МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

|  |  |  |  |
| --- | --- | --- | --- |
| Кафедра | *Вычислительной техники* | | |
|  | (полное название кафедры) | | |
|  |  | Утверждаю | |
| **Зав. кафедрой** | *Якименко А.А.* |
|  | |
| (подпись, инициалы, фамилия) | |
| «ㅤㅤ» ㅤㅤㅤㅤㅤㅤ 2024 г. | |

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА**

|  |  |
| --- | --- |
| *Гана Алексея Андреевича* | |
| (фамилия, имя, отчество студента – автора работы) | |
| *Клиент-серверное приложение* | |
| (тема работы) | |
| *для автоматизации взаимодействия персонала точек питания* | |
| *Факультет автоматики и вычислительной техники* | |
| (полное название факультета) | |
| Направление подготовки | *09.03.04 Программная инженерия* |
|  | (код и наименование направления подготовки бакалавра) |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Руководитель**  **от НГТУ** |  | | |  | | | | |  | | | | **Автор выпускной квалификационной работы** | |
| *Малявко Александр Антонович* |  | | | |  |  | | | | | *Ган Алексей Андреевич* | | | |
| (фамилия, имя, отчество) |  |  | | | | |  | | | (фамилия, имя, отчество) | | | |
| *к.т.н., доцент* |  | |  | | | | |  | | | | *АВТФ, АВТ-043* | | |
| (ученая степень, ученое звание) |  |  | | | | |  | | | (факультет, группа) | | | |
|  |  | |  | | | | |  | | | |  | | |
| (подпись, дата) |  |  | | | | |  | | | (подпись, дата) | | | |

|  |
| --- |
|  |

Новосибирск 2024

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

|  |  |  |  |
| --- | --- | --- | --- |
| Кафедра | *Вычислительной техники* | | |
|  | (полное название кафедры) | | |
|  |  | УТВЕРЖДАЮ | |
| Зав. кафедрой | *Якименко А.А.* |
| (фамилия, имя, отчество) | |
|  |  |
|  |  |
|  | (подпись, дата) |

**ЗАДАНИЕ  
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА**

|  |  |  |  |
| --- | --- | --- | --- |
| студенту | *Ган Алексею Андреевичу* | | |
|  | (фамилия, имя, отчество) | | |
| Направление подготовки | | *09.03.04 Программная инженерия* | |
|  | | (код и наименование направления подготовки бакалавра) | |
| *Факультет автоматики и вычислительной техники* | | | |
| (полное название факультета) | | | |
| Тема | *Клиент-серверное приложение* | | |
|  | (полное название темы выпускной квалификационной работы бакалавра) | | |
| *для автоматизации взаимодействия персонала точек питания* | | | |
| Исходные данные (или цель работы) | | | *Целью работы является разработка* |
| *клиент-серверного приложения для автоматизации взаимодействия персонала точек* | | | |
| *питания* | | | |
|  | | | |
|  | | | |
|  | | | |
|  | | | |
| Структурные части работы | | |  |
| *Введение* | | | |
| *1. Системная аналитика: обзор существующих решений, формирование бизнес-* | | | |
| *требований к системе, моделирование предметной области* | | | |
| *2. Выбор средств разработки* | | | |
| *3. Архитектурное проектирование* | | | |
| *4. Функциональное проектирование* | | | |

|  |
| --- |
| *5. Руководство пользователя* |
| *Заключение* |
| Список используемых источников |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

Задание согласовано и принято к исполнению.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Руководитель**  **от НГТУ** |  |  |  | **Студент** |
| *Малявко Александр Антонович* |  |  |  | *Ган Алексей Андреевич* |
| (фамилия, имя, отчество) |  |  |  | (фамилия, имя, отчество) |
| *к.т.н.* |  |  |  | *АВТФ, АВТ-043* |
| (ученая степень, ученое звание) |  |  |  | (факультет, группа) |
|  |  |  |  |  |
| (подпись, дата) |  |  |  | (подпись, дата) |

|  |  |  |  |
| --- | --- | --- | --- |
| Тема утверждена | приказом по НГТУ № | 1041/2 | от « 27 » февраля 2024 г. |
| изменена | приказом по НГТУ № | ㅤㅤㅤㅤ | от «ㅤㅤ» ㅤㅤㅤㅤㅤㅤ 2024 г. |

| ВКР сдана в ГЭК №ㅤㅤㅤㅤ, тема сверена с данными приказа | |
| --- | --- |
|  |  |
|  | (подпись секретаря государственной экзаменационной комиссии по защите ВКР, дата) |
|  |  |
|  | (фамилия, имя, отчество секретаря государственной  экзаменационной комиссии по защите ВКР) |

**АННОТАЦИЯ**

Пояснительная записка к выпускной квалификационной работе на тему «Клиент-серверное приложение для автоматизации взаимодействия персонала точек питания».

Руководителем работы является Малявко Александр Антонович.

Целью данной бакалаврской работы является разработка мобильного клиент-серверного приложения для автоматизации взаимодействия персонала точек питания, что включает в себя разработку серверной части приложения и клиентской на платформе Android.

Пояснительная записка содержит 73 страницы, 67 рисунков, 9 таблиц и 15 источников.

Ключевые слов: Java, Android, фреймворк Spring, ORM Hibernate, Android, RestAPI, WebSocket, клиент-сервер.

Оглавление

[Введение 6](#_Toc168488833)

[1 Системная Аналитика 7](#_Toc168488834)

[1.1 Постановка задачи 7](#_Toc168488835)

[1.2 Обзор существующих решений 7](#_Toc168488836)

[1.2.1 r\_keeper Waiter. 7](#_Toc168488837)

[1.2.2 Iiko Waiter. 8](#_Toc168488838)

[1.2.3 Сравнительный анализ 9](#_Toc168488839)

[1.3 Формирование бизнес-требований 10](#_Toc168488840)

[1.4 Формирование функциональных требований 10](#_Toc168488841)

[1.5 Моделирование предметной области 12](#_Toc168488842)

[2 Выбор средств разработки 17](#_Toc168488843)

[2.1 Выбор среды разработки 17](#_Toc168488844)

[2.2 Выбор языка программирования 18](#_Toc168488845)

[2.3 Выбор СУБД 18](#_Toc168488846)

[2.4 Выбор фреймворков 19](#_Toc168488847)

[3 Архитектурное проектирование 21](#_Toc168488848)

[3.1 Общая архитектура 21](#_Toc168488849)

[3.2 Объектная модель 21](#_Toc168488850)

[3.3 Структура базы данных 27](#_Toc168488851)

[3.4 Аутентификация 28](#_Toc168488852)

[3.5 WebSocket 31](#_Toc168488853)

[3.6 Model-View-ViewModel 32](#_Toc168488854)

[3.7 Single Activity 33](#_Toc168488855)

[3.8 Retrofit 33](#_Toc168488856)

[4. ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ 35](#_Toc168488857)

[4.1 Функциональное проектирование серверной части 35](#_Toc168488858)

[4.1.1 Сущности 35](#_Toc168488859)

[4.1.2 Api сервера 37](#_Toc168488860)

[4.1.3 JWT токены 43](#_Toc168488861)

[4.1.4 WebSocket 44](#_Toc168488862)

[4.2 Функциональное проектирование клиентской части 45](#_Toc168488863)

[4.2.1 POJO классы 45](#_Toc168488864)

[4.2.2 Запуск приложения 46](#_Toc168488865)

[4.2.3 Токены доступа и обновления 46](#_Toc168488866)

[4.2.4 Запуск Activity 48](#_Toc168488867)

[4.2.5 Взаимодействие с сервером. 49](#_Toc168488868)

[4.2.6 Отображение получаемых данных 50](#_Toc168488869)

[4.2.7 WebSocket 53](#_Toc168488870)

[4.2.8 Отправка уведомлений 55](#_Toc168488871)

[5 Руководство пользователя 57](#_Toc168488872)

[5.1 Вход в систему 57](#_Toc168488873)

[5.2 Интерфейс администратора 58](#_Toc168488874)

[5.2.1 Начальный экран 58](#_Toc168488875)

[5.2.2 Кнопка удаления 58](#_Toc168488876)

[5.2.3 Кнопка добавления 59](#_Toc168488877)

[5.2.4 Кнопка редактирования 59](#_Toc168488878)

[5.2.5 Создание Блюда 59](#_Toc168488879)

[5.2.6 Кнопка возврата 60](#_Toc168488880)

[5.2.7 Добавление сотрудника 60](#_Toc168488881)

[5.2.8 Редактирование сотрудника 61](#_Toc168488882)

[5.2.9 Вкладка выручки 62](#_Toc168488883)

[5.2.10 Смена вкладок 64](#_Toc168488884)

[5.2.11 Выход из аккаунта 64](#_Toc168488885)

[5.3 Интерфейс официанта 64](#_Toc168488886)

[5.3.1 Начальный экран 64](#_Toc168488887)

[5.3.2 Cтолы 65](#_Toc168488888)

[5.3.2 Выбранный стол 66](#_Toc168488889)

[5.3.3 Детали заказа 67](#_Toc168488890)

[Заключение 70](#_Toc168488891)

[Список источников 71](#_Toc168488892)

# Введение

Каждый человек так или иначе сталкивается с индустрией общественного питания, будь то поход в ресторан с друзьями или просто посещение кафе в обеденный перерыв, и каждый человек, заходя туда, хочет получить достойное обслужив за свои деньги.

Обслуживанием клиентов занимается персонал общественного питания: официанты, повара, бармены и администраторы. Нередки случаи, когда повар забыл приготовить заказа, или официант забыл вынести блюдо и оно остыло, или вообще заказ просто не передали на кухню. Думаю, многие лично сталкивались данной ситуацией, либо выслушивали недовольные рассказы своих знакомых. Данная проблема связана с тем, что работа персонала довольно изнурительна, и во время сильной загруженности заведения уставшие под конец смены сотрудники допускают ошибки. Всё это ведет к недовольству со стороны клиентов и потере денег со стороны владельцев.

Чтобы облегчить работу персонала и улучшить процесс обслуживания клиента необходима система, работающая на мобильных устройствах и позволяющая принимать заказы, отслеживать статусы заказов, а также получать уведомления об их изменениях.

Целью данной работы является разработка клиент-серверное приложение для автоматизации точек питания, для достижения желаемого результата нужно выполнить ряд задач: изучить имеющиеся решения, обозначить их возможности, достоинства и недостатки, сформировать бизнес-требования к мобильному приложению, изучить сопутствующую литературу, составить техническое задание, смоделировать модель предметной области, выполнить архитектурное и функциональное проектирование системы, а также разработать руководство пользователя.

# 1 Системная Аналитика

## Постановка задачи

Цель работы – разработка мобильного клиент-серверного приложения для автоматизации взаимодействия персонала точек питания. Необходимо разработать как серверную часть, так и клиентскую на платформе Android.

Приложение должно предусматривать наличие четырех ролей пользователей: официант, повар, бармен, администратор, а также иметь соответствующий каждой роли функционал.

Администратор должен иметь возможность заполнять и редактировать базу данных сотрудников, блюд, столов и категорий, а также просматривать выручку за указные период.

Официанты должны иметь возможность просмотра текущих заказов, добавления и редактирования новых заказов, а также должны получать уведомления при изменении статуса блюд в заказе

Повара и бармены должны иметь возможность просматривать текущий заказы, менять статус блюд в заказе, а также получать уведомления о новых заказах.

## Обзор существующих решений

### 1.2.1 r\_keeper Waiter.

r\_keeper Waiter [2] – это мобильное приложение для официантов разработанное компании USC, специализирующейся на разработке программного обеспечения и программно-аппаратных комплексов для автоматизации работы официантов в ресторанах. Данное решение представляет собой инновационную разработку для оптимизации процессов обслуживания гостей, управления заказами, а также повышения эффективности работы персонала.

В отличие от стандартного r\_Keeper, ориентированного на управление всеми аспектами работы ресторана, r\_keeper Waiter сконцентрирован именно на сфере обслуживания и взаимодействия с клиентами, что позволяет увеличить удовлетворенность клиентов, повышая скорость обслуживания, также его качество.

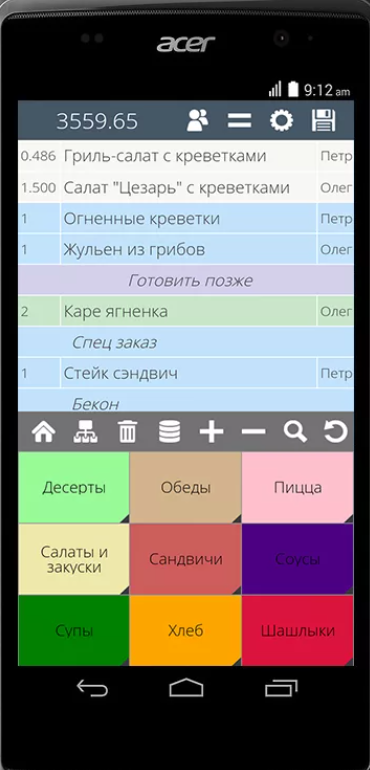


Рисунок 1 – Интерфейс r\_keeper Waiter

### 1.2.2 Iiko Waiter.

iiko Waiter [3] – это мобильное приложение для официантов, разработанное компанией iiko, являющейся лидером в области разработки инновационных технологий для ресторанного бизнеса. Основанная Давидом Яном и Максимом Нальским в 2005 году, компания iiko предлагает комплексные решения для автоматизации работы предприятий общественного питания. Приложение iiko Waiter предназначено для оптимизации процесса обслуживания гостей в ресторанах и кафе, благодаря удобному интерфейсу и мобильности, официанты могут принимать заказы, отправлять их на кухню, а также управлять столиками и следить за статусом заказов непосредственно на месте обслуживания.

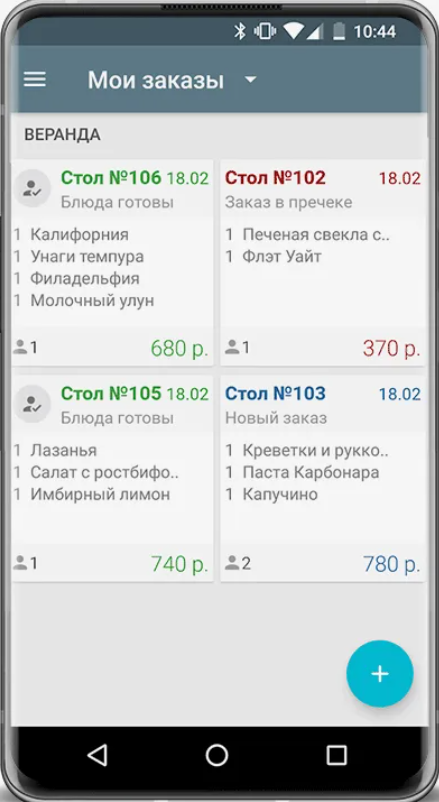


Рисунок 2 – Интерфейс iiko Waiter

### 1.2.3 Сравнительный анализ

Таблица 1 – Сравнительный анализ

|  |  |  |
| --- | --- | --- |
| Параметр | iiko Waiter | R-Keeper Waiter |
| Поддержка iOS | + | + |
| Поддержка Android | + | + |
| Облачное хранение данных | + | + |
| Поддержка множества языков | + | + |
| Работа в офлайн режиме | + | + |
| Стоимость лицензии в месяц | 590р/устройство | 580р/устройство |
| База данных | SQL | Закрытый язык |

Также в ходе анализа отзывов пользователей установлено, что iiko обладает более понятным и удобный интерфейсом, что непосредственно влияет на скорее обучения сотрудников и качество обслуживания

По результатам сравнительного анализа решений для официантов от iiko и r\_keeper, можно сделать вывод, что обе системы имеют внушительный функционал, однако, пусть iiko немного и уступает по параметру стоимости, его интерфейс более понятный и удобный, нежели у конкурента, а также база данных написана на известном многим языке SQL, поэтому можно сказать, что iiko waiter более предпочтителен для обслуживания посетителей.

## Формирование бизнес-требований

Приложение должно быть настраиваемым под различные заведения, это означает, что для пользователей с ролью “Администратор” должен быть реализован функционал изменения параметров, таких как: категории и блюда в меню, количество столов в зависимости от задачи заведения общественного питания. Также они должны иметь возможность регистрации новых сотрудников, поскольку данная функция недоступно для посторонних.

Пользователи с ролью “Официант” должны иметь возможность создавать и редактировать заказы, добавлять туда блюда из меню, и редактировать их в случае необходимости, а также должны получать уведомления об изменения статуса блюд в заказе.

Пользователи с ролью “Повар” и “Бармен” должны получать информацию о заказах, созданных официантами, и блюдах в них, соответствующим их ролям, должны иметь возможность меняться состояние готовности блюд в заказе, а также должны получать уведомление о новых заках.

## Формирование функциональных требований

**Пользователи-сотрудники.**

Система должна предоставлять возможность:

* Создания пользователей-сотрудников;
* Аутентификации и авторизации пользователей-сотрудников;
* Просмотра списка пользователей-сотрудников;
* Назначения пользователям-сотрудникам одной или нескольких необходимых ролей:
  + Администратор;
  + Официант;
  + Повар;
  + Бармен.
* Изменение и просмотр информации о пользователях-сотрудниках:
  + Имя;
  + Фамилия;
  + Отчество;
  + Email;
  + Номер телефона;
  + Назначенные роли.

**Категории.**

Система должна предоставлять возможность:

* Создания новых категорий;
* Просмотра списка категорий;
* Удаления и изменения информации о категориях.

**Блюда.**

Система должна предоставлять возможность:

* Создания новых блюд;
* Просмотра списка блюд;
* Удаления и изменения информации о блюдах.

**Столы.**

Система должна предоставлять возможность:

* Добавления новых столов;
* Удаления и изменения информации о столах;
* Просмотра списка столов;
* Отслеживания статуса столов.

**Заказы.**

Система должна предоставлять возможность:

* Добавления новых заказов;
* Отслеживания статусов заказов;
* Просмотра списка текущих заказов;
* Добавления и изменения деталей заказа;
* Изменение статусы заказов.

**Детали заказа.**

Система должна предоставлять возможность:

* Добавления новых деталей в выбранный заказ;
* Просмотра деталей выбранного заказа;
* Удаление или изменения деталей в выбранном заказе.

## Моделирование предметной области

Для реализации системы выделены четыре основные группы пользователей, каждая из которых выполняет свои специфические задачи:

* Официант – пользователь, который выполняет задачи:
  + Управление столами;
  + Управление заказами;
  + Отмечает завершение заказа;
  + Отмечает доставку блюда;
  + Отслеживает статус блюд и заказов.
* Повар – пользователь, который выполняет задачи:
  + Отмечает начало приготовление блюда
  + Отмечает доставку блюда;
  + Отслеживает появление новых заказов.
  + Бармен – пользователь, который выполняет задачи:
  + Отмечает начало приготовление блюда
  + Отмечает доставку блюда;
  + Отслеживает появление новых заказов.
* Администратор – пользователь, который выполняет задачи:
  + Заполнение базы данных о сотрудниках, категориях, блюдах в меню, столах.

Для наглядности рассмотрим диаграммы прецедентов для каждой из ролей.

На рисунке 3 представлена диаграмма прецедентов роли “Официант”:

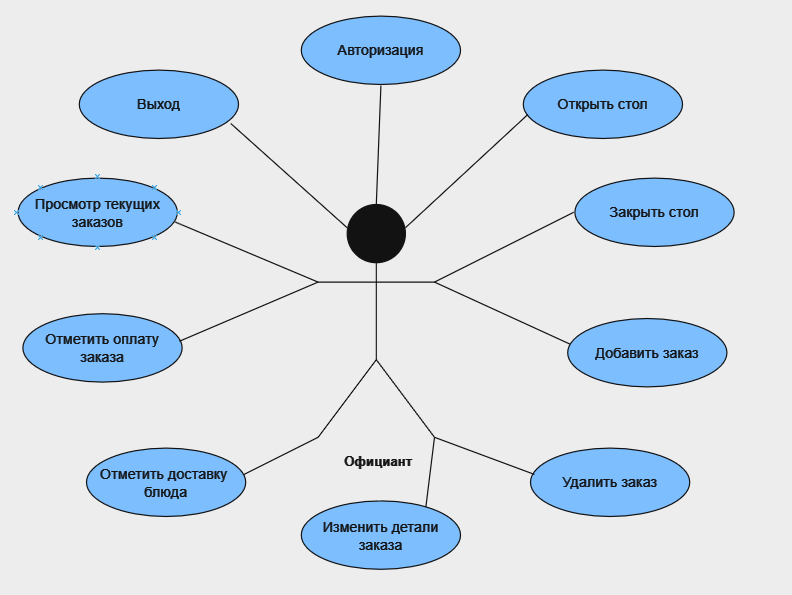


Рисунок 3 – Диаграмма прецедентов для роли официант

Описание прецедентов:

* Авторизация – вход пользователя в систему, после чего ему доступны возможности для его роли;
* Выход – выход пользователя из системы:
* Открыть стол – занимает доступный стол, меняя его статус на «Занят», автоматически создает пустой заказ;
* Закрыть стол – освобождение занятый стол, меняя его статус на «Свободен», и изменят статус всех заказов, прикреплённых к столу на «Завершен»;
* Создать заказа – создание заказ у выбранного стола;
* Изменить детали заказа – добавление, удаление позиций из меню в заказе;
* Удалить заказ – удаляет заказ у выбранного стола;
* Отметить доставку блюда – изменить статус блюда на «Доставлено».
* Отметить оплату заказа – изменить статус заказа на «Оплачен».

На рисунке 4 представлена диаграмма прецедентов роли “Повар”:

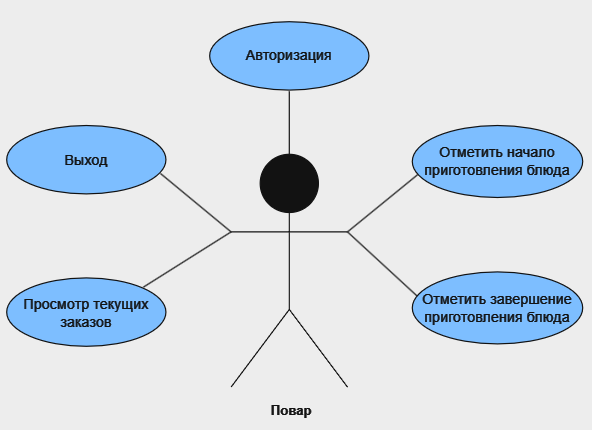


Рисунок 4 – Диаграмма прецедентов для роли повар

На рисунке 5 представлена диаграмма прецедентов роли “Бармен”:

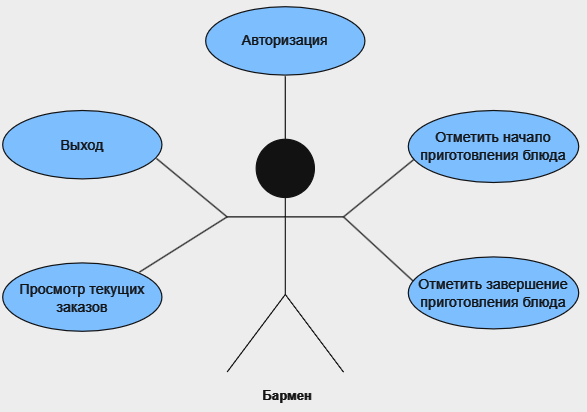


Рисунок 5 – Диаграмма прецедентов для роли бармен

Описание прецедентов:

* Отметить начало приготовление блюда – изменить статус блюда на «Готовится»
* Отметить завершение приготовление блюда – изменить статус блюда на «Ожидает доставки»

На рисунке 6 представлена диаграмма прецедентов роли “Администратор”:

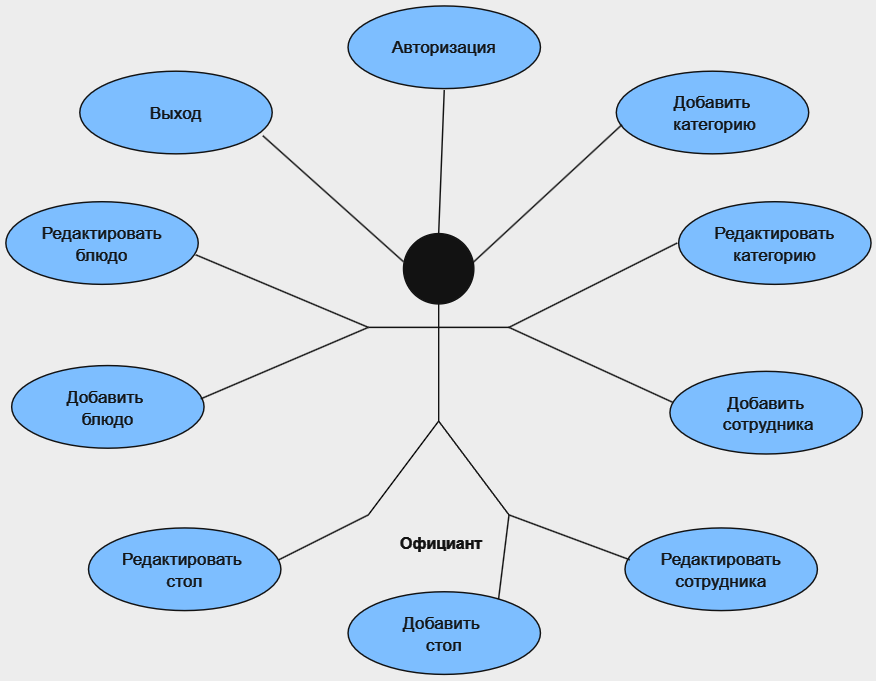


Рисунок 6 – Диаграмма прецедентов для роли администратор

Описание прецедентов:

* Добавить сотрудника – добавление нового сотрудника с заполненными данными;
* Добавить стол – добавляет новый стол с заполненными данными;
* Добавить категорию – добавление новую категорию с заполненными данными;
* Добавить блюдо – добавление новое блюдо с заполненными данными;
* Редактировать сотрудника – удаление сотрудника или изменение информации о нём;
* Редактировать стол – удаление стола или изменение информации о нём;
* Редактировать категории – удаление категории или изменение информации о ней;
* Редактировать блюдо – удаление блюда или изменение информации о нём;

Также на рисунке 7 представлена диаграмма бизнес-процесса обслуживания гостя.

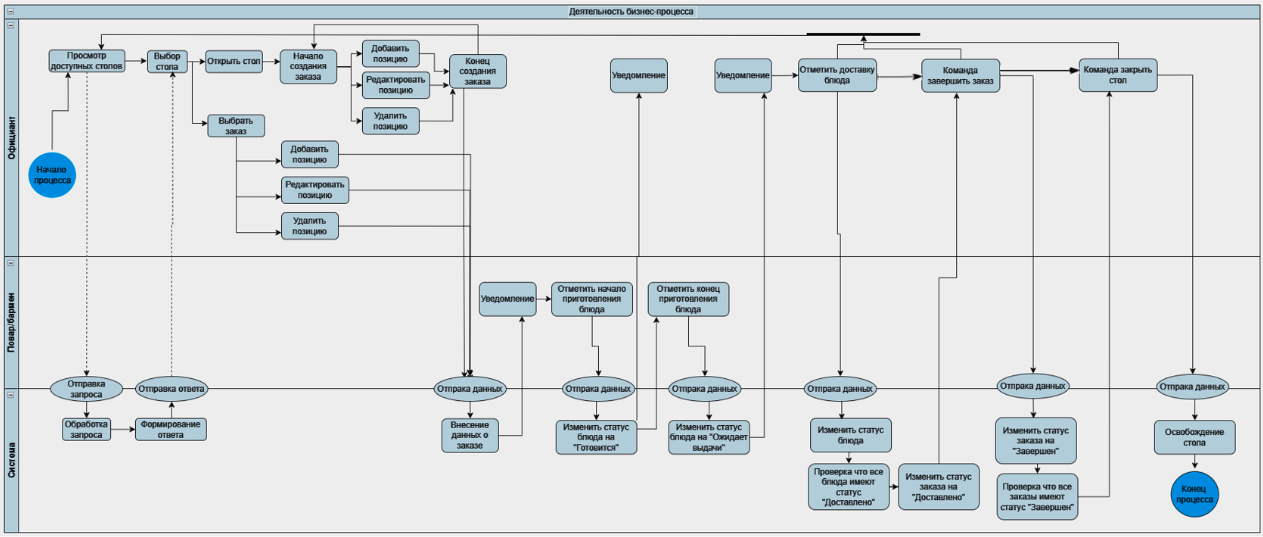


Рисунок 7 – Диаграмма бизнес-процесса обслуживания гостя.

# Выбор средств разработки

## Выбор среды разработки

Интегрированная среда разработки (IDE) – это программное обеспечение, предназначенное для облегчения разработки программного обеспечения, как правило состоит из редактора кода, компилятора, средств автоматизации сборки и отладчика.

**IntelliJ IDEA** – одна из наиболее популярных сред разработки созданная компанией JetBrains В 2001 году, является популярной средой, поддерживающей разработку на таких языках, как Java, Python, JavaScript, предоставляющая для разработчиков функции авто дополнения и анализа кода, удобный и понятный графический интерфейс, а также интегрирована с системами контроля версий.

Данная программа имеет бесплатную версию для обучения и платную для коммерческого использования.

**Android Studio** – официальная среда разработки для создания Android приложений, разработанная компанией Google в 2013 году, включающая в себя все необходимые инструменты для разработки, тестирования и отладки Android-приложений, включая интеграцию с Android SDK, визуальные инструменты для дизайна интерфейсов, систему сборки Gradle, а также встроенные эмуляторы что позволяет запускать приложение прямо из IDE на различных устройствах, что , по моему мнению, крайне облегчает процесс разработки. В её основу легла среда, разработанная компанией JetBrains – IntelliJ IDEA. Но, в отличие от неё, является полностью бесплатной.

**Eclipse** – среда разработки с открытым исходным кодом, созданная в 2001 году компанией IBM и обретшая большую популярность. Она поддерживает разработку на различных языках программирования, хотя изначально задумывалась как Java IDE. Для разработки Android приложение необходимо установить дополнительный плагин ADT (Android Development Tools), добавляющий необходимые инструменты для работы с Android SDK.

Для разработки клиентской части приложения была выбрана Android Studio, ввиду наличия встроенного эмулятора, упрощающего тестирование приложений на различных устройствах.

Для серверной части был выбран IntelliJ IDEA благодаря удобству рефакторинга и наличию опыта разработки в данной среде.

## Выбор языка программирования

Для разработки Android приложений в Anroid Studio используют два основных языка Java и Kotlin.

**Java** [6] – это один из самых популярных языков программирования, разработанный компанией Sun Microsystems в 1995 году, получивший широкую известность, благодаря технологии (JVM). Java Virtual Machine – интерпретатор, обрабатывающий полученный байт-код и передающий соответствующие инструкции устройству.

Данных подход позволяет писать Java-приложения, не задумываясь о том, на каком устройстве оно будет запущено**,** главное лишь наличие на нём JVM.

Также Java имеет большое количество серверных фреймворков, таких как Spring и Hibernate, что будет хорошим подспорьем при разработке серверной части.

**Kotlin** – современный язык программирования, разработанный компанией JetBrains в 2011 году и созданный на замену Java, он полностью совместим с Java и также использует JVM.

Еще одной особенностью Kotlin является наличие лаконичного синтаксиса, а также безопасные типы, что исключает ошибку с null-значениями.

Для разработки приложения был выбран язык программирования Java. Этот выбор обусловлен опытом разработки на Java, а также наличие удобных для разработки серверной части фреймворков.

## Выбор СУБД

Система управления базами данных – программное обеспечение, используемое для создания, управления и манипулирования базами данных, также обеспечивает интерфейс между пользователями и БД, позволяя хранить, извлекать и управлять данными.

**MySQL** – свободная система управления реляционными базами данных, поддерживаемая корпорацией Oracle, и написанная на языках Си и C++ с открытым исходным кодом, использующая язык SQL для управления данными и доступа к ним. На данным момент является одной из самых популярных открытых СУБД.

Из преимуществ стоит отметить высокую скорость работы и популярность, из минусов же неполное соответствие требования ASID.

**SQLite** – встраиваемая СУБД, хранящая все данные в одном файле, поставляется в виде библиотеки, встраиваемой в код приложения, работающего с СУБД [12].

Из плюсов стоит отметить отсутствие необходимости устанавливать дополнительный сервер СУБД, т.к. SQLite является встраиваемой системой, а из минусов стоит выделить малое количество поддерживаемых данных, а также отсутствие возможности горизонтально масштабировать БД.

**PostgreSQL [9]** – свободная объектно-реляционная система управления базами данных, поддерживаемая сообществом postgresql, написанная на языке Си. Существует в реализациях для множества UNIX-подобных платформ, включая AIX, различные BSD-системы, HP-UX, IRIX, Linux, macOS, Solaris/OpenSolaris, Tru64, QNX, а также для Microsoft Windows.

Их плюсов стоит отметить поддержку NoSql типов данных, а также полное соответствие требованиям ACID, у PostgreSQL тяжело выделить прям явные минуса, однако стоит отметить сложность конфигурации и использовании относительно других рассматриваемых СУБД.

PostgreSql была выбрана по причине наличия в ней поддержки Nosql типов данных, а также наличия опыта работы с ней.

## Выбор фреймворков

Для разработки серверной части приложения были выбраны два фреймворка: Spring Framework и Hibernate. Эти фреймворки хорошо зарекомендовали себя в индустрии.

**Spring Framework [13]**– фреймворк для разработки корпоративных приложений на языке Java, разработанный компанией Rod Johnson в 2003 году, Spring стал одним из самых популярных фреймворков для создания серверных приложений, благодаря своей модульной архитектуре.

Spring состоит из множества модулей, позволяющих разработчикам использовать необходимые для конкретного проекта компоненты. Из основных модулей можно назвать Spring Core, Spring MVC, Spring Data, а также Spring Security.

Также существует подмножество Spring Boot, позволяющие значительно упростить создании и развёртывание приложений, что, как я считаю, крайне важно для разработки.

**Hibernate** [4] – ORM фреймворк, разработанный в 2001 году компанией Gavin King и позволяющий взаимодействовать с базами данных на уровне объектов, а не таблиц.

С помощью аннотаций, как на рисунке 7 Hibernate позволяет работать не с SQL запросами к БД, а с объектами Java, что значительно проще и интуитивно понятнее.

Помимо прочего Hibernate легко интегрируется со Spring Framework, что позволяет использовать возможности обоих фреймворков для создания приложений.

# Архитектурное проектирование

## Общая архитектура

Приложение является RESTful - сервисом. Сервер хранит все модели, а клиент при необходимости запрашивает их с использование HTTP- запросов.

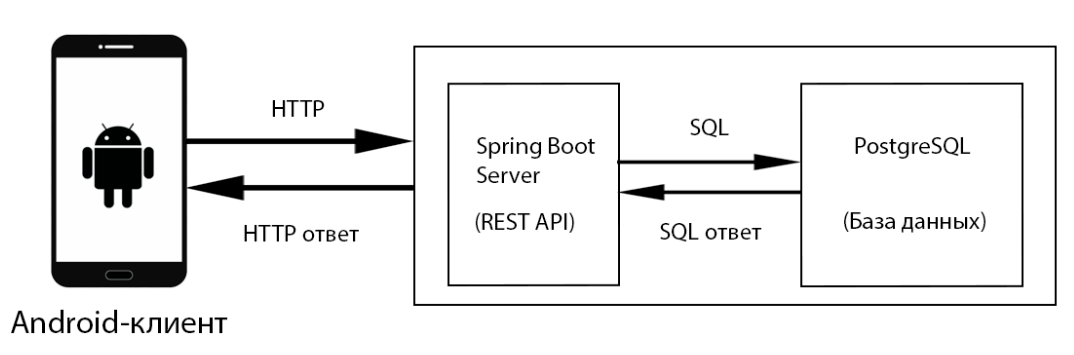


Рисунок 8 – Архитектура системы

**REST.**

REST (Representational State Transfer) [15] – это архитектурный стиль программирования, используемый для создания веб-сервисов, которые взаимодействуют по протоколу HTTP. Основные принципы REST включают:

* Клиент-серверная архитектура: Разделение клиента и сервера, что позволяет им развиваться независимо друг от друга.
* Без состояния (stateless): Каждый запрос от клиента к серверу должен содержать всю информацию, необходимую для его обработки. Сервер не хранит состояния между запросами.
* Кэшируемость: Ответы сервера могут быть помечены как кэшируемые, что позволяет клиентам и промежуточным серверам сохранять и повторно использовать их.
* Единообразие интерфейса: Использование стандартных методов HTTP (GET, POST, PUT, DELETE) для выполнения операций над ресурсами, представленными в виде URI (Uniform Resource Identifier).

## Объектная модель

На основе анализа предметной области было выделены 6 основных сущностей:

* + Пользователь;
  + Аккаунт;
  + Категория;
  + Стол;
  + Блюдо;
  + Деталь заказа;
  + Заказ;
  + Роль;
  + Статус заказа;
  + Статус стола;
  + Статус детали заказа.

Общая диаграмма сущностей представлена на рисунке 9:

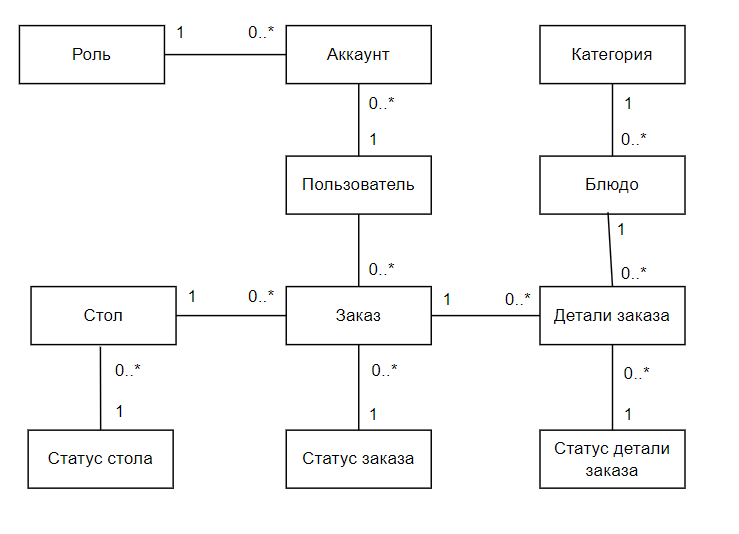


Рисунок 9 – Общая диаграмма сущностей

Диаграмма сущности “Пользователь” представлена на рисунке 10:

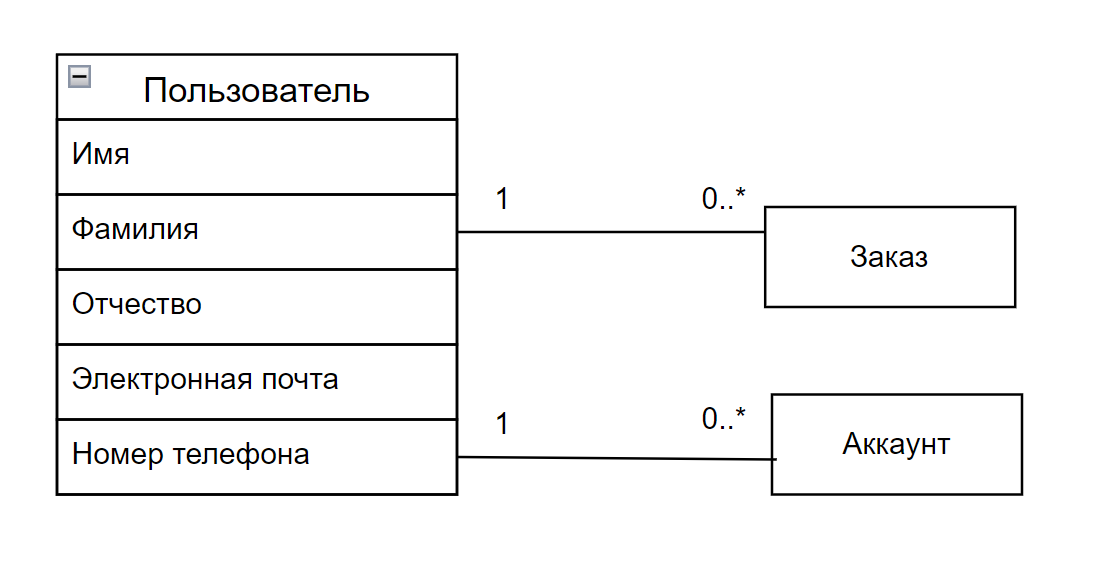


Рисунок 10 – Диаграмма сущности “Пользователь”

Данная сущность включает в себя:

* + Имя – имя пользователя для профиля;
  + Фамилия – фамилия пользователя для профиля;
  + Электронный адрес – адрес электронной почты пользователя;
  + Номер телефона – номер телефона пользователя.

Диаграмма сущности “Аккаунт” представлена на рисунке 11:

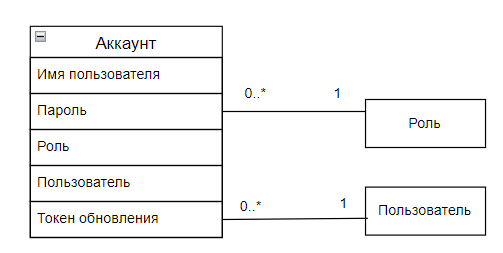


Рисунок 11 – Диаграмма сущности “Аккаунт”

Данная сущность включает в себя:

* + Имя пользователя – псевдоним, необходимый для авторизации;
  + Пароль – секретный ключ, необходимый для авторизации, хранится в зашифрованном виде
  + Роль – права, которые соответствует данному аккаунту;
  + Пользователь – владелец данного аккаунта;
  + Токен обновления – токен обновления, используется для аутентификации.

Диаграмма сущности “Роль” представлена на рисунке 12

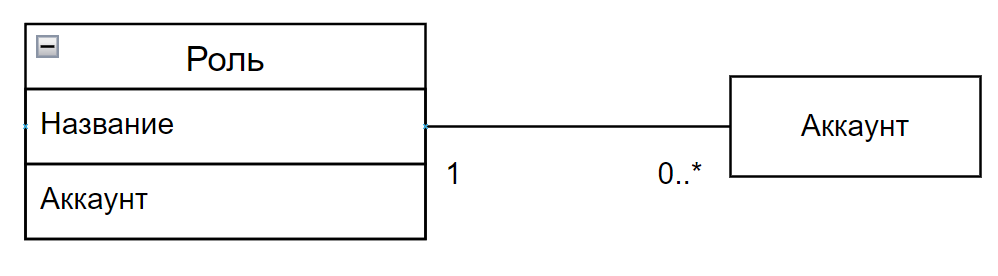


Рисунок 12 – Диаграмма сущности “Роль”

Данная сущность включает в себя:

* + Название – наименование роли;
  + Аккаунт – аккаунты, имеющие данную роль.

Диаграмма сущности “Блюдо” представлена на рисунке:

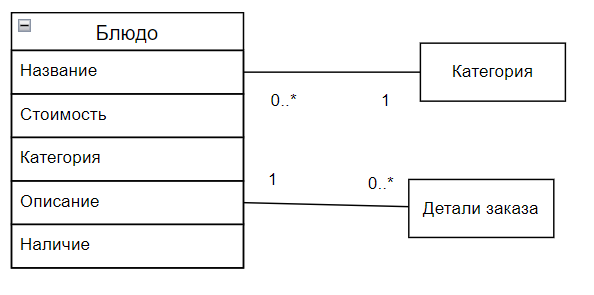


Рисунок 13 – Диаграмма сущности “Блюдо”

Данная сущность включает в себя:

* + Название – наименование блюда;
  + Стоимость – цена на данное блюдо
  + Категория – категория на данное блюдо.
  + Описание – второстепенная информация о блюде.
  + Наличие – значение, отражающее есть ли данное блюдо в наличии.
  + Место приготовления – значение, отражающее кто будет готовить блюдо

Диаграмма сущности “Категория” представлена на рисунке 14:

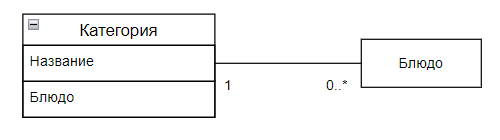


Рисунок 14 – Диаграмма сущности “Категория”

Данная сущность включает в себя:

* + Название – наименование категории;
  + Блюда – блюда, имеющие данную категорию.

Диаграмма сущности “Детали заказа” представлена на рисунке 15:

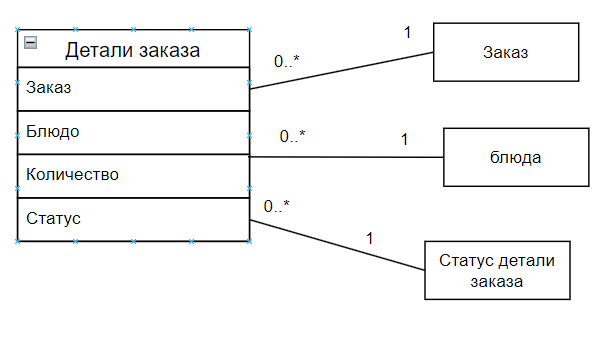


Рисунок 15 – Диаграмма сущности “Детали заказа”

Данная сущность включает в себя:

* + Заказ –заказа, которому принадлежит данная деталь заказа;
  + Блюдо – блюдо, которое находится в данной детали заказа;
  + Количество – количество блюд в детали заказа.
  + Статус – информация о состоянии заказа.

Рисунок 15 – Диаграмма сущности “Заказ” представлена на рисунке 16:

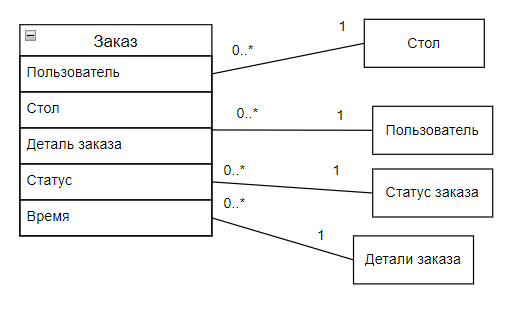


Рисунок 16 – Диаграмма сущности “Заказ”

Данная сущность включает в себя:

* + Пользователь – официант, который создал данный заказ;
  + Стол – номер стола, за которым создан заказ;
  + Деталь заказа – детали заказа, принадлежащие данному заказу;
  + Статус – информация о состоянии заказа;
  + Время – дата и время создания заказа.

Диаграмма сущности “Стол” представлена на рисунке 17:

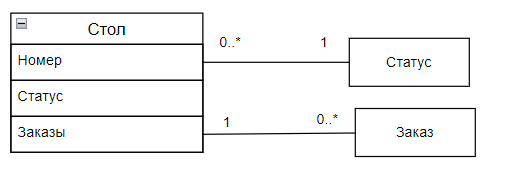


Рисунок 17 – Диаграмма сущности “Стол”

Данная сущность включает в себя:

* + Номер – номер данного стола;
  + Статус – информация о состоянии стола;
  + Заказы – заказа, принадлежащие данному столу.

## Структура базы данных

На основе сформированных объектных моделей спроектирована база данных, структура которой представлена на рисунке 18.

Объектные модели в базе данных представлены в качестве таблиц, атрибуты моделей – как колонки этих таблиц определенных типов.

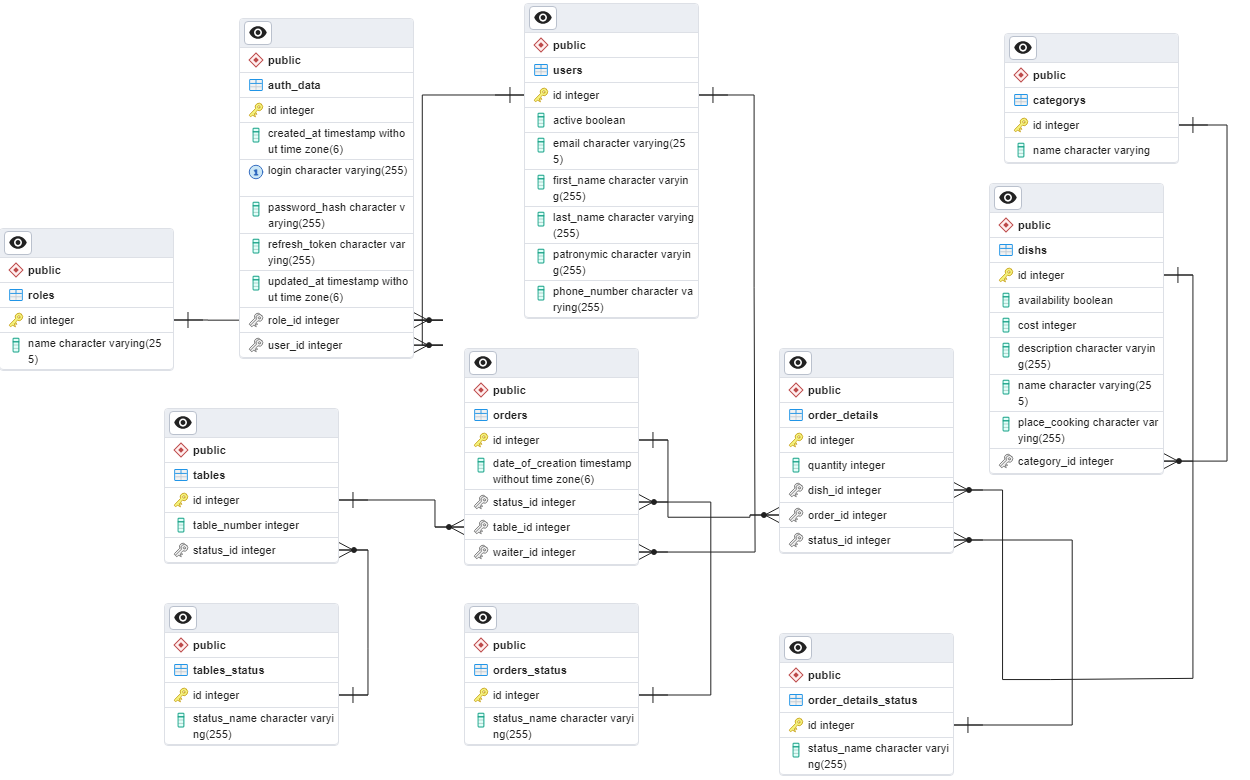


Рисунок 18 – Структура базы данных

## **Аутентификация**

Аутентификация реализована с помощью **JSON Web Token (JWT)** [12] — это стандарт открытого представления токенов в формате JSON, используемый для передачи информации между сторонами в компактной и защищенной форме. JWT широко используется для аутентификации и авторизации в веб-приложениях.

Основные компоненты JWT

JWT состоит из трех частей, разделенных точками (.):

* + **Header (Заголовок)**: содержит информацию о типе токена и алгоритме подписи.
  + **Payload (Полезная нагрузка)**: содержит утверждения (claims), представляющие собой данные, которые нужно передать. Обычно это информация о пользователе и метаданные.
  + **Signature (Подпись)**: используется для верификации подлинности токена и целостности данных. Создается путем кодирования заголовка и полезной нагрузки с использованием секретного ключа или приватного ключа.

Пример JWT токена изображен на рисунке 19

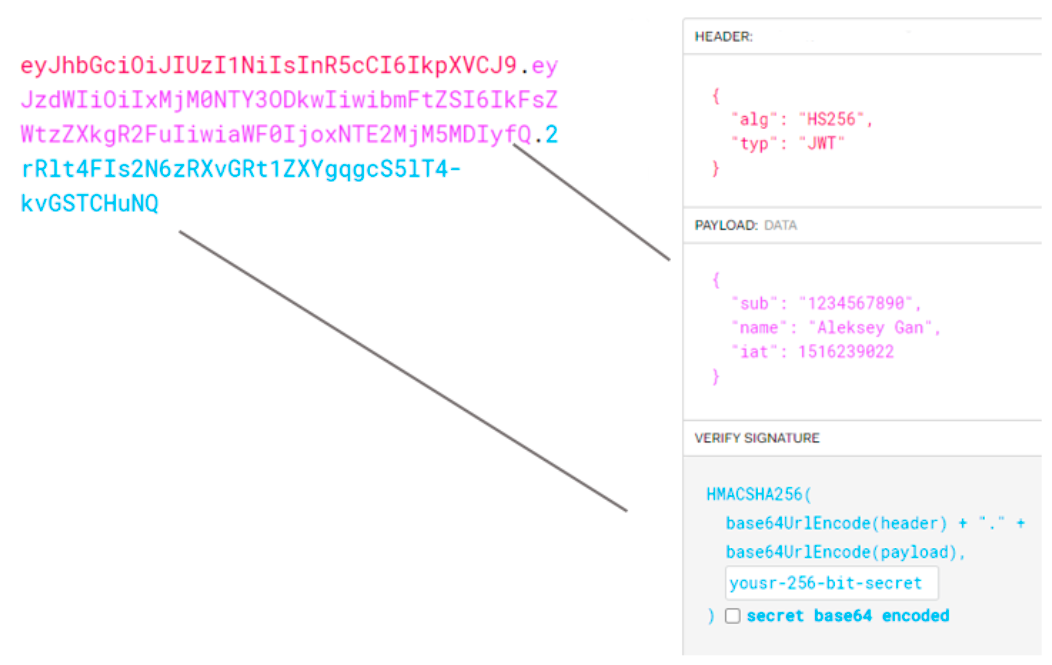


Рисунок 19 – Кодированный и декодированный JWT токен

Существует два вида JWT токена:

**Токен доступа**

**Токен доступа (Access Token)** – это короткоживущий токен, который используется для доступа к защищенным ресурсам API. Обычно он имеет ограниченный срок действия (несколько минут или часов). Основные характеристики Access Token:

* + **Короткий срок действия**: обычно действует от нескольких минут до нескольких часов.
  + **Использование**: Каждый запрос к защищенному ресурсу API включает Access Token в заголовке Authorization.
  + **Безопасность**: из-за короткого срока действия, даже если токен будет скомпрометирован, злоумышленник сможет использовать его лишь в течение короткого времени.

После успешной аутентификации пользователь получает Access Token, который используется для всех последующих запросов к защищенным ресурсам.

**Токен обновления**

**Токен обновления (Refresh Token)** – это долговременный токен, который используется для получения нового Access Token после того, как текущий Access Token истечет. Refresh Token обычно имеет более длительный срок действия (несколько дней или месяцев). Основные характеристики Refresh Token:

**Долгий срок действия**: обычно действует от нескольких дней до нескольких месяцев.

Когда Access Token истекает, клиент использует Refresh Token для получения нового Access Token от сервера аутентификации.

При аутентификации процесс получения токенов выглядит следующим образом:

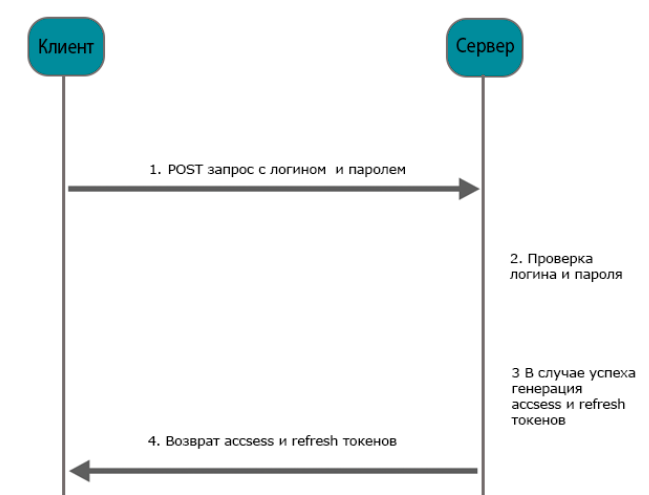


Рисунок 20 – Процесс получения JWT токенов

При последующих запросах клиент включает JWT в заголовок Authorization (Authorization: Bearer <token>).

Сервер проверяет валидность JWT и извлекает информацию о пользователе для обработки запроса.

## WebSocket

Официанты и повара должны получать актуальную информацию о состоянии заказов и блюд, но REST не очень подходит для, после CRUD операции не позволяет обновлять данные у всех клиентов. Поэтому для реализации функции постоянного обновления информации о заказах в приложении принято решение использовать протокол WebSocket [11], являющийся протоколом связи и предоставляющий постоянное двунаправленное соединение между клиентом и сервером. В отличие от традиционных HTTP-запросов, WebSocket позволяет передавать данные в реальном времени без необходимости повторного установления соединения.

Постоянное соединение позволяет поддерживать одно открытое соединение между клиентом и сервером на протяжении всего времени их взаимодействия, что дает серверу возможность отправлять данные клиенту в любое время, не дожидаясь запроса от него. Также двунаправленность соединения позволяет клиенту и серверу общаться между собой по одному и тому же каналу связи, что улучшает скорость и эффективность обмена данными по сравнению с традиционными HTTP-запросами.

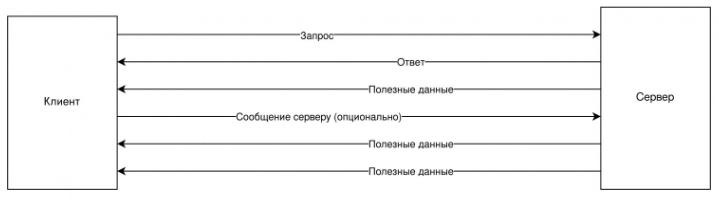
Процесс работы с WebSocket представлена на рисунке 21:  


Рисунок 21 – Процесс работы с WebSocket

Описание процесса:

После успешной аутентификации пользователя приложение устанавливает WebSocket-соединение с сервером, используемое для передачи данных в реальном времени. При изменении статуса заказа или другого события, сервер отправляет соответствующее сообщение через WebSocket всем подключенным клиентам, после чего клиентские приложения получают эти сообщения и обновляют пользовательский интерфейс, отображая актуальную информацию.

## Model-View-ViewModel

Для разработки клиентской части приложения принято решение использовать архитектурный шаблон MVVM [14], чья аббревиатура расшифровывается, как (Model-View-ViewModel). Отличительная особенность от таких шаблонов, как MVC и MVP заключается в отделении логики представления данных от бизнес-логики, путем вынесения её в отдельный класс для более четкого разграничения. Теперь определим, что же обозначает каждое слова в названии.

Model – подразумевает под собой логику работы с данными приложения, обычно это POJO классы.

View – layout экрана, располагающий все необходимые виджеты для отображения информации.

View-Model – связующее звено между моделью и представлением, содержащее логику для обновления данных в модели и их извлечения для передачи в представление, однако ViewModel ничего не знает о графическом представлении. Графический интерфейс автоматически меняется при изменении состояния ViewModel.

На рисунке 22 представлен принцип работы MVVM:

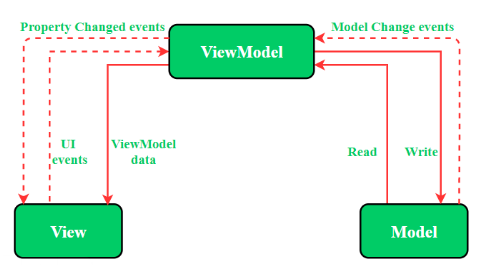


Рисунок 22 – Принцип работы MVVM

## Single Activity

Также для создания графического интерфейс приложение было принято решение использовать паттерн **Single Activity [7], отличительной особенностью которого является** использование одной основной Activity в котором имеется контейнер для фрагментов, в свою очередь фрагменты являются отдельными экранами и сменяют друг друга при переходе пользователя с одного экрана на другой.

Данный паттерн позволяет создавать плавные анимации между переходами, а также прощает передачу данных между экранами

Также наличие одного activity позволило без особых затруднений обмениваться данными между экранами. Это возможно благодаря общему контейнеру для ViewModel, привязанному к activity. И поскольку на всё приложение одно activity, то и все ViewModel будут существовать до конца работы приложения, что позволяет не беспокоиться о сохранении данных при переходе со экрана на экран.

## Retrofit

Для передачи данных с клиент на сервер в приложение используется библиотека Retrofit [5], являющаяся типобезопасным HTTP–клиентом для Android и Java, разработанный компанией Square в 2013 году.

Retrofit позволяет отправлять HTTP-запросы типа POST, GET, PUT и DELETE, обеспечивая простое взаимодействие с веб-сервисами.

Кроме того, библиотека упрощает сериализацию данных используя различные конвертеры, такие как Gson, преобразующий JSON в обычный Java объект.

С помощью аннотаций указываются web-команды. В самой аннотации указывается метод, используемый на сервере:

* @GET - GET-запрос;
* @POST – POST-запрос;
* @FormUrlEncoded – необходим для пар “ключ/значение”;
* @Query – необходим для задания ключа запроса со значением параметра.
* @Field – необходим для пар “ключ/значение”;
* @Multipart – необходим для загрузки видео и изображений;
* @Headers – добавляет заголовки в запрос;
* @PUT – PUT-запрос;
* Для использования динамически параметров необходимо указать значение в фигурных скобках. Пример представлен на рисунке 23:

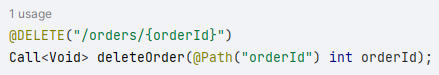


Рисунок 23 – Использование динамических параметров в запросе

# 4. ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

## Функциональное проектирование серверной части

### 4.1.1 Сущности

Рассмотрим на примере сущности *“*Блюдо”, на рисунке 24 изображен код класса *Dishes* с его полями.



Рисунок 24 – Код сущности Dishes

С помощью аннотаций Hibernate отображает объект на реляционные БД и управляет взаимодействие с ней, для наглядности рассмотрим все аннотации на рисунке

Аннотация *@Entity* позволяет связать класс *Dishes* с таблицей, указывая, что это сущность (entity), далее идёт аннотация *@Table* с параметрами name, schema, catalog, данная аннотация даёт дополнительную информацию о том, как сущность должна быть сопоставлена с таблицей, указывая имя таблицы, имя схемы, а также имя каталога БД,

Аннотация *@Id* указывает что поле является первичным ключом, а аннотация *@GeneratedValue* указывает, что он будет создаваться автоматически и с авто инкрементированием, благодаря параметру *GenerationType.IDENTITY.*

Аннотация *@Column* позволяет указать имя для столбцов в таблице БД, соответствующим полям сущности, а с помощью аннотации *@JoinColumn* и *@ManyToOne* позволяет указать, что поле является первичным ключом, имеющим связь многие к одному.

Для изменения полей сущности используются геттеры и сеттеры, расположенные на рисунке 25:

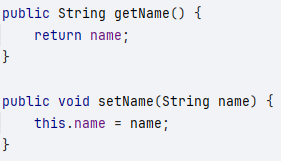


Рисунок 25 – Геттеры и сеттеры сущности Dishes

Рассмотрим на рисунке 26 реализацию класса *DishesDao*, включающий в себя методы для работы с репозиторием *DishesRepository*, позволяющим выполнять запросы к БД.

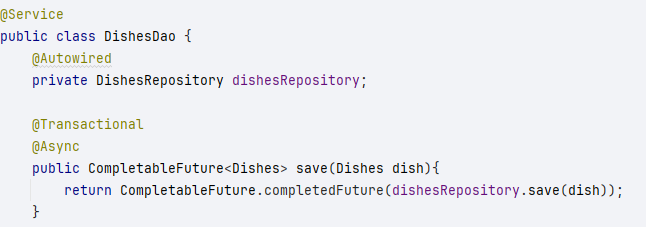


Рисунок 26 – Реализация класса DishesDao

Аннотация *@Service* означает, что это сервисный компонент Spring, как правило в них реализуется взаимодействие с репозиторием.

Класс *DishesController* позволяет обрабатывать обрабатывает HTTP-запросы и возвращает данные в формате JSON. В данном контроллере реализованные конечные точки для модели *Dishes*.



Рисунок 27 – Реализация класс DishesController

Аннотация *@RestController* говорит Spring Framework, что этот класс является контроллером, который обрабатывает HTTP-запросы и возвращает данные в формате JSON, в свою очередь аннотация *@GetMapping* указывает, что метод *getAllDishes* обрабатывает HTTP-запросы по URL адресу “/dishes/get-all”

### 4.1.2 Api сервера

На сервере реализованы конечные точки (endpoint), реализующие CRUD операции. CRUD – это акроним, обозначающий четыре базовые функции при работе с данными: create, read, update, delete, рассмотрим endpoint для всех сущностей:

Конечные точки для работы с сущностью “Категория” представлены в таблице 2:

Талица 2 – Конечные точки сущности “Категория”

|  |  |  |  |
| --- | --- | --- | --- |
| URL | Метод | Параметры запроса тело запроса, часть URL | Описание |
| /caterogies/get-all | GET |  | Получение списка блюд. |
| /caterogies/save | POST | Тело запроса:  category- категория для добавления | Добавление новой категории. |
| /caterogies/update/{id} | PUT | Часть URL:  id – индивидуальный номер категории.  Тело запроса:  category- категория для обновления | Обновление категории. |
| /caterogies/ id} | DELETE | Часть URL:  id – индивидуальный номер. категории | Удалении категории. |
| URL | Метод | Параметры запроса тело запроса, часть URL | Описание |

Конечные точки для работы с сущностью “Блюдо” представлены в таблице 3:

Талица 3 – Конечные точки сущности “ Блюдо”

|  |  |  |  |
| --- | --- | --- | --- |
| **URL** | **Метод** | **Параметры запроса тело запроса, часть URL** | **Описание** |
| /dishes | POST | Тело запроса:  dish - блюдо для добавления | Добавление нового блюда. |
| /dishes/{id} | PUT | Часть URL:  id – индивидуальный номер блюда.  Тело запроса:  dish - блюдо для обновления | Обновление блюда. |

Продолжение таблицы 3

|  |  |  |  |
| --- | --- | --- | --- |
| /dishes/{id} | DELETE | Часть URL:  id – индивидуальный номер блюда. | Удалении блюда. |
| /dishes/{categoryId} | GET | Часть URL:  category – индивидуальный номер категории. | Получения списка всех блюд в выбранной категории. |
| /dishes/get-all | GET |  | Получение списка блюд. |

Конечные точки для работы с сущностью “Стол” представлены в таблице 4:

Талица 4 – Конечные точки сущности “ Стол”

|  |  |  |  |
| --- | --- | --- | --- |
| URL | Метод | Параметры запроса тело запроса, часть URL | Описание |
| /tables/get-all | GET |  | Получение списка столов. |
| /tables | POST | Тело запроса:  table - стол для добавления | Добавление нового стола. |
| /tables/{id} | PUT | Часть URL:  id – индивидуальный номер стола.  Тело запроса:  table - стол для обновления | Обновление стола. |
| /tables/{id} | DELETE | Часть URL:  id – индивидуальный номер стола. | Удалении стола. |

Конечные точки для работы с сущностью “Заказ” представлены в таблице 5:

Талица 5 – Конечные точки сущности “ Заказ”

|  |  |  |  |
| --- | --- | --- | --- |
| URL | Метод | Параметры запроса тело запроса, часть URL | Описание |
| /orders/get-all | GET |  | Получение списка всех заказов. |
| /orders | POST | Тело запроса:  order - заказ для добавления | Добавление нового заказа. |
| /orders/{id} | PUT | Часть URL:  id – индивидуальный номер заказа.  Тело запроса:  order - стол для обновления | Обновление заказа. |
| /orders/{id} | DELETE | Часть URL:  id – индивидуальный номер заказа. | Удалении заказа. |
| /orders/{userId} | GET | Часть URL:  userId – индивидуальный номер пользователя. | Получение списка всех заказов, созданных выбранным официантом. |
| /orders/{userId}/{tableId} | GET | Часть URL:  userId – индивидуальный номер пользователя.  tableId – индивидуальный номер стола | Получения всех заказов выбранного официанта за выбранным столом. |
| /orders/status/  {orderId}/{statusId} | PUT | Часть URL:  orderId – индивидуальный номер заказа.  statusId – индивидуальный номер статуса заказа. | Изменение статуса у выбранного заказа. |

Конечные точки для работы с сущностью “Пользователь” представлены в таблице 6:

Талица 6 – Конечные точки сущности “Пользователь”

|  |  |  |  |
| --- | --- | --- | --- |
| URL | Метод | Параметры запроса тело запроса, часть URL | Описание |
| /users/get-all | GET |  | Получение списка всех пользователей. |
| /users | POST | Тело запроса:  user - пользователь для добавления | Добавление нового пользователя. |
| /users/{id} | PUT | Часть URL:  id – индивидуальный номер пользователя  Тело запроса:  order - пользователь для обновления | Обновление пользователя. |
| /users/{id} | DELETE | Часть URL:  id – индивидуальный номер пользователя. | Удалении пользователя. |

Конечные точки для работы с сущностью “Роль” представлены в таблице 7:

Талица 7 – Конечные точки сущности “Роль”

|  |  |  |  |
| --- | --- | --- | --- |
| **URL** | **Метод** | **Параметры запроса тело запроса, часть URL** | **Описание** |
| /roles{userId} | GET | Часть URL:  id – индивидуальный номер пользователя. | Получение списка ролей у выбранного сотрудника |
| /roles/get-all | GET |  | Получение списка всех ролей. |

Конечные точки для работы с сущностью “Аккаунт” представлены в таблице 8:

Талица 8 – Конечные точки сущности “Аккаунт”

|  |  |  |  |
| --- | --- | --- | --- |
| **URL** | **Метод** | **Параметры запроса тело запроса, часть URL** | **Описание** |
| /auth | POST | Тело запроса:  user – пользователь, которому будет принадлежать аккаунт;  role – выбранная роль | Создание аккаунты выбранной роли для выбранного пользователя. |
| /auth/login | POST | Параметры запроса:  login – имя пользователя;  password – пароль пользователя | Вход в аккаунт. |
| /auth/refresh | POST | Параметры запроса:  refreshToken – токен обновления | Получение нового токена обновления. |

Конечные точки для работы с сущностью “Детали Заказа” представлены в таблице 9:

Талица 9 – Конечные точки сущности “Детали заказа”

|  |  |  |  |
| --- | --- | --- | --- |
| URL | Метод | Параметры запроса тело запроса, часть URL | Описание |
| /ordersDetails/{orderId} | GET | Часть URL:  id – индивидуальный номер заказа. | Получить список деталей заказа. |
| /ordersDetails | POST | Параметры запроса:  quantity – количество блюд;  dishId - индивидуальный номер блюда;  status – статус;  orderId - индивидуальный номер заказа. | Добавление новой детали в заказ. |

Продолжение таблицы 9

|  |  |  |  |
| --- | --- | --- | --- |
| /ordersDetails | POST | Тело запроса:  orderDetail – деталь выбранного заказа | Изменение детали заказа |
| /ordersDetails/{id} |  |  |  |
| / ordersDetails /{id} | DELETE | Часть URL:  id – индивидуальный номер пользователя. | Удалении пользователя. |

### 4.1.3 JWT токены

Реализация процесса получения токенов доступа и обновления представлена на рисунке:



Рисунок 28 – Программная реализация получения JWT токенов

Описание действий:

Сначала проверяется наличие пользователя с введенным именем пользователя, далее при помощи алгоритма Argon2 пароль проходит верификацию, так как в БД он храниться в хэш-виде, если всю успешно, то генерируются токены доступа и обновления, затем в ответ записываются то токены, имя роли, а также пользователь.

Реализация процесса обновления токенов доступа и обновления представлена на рисунке 29:



Рисунок 29 – Программная реализация обновления JWT токенов

Описание действий:

Из токена обновления выделяется имя пользователя, затем в базе данных находится запись о пользователе, после этого сравнивается токен из БД и переданный в параметрах запроса, если проверка выполнилась успешно, то отправляет ответ с новыми токенами доступа и обновления.

### 4.1.4 WebSocket

Реализация процесса отправки сообщений через протокол WebSocket представлена на рисунке, для примера используем отправку сообщений при добавлении нового блюда;



Рисунок 30 – Программная реализация отправки сообщений через WebSocket

Описание действий:

Создает в базе данных блюдо, с параметрами полученными от клиента, затем преобразует добавленный объект JSON и с помощью метода *broadcastMessage* у класса *webSocketHandler* WebSocket отправляет всем клиентам сообщение, что в БД появилось новое блюдо.

## Функциональное проектирование клиентской части

### 4.2.1 POJO классы

Для представления данных, получаемых от сервера в формате JSON созданы POJO классы. POJO (Plain Old Java Object) – простые Java-классы, которые не зависят от каких-либо специфических библиотек или фреймворков. Для сериализации и десериализации объектов используется библиотека Gson.

На рисунке 31 представлен пример модели сущности "Блюдо".



Рисунок 31 – Программная реализация POJO класса Dishes

Чтобы правильно выполнить конвертацию, необходимо указать соответствие между полями класса и параметрами JSON-объекта. Это можно выполнить с помощью аннотации *@SerializedName*, где указывается имя параметра в JSON. Также в классе реализованы геттеры и сеттеры, а также метод equals. Остальные класс сущности реализована похожим образом.

### 4.2.2 Запуск приложения

При запуске приложения создаётся класс *MyApplication*, наследуемый от класса Application, данный класс необходим инициализация объекта *SharedPrefrences*, отвечающего за хранение в памяти устройства данных приложения, класс *ServerApi* для отправки запросов к серверу, а также поля для пользователя и его роли, изначально они равны null, так как при старте приложения пользователь не авторизован.

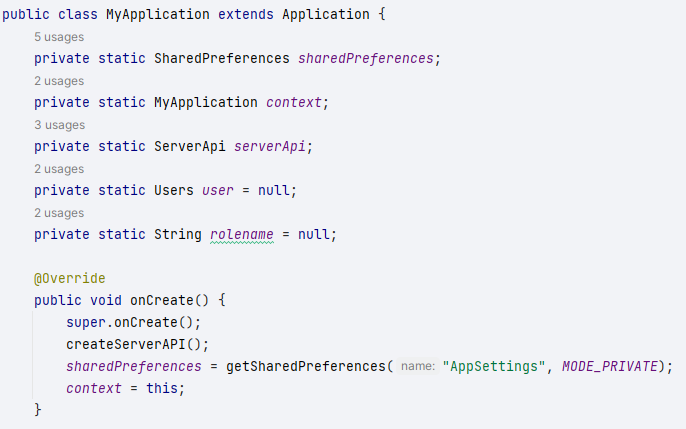


Рисунок 32 – Инициализация MyApplication

### 4.2.3 Токены доступа и обновления

При идентификации пользователя для получения доступа к данным, хранящимся на сервере, используются токен доступа и обновления.

Все токены хранятся в общих настройках устройства с использованием SharedPreferences, общие настройки являются частными, поэтому

Для выполнения большинства запросов требуется токен доступа, поэтому в каждый HTTP запрос необходимо устанавливать заголовок, а также сам токен доступа, для этого используется *OkHttpClient* с Interceptor для перехвата HTTP запрос.

Реализация процесса перехвата HTTP запросов продемонстрирована на рисунке 33:

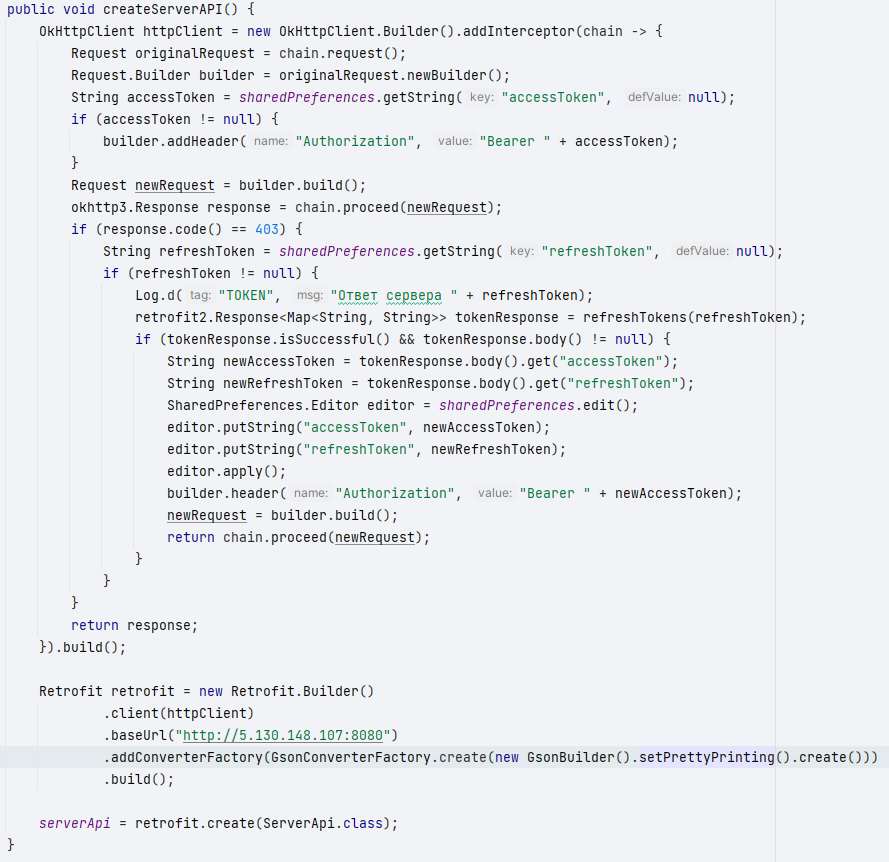


Рисунок 33 – Программная реализация перехвата HTTP запросов

Описание процесса:

Создаётся *OkHttpClient* с Interceptor, далее добавляем токен о отправляем запрос, если в ответе пришел код 403, то отправляем запрос на обновления токена с помощью метода *refreshTokents* с переданным токеном обновления, код метода представлен на рисунке 34

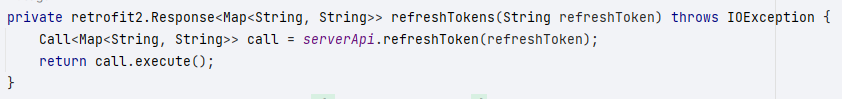


Рисунок 34 – Метод refreshToken

### 4.2.4 Запуск Activity

После создания объекта приложения происходит создание *MainActivity*, которое будет хранить все фрагменты в контейнере. При его инициализации создаётся *MainViewModel*, там будут храниться сообщения получаемые от WebSocket.



Рисунок 35 – Инициализация MainActivity

Также при создании Activity вызывается метод *loadFragment*, данный метод открывает нужный фрагмент.

Реализация метода *loadFragment* представлена на рисунке 36:

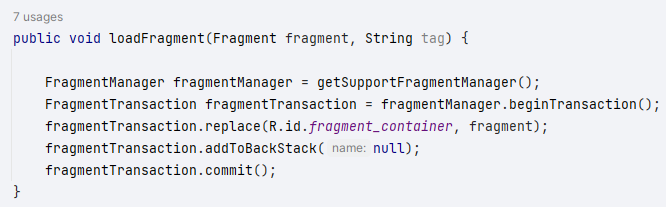


Рисунок 36 – Реализация метода loadFragment

В данном методе фрагмент в контейнере с идентификатором *R.id.fragment\_container* заменяется на новый фрагмент, если в контейнере уже есть фрагмент, он будет удалён, а новый фрагмент добавлен. При создании Activity данным метод позволяет перейти к фрагменту *LoginFragment* для дальнейшей авторизации.

### 4.2.5 Взаимодействие с сервером.

Для взаимодействия с сервером реализован интерфейс *ServerAPI*, он содержит методы, реализующие запросы к серверу.

На рисунке 37 изображен интерфейс с одним из своих методов, позволяющим добавить новую категорию в базу данных

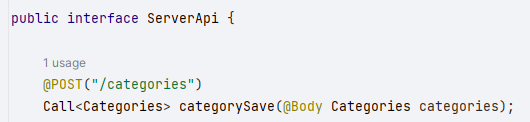


Рисунок 37 – Интерфейс ServerApi с методом categorySave

Для написания запроса к конечной точке на сервере необходимо указать URL путь, а также один из типов запроса: PUT, GET, DELETE, POST.

В методе в качестве возвращаемого значения задается Call<T>. В качестве T должен выступать необходимый класс модели или Response. Например, в методе на рисунке 38 изображен метод получения всех категорий.

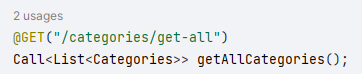


Рисунок 38 – Метод getAllCategories

В данном методе возвращается список объектов *Categories*.

Для реализации асинхронных запросов к серверу, которое не будут останавливать основной поток приложения, требуется вызвать метод у объекта *ServerApi*, а далее сформировать Callback, позволяющий при успешном запросе сериализовать данные в объект модели. На рисунке 39 представлен пример асинхронного запроса для получения всех категорий.

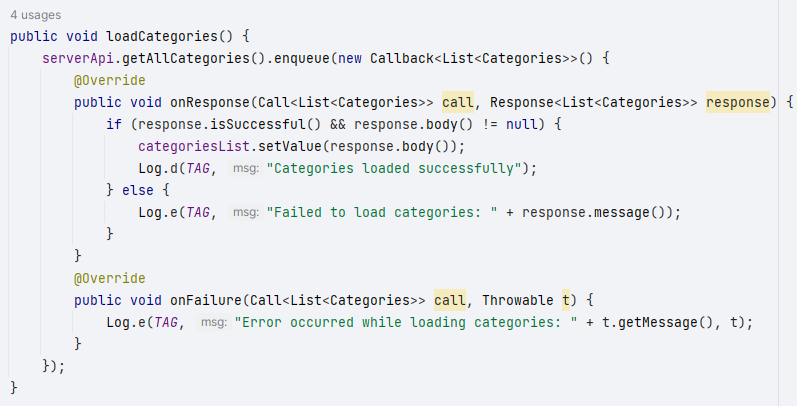


Рисунок 39 – Асинхронный запрос к серверу на получение всех категорий

### 4.2.6 Отображение получаемых данных

Каждый экран состоит с класса *Fragment*, класса *ViewModel* и XML разметки.

Пример разметки фрагмента *AdminDishesFragment*:



Рисунок 40 – XML разметка фрагмента Dishes

Также на рисунке 41 изображен графический интерфейс, соответствующий XML разметке на рисунке 40:



Рисунок 41 – XML графический интерфейс фрагмента Dishes.

Для хранения *ViewModel* используется *ViewModelProvider*. При запросе на получение *ViewModel* контейнер либо создаст новую, если она отсутствует, либо вернет уже существующую. Контейнер использует жизненный цикл переданного Activity или фрагмента. Благодаря паттерну Single Activity, можно привязать хранилище к Activity, что позволит *ViewModel* существовать на протяжении всего жизненного цикла приложения, данная функция крайне полезна для работы с WebSocket.

Пример создания *ViewModel* изображён на рисунке 42:  


Рисунок 42 – Создание mainViewModel

*ViewModel* имеют поля с типом *LiveData*, где локально хранятся модели до следующей загрузке с сервера.

На рисунке 43 изображен пример полей *LiveData* в *MainViewModel.*



Рисунок 43 – Класс MainViewModel с полями

На данные поля можно подписаться и при изменении *LiveData*, все подписавшиеся получат обновлённые данные.

Пример подписки на поле *LiveData* изображен на рисунке 44.



Рисунок 44. – Код подписки на LiveData поле

В данном примере происходит подписка на сообщения от WebSocket во фрагменте *AdminDishesFragment.*

### 4.2.7 WebSocket

Поскольку для подключения по WebSocket также необходим токен доступа, то подключаться следует после прохождения авторизации, и в зависимости от роли аккаунта, на который зашёл пользовать, подключение будет происходить в главном фрагменте соответствующей роли.

На рисунке 45 представлен процесс подключения по WebSocket в главном фрагменте роли “администратор” AdminMainFragment:



Рисунок 45 – Процесс подключения к серверу по протоколу WebSocket

При получении сообщения от сервера по WebSocket изменяется *LiveDate* в *MainViewModel*, все подписавшиеся также узнают об этом сообщении, после чего срабатывает метод *handleWebSocketMessage*, который выполняющий соответствующие функции в зависимости от текущего окна.

Для примера реализации обработки сообщений рассмотрим метод во фрагменте *WaiterOrdersDetailsFragment*. Данный метод продемонстрирован на рисунке 46.



Рисунок 46 – Реализация метода handleWebSocketMessage

### Отправка уведомлений

В зависимости от содержания полученного от WebSocket сообщения приложение создаёт уведомление на самом устройстве для пользователя.

При запуске *MainActivity* в методе *createNotificationChanel* создаётся канал уведомлений с заданными параметрами. После получения сообщения от WebSocket в зависимости от его содержания с помощью метода *sendNotification* отправляется уведомление на устройство с нужным заголовком и текстом сообщения.

Реализация данных методов продемонстрирована на рисунке 47



Рисунок 47 – Реализация методов отправки уведомления

Пример получения уведомления продемонстрирован на рисунке 48.

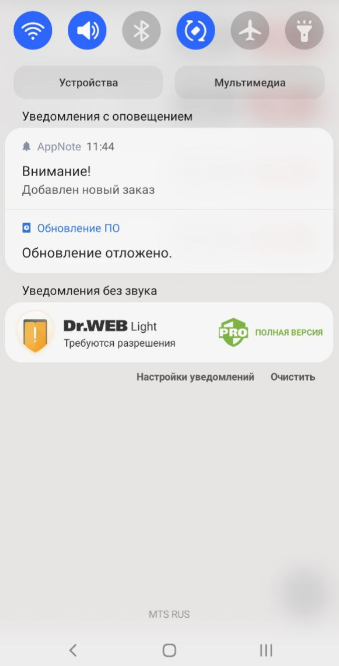


Рисунок 48 – Пример получения уведомления.

# 5 Руководство пользователя

## Вход в систему

При запуске приложение открывается начальное окно авторизации. Для входа в систему необходимо ввести логин и пароль из письма, полученного по электронной почте.

После ввода логина и пароля необходимо нажать кнопку “Войти”, если нет аккаунта с таким именем пользователя и паролем, то появится сообщение, сообщающее это, при верном вводе откроется одно из 3 окон, соответствующих роли пользователя на данном аккаунте.

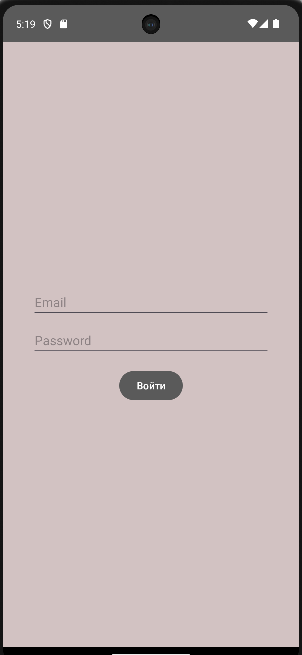


Рисунок 49 – Начальный экран при запуске приложения

## Интерфейс администратора

### 5.2.1 Начальный экран

Данное окно открывается при входе в аккаунт администратора.

По умолчанию открывается вкладка “Блюда”, в которой находится список блюд в меню. Интерфейс, описанный ниже для данной вкладки аналогичен и для других вкладок “Столы”, “Категории”. Пользователи.

Интерфейс данного окна представлено на рисунке 50.

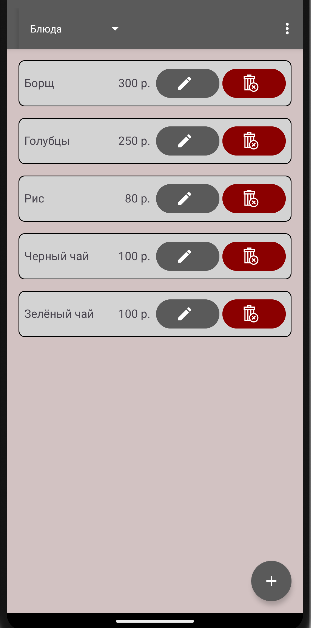


Рисунок 50 – Начальный экран при входе в аккаунт администратора.

### 5.2.2 Кнопка удаления

По нажатию кнопки с иконкой корзинки происходит удаление блюда из базы данных и после этого обновляется список блюд в приложении.

Внешний вид кнопки удаления на рисунке 51:



Рисунок 51 – Внешний вид кнопки добавления

### 5.2.3 Кнопка добавления

При нажатии на кнопку добавления, открывается экран создания нового блюда.

Внешний вид кнопки добавления на рисунке 52:



Рисунок 52 – Внешний вид кнопки добавления

### 5.2.4 Кнопка редактирования

При нажатии на кнопку редактирования, открывается экран редактирования выбранного блюда.

Внешний вид кнопки редактирования на рисунке 53:



Рисунок 53 – Внешний вид кнопки редактирования

### 5.2.5 Создание Блюда

В окне создания блюда необходимо заполнить текстовое поле “Название блюда”, “Стоимость блюда” и “Описание блюда”, также ниже представлены 2 списка с выбором категории блюда и места приготовления.

По нажатию на кнопку с текстом “Добавить блюдо” происходит добавление нового блюда в базу данных.

Интерфейс окна создания блюда представлен на рисунке 54:

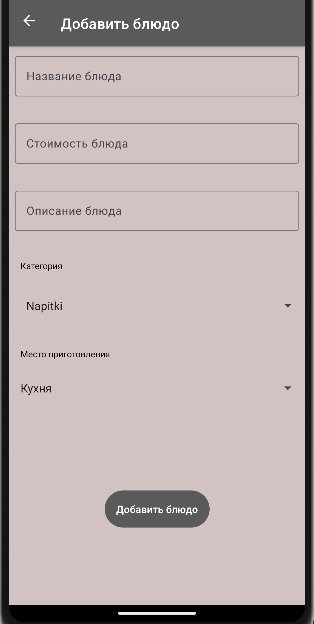


Рисунок 54 – Внешний вид окна добавления блюда

### 5.2.6 Кнопка возврата

По нажатию на кнопку в верхней панели с иконой стрелочки происходит возврат к предыдущему окну.

Внешний вид кнопки возврата представлен на рисунке 55.



Рисунок 55 – Внешний вид кнопки окна

### 5.2.7 Добавление сотрудника

Во вкладке “Пользователи” при нажатии на кнопку добавления открывается окно добавления сотрудника. При заполнении обязательных полей, а это все кроме отчества и нажатии на кнопку добавить сотрудника добавляется сотрудник с одной ролью на выбор.

Интерфейс окна добавления сотрудника представлен на рисунке 56:

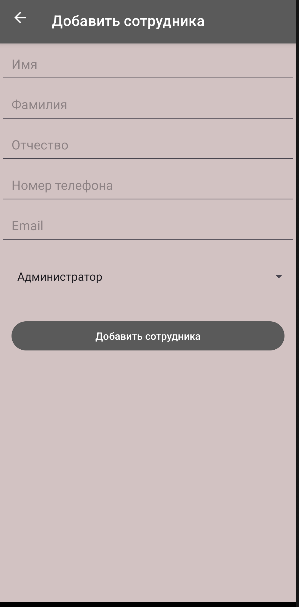


Рисунок 56 – Внешний вид окна добавления сотрудника

### 5.2.8 Редактирование сотрудника

Во вкладке “Пользователи” при нажатии на кнопку редактирования открывается окно редактирования сотрудника. Тут можно изменить параметры сотрудника, а также добавить или удалить его роли.

Интерфейс окна редактирования сотрудника представлен на рисунке 57:

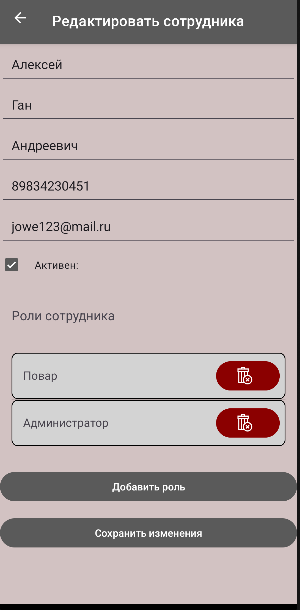


Рисунок 57 – Внешний вид окна редактирования сотрудника

### 5.2.9 Вкладка выручки

При открытии вкладки выручки открывается окно с двумя текстовыми полями для ввода даты начала и даты конца. Дата вводится с помощью календаря, который продемонстрирован на рисунке. После ввода даты и нажатия кнопки подсчитать выручку появляется список заказов за указанный период, а также считается их общая стоимость.

Внешний вид окна после подсчёта выручки представлен на рисунке 58

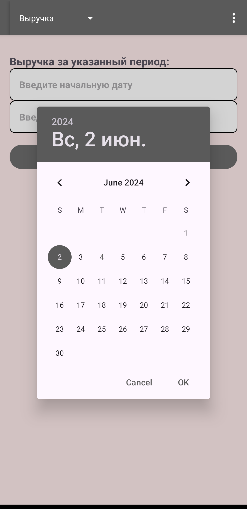


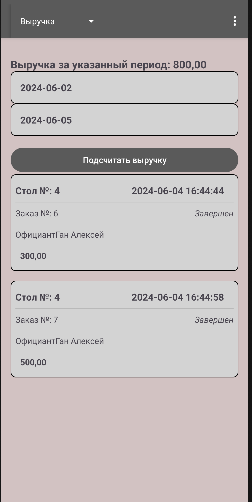
Рисунок 58 – Внешний вид календаря  


Рисунок 59 – Внешний вид окна выручки

### 5.2.10 Смена вкладок

Для смены вкладки необходимо нажать на список в верхней панели, после чего выбрать необходимую вкладку

Внешний вид нераскрытого списка представлен на рисунке 60



Рисунок 60 – Внешний вид нераскрытого списка

Внешний вид раскрытого списка представлен на рисунке 61

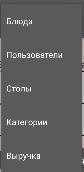


Рисунок 61 – Внешний вид раскрытого списка

### 5.2.11 Выход из аккаунта

При нажатии на три точки в правом верхнем углу открывается меню, где сначала указаны фамилия и имя пользователя, а затем кнопка “Выход”, нажав её, пользователь выйдет из аккаунта, и произойдет переход на экран авторизации.

Внешний вид меню выхода из аккаунта представлен на рисунке 62.

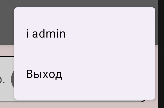


Рисунок 62 – Внешний вид меню выхода из аккаунта

## Интерфейс официанта

### 5.3.1 Начальный экран

Данное окно открывается при входе в аккаунт официанта.

По умолчанию открывается вкладка “Заказы” с открытой страницей ”Мои”, в котором отображены заказы, созданные данным официантом. На странице “Все” будут отображены все заказы.

Интерфейс начального экрана официанта представлен на рисунке 63:

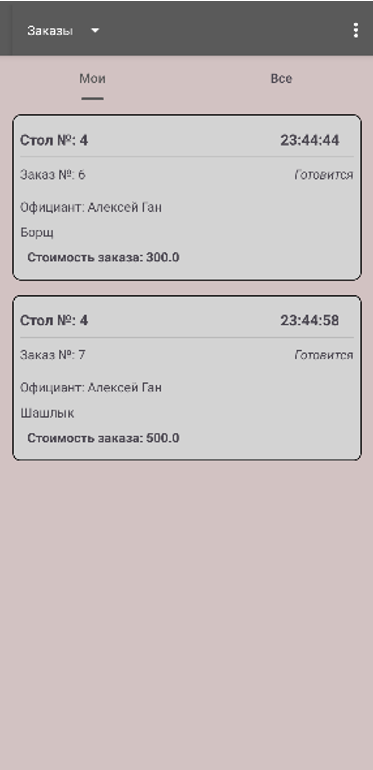


Рисунок 63 – Интерфейс окна редактирования сотрудника

### 5.3.2 Cтолы

Данное окно содержит все список столов в заведении и открывается при выборе вкладки столы. Также более светлым отображены столы со статусом “Занят”, а более темным столы со статусом “Свободен”. При нажатии на занятый стол появится диалоговое окно, сообщающее, что стол занят, но можно перейти к окну этого стола, а при нажатии на свободный стол, стол станет занятым и автоматически создастся пустой заказ.

Внешний вид вкладки “Столы” представлен на рисунке 64:

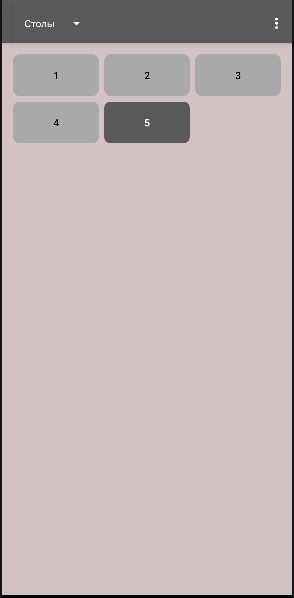


Рисунок 64 – Внешний вид вкладки столы

### 5.3.2 Выбранный стол

При выборе стола открывается окно, где отображены заказы данного стола, а также кнопки добавления нового заказа и закрытия стола. Если при нажатии на кнопку “Закрыть стол” есть какой-то неоплаченный заказ, то выдаст сообщение об ошибке. Чтобы изменить статус заказа на “Оплачен” нужно нажать по тексту статуса заказа.

Интерфейс окна выбранного стола представлен на рисунке 65:

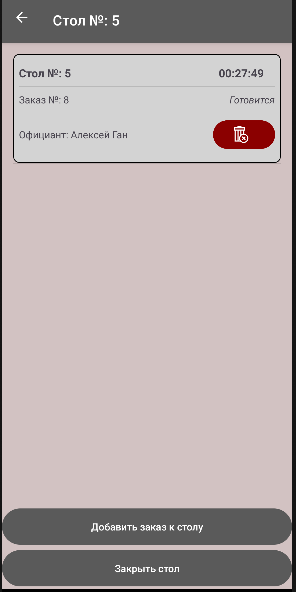


Рисунок 65 – Интерфейс окна выбранного стола

### 5.3.3 Детали заказа

При выборе заказа открывается окно с его деталями. Отображается количество данного блюда, название, стоимость, статус блюда, а также кнопку удаления блюда. Если статус блюда “Ожидает доставки”, то при нажатии на него статус изменится на “Доставлено”. Менять статус блюд могут повара и официанты. В нижней половине экрана находится интерфейс для добавления блюда в заказ. Для добавления необходимо выбрать категорию блюда и тогда экран сменится на

Внешний вид окна детали заказа представлен на рисунке 66:

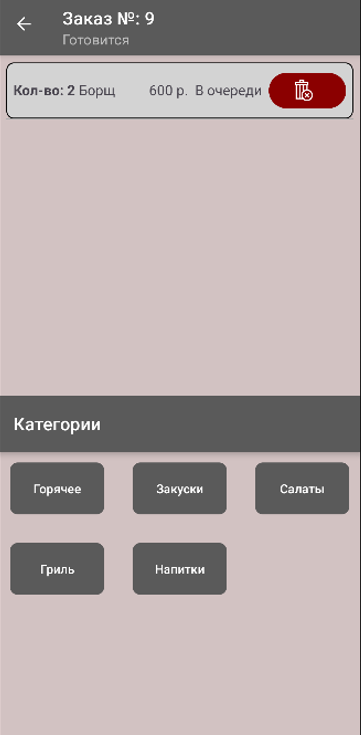


Рисунок 66 – Внешний вид окна деталей заказа.

В нижней половине экрана находится интерфейс для добавления блюда в заказ. Для добавления необходимо выбрать категорию блюда и тогда в списке будут отображаться блюда в данной категории, после чего выбрать уже само блюдо.

Интерфейс окна выбора блюд представлен на рисунке 67:

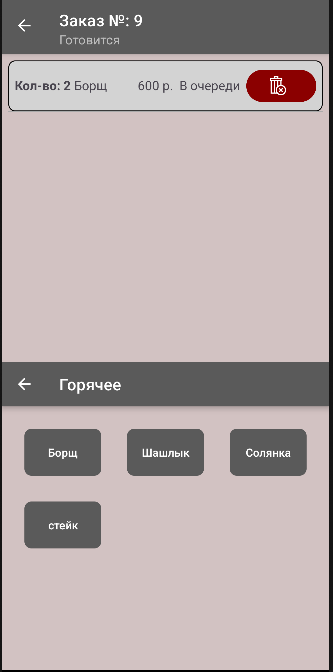


Рисунок 67 – Интерфейс окна выбора блюд

# Заключение

Результаты работы:

Была смоделирована предметная область, проанализированные аналогичный разработки по автоматизации заведений общественного питания, сформулированы бизнес-требования и функциональные требования к системе.

Были проанализированы, а в дальнейшим использованы средства разработки, среди которых: среда IntelliJ IDEA для серверной части и Android Studio для клиентской, язык программирования Java, СУБД PostgreSQL, фреймворки Spring и Hibernate.

Выполнено архитектурное проектирование системы.

Выполнено функциональное проектирование системы

Создано руководство пользователя, описывающее графический интерфейс клиентской части приложения.

Реализована как клиентская на платформе Android, так и серверная часть приложения для автоматизации взаимодействия персонала точек питания, в котором предусмотрены 4 роли: официант, администратор, повар и бармен. Для каждой роли реализован соответствующей функционал.

# Список источников

1. Руководство пользователя Iiko – URL: <https://ru.iiko.help/resources/Storage/archive/PDF/UG_iikoOffice_5.2.pdf> (дата обращения 23.04.2024)
2. Официальный сайт R-Keeper – URL: <https://rkeeper.ru/> (дата обращения 26.04.2024)
3. Официальный сайт Iiko – URL: <https://iiko.ru/> (дата обращения 29.04.2024)
4. Волушкова В.Л. Архитектурные решения java для доступа к данным: учеб. пособие. – Тверь: Твер. гос. ун-т, 2019. – 137 с. – URL: <http://lib.jizpi.uz/pluginfile.php/7130/mod_resource/content/0/Архитектурные_решения_java_для_доступа_к_данным_учеб_пособи.pdf> (дата обращения 1.05.2024)
5. Использование Retrofit 2.x в качестве REST клиента – Tutorial. // Habr – URL: <https://habr.com/ru/post/428736/>(дата обращения 3.05.2024)
6. Блинов, И. Н., Романчик, В. С. Java from EPAM : учеб.-метод. пособие / И. Н. Блинов, В. С. Романчик. — 2-е издание Минск:, 2021. – 560 – URL: <https://training.epam.com/static/news/Books/Ihar_Blinou.Java.from.EPAM.v2.2021.pdf> (дата обращения 5.05.2024)
7. Лицензия на вождение болида, или почему приложения должны быть Single-Activity // Habr – URL: <https://habr.com/ru/company/redmadrobot/blog/426617/>(дата обращения 8.05.2024)
8. Наир В. Н20 Предметно-ориентированное проектирование в Enterprise Java с помощью Jakarta EE, Eclipse MicroProfile, Spring Boot и программной среды Axon Framework / пер. с англ. А. В. Снастина. – М.: ДМК Пресс, 2020. – 306 с.: ил. (дата обращения 10.05.2024)
9. Редмонд Э., Уилсон Д. Семь баз данных за семь недель. Введение в современные базы данных и идеологию NoSQL – URL: <https://library-it.com/wp-content/uploads/2020/11/redmond_je_uilson_d_sem_baz_dannyh.pdf> (дата обращения 12.05.2024)
10. Муравьев Е. Г., Рыбанов А.А. Автоматизированные системы управления бизнес-функциями в организациях общественного питания и условия их оптимизации – URL: <https://cyberleninka.ru/article/n/avtomatizirovannye-sistemy-upravleniya-biznes-funktsiyami-v-organizatsiyah-obschestvennogo-pitaniya-i-usloviya-ih-optimizatsii> (дата обращения 17.05.2024)
11. Хабаров, С. П. Клиент-серверная экспертная система на основе технологии WebSocket / С. П. Хабаров, К. С. Голубев // Информационные системы и технологии: теория и практика : Сборник научных трудов научно-технической конференции института леса и природопользования, Санкт-Перербург, 01 февраля 2017 года. Том Выпуск 9. – Санкт-Перербург: Санкт-Петербургский государственный лесотехнический университет им. С.М. Кирова, 2017. – С. 115-119. – EDN VYETTV. – URL: <https://www.elibrary.ru/item.asp?id=32569998> (дата обращения 21.05.2024)
12. Матюшин, Д. С. Генерация токена авторизации посредством библиотеки JWT / Д. С. Матюшин, А. В. Богорадникова // Актуальные вопросы в науке и практике : сборник статей по материалам XII международной научно-практической конференции, Самара, 12 ноября 2018 года. Том Часть 1(4). – Самара: Общество с ограниченной ответственностью Дендра, 2018. – С. 91-94. – EDN POPGFT. – URL: <https://www.elibrary.ru/item.asp?id=36977230> (дата обращения 24.05.2024)
13. Что такое Spring Framework? От внедрения зависимостей до Web MVC. // Habr– URL: <https://habr.com/ru/articles/490586/> (дата обращения 24.05.2024)
14. Грузин, Н. А. Сравнение шаблонов проектирования архитектуры приложения: MVC, MVP и MVVM / Н. А. Грузин // Modern Science. – 2021. – № 1-1. – С. 434-440. – EDN VWUTGG. – URL: <https://elibrary.ru/item.asp?id=44555466> (дата обращения 24.05.2024)
15. Ананченко, И. В. Подходы к разработке веб-приложений с использованием API: сравнительный анализ restful и graphql / И. В. Ананченко, И. А. Проснеков // Научные исследования 2024 : сборник статей X Международной научно-практической конференции, Пенза, 23 февраля 2024 года. – Пенза: Наука и Просвещение (ИП Гуляев Г.Ю.), 2024. – С. 17-20. – EDN ESYHOY. – URL: <https://elibrary.ru/item.asp?id=61781182> (дата обращения 25.05.2024)