почты должны быть уникальными;
- все телефонные номера должны быть уникальными;
- названия должностей, типы способов доставки и способы оплаты, сборки должны быть уникальными;
- текстовое поле для ввода номера имеет фиксированное значение по длине в двенадцать символов;
- все цены должны быть указаны в целочисленных значениях;
- все текстовые поля, за исключением текстовых полей для ввода номера телефона и адресов, имеют максимальное значение по длине в двадцать символов.

#### **1.4 Постановка решаемой задачи**

Основной целью курсового проекта является проектирование и создание базы данных для поддержки работы производства персональных компьютеров. Проектируемая база данных должна отвечать требованиям надежности, минимальной избыточности, целостности данных и ее схема должна быть приведена к третьей нормальной форме. База данных должна поддерживать основные современные средства для работы и администрирования. Также необходимо реализовать клиентское приложение для взаимодействия с разработанной базой данных.

В качестве клиента для данного курсового проекта было выбрано мобильное приложение на базе системы Android, а в качестве СУБД *MySQL*. При разработке мобильного приложения было нужно реализовать взаимодействие между СУБД и самим приложением благодаря разработанному интерфейсу доступа к данным – *REST API*.

В результате курсового проектирования ожидается создание базы данных для поддержки производства персональных компьютеров, взаимодействие с которой осуществляется при помощи мобильного приложения.

## 2 ПРОЕКТИРОВАНИЕ БАЗЫ ДЛЯ ОСНОВНОГО ВИДА ДЕЯТЕЛЬНОСТИ РАССМАТРИВАЕМОЙ ПРЕДМЕТНОЙ ОБЛАСТИ

### 2.1 Разработка инфологической модели предметной области базы данных

Проектирование БД – одна из наиболее сложных и ответственных задач, связанных с созданием информационной системы. В результате решения этой задачи должны быть определены содержание БД, эффективный для всех ее будущих пользователей способ организации данных и инструментальные средства управления данными.

Цель инфологического моделирования – обеспечение наиболее естественных для человека способов сбора и представления той информации, которую предполагается хранить в создаваемой базе данных. Храниться данные будут в диаграмме «сущность – связь», которая будет создана с помощью СУБД.

Диаграмма «сущность – связь» (ER-диаграмма) позволяет графически представить все элементы информационной модели согласно простым, интуитивно понятным, но строго определенным правилам – нотациям.

При разработке базы данных проводилась в программе-инструменте для визуального проектирования баз данных *MySQL Workbench*.

Создание новой модели производится при нажатии на пункт «*New Model*» в выпадающем меню пункта «*File*», как показано на рисунке 2.1, либо же благодаря нажатии комбинации клавиш «*Ctrl + N*».

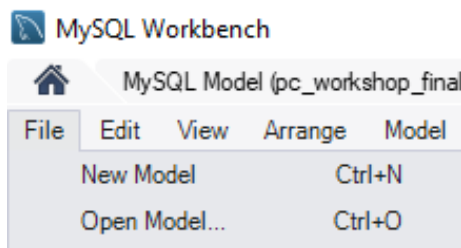


Рисунок 2.1 – Создание новой модели

При создании новой модели, открывается окно создания диаграммы, представленное на рисунке 2.2.

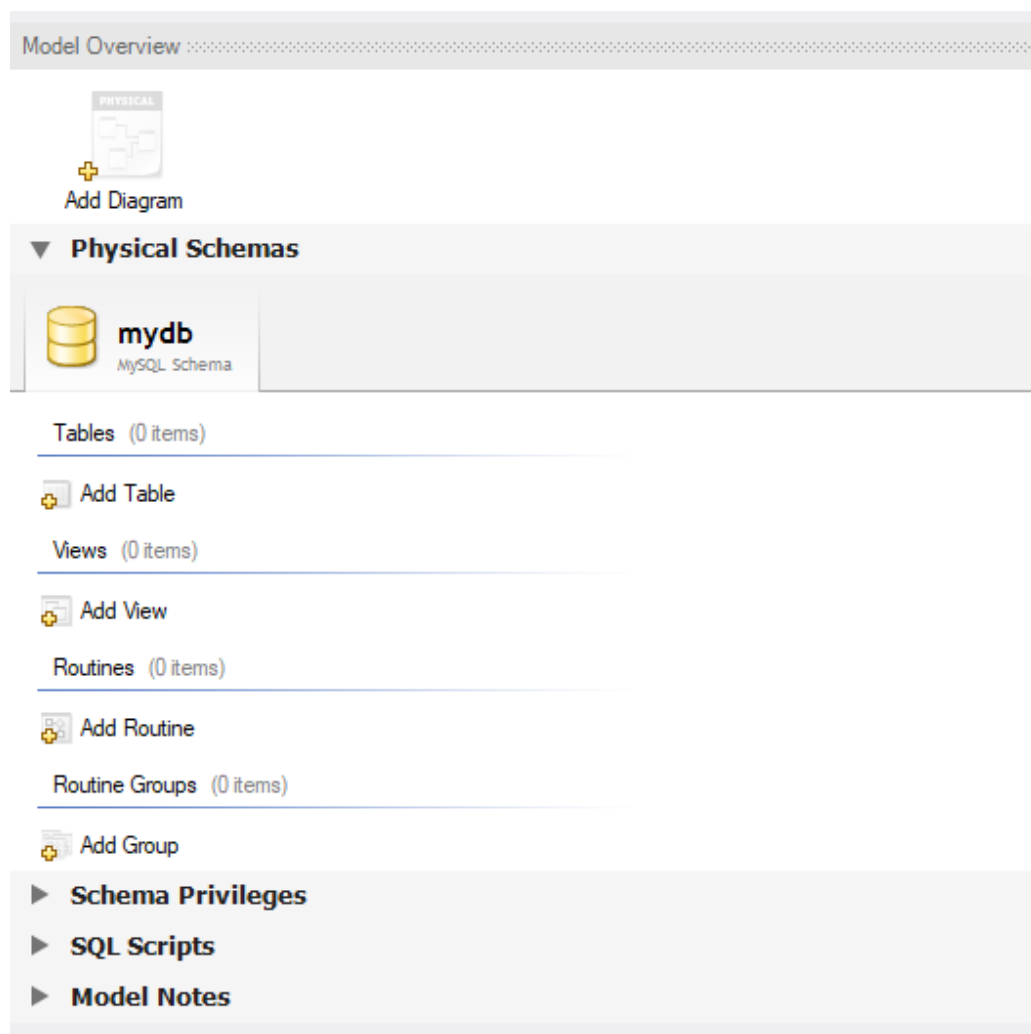


Рисунок 2.2 – Окно создания диаграммы.

В окне создания диаграммы можно, как и создать саму диаграмму, так и выставить определенные параметры для неё.

Для начала работы с диаграммой необходимо дважды нажать на пункт «*Add Diagram*», после чего откроется окно проектирования диаграммы, где можно осуществить множество различных действий:

- создание новых таблиц;
- редактирование существующих таблиц;
- изменение свойств таблицы;
- создание текстовых заметок.

Для создания таблиц в окне проектирования диаграммы стоит выбрать инструмент «*Place a New Table*» (см. рисунок 2.3).

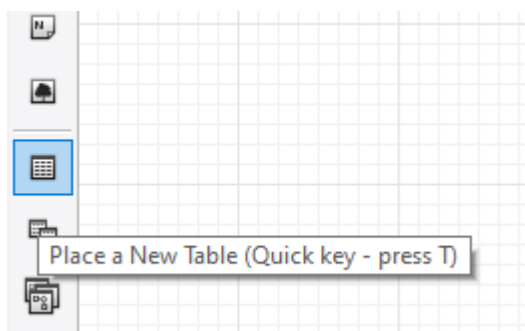


Рисунок 2.3 – Создание новой таблицы

После создания таблицы, необходимо добавить к ней атрибуты. Для выполнения данного действия стоит дважды кликнуть по созданной таблице. При исполнении этих действий откроется окно просмотра и редактирования свойств таблицы (см. рисунок 2.4).

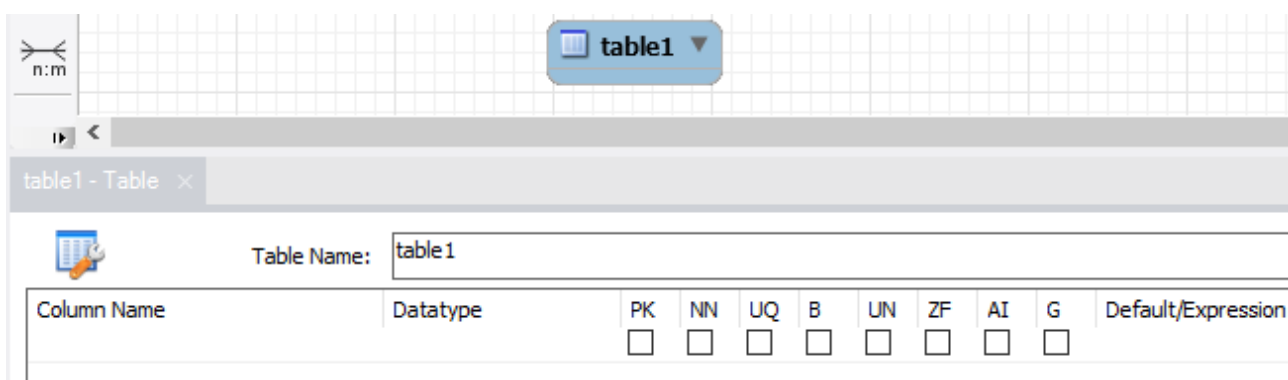


Рисунок 2.4 – Просмотр и редактирование свойств таблицы

В представленном на рисунке выше окне можно выполнять следующие действия:

- изменять имя таблицы;
- добавлять атрибуты к таблице;
- выбирать свойств для атрибутов.

Таким образом нужно создать все необходимые таблицы для базы данных и выставить связи между ними (см. рисунок 2.5).

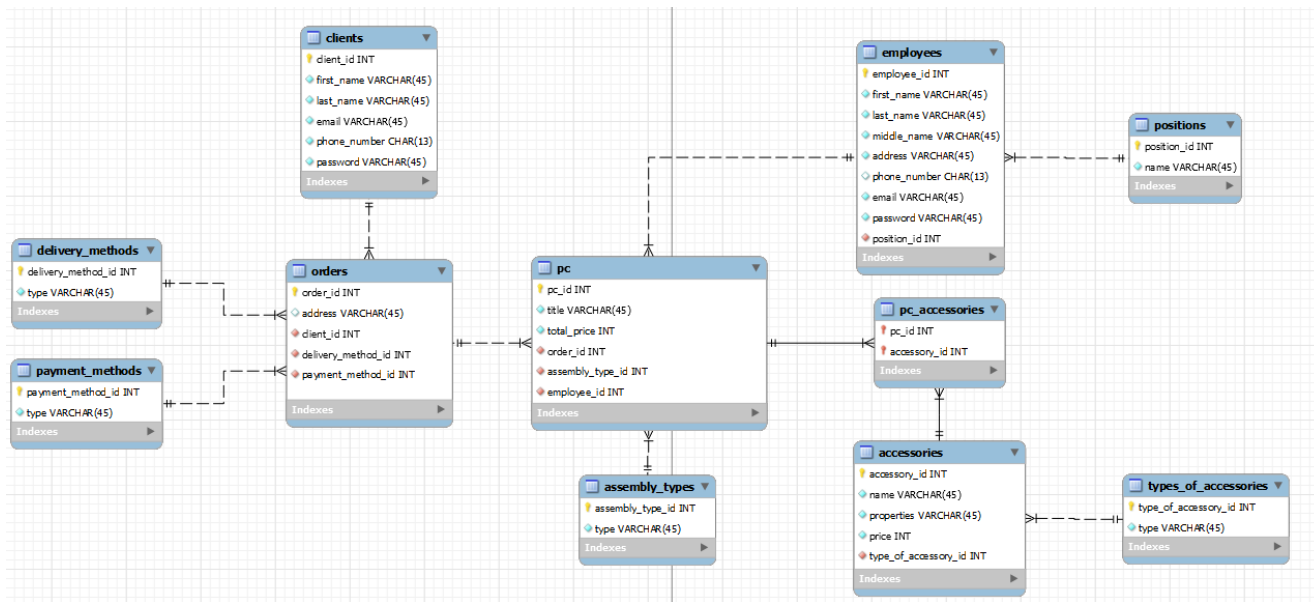


Рисунок 2.5 – Созданная диаграмма «сущность – связь»

## 2.2 Выбор и обоснование используемых типов данных и ограничений (доменов)

В *MySQL* существует множество типов данных, которые можно разделить на шесть групп: целые, вещественные, строковые, бинарные, даты и времени, перечисления и множества.

В ходе проектирования базы данных используются следующие типы данных из *MySQL*:

- *VARCHAR*;
- *INT*.

*VARCHAR* – строковые данные переменного размера. Используются значения для определения размера строки в байтах (допускаются значения от 1 до 8000) или указание предельного размера столбца, вплоть до максимального размера хранилища, что составляет  $2^{31}-1$  байт (2 ГБ).

*INT* – целочисленный тип данных, один из простейших и самых распространённых типов данных в базах данных и программировании. Служит для представления целых чисел [2].

## 2.3 Проектирование запросов к базе данных

Запрос – объект базы данных, который используется для извлечения информации из одной или нескольких таблиц, или для выполнения определенных действий с данными [3].

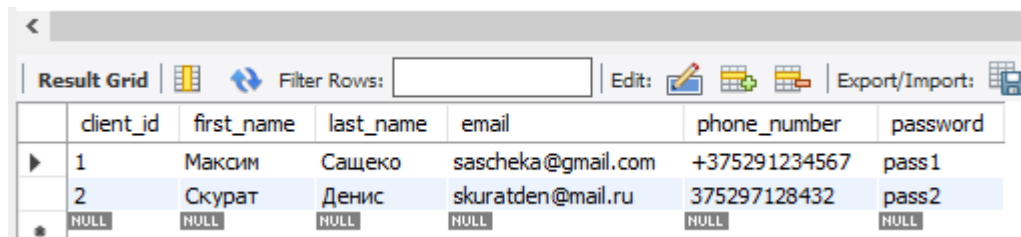
Запрос (команда) строится на основе одной или нескольких взаимосвязанных таблиц, позволяя комбинировать содержащуюся в них информацию. При этом могут использоваться как таблицы БД, так и сохраненные таблицы, полученные в результате выполнения других запросов. Кроме того, запрос может строиться непосредственно на другом запросе.

Чтобы просмотреть данные из таблицы следует воспользоваться оператором *SELECT*. На рисунке 2.6 приведен пример просмотра таблицы клиентов.

```
1 • SELECT * FROM clients;
```

Рисунок 2.6 – Запрос данных для таблицы клиентов

Результат выполнения запроса представлен на рисунке 2.7.



|   | client_id | first_name | last_name | email              | phone_number  | password |
|---|-----------|------------|-----------|--------------------|---------------|----------|
| ▶ | 1         | Максим     | Сащeko    | sascheka@gmail.com | +375291234567 | pass1    |
|   | 2         | Скурат     | Денис     | skuratden@mail.ru  | 375297128432  | pass2    |
| * | NULL      | NULL       | NULL      | NULL               | NULL          | NULL     |

Рисунок 2.7 – Выполнения запроса вывода информации о клиентах

Для получения информации из таблицы по определенному значению используется оператор *SELECT*, где дополнительно нужно указать желаемый атрибут (см. рисунок 2.8).

```
1 • SELECT first_name FROM clients;
```

Рисунок 2.8 – Запрос на получение имен из таблицы клиентов

Результат выполнения запроса представлен на рисунке 2.9.

|   | first_name |
|---|------------|
| ▶ | Максим     |
|   | Денис      |

Рисунок 2.9 – Выполнения запроса вывода имен клиентов

Для добавления данных в таблицу используется оператор *INSERT INTO*. На рисунке 2.10 приведен пример добавления данных в таблицу клиентов.

```
insert into clients (first_name, last_name, email, phone_number, `password`)
values("Максим", "Сащеко", "sascheka@gmail.com", "+375291234567", "pass1")
```

Рисунок 2.10 – Добавление данных в таблицу клиентов

В результате выполнения операции таблица клиентов заполнена введенными значениями (см. рисунок 2.11)

|   | client_id | first_name | last_name | email              | phone_number  | password |
|---|-----------|------------|-----------|--------------------|---------------|----------|
| ▶ | 1         | Максим     | Сащеко    | sascheka@gmail.com | +375291234567 | pass1    |

Рисунок 2.11 – Заполненная таблица клиентов

Для обновления данных в таблице используется оператор *UPDATE* (см. рисунок 2.12).

```
UPDATE clients SET `first_name` = "Никита" WHERE client_id = 1;
```

Рисунок 2.12 – Выполнения запроса на редактирование таблицы клиентов

Результат выполнения представлен на рисунке 2.13.

Result Grid

Filter Rows:

Edit:

Export/Import:

|   | client_id | first_name | last_name | email              | phone_number  | password |
|---|-----------|------------|-----------|--------------------|---------------|----------|
| ▶ | 1         | Никита     | Сашеко    | sascheka@gmail.com | +375291234567 | pass1    |

Рисунок 2.13 – Измененная таблица клиентов

Для удаления данных из таблицы используется оператор *DELETE*. На рисунке 2.14 приведен пример удаления данных из таблицы клиентов.

```
DELETE FROM clients WHERE client_id = 1;
```

Рисунок 2.14 – Удаление данных из таблицы клиентов

После этого из таблицы клиентов были удалены выбранные данные (см. рисунок 2.15)

Result Grid

Filter Rows:

Edit:

Export/Import:

|   | client_id | first_name | last_name | email             | phone_number | password |
|---|-----------|------------|-----------|-------------------|--------------|----------|
| ▶ | 2         | Денис      | Скупат    | skuratden@mail.ru | 375297128432 | pass2    |

Рисунок 2.15 – Таблица клиентов после выполнения операции удаления

На основе представленных выше запросов были созданы все остальные запросы к базе данных производства персональных компьютеров.

## 2.4 Программная реализация и документирование базы данных

Так как данная курсовая работа разрабатывалась под операционную систему *Android*, то выбор языков программирования был не столь большим, а именно *Java* и *Kotlin*.

Язык программирования *Kotlin* впервые был представлен в 2011 году, более чем на 20 лет после появления языка *Java*. В 2017 году компания *Google* в своей официальной документации признала *Kotlin* наиболее подходящим языком для создания *Android*-приложений и рекомендовала его к использованию программистам [4].

Основными преимуществами языка программирования *Kotlin* можно смело назвать следующие пункты:

- полная совместимость с *Java*;



- чистота;
- безопасность;
- простота синтаксиса [5].

Если говорить про преимущества языка *Java* перед *Kotlin* то таковыми стоит отметить:

- более быструю компиляцию;
- большое число пользователей.

После непродолжительного анализа в качестве языка для написания приложения был выбран *Kotlin*, так как в данных условия он, очевидно, более подходящие по условию задачи, особенно благодаря своей простоте.

В качестве среды разработки была выбрана *Android Studio*, являющийся основной средой разработки *Android*-приложений от *Google*.

## 3 ПРИМЕНЕНИЕ РАЗРАБОТАННОЙ БАЗЫ ДАННЫХ

### 3.1 Руководство пользователя

В разработанном приложении реализована возможность клиент-серверного взаимодействия базы данных производства персональных компьютеров и приложения на базе операционной системы *Android*.

При открытии приложения, пользователя встречает экран приветствия, в котором предлагается войти в система зарегистрированным клиентам, либо же создать новый аккаунт (см. рисунок 3.1).

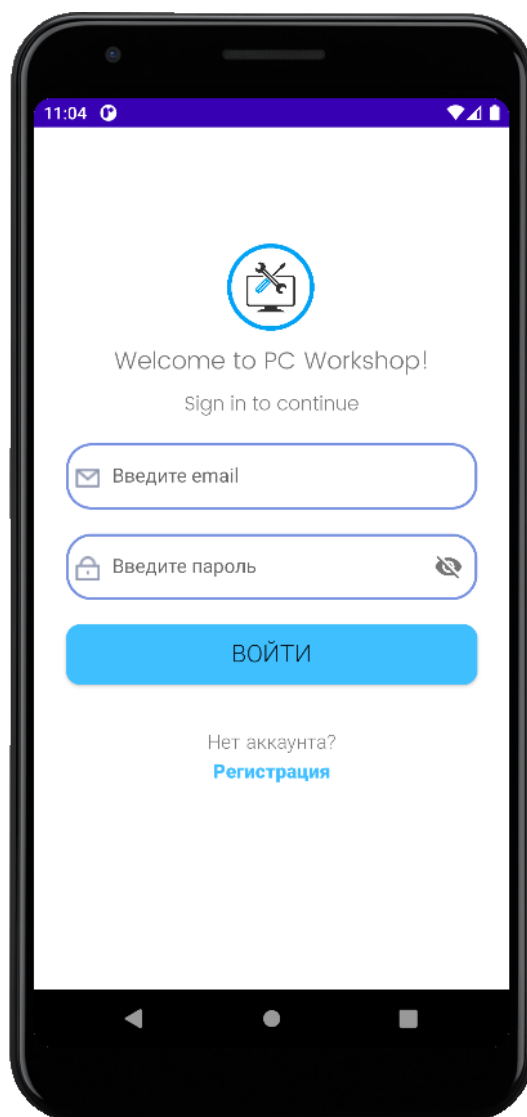


Рисунок 3.1 – Экран приветствия

Если нужно зарегистрироваться в системе, то нужно кликнуть на текстовое поле «Регистрация» (см. рисунок 3.2).

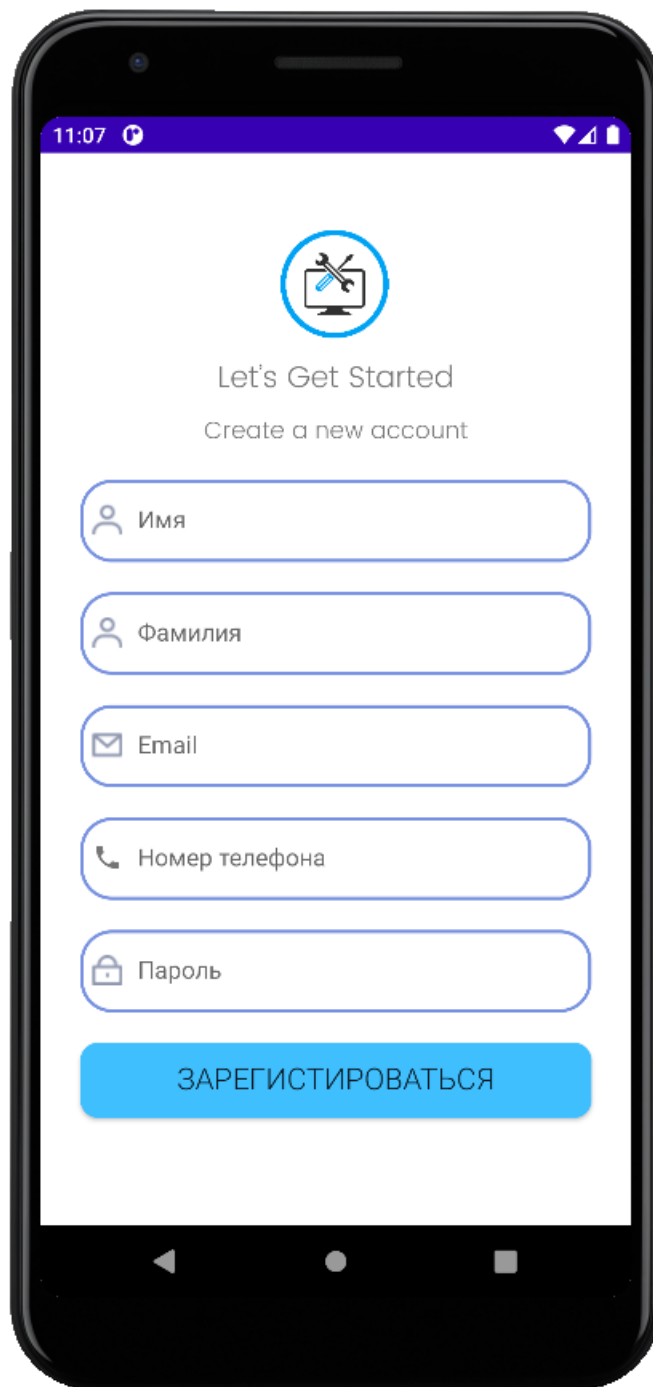


Рисунок 3.2 – Экран регистрации

При входе пользователя в систему с верно введенными данными (при допущении ошибки в написании логина или пароля выдаст предупреждение о

некорректности введенных данных), перед ним открывается экран просмотра информации о заказах, относящихся именно к этому пользователю (см. рисунок 3.3).

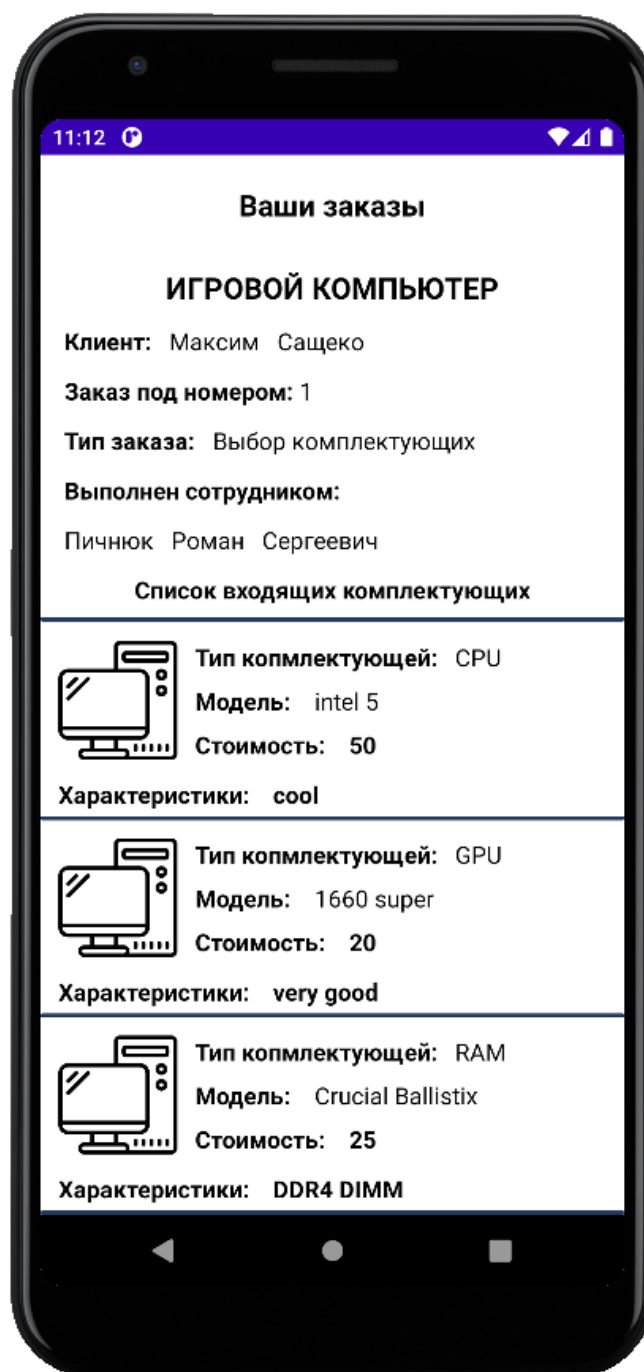


Рисунок 3.3 – Экран просмотра заказов клиента

Если при входе были корректно введены данные для входа работника в систему, это в таком случае будет открыт абсолютно иной экран, представленный на рисунке 3.4.

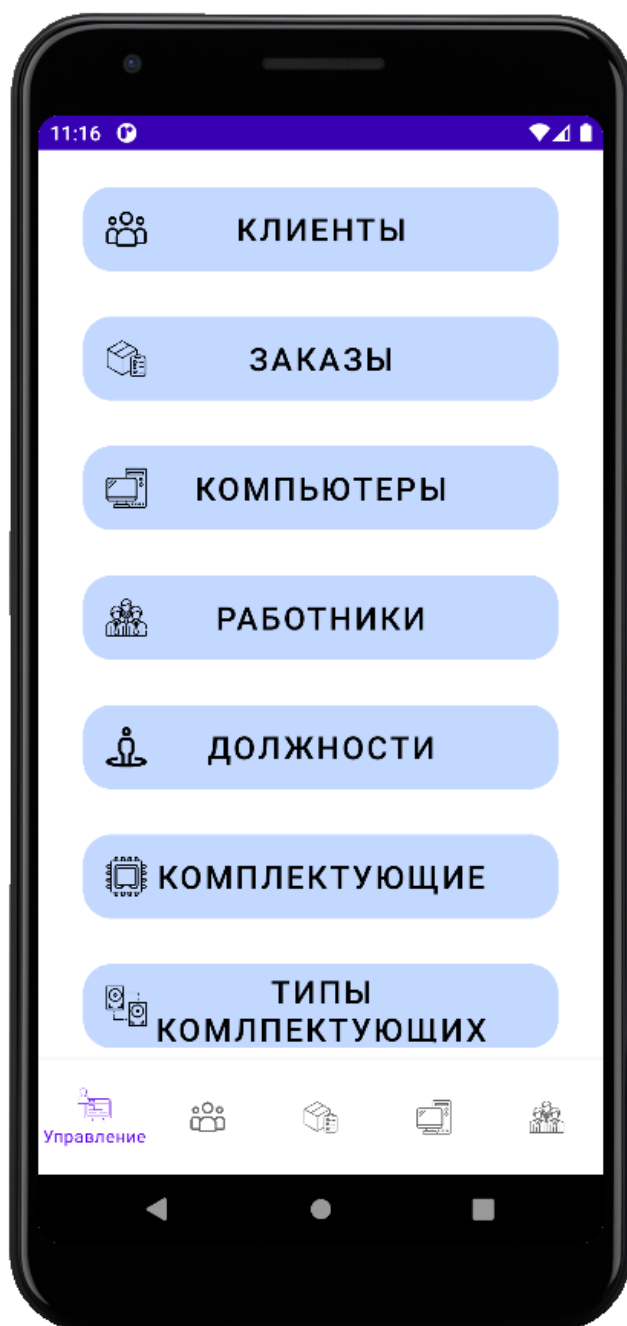


Рисунок 3.4 – Экран при входе в систему работника

Навигация работниками осуществляется благодаря нижней панели навигации и экрану «Управление», представленному на рисунке 3.4.

В нижней панели навигации, кроме пункта «Управление» присутствуют пункты «Клиенты», «Заказы», «Компьютеры» и «Работники». Экраны просмотра данных пунктов представлены на рисунках 3.5 и 3.6.

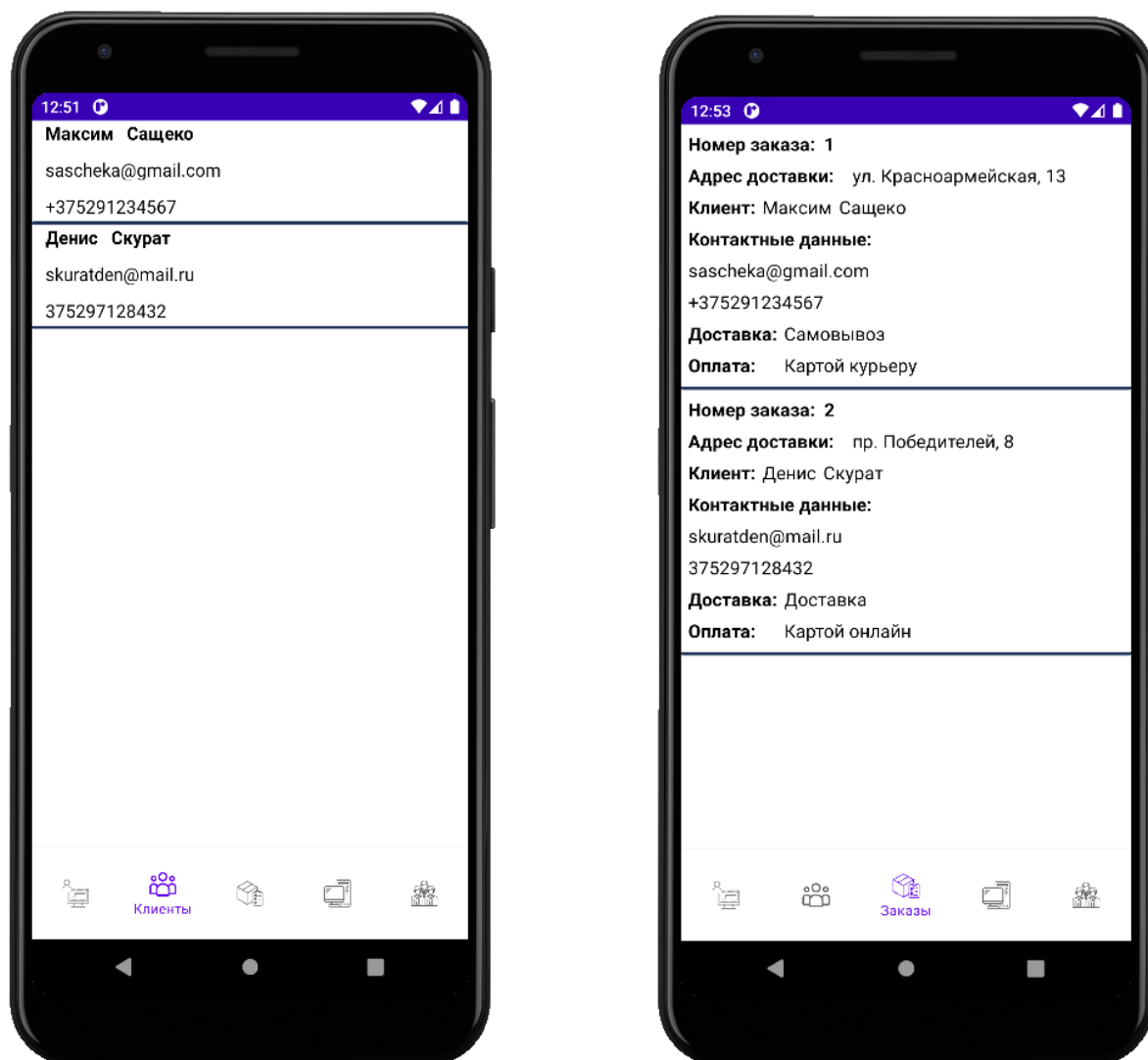


Рисунок 3.5 – Экраны просмотров клиентов и заказов

На этих экранах предоставлены списки клиентов и заказов, динамически обновляющиеся динамически (при изменении параметров в базе данных данные на экране изменяются).

Оставшиеся два пункта на панели навигации отвечают за отображение списков компьютеров и работников (см. рисунок 3.6).

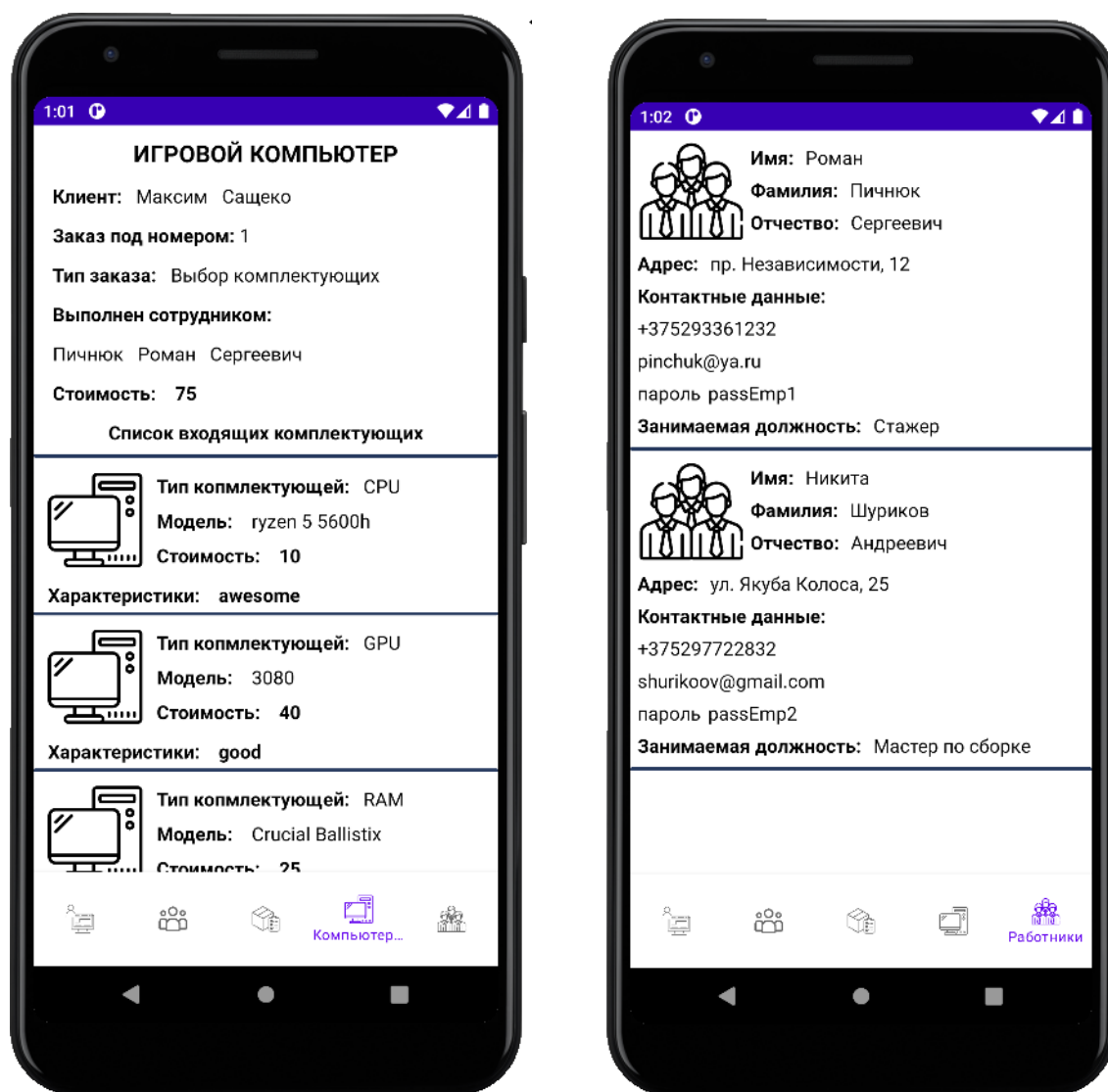


Рисунок 3.6 – Экраны просмотра компьютеров и работников

На этих экранах отображаются списки компьютеров и работников, изменяющихся динамически, как и предыдущие два экрана.

Кроме простого отображения, элементы в списках являются кликабельными, а при клике открывается окно редактирования и удаления элемента, в которое передаются изначально заданные параметры (см. рисунок 3.7).

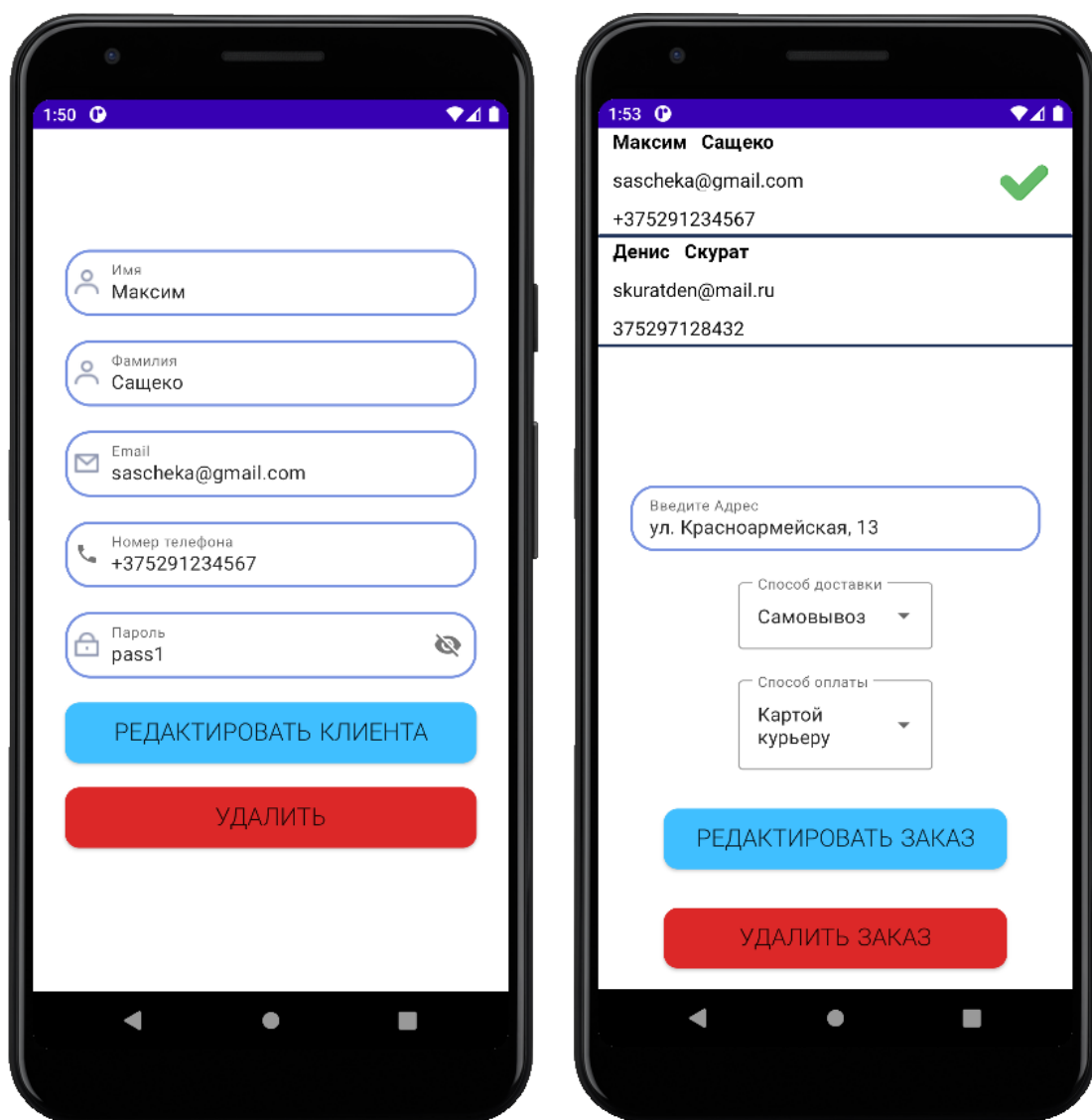


Рисунок 3.7 – Экраны редактирования элементов списков

В пункте навигации «Управление» отображается список кнопок. При нажатии на любую из кнопок открывается новый экран, в котором можно осуществить добавление выбранного элемента в таблицу из базы данных. Примеры нескольких добавлений приведены на рисунках 3.8, 3.9, 3.10.

На рисунке 3.8 приведён пример добавления заказа в базу данных. Список клиентов вверху экрана является пролистываемым, при нажатии на элемент которого заказ определяется конкретному клиенту.



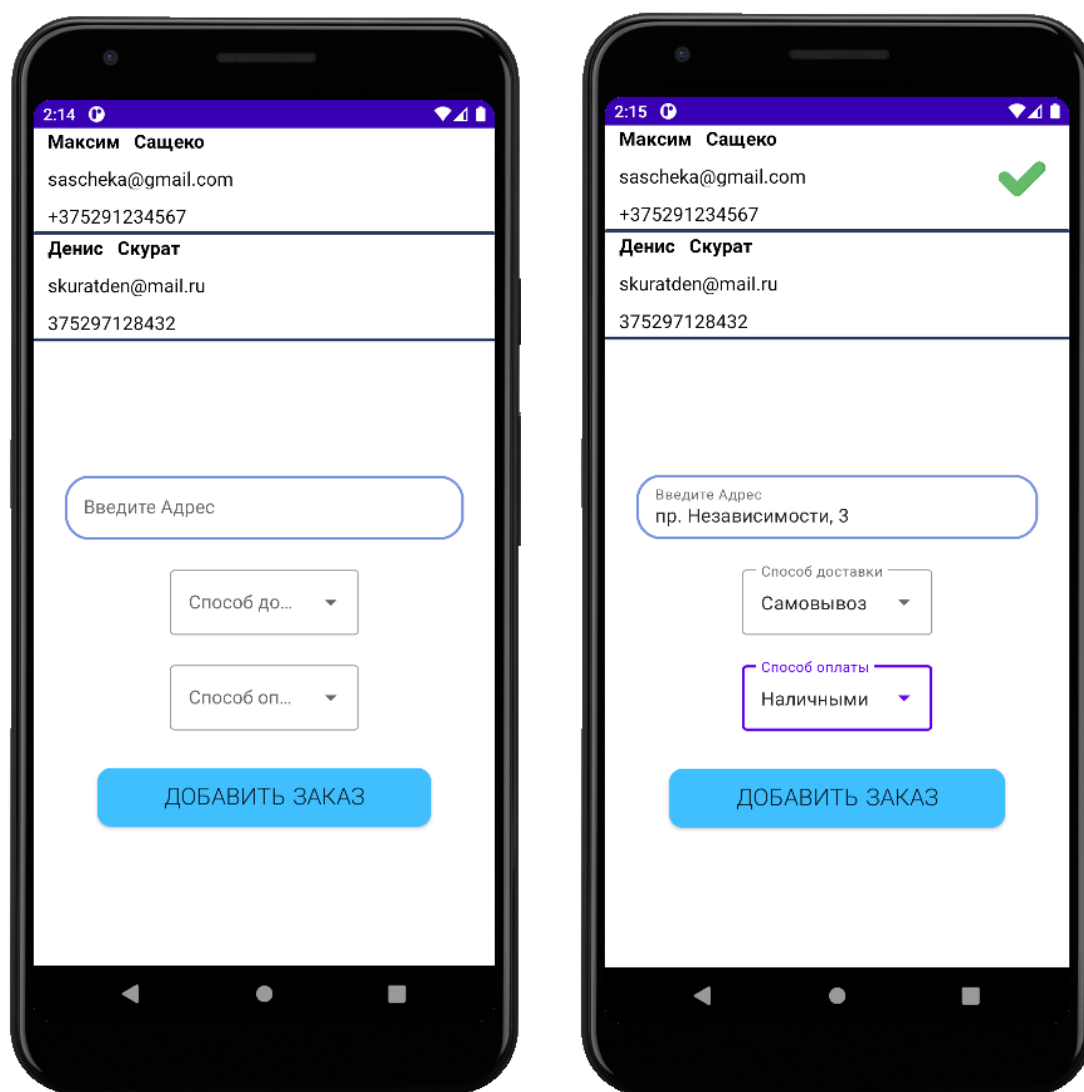


Рисунок 3.8 – Добавление заказа

Следующий экран – экран добавления нового компьютера. Внутри него кроме обычных полей присутствуют два пролистываемых списка, в которых можно выбрать работника, выполняющего сборку компьютера, а также комплектующие, входящие в сборку (см. рисунок 3.9).

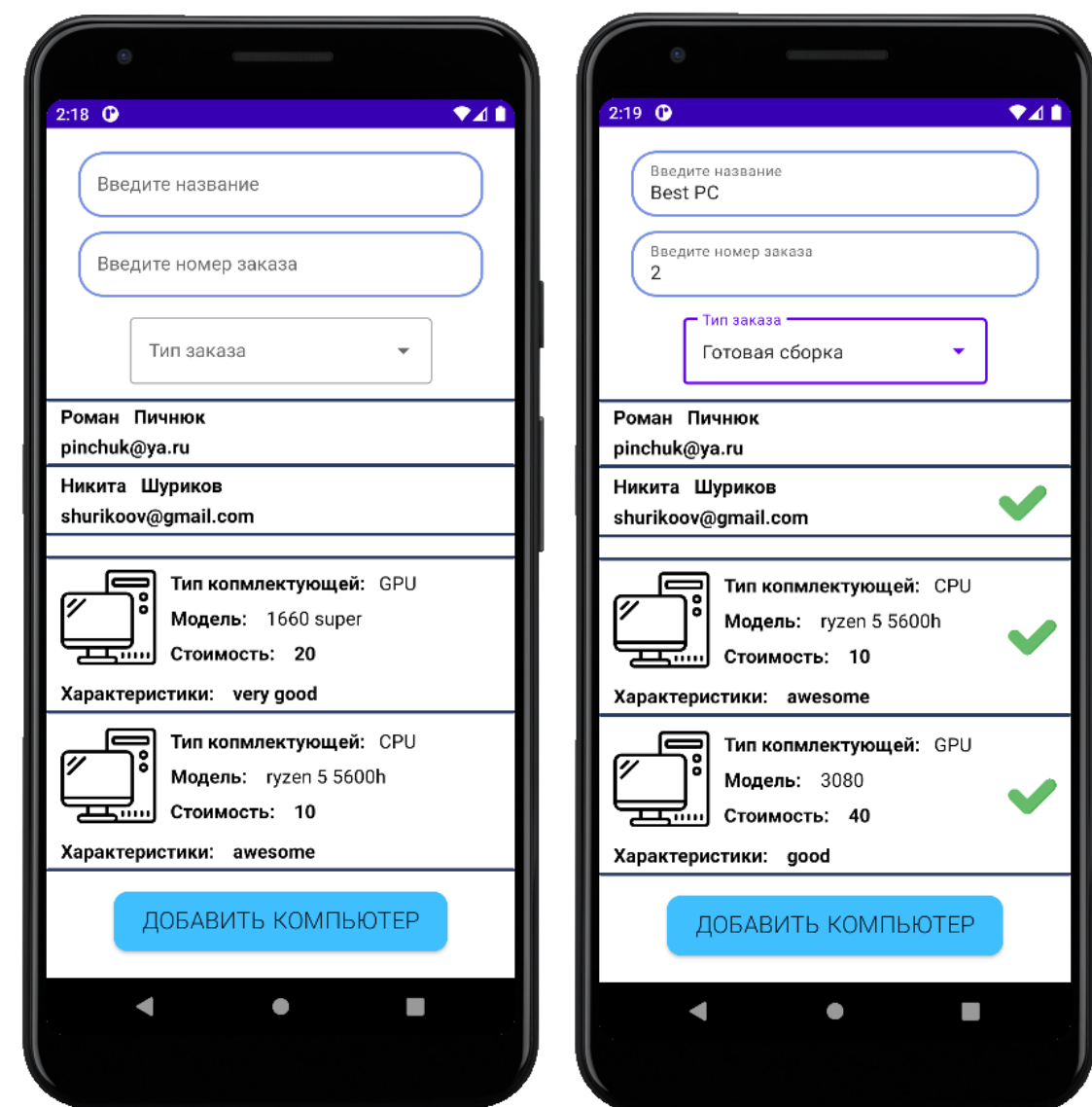


Рисунок 3.9 – Добавление компьютера

На рисунке 3.10 изображен экран добавления новой комплектующей. Кроме обычных полей и кнопки добавления на экране присутствует пролистываемый список уже существующих комплектующих.

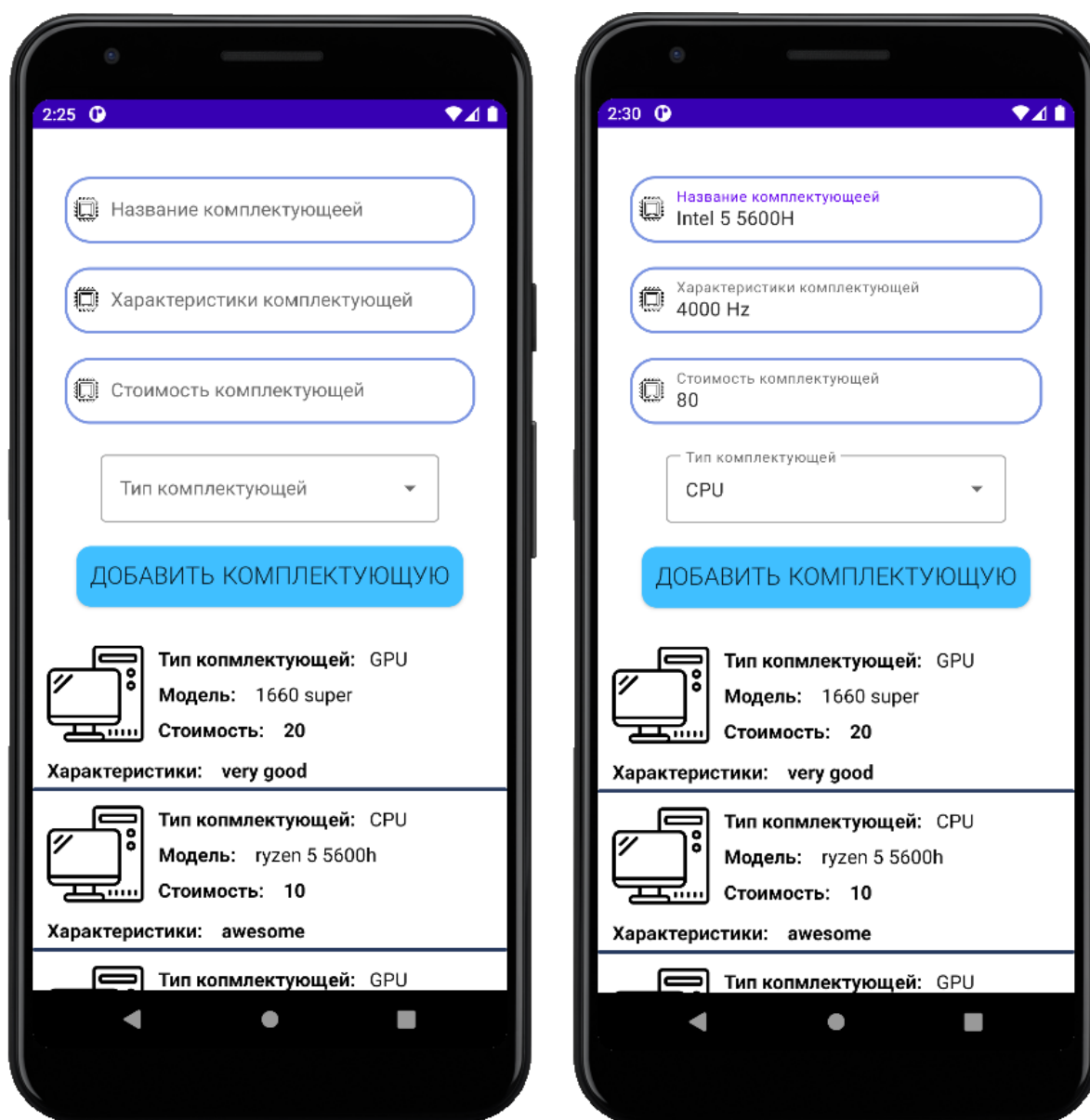


Рисунок 3.10 – Добавление комплектующей

Остальные экраны добавления выполняют схожий функционал и имеют примерно похожий интерфейс.

### 3.2 Администрирование базы данных

Разработанное *Android*-приложение имеет разграничение функционала для пользователей.

Сотруднику выданы полные права доступа к приложению. Благодаря этому, сотрудник может проводить любые операции с базой данных.

При входе в систему под аккаунтом пользователя можно лишь просматривать информацию о заказах, принадлежащих этому аккаунту.

### **3.3 Реализация клиентских запросов**

Разработанное приложение в первую очередь предназначено в первую очередь для администраторов, так как подразумевается, что сам заказ не осуществляется непосредственно самим клиентом, а сотрудником компании. При добавлении заказа администратор сам может выбрать свободного сотрудника и назначить ему выполнение конкретного заказа, что никак не может быть выполнено клиентом.

Клиент имеет доступ только лишь к списку своих заказов, для просмотра определенного рода информации. Доступа к добавлению, редактированию и удалению он не имеет.

Исходя из данной информации, при создании приложения большее внимание уделялось реализации интерфейса администратора.

### **3.4 Обоснование и реализация механизма обеспечения безопасности и сохранности данных**

Подключение к клиентской части на базе операционной системы *Android* осуществляется через созданную серверную часть, которая выполняет роль связующего звена между БД и клиентом. Непосредственно из *Android*-приложения посылаются *HTTPS* запросы на сервер. Так как используемый протокол обеспечивает защищенную передачу данных, то за гипотетический перехват *HTTP*-пакетов можно не беспокоиться, так как информация в них будет надежно зашифрована.

В приложении полностью разделены экраны администраторов и клиентов, что значит, без наличия логина и пароль администратора получить доступ к информации не предоставляется возможным.

Что касается самих экранов ввода паролей, во время ввода можно нажать на иконку в правом углу, которая скрывает пароль, заменяя его отображение на символы звездочек.

## ЗАКЛЮЧЕНИЕ

В ходе курсового проекта была изучена основная теория реляционных баз данных и разработки клиент-серверных приложений.

Получены практические и теоретические навыки работы с следующим стеком технологий:

- *MySQL*;
- *Kotlin*;
- *Retrofit*;
- *Android Studio*.

Была разработана база данных для поддержки работы производства персональных компьютеров.

Созданная база данных отвечает всем требованиям курсового проекта.

Для связи базы данных с клиентским приложением было написано *API* внутреннего использования.

Была спроектирована и разработана клиентская часть (приложение) на базе операционной системы *Android*. Созданное приложение имеет визуально приятный и интуитивно понятный интерфейс, соответствует всем требованиям курсового проекта.

## СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

- [1] Структуры и базы данных [Электронный ресурс]. – Режим доступа: [https://libeldoc.bsuir.by/bitstream/123456789/27723/1/Alekseev\\_struk.pdf](https://libeldoc.bsuir.by/bitstream/123456789/27723/1/Alekseev_struk.pdf).
- [2] Целочисленные типы данных [Электронный ресурс]. – Режим доступа: [https://ru.wikipedia.org/wiki/Целое\\_\(тип\\_данных\)](https://ru.wikipedia.org/wiki/Целое_(тип_данных))
- [3] Понятие запроса [Электронный ресурс]. – Режим доступа: <http://lab314.brsu.by/roleg/bio/bio/bit/access/lr3tnew.html>
- [4] Язык программирования *Kotlin* [Электронный ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/Kotlin>
- [5] Почему следует полностью переходить на *Kotlin* [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/company/vk/blog/329294>
- [6] Клиент-сервер на Android [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/post/269135/>

# ПРИЛОЖЕНИЕ А

## (обязательное)

### Отчёт о проверке на уникальность в системе «Антиплагиат»

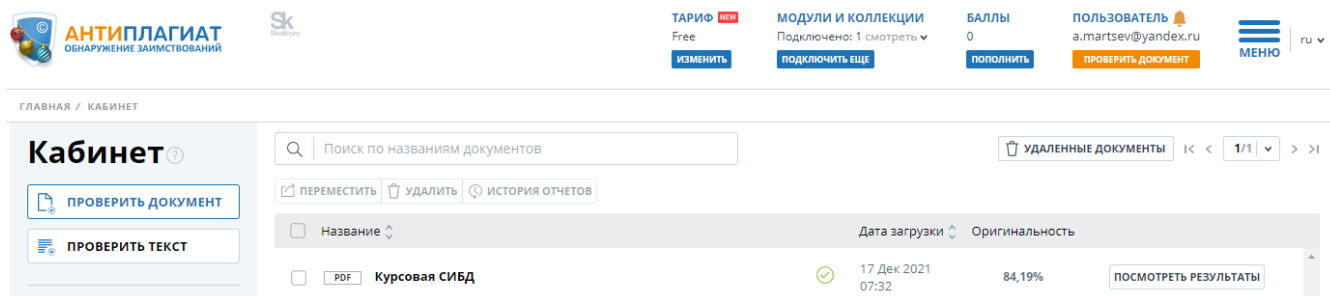


Рисунок А.1 – Отчет о проверке в системе «Антиплагиат»

## ПРИЛОЖЕНИЕ Б

### (обязательное)

### Скрипт генерации БД

```
-- MySQL Script generated by MySQL Workbench
-- Fri Dec 17 03:10:24 2021
-- Model: New Model    Version: 1.0
-- MySQL Workbench Forward Engineering

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

--
-- Schema pc_workshop
--

--
-- Schema pc_workshop
--

CREATE SCHEMA IF NOT EXISTS `pc_workshop` DEFAULT CHARACTER SET utf8 ;
USE `pc_workshop` ;

--
-- Table `pc_workshop`.`clients`
--

CREATE TABLE IF NOT EXISTS `pc_workshop`.`clients` (
  `client_id` INT NOT NULL AUTO_INCREMENT,
  `first_name` VARCHAR(45) NOT NULL,
  `last_name` VARCHAR(45) NOT NULL,
  `email` VARCHAR(45) NOT NULL,
  `phone_number` CHAR(13) NOT NULL,
  `password` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`client_id`),
  UNIQUE INDEX `email_UNIQUE` (`email` ASC) VISIBLE)
ENGINE = InnoDB;

--
-- Table `pc_workshop`.`delivery_methods`
--

CREATE TABLE IF NOT EXISTS `pc_workshop`.`delivery_methods` (
  `delivery_method_id` INT NOT NULL AUTO_INCREMENT,
  `delivery_type` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`delivery_method_id`),
  UNIQUE INDEX `delivery_type_UNIQUE` (`delivery_type` ASC) VISIBLE)
ENGINE = InnoDB;

--
-- Table `pc_workshop`.`payment_methods`
--

CREATE TABLE IF NOT EXISTS `pc_workshop`.`payment_methods` (
  `payment_method_id` INT NOT NULL AUTO_INCREMENT,
```



```

        `payment_type` VARCHAR(45) NOT NULL,
        PRIMARY KEY (`payment_method_id`),
        UNIQUE INDEX `payment_type_UNIQUE` (`payment_type` ASC) VISIBLE)
ENGINE = InnoDB;

-- -----
-- Table `pc_workshop`.`orders`
-- -----
CREATE TABLE IF NOT EXISTS `pc_workshop`.`orders` (
  `order_id` INT NOT NULL AUTO_INCREMENT,
  `address` VARCHAR(45) NULL,
  `client_id` INT NOT NULL,
  `delivery_method_id` INT NOT NULL,
  `payment_method_id` INT NOT NULL,
  PRIMARY KEY (`order_id`),
  INDEX `fk_orders_clients_idx` (`client_id` ASC) VISIBLE,
  INDEX `fk_orders_delivery_methods1_idx` (`delivery_method_id` ASC)
VISIBLE,
  INDEX `fk_orders_payment_methods1_idx` (`payment_method_id` ASC) VISIBLE,
  CONSTRAINT `fk_orders_clients`
    FOREIGN KEY (`client_id`)
      REFERENCES `pc_workshop`.`clients` (`client_id`)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION,
  CONSTRAINT `fk_orders_delivery_methods1`
    FOREIGN KEY (`delivery_method_id`)
      REFERENCES `pc_workshop`.`delivery_methods` (`delivery_method_id`)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION,
  CONSTRAINT `fk_orders_payment_methods1`
    FOREIGN KEY (`payment_method_id`)
      REFERENCES `pc_workshop`.`payment_methods` (`payment_method_id`)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION)
ENGINE = InnoDB;

-- -----
-- Table `pc_workshop`.`assembly_types`
-- -----
CREATE TABLE IF NOT EXISTS `pc_workshop`.`assembly_types` (
  `assembly_type_id` INT NOT NULL AUTO_INCREMENT,
  `type` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`assembly_type_id`),
  UNIQUE INDEX `type_UNIQUE` (`type` ASC) VISIBLE)
ENGINE = InnoDB;

-- -----
-- Table `pc_workshop`.`positions`
-- -----
CREATE TABLE IF NOT EXISTS `pc_workshop`.`positions` (
  `position_id` INT NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`position_id`),
  UNIQUE INDEX `name_UNIQUE` (`name` ASC) VISIBLE)
ENGINE = InnoDB;

```

```

-----
-- Table `pc_workshop`.`employees`
-----
CREATE TABLE IF NOT EXISTS `pc_workshop`.`employees` (
  `employee_id` INT NOT NULL AUTO_INCREMENT,
  `first_name` VARCHAR(45) NOT NULL,
  `last_name` VARCHAR(45) NOT NULL,
  `middle_name` VARCHAR(45) NOT NULL,
  `address` VARCHAR(45) NOT NULL,
  `phone_number` CHAR(13) NULL,
  `email` VARCHAR(45) NOT NULL,
  `password` VARCHAR(45) NOT NULL,
  `position_id` INT NOT NULL,
  PRIMARY KEY (`employee_id`),
  INDEX `fk_employees_positions1_idx` (`position_id` ASC) VISIBLE,
  UNIQUE INDEX `email_UNIQUE` (`email` ASC) VISIBLE,
  CONSTRAINT `fk_employees_positions1`
    FOREIGN KEY (`position_id`)
      REFERENCES `pc_workshop`.`positions` (`position_id`)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION)
ENGINE = InnoDB;

-----
-- Table `pc_workshop`.`pc`
-----
CREATE TABLE IF NOT EXISTS `pc_workshop`.`pc` (
  `pc_id` INT NOT NULL AUTO_INCREMENT,
  `title` VARCHAR(45) NOT NULL,
  `total_price` INT NOT NULL,
  `order_id` INT NOT NULL,
  `assembly_type_id` INT NOT NULL,
  `employee_id` INT NOT NULL,
  PRIMARY KEY (`pc_id`),
  INDEX `fk_pc_orders1_idx` (`order_id` ASC) VISIBLE,
  INDEX `fk_pc_assembly_types1_idx` (`assembly_type_id` ASC) VISIBLE,
  INDEX `fk_pc_employees1_idx` (`employee_id` ASC) VISIBLE,
  UNIQUE INDEX `title_UNIQUE` (`title` ASC) VISIBLE,
  CONSTRAINT `fk_pc_orders1`
    FOREIGN KEY (`order_id`)
      REFERENCES `pc_workshop`.`orders` (`order_id`)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION,
  CONSTRAINT `fk_pc_assembly_types1`
    FOREIGN KEY (`assembly_type_id`)
      REFERENCES `pc_workshop`.`assembly_types` (`assembly_type_id`)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION,
  CONSTRAINT `fk_pc_employees1`
    FOREIGN KEY (`employee_id`)
      REFERENCES `pc_workshop`.`employees` (`employee_id`)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION)
ENGINE = InnoDB;
-----

```

```

-- Table `pc_workshop`.`types_of_accessories`
-----
CREATE TABLE IF NOT EXISTS `pc_workshop`.`types_of_accessories` (
  `type_of_accessory_id` INT NOT NULL AUTO_INCREMENT,
  `type` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`type_of_accessory_id`),
  UNIQUE INDEX `type_UNIQUE` (`type` ASC) VISIBLE)
ENGINE = InnoDB;

-- Table `pc_workshop`.`accessories`
-----
CREATE TABLE IF NOT EXISTS `pc_workshop`.`accessories` (
  `accessory_id` INT NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(45) NOT NULL,
  `properties` VARCHAR(45) NOT NULL,
  `price` INT NOT NULL,
  `type_of_accessory_id` INT NOT NULL,
  PRIMARY KEY (`accessory_id`),
  INDEX `fk_accessories_types_of_accessories1_idx` (`type_of_accessory_id`
ASC) VISIBLE,
  UNIQUE INDEX `name_UNIQUE` (`name` ASC) VISIBLE,
  CONSTRAINT `fk_accessories_types_of_accessories1`
    FOREIGN KEY (`type_of_accessory_id`)
      REFERENCES `pc_workshop`.`types_of_accessories`
        (`type_of_accessory_id`)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION)
ENGINE = InnoDB;

-- Table `pc_workshop`.`pc_accessories`
-----
CREATE TABLE IF NOT EXISTS `pc_workshop`.`pc_accessories` (
  `pc_id` INT NOT NULL,
  `accessory_id` INT NOT NULL,
  PRIMARY KEY (`pc_id`, `accessory_id`),
  INDEX `fk_pc_has_accessories_accessories1_idx` (`accessory_id` ASC)
VISIBLE,
  INDEX `fk_pc_has_accessories_pc1_idx` (`pc_id` ASC) VISIBLE,
  CONSTRAINT `fk_pc_has_accessories_pc1`
    FOREIGN KEY (`pc_id`)
      REFERENCES `pc_workshop`.`pc` (`pc_id`)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION,
  CONSTRAINT `fk_pc_has_accessories_accessories1`
    FOREIGN KEY (`accessory_id`)
      REFERENCES `pc_workshop`.`accessories` (`accessory_id`)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION)
ENGINE = InnoDB;

USE `pc_workshop`;
DELIMITER $$
USE `pc_workshop`$$

```

```

CREATE DEFINER = CURRENT_USER TRIGGER `pc_workshop`.`orders_BEFORE_DELETE`
BEFORE DELETE ON `orders` FOR EACH ROW
BEGIN
SET @orderid = OLD.order_id;
DELETE FROM pc WHERE pc.order_id like @orderid;
END$$

USE `pc_workshop`$$
CREATE DEFINER = CURRENT_USER TRIGGER
`pc_workshop`.`pc_accessories_AFTER_INSERT` AFTER INSERT ON `pc_accessories` FOR
EACH ROW
BEGIN
SET @pcid = NEW.pc_id;
SET @accessoryid = NEW.accessory_id;

set @price = (SELECT price FROM accessories WHERE accessory_id like
@accessoryid);

update pc set pc.total_price = (pc.total_price + @price) WHERE pc.pc_id
like @pcid;

END$$

USE `pc_workshop`$$
CREATE DEFINER = CURRENT_USER TRIGGER
`pc_workshop`.`pc_accessories_BEFORE_DELETE` BEFORE DELETE ON `pc_accessories`
FOR EACH ROW
BEGIN
SET @pcid = OLD.pc_id;
SET @accessoryid = OLD.accessory_id;

set @price = (SELECT price FROM accessories WHERE accessory_id like
@accessoryid);

update pc set pc.total_price = (pc.total_price - @price) WHERE pc.pc_id
like @pcid;
END$$

DELIMITER ;
USE `pc_workshop`;
DROP procedure IF EXISTS `getEmployeesWithPositions`;
USE `pc_workshop`;
DROP procedure IF EXISTS `pc_workshop`.`getEmployeesWithPositions`;
DELIMITER $$
USE `pc_workshop`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `getEmployeesWithPositions`()
BEGIN
SELECT * FROM employees LEFT OUTER JOIN positions ON positions.position_id
= employees.position_id;
END$$
DELIMITER ;

USE `pc_workshop`;
DROP procedure IF EXISTS `pc_workshop`.`getOrderedOrders`;
DELIMITER $$
USE `pc_workshop`$$

```

```

CREATE DEFINER=`root`@`localhost` PROCEDURE `getOrderedOrders`()
BEGIN
SELECT * FROM orders ORDER BY client_id ASC;
END$$
DELIMITER ;

USE `pc_workshop`;
DROP procedure IF EXISTS `getOrderedAccessories`;
DELIMITER $$
USE `pc_workshop`$$
CREATE PROCEDURE `getOrderedAccessories` ()
BEGIN
SELECT * FROM accessories ORDER BY price ASC;
END$$
DELIMITER ;

SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;

```

**ПРИЛОЖЕНИЕ В**  
**(обязательное)**  
**Листинг программного кода**

```
package com.example.pcworkshop.services

import retrofit2.Retrofit
import retrofit2.converter.gson.GsonConverterFactory

class RetrofitInstance {

    companion object {
        private const val URL = "http://10.0.2.2:3000/"
        // private const val URL = "http://192.168.0.13:3000/"

        fun getRetrofitInstance(): Retrofit {
            return Retrofit.Builder()
                .baseUrl(URL)
                .addConverterFactory(GsonConverterFactory.create())
                .build()
        }
    }
}
```

```
package com.example.pcworkshop.services

import com.google.gson.Gson
import com.google.gson.GsonBuilder
import retrofit2.Retrofit
import retrofit2.converter.gson.GsonConverterFactory
import retrofit2.converter.moshi.MoshiConverterFactory

object ServiceBuilder {

    private const val URL = "http://10.0.2.2:3000/"

    private val gson: Gson = GsonBuilder()
        .setLenient()
        .create()

    private val retrofit: Retrofit = Retrofit.Builder()
        .baseUrl(URL)
        .addConverterFactory(GsonConverterFactory.create(gson))
        .build()

    fun <T> buildService(serviceType: Class<T>): T {
        return retrofit.create(serviceType)
    }
}
```

```
package com.example.pcworkshop.models.clients.dao
```

```

import com.example.pcworkshop.models.clients.Clients
import com.example.pcworkshop.models.clients.PostClient
import retrofit2.Call
import retrofit2.Response
import retrofit2.http.*

interface ClientsDao {

    @GET("clients")
    suspend fun getAllClients(): Response<List<Clients>>

    @GET("clients/{id}")
    suspend fun getClient(@Path("id") id: Int): Response<Clients>

    @POST("clients/create")
    fun addClient(@Body client: PostClient): Call<PostClient>

    @FormUrlEncoded
    @PUT("clients/update/{client_id}")
    fun updateClient(
        @Path("client_id") id: Int,
        @Field("first_name") firstName: String,
        @Field("last_name") lastName: String,
        @Field("email") email: String,
        @Field("phone_number") phoneNumber: String,
        @Field("password") password: String
    ): Call<PostClient>

    @DELETE("clients/delete/{id}")
    fun deleteClient(@Path("id") id: Int): Call<Unit>

}

package com.example.pcworkshop.screen.clients.repository

import com.example.pcworkshop.models.clients.dao.ClientsDao
import com.example.pcworkshop.models.clients.Clients
import com.example.pcworkshop.models.clients.PostClient
import com.example.pcworkshop.services.RetrofitInstance
import retrofit2.Call
import retrofit2.Response

class ClientsRepository {

    suspend fun getAllClients(): Response<List<Clients>> {
        val retrofitInstance =
RetrofitInstance.getRetrofitInstance().create(ClientsDao::class.java)
        return retrofitInstance.getAllClients()
    }

    suspend fun getClient(clientId: Int): Response<Clients> {
        val retrofitInstance =
RetrofitInstance.getRetrofitInstance().create(ClientsDao::class.java)
        return retrofitInstance.getClient(clientId)
    }

    fun addClient(client: PostClient): Call<PostClient> {

```

```

        val retrofitInstance =
RetrofitInstance.getRetrofitInstance().create(ClientsDao::class.java)
        return retrofitInstance.addClient(client)
    }

    fun updateClient(clientId: Int,
                    firstName: String,
                    lastName: String,
                    email: String,
                    phoneNumber: String,
                    password: String): Call<PostClient> {
        val retrofitInstance =
RetrofitInstance.getRetrofitInstance().create(ClientsDao::class.java)
        return retrofitInstance.updateClient(clientId, firstName, lastName,
email, phoneNumber, password)
    }

    fun deleteClient(clientId: Int): Call<Unit> {
        val retrofitInstance =
RetrofitInstance.getRetrofitInstance().create(ClientsDao::class.java)
        return retrofitInstance.deleteClient(clientId)
    }
}

```



**ПРИЛОЖЕНИЕ Г**  
**(обязательное)**  
**Ведомость курсового проекта**