МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Южно-Уральский государственный университет
(национальный исследовательский университет)»**
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н.,
профессор

_____ Л.Б. Соколинский

«___»_____ 2024 г.

Разработка серверной части приложения для столовой «Восточный дракон»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 02.03.02.2024.308-304.ВКР

Научный руководитель,
ст. преподаватель кафедры СП
_____ Н.С. Силкина

Автор работы,
студент группы КЭ-401
_____ И.С. Лазарев

Ученый секретарь
(нормоконтролер)
_____ И.Д. Володченко
«___»_____ 2024 г.

Челябинск, 2024 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

УТВЕРЖДАЮ
Зав. кафедрой СП
_____ Л.Б. Соколинский
29.01.2024 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра
студенту группы КЭ-401
Лазареву Ивану Сергеевичу,
обучающемуся по направлению
02.03.02 «Фундаментальная информатика и информационные технологии»

1. **Тема работы** (утверждена приказом ректора от 22.04.2024 г. № 764-13/12)

Разработка серверной части приложения для столовой «Восточный дракон».

2. **Срок сдачи студентом законченной работы:** 03.06.2024 г.

3. **Исходные данные к работе**

3.1. Документация PHP. [Электронный ресурс] URL: https://www.php.net (дата обращения: 20.01.2024 г.).

3.2. Документация 1C-Bitrix. [Электронный ресурс] URL: https://dev.1c-bitrix.ru/docs/ (дата обращения: 30.01.2024 г.).

3.3. Документация API Yookassa. [Электронный ресурс] URL: https://yookassa.ru/developers (дата обращения: 10.03.2024 г.).

3.4. Firebase REST API Documentation. [Электронный ресурс] URL: https://firebase.google.com/docs/reference/fcm/rest/v1/projects.messages (дата обращения: 11.04.2024 г.).

4. **Перечень подлежащих разработке вопросов**

4.1. Провести анализ предметной области и разработать необходимые требования к системе.

4.2. Спроектировать базу данных.

4.3. Спроектировать API для приложения.

4.4. Реализовать и протестировать спроектированное API.

5. Дата выдачи задания: 29.01.2024 г.

Научный руководитель,
ст. преподаватель кафедры СП Н.С. Силкина

Задание принял к исполнению И.С. Лазарев

ГЛОССАРИЙ

1. *CMS (Content Management System)* – это система создания и управления сайтом. Это визуально удобный интерфейс, с помощью которого можно добавлять и редактировать содержимое сайта [1].

2. *API (Application Programming Interface)* – это набор способов и правил, по которым различные программы общаются между собой и обмениваются данными [2].

3. *SSL (Secure Sockets Layer)* – это протокол безопасности, создающий зашифрованное соединение между веб-сервером и веб-браузером [3].

4. *CRUD (Create, Read, Update и Delete)* – представляет собой четыре основные операции, выполняемые при управлении данными и манипулировании ими в рамках разработки программного обеспечения [4].

5. *HTTP (Hypertext Transfer Protocol)* – это протокол, позволяющий получать различные ресурсы, например HTML-документы. Лежит в основе обмена данными в интернете [5].

6. *JSON (Javascript Object Notation)* – текстовый формат обмена данными, основанный на JavaScript и легко читающийся людьми [6].

7. *ORM (Object-relational mapping)* – технология программирования, которая связывает базы данных с концепциями объектно-ориентированных языков программирования, создавая «виртуальную объектную базу данных» [7].

8. *Коллекция* – это умный массив, позволяющий оптимизировать групповые операции с объектами одного типа [8].

9. *Идемпотентность* – свойство объекта или операции при повторном применении операции к объекту давать тот же результат, что и при первом [9].

10. *Вебхук (Webhook)* – метод уведомлений, который интегрируется на сайт или в сервис для получения мгновенной информации о событиях [10].

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность

С развитием информационных технологий и цифровизации бизнес-процессов, значительная часть предприятий общественного питания стремится автоматизировать свою деятельность с помощью различных информационных систем. В рамках данной выпускной квалификационной выпускной работы исследуется процесс разработки серверной части приложения для китайской столовой «Восточный дракон». Акцент делается на анализе потребностей бизнеса, выборе решений, технологий и разработке функционала, способного оптимизировать процесс оформления заказов и управления ими. Результаты работы направлены на повышение эффективности работы ресторана и улучшение услуг для клиентов.

Постановка задачи

Целью выпускной квалификационной работы является разработка серверной части приложения для столовой «Восточный дракон». Для достижения поставленной цели необходимо решить следующие задачи:

1) провести обзор аналогичных систем;
2) разработать структуру таблиц базы данных;
3) разработать API;
4) реализовать API;
5) в рамках реализации API реализовать интеграцию с сервисом онлайн-оплаты;
6) в рамках реализации API реализовать интеграцию с сервисом отправки push-уведомлений;
7) протестировать API.

Структура и содержание работы

Работа состоит из введения, четырех глав, заключения и списка литературы. Объем работы составляет 47 страниц, объем списка литературы – 22 источника.

В первой главе «Анализ предметной области» содержится разбор предметной области и обзор аналогичных проектов.

Вторая глава «Проектирование» посвящена описанию вариантов использования, компонентов системы, проектированию базы данных и API.

В третьей главе «Реализация» содержатся описание инструментов для реализации проекта, реализация API, реализация интеграции с сервисом онлайн-оплаты и реализация интеграции с сервисом отправки push-уведомлений.

Четвертая глава «Тестирование» посвящена тестированию разработанного API.

В заключении подводятся итоги полученных в процессе работы результатов, делаются выводы о работе приложения, описываются планы для будущих улучшений серверной части приложения.

После заключения следует список использованной литературы для реализации проекта.

В приложениях содержатся диаграмма вариантов использования, диаграмма компонентов, схема базы данных, листинги с реализацией классов хранилища для работы с данными товаров и взаимодействия с сервисом push-уведомлений.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Описание предметной области

Предметная область данного проекта охватывает разработку серверной части приложения для столовой, которая предлагает пользователям возможность делать предзаказы еды. Это означает, что клиенты смогут заранее выбирать блюда, оплачивать свой заказ онлайн и отслеживать статус заказа.

Предметная область проекта включает в себя следующие аспекты:

1) управление меню столовой, включая добавление, изменение и удаление блюд, а также учет их стоимости и наличия;

2) прием предзаказов от клиентов, включая обработку заказов, расчет стоимости и оформление оплаты;

3) разработка API для взаимодействия с приложением из внешних систем.

Таким образом, в результате выполнения проекта будет разработана и внедрена серверная часть приложения для столовой по предзаказам еды, которая позволит автоматизировать основные процессы столовой, обеспечит пользователям удобный интерфейс для управления и контроля, а также предоставит API для взаимодействия с приложением из внешних систем и возможностью дальнейшего масштабирования.

1.2. Анализ аналогичных проектов

Для анализа были выбраны местные компании из Челябинска – Gordan и Kamakura предлагающие сервис по заказу еды в заведении и доставке по городу Челябинск. Рассмотрев визуальную составляющую данных приложений, можно будет более четко сделать выводы о функциональности будущего приложения.

Главная страница сервиса Gordan представлена в виде каталога с категориями товаров, текущими акциями и новостями. Каждый товар сразу можно добавить в корзину. Данная страница приложения представлена на

рисунке 1. По клику на товар открывается карточка с детальным описанием товара и составом.

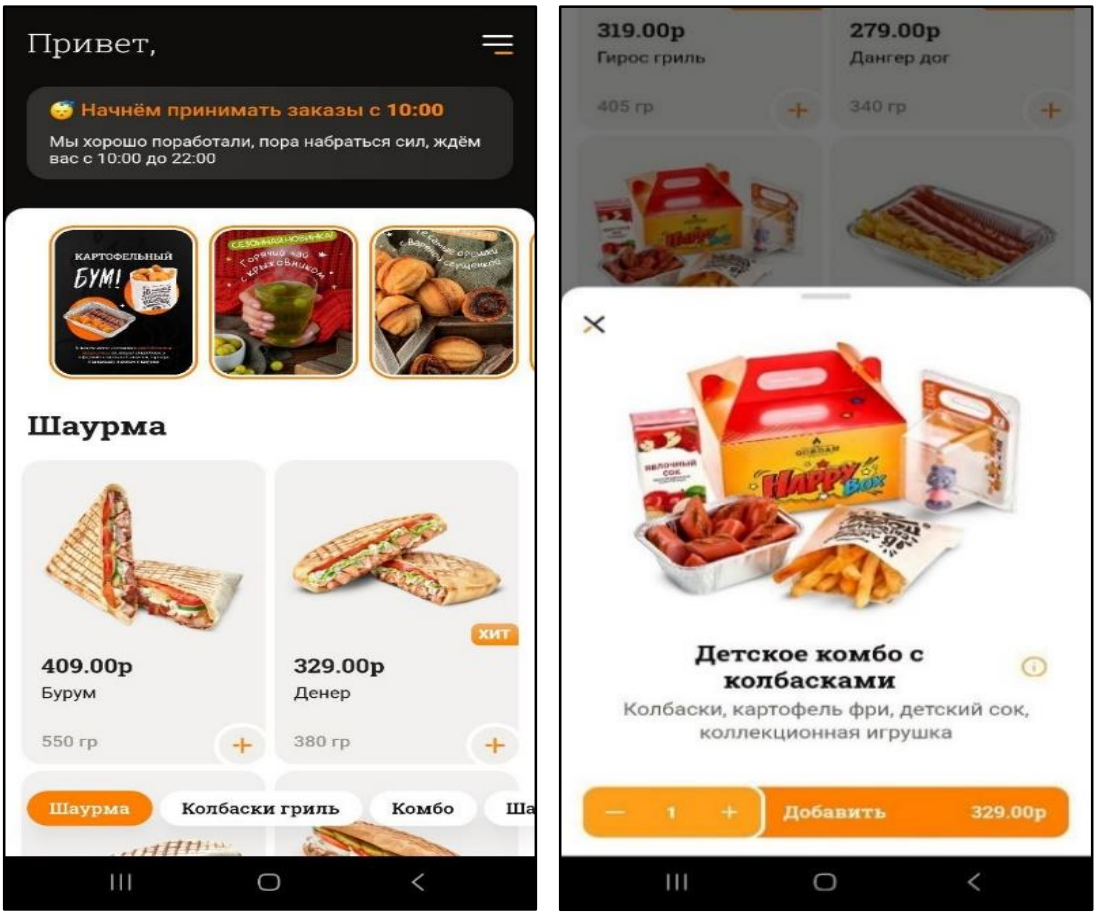

Рисунок 1 – Скриншоты приложения Gordan

Главная страница сервиса Kamakura (рисунок 2) тоже представляет из себя каталог с категориями и карточками товара, где можно сразу добавить товар в корзину, содержит вывод новостей и акций, но включает в себя еще и поисковую строку, и меню внизу экрана.

По клику на товар в каталоге приложения Kamakura как и у предыдущего приложения открывается детальная карточка товара с изображением, весом товара, составом товара. Внизу экрана присутствует кнопка для добавления товара в корзину. На карточке присутствуют кнопки добавления товара в избранное и возможность поделиться в социальных сетях (рисунок 2).

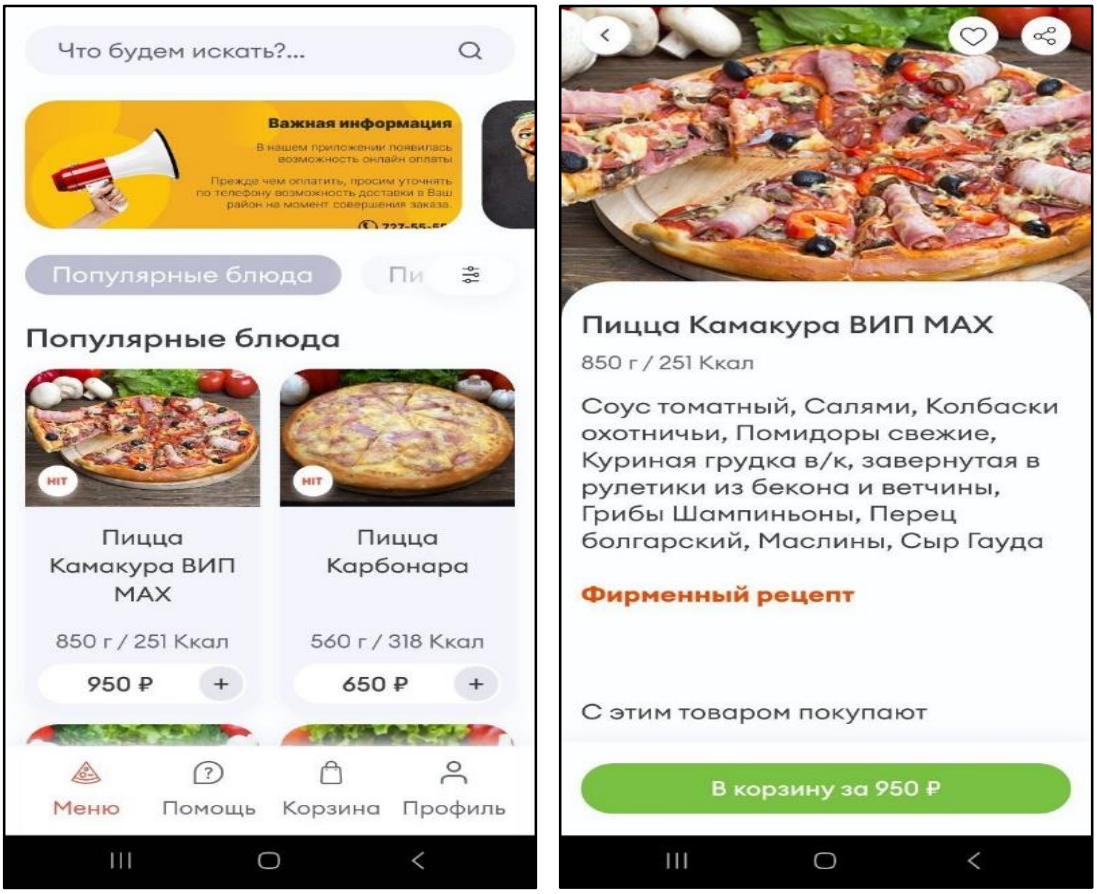

Рисунок 2 – Скриншоты приложения Kamakura

Сравнение функционала выбранных приложений поможет сделать вывод о функциональности разрабатываемого приложения. Сравнение функциональности приложений представлено в таблице 1.

Таблица 1 – Сравнение функциональности приложений

№	Функционал	Gordan	Kamakura	Разрабаты-ваемое приложение
1.	Избранное	–	+	+
2.	Оформление заказа	+	+	+
3.	Вывод каталога	+	+	+
4.	Вывод детальной карточки товара	+	+	+
5.	Редактирование комбо товара	+	–	+
6.	Вывод новостей и акций	+	+	+
7.	Авторизация	+	+	+
8.	Редактирование личного кабинета	+	+	+
9.	Вывод контактной информации	+	+	+
10.	Просмотр заказа	+	+	+
11.	Просмотр истории заказов	+	+	+

Окончание таблицы 1

№	Функционал	Gordan	Kamakura	Разрабатываемое приложение
12.	Поиск	–	+	+
13.	Управление корзиной товаров	+	+	+
14.	Уведомления	+	+	+
15.	Онлайн-оплата	+	+	+

Оба сервиса имеют довольно схожую функциональность и позволяют пользователям удобно и быстро осуществлять заказ еды, редактировать свои предпочтения, следить за новостями и акциями, редактировать личный профиль, просматривать историю заказов, следить за статусом заказа.

При этом отмечается, что мобильное приложение Kamakura предоставляет дополнительные возможности, такие как поиск позиций, добавление товаров в избранное. Эти функции могут привлечь дополнительных пользователей и способствовать улучшению пользовательского опыта.

Анализ показал, что оба приложения обеспечивают основные требования для заказа еды, но дополнительные функции Kamakura могут повысить конкурентоспособность и привлечь больше клиентов, благодаря немного большему функционалу.

1.3. Парадигмы проектирования API

REST API

REST [14] (Representational State Transfer) – это архитектурный стиль для проектирования сетевых приложений. Он основывается на принципе использования различных методов HTTP-протокола для совершения операций над ресурсами. Основными преимуществами REST API являются его простота, масштабируемость и гибкость, что делает его широко применимым в различных сценариях разработки.

При создании REST API следуют следующим основным принципам.

1. Необходимо идентифицировать основные ресурсы приложения. Например, для управления заказами, ресурсы могут включать задачи, список заказов, корзину товаров, пользователей и т.д.

2. Для каждого ресурса необходимо определить точки доступа (URL-адреса) для выполнения различных операций. Например, «/order/list» для получения всех заказов, «/order/{id}» для получения данных о конкретном заказе и т. д.

3. Необходимо сопоставить операции с методами HTTP для каждой точки доступа. Например, использовать GET для получения данных, POST для создания новых записей, PUT для обновления существующих данных и DELETE для удаления.

4. Учитывать возможные ошибки и исключения при работе клиента с API. Возвращать соответствующий HTTP-статус код и информативное сообщение об ошибке.

RPC API

RPC [15] (Remote Procedure Call) – это парадигма, в которой клиент вызывает удаленные процедуры или функции на сервере, в то время как для клиента они выглядят как локальные вызовы функций. Функция, которая вызывается сервером, может определяться в теле запроса или быть частью URL. Параметры передаются в теле запроса (рисунок 3). В сравнении с REST, RPC более ограничен в терминологии и типах запросов.

```
{
    "function": 'Authorization',
    "params": {
       "login": "foo",
       "password": "bar"
    }
}
```

Рисунок 3 – Пример тела RPC запроса

GraphQL API

GraphQL [16] – это язык запросов и среда выполнения, разработанные внутри Facebook в 2012 году и открытые для публичного использования с

2015 года. GraphQL предоставляет клиентам возможность запрашивать только те данные, которые им необходимы. Имеет систему типизации, стандартизированный формат запросов, прозрачную семантику, систему статической проверки и анализа типов. Пример запроса и ответа представлен на рисунке 4.

Запрос

```
{
    food {
        name
        descr
        price
    }
}
```

Ответ

```
{
    "food": {
        "name": "Rice",
        "descr": "tasty",
        "price":  450
      }
}
```

Рисунок 4 – Пример тела и ответа GraphQL запроса

Выбор парадигмы проектирования

Одним из главных преимуществ REST API является его гибкость. REST позволяет создавать различные интерфейсы, предоставляя доступ к отдельным ресурсам посредством уникальных URI (Uniform Resource Identifier). Это позволяет разработчикам легко масштабировать и изменять функционал API без необходимости изменения клиентского кода. Гибкость REST API делает его привлекательным выбором для широкого спектра приложений.

Еще одним преимуществом REST API является его простота и понятность. REST использует стандартные методы и структуры запросов, что делает его легким для понимания и использования как для разработчиков, так и для клиентов API. Этот аспект делает REST популярным выбором при разработке веб-сервисов и взаимодействии между сервером и клиентом.

Выбор REST API в качестве парадигмы проектирования API оправдан по ряду причин. Его гибкость, простота и универсальность делают этот стиль API привлекательным для различных проектов, особенно в контексте веб-разработки и создания мобильных приложений.

2. ПРОЕКТИРОВАНИЕ

2.1. Варианты использования

В соответствии с требованиями к проектируемому сервису были разработаны варианты использования. Диаграмма вариантов использования представлена на рисунке 1 в приложении В.

В данной системе есть два актера – авторизованный клиент и неавторизованный клиент, использующие систему.

Неавторизованному клиенту доступна только функция авторизация – возможность войти в систему и получить доступ ко всем функциям.

Авторизованному клиенту доступны следующие функции.

1. Управление каталогом – получение информации о текущем каталоге товаров и конкретных товарах.
2. Управление пользовательским аккаунтом – получение информации о пользователе и взаимодействие с данными пользователя.
3. Управление корзиной – получение информации о корзине товаров пользователя и взаимодействие с данными корзины товаров.
4. Управление заказами – получение информации о заказе/заказах пользователя и взаимодействие с данными заказов.
5. Управление избранными товарами – получение информации об избранных товарах и взаимодействие с данными избранных товаров.
6. Управление платежами – возможность осуществлять оплату заказов и получать информацию о платежах.
7. Поиск – возможность поиска товаров.
8. Управление сервисом push-уведомлений – возможность устанавливать/удалять токен устройства для отправки уведомлений, получать информацию об уже отправленных уведомлениях.
9. Управление контентом – возможность получить информацию о новостях.

2.2. Компоненты

При проектировании системы были определены компоненты системы, которые представлены в диаграмме компонентов на рисунке 2 в приложении Г.

Клиентская часть может взаимодействовать посредством HTTP запросов с веб-сервером, который предлагает программный интерфейс, т.е. способы и правила взаимодействия с данными. Веб-сервер обращается к базе данных для получения или отправки данных.

База данных является хранилищем для данных, необходимых для работы приложения. Работа с базой данных происходит за счет CRUD операций, которые выполняются для получения или изменения данных.

При получении запроса, веб-сервер обрабатывает его и, если необходимо, обращается к базе данных для проведения необходимых операций. После чего он формирует ответ в формате JSON и возвращает его клиенту.

Контроллер API предназначен для обработки запросов протокола HTTP. В диаграмме представлены следующие контроллеры.

1. AuthController – компонент, отвечающий за авторизацию пользователей.
2. CatalogController – управляет информацией о доступных блюдах и продуктах.
3. PaymentController – обрабатывает платежные транзакции и взаимодействует с платежной системой.
4. UserController – управляет информацией о пользователях.
5. CartController – обрабатывает операции, связанные с корзиной покупок.
6. PushController – отправляет уведомления пользователю через push-сервис.
7. OrderController – управляет информацией о заказах и операциях с ними.

8. NewsController – обеспечивает доступ к информации о новостях и акциях.

9. SearchController – отвечает за поиск товаров в базе данных.

10. FavoriteController – управляет операциями, связанными с избранными товарами.

2.3. Проектирование базы данных

В процессе проектирования была разработана база данных. Схема базы данных представлена на рисунке 3 в приложении Д.

Таблица `hlb_goods` (таблица 2) содержит информацию о товарах. Данная таблица связана с таблицами `hlb_good_categories` (таблица 3) как «много-к-одному», `hlb_cart` (таблица 10) как «один-ко-многим», `hlb_favorite` (таблица 9) как «один-ко-многим».

Таблица 2 – Описание таблицы `hlb_goods`

Наименование поля	Тип поля	Семантика
ID	INTEGER (11)	Первичный ключ
UF_NAME	TEXT	Название продукта
UF_DESCR	TEXT	Описание товара
UF_PRICE	DOUBLE	Стоимость товара
UF_CATEGORY	INTEGER(11)	Внешний ключ к таблице `hlb_good_categories`
UF_ACTIVE	INTEGER(11)	Активность товара
UF_IMAGE	INTEGER(11)	ID изображения товара, хранящегося в таблицах 1C-Bitrix
UF_WEIGHT	DOUBLE	Вес товара

Таблица `hlb_good_categories` (таблица 3) определяет наименования категорий, которые присваиваются товарам. Данная таблица связана с таблицей `hlb_goods` (таблица 2) как «один-ко-многим».

Таблица 3 – Описание таблицы `hlb_good_categories`

Наименование поля	Тип поля	Семантика
ID	INTEGER (11)	Первичный ключ
UF_NAME	TEXT	Название категории продуктов
UF_ACTIVE	INTEGER(11)	Активность категории

Таблица hlb_dinners (таблица 4) содержит информацию об обедах. Данная таблица связана с таблицами hlb_dinner_compound (таблица 5) как «много-к-одному», hlb_cart (таблица 10) как «один-ко-многим», hlb_favorite (таблица 9) как «один-ко-многим».

Таблица 4 – Описание таблицы hlb_dinners

Наименование поля	Тип поля	Семантика
ID	INTEGER (11)	Первичный ключ
UF_NAME	TEXT	Название обеда
UF_DESCR	TEXT	Описание обеда
UF_PRICE	DOUBLE	Стоимость обеда
UF_ACTIVE	INTEGER(11)	Активность обеда
UF_IMAGE	INTEGER(11)	ID изображения обеда, хранящегося в таблицах 1C-Bitrix

Таблица hlb_dinner_compound (таблица 5) определяет состав обеда. Данная таблица связана с таблицами hlb_dinners (таблица 4) как «один-к-многим», hlb_ingredients_category (таблица 7) как «один-ко-многим».

Таблица 5 – Описание таблицы hlb_dinner_compound

Наименование поля	Тип поля	Семантика
ID	INTEGER (11)	Первичный ключ
UF_NAME	TEXT	Название состава обеда
UF_FK_DINNER	INTEGER(11)	Внешний ключ к таблице hlb_dinners
UF_ACTIVE	INTEGER(11)	Активность состава обеда
UF_COMPOUND	TEXT	Состав обеда

Таблица hlb_ingredients (таблица 6) содержит информацию об ингредиентах, которые входят в состав обедов. Данная таблица связана с таблицами hlb_ingredients_category (таблица 7) как «много-к-одному», hlb_cart (таблица 10) как «много-к-одному».

Таблица 6 – Описание таблицы hlb_ingredients

Наименование поля	Тип поля	Семантика
ID	INTEGER (11)	Первичный ключ
UF_NAME	TEXT	Название ингредиента
UF_DESCR	TEXT	Описание ингредиента
UF_ACTIVE	INTEGER(11)	Активность ингредиента
UF_FK_CATEGORY	INTEGER (11)	Внешний ключ к таблице hlb_ingredients_category

Окончание таблицы 6

Наименование поля	Тип поля	Семантика
UF_EXTRA_PRICE	DOUBLE	Наценка на ингредиент
UF_IMAGE	INTEGER(11)	ID изображения ингредиента, хранящегося в таблицах 1C-Bitrix

Таблица `hlb_ingredients_category` (таблица 7) определяет наименования категорий, которые присваиваются ингредиентам. Данная таблица связана с таблицами `hlb_ingredients` (таблица 6) как «один-ко-многим», `hlb_dinner_compound` (таблица 5) «много-к-одному».

Таблица 7 – Описание таблицы `hlb_ingredients_category`

Наименование поля	Тип поля	Семантика
ID	INTEGER (11)	Первичный ключ
UF_NAME	TEXT	Название категории
UF_ACTIVE	INTEGER(11)	Активность категории

Таблица `b_user` (таблица 8) содержит информацию о зарегистрированных пользователях. Данная таблица связана с таблицами `hlb_favorite` (таблица 9) как «один-ко-многим», `hlb_cart` (таблица 10) как «один-ко-многим», `hlb_orders` (таблица 11) как «один-ко-многим», `hlb_pushes` (таблица 13) как «один-ко-многим».

Таблица 8 – Описание таблицы `b_user`

Наименование поля	Тип поля	Семантика
ID	INTEGER (11)	Первичный ключ
LOGIN	TEXT	Логин пользователя
PASSWORD	TEXT	Пароль пользователя
ACTIVE	INTEGER(11)	Активность пользователя
EMAIL	TEXT	Электронная почта пользователя
NAME	TEXT	Имя пользователя
UF_CODE	TEXT	Последний использованный код подтверждения
UF_DEVICE_TOKE	TEXT	Токен устройства

Таблица `hlb_favorite` (таблица 9) содержит информацию о товарах и обедах, которые пользователь добавил в избранное. Данная таблица связана с таблицами `b_user` (таблица 8) как «много-к-одному», `hlb_dinners`

(таблица 4) как «много-к-одному», hlb_goods (таблица 2) как «много-к-одному».

Таблица 9 – Описание таблицы hlb_favorite

Наименование поля	Тип поля	Семантика
ID	INTEGER (11)	Первичный ключ
UF_USER_ID	INTEGER(11)	Внешний ключ к таблице b_user
UF_FK_DINNER	INTEGER(11)	Внешний ключ к таблице hlb_dinners
UF_FK_PRODUCT	INTEGER(11)	Внешний ключ к таблице hlb_goods
UF_DATE_CREATED	DATETIME	Дата и время добавления

Таблица hlb_cart (таблица 10) содержит информацию о товарах, которые пользователь добавил в корзину. Данная таблица связана с таблицами b_user (таблица 8) как «много-к-одному», hlb_orders (таблица 11) как «много-к-одному», hlb_goods (таблица 2) как «много-к-одному», hlb_dinners (таблица 4) как «много-к-одному», hlb_ingredients (таблица 6) как «один-ко-многим».

Таблица 10 – Описание таблицы hlb_cart

Наименование поля	Тип поля	Семантика
ID	INTEGER (11)	Первичный ключ
UF_USER_ID	INTEGER(11)	Внешний ключ к таблице b_user
UF_ORDER_ID	INTEGER(11)	Внешний ключ к таблице hlb_orders
UF_PRODUCT_ID	INTEGER(11)	Внешний ключ к таблице hlb_goods
UF_COMBO_ID	INTEGER(11)	Внешний ключ к таблице hlb_dinners
UF_QUANTITY	INTEGER(11)	Количество товара
UF_DATE_CREATED	DATETIME	Дата и время добавления
UF_PRICE	DOUBLE	Стоимость товаров
UF_INGREDIENTS_ID	TEXT	Внешний ключ к таблице hlb_ingredients

Таблица hlb_orders (таблица 11) содержит информацию о заказе пользователя. Данная таблица связана с таблицами b_user (таблица 8) как «много-к-одному», hlb_payment (таблица 12) как «один-к-одному», hlb_cart (таблица 10) как «один-ко-многим».

Таблица 11 – Описание таблицы hlb_orders

Наименование поля	Тип поля	Семантика
ID	INTEGER (11)	Первичный ключ
UF_USER_ID	INTEGER(11)	Внешний ключ к таблице b_user

Окончание таблицы 11

Наименование поля	Тип поля	Семантика
UF_PRICE	DOUBLE	Стоимость заказа
UF_DATE_CREATED	DATETIME	Дата и время создания заказа
UF_PAYMENT_ID	INTEGER(11)	Внешний ключ к таблице hlb_payment
UF_STATUS	TEXT	Статус заказа
UF_COMMENT	TEXT	Комментарий к заказу
UF_TABLEWARE	INTEGER(11)	Количество приборов
UF_SEND_PUSH	INTEGER(11)	Отправить уведомления

Таблица hlb_payment (таблица 12) содержит информацию о платежах пользователей. Данная таблица связана с таблицей hlb_orders (таблица 11) как «один-к-одному».

Таблица 12 – Описание таблицы hlb_payment

Наименование поля	Тип поля	Семантика
ID	INTEGER (11)	Первичный ключ
UF_YOOKASSA_ID	TEXT	ID платежа в сервисе Yookassa
UF_STATUS	TEXT	Статус платежа
UF_DATE_CREATED	DATETIME	Дата платежа
UF_NAME	TEXT	Название платежа

Таблица hlb_pushes (таблица 13) содержит информацию об отправленных push-уведомлениях. Данная таблица связана с таблицей b_user (таблица 8) как «много-к-одному».

Таблица 13 – Описание таблицы hlb_pushes

Наименование поля	Тип поля	Семантика
ID	INTEGER (11)	Первичный ключ
UF_TITLE	TEXT	Заголовок отправленного уведомления
UF_TEXT	TEXT	Текст отправленного уведомления
UF_DATE_SEND	DATETIME	Дата отправки
UF_USER_ID	INTEGER (11)	Внешний ключ к таблице b_user

Таблица hlb_news (таблица 14) содержит информацию о новостях.

Таблица 14 – Описание таблицы hlb_news

Наименование поля	Тип поля	Семантика
ID	INTEGER (11)	Первичный ключ
UF_NAME	TEXT	Заголовок новости

Окончание таблицы 14

Наименование поля	Тип поля	Семантика
UF_DESCRIPTION	TEXT	Текст новости
UF_IMAGE	INTEGER(11)	ID изображения ингредиента, хранящегося в таблицах 1C-Bitrix
UF_ACTIVE	INTEGER(11)	Активность новости
UF_SEND_PUSH	INTEGER(11)	Отправить уведомления пользователям
UF_PUSH_TITLE	TEXT	Заголовок уведомления
UF_PUSH_TEXT	TEXT	Текст уведомления

2.4. Проектирование REST API

При проектировании API необходимо определить перечень и функциональность, которую будут предоставлять точки входа (эндпоинты), методы HTTP (GET, POST, PUT, DELETE) для выполнения различных операций и формат передачи данных.

Формат передачи данных – multipart/form-data, стандартный тип содержимого (Content-Type), который используется для передачи данных в HTTP запросах. Этот формат позволяет отправлять данные с использованием заголовков, указывающих границы между частями данных.

API должно позволять клиентской части приложения взаимодействовать со всеми необходимыми функциями, такими как создание заказа, получение информации о меню и акциях, управление учетной записью пользователя и другие операции. В таблице 15 представлен перечень запросов к API.

Таблица 15 – Перечень запросов к API

№	Семантика запроса	Пример запроса
Каталог		
1.	Получить каталог	GET: {API_URL}/api/catalog/
2.	Получить товар	GET: {API_URL}/api/catalog/:id
3.	Получить обед	GET: {API_URL}/api/catalog/dinner/:id
Авторизация		
4.	Аутентификация по почте	POST: {API_URL}/api/user/auth/
5.	Проверка кода	POST: {API_URL}/api/user/code/
Пользователь		
6.	Получить информацию о пользователе	GET: {API_URL}/api/user/

Окончание таблицы 15

№	Семантика запроса	Пример запроса
7.	Изменить данные пользователя	PUT: {API_URL}/api/user/
8.	Выйти из аккаунта	GET: {API_URL}/api/user/logout/
Корзина		
10.	Получить корзину пользователя	GET: {API_URL}/api/cart/
11.	Добавить товар	POST: {API_URL}/api/cart/:id
12.	Очистить корзину	POST: {API_URL}/api/cart/clear/
13.	Удалить товар	DELETE: {API_URL}/api/cart/:id
14.	Изменить товар	PUT: {API_URL}/api/cart/:id
Заказ		
15.	Отменить заказ	POST: {API_URL}/api/order/cancel/:id
16.	Создать заказ	POST: {API_URL}/api/order/create/
17.	Повторить заказ	POST: {API_URL}/api/order/repeat/
18.	Данные заказа	GET: {API_URL}/api/order/:id
19.	История заказов	GET: {API_URL}/api/order/history/
Избранное		
20.	Получить список избранных товаров	GET: {API_URL}/api/favorite/
21.	Добавить в избранное	POST: {API_URL}/api/favorite/:id
22.	Удалить из избранного	DELETE: {API_URL}/api/favorite/:id
Оплата		
23.	Создание платежа	POST: {API_URL}/api/payments/
24.	Список платежей	GET: {API_URL}/api/payments/
25.	Информация о платеже	GET: {API_URL}/api/payments/:id
Поиск		
26.	Поиск товаров по названию	GET: {API_URL}/api/search/?prompt=
Новость		
27.	Список новостей	GET: {API_URL}/api/news/
Push-уведомление		
28.	Установить токен устройства	POST: {API_URL}/api/push/token/
29.	Удалить токен устройства	DELETE: {API_URL}/api/push/token/
30.	Список уведомлений	GET: {API_URL}/api/push/list/

3. РЕАЛИЗАЦИЯ

3.1. Инструменты для реализации проекта

CMS

Для реализации данного проекта была выбрана отечественная разработка CMS 1С-Bitrix [11]. У данной системы есть следующие преимущества.

1. Локализованная поддержка. 1С-Битрикс обладает локализованной поддержкой и документацией на русском языке, что облегчает использование и обучение персонала в российских компаниях.

2. Соответствие требованиям. 1С-Битрикс ориентирован на соответствие российским законодательным и бухгалтерским требованиям, включая налоговую отчетность и стандарты безопасности данных. Это делает его привлекательным выбором для российских компаний, работающих в рамках местного законодательства.

3. Интеграция с 1С. Многие российские компании используют 1С в качестве системы управления предприятием, и 1С-Битрикс предоставляет удобные средства интеграции с этой платформой, обеспечивая единый информационный пространство для бизнеса. На данный момент нет необходимости в данной интеграции, но наличие такой возможности – это хорошее дополнение.

4. Техническая поддержка. 1С-Битрикс предоставляет качественную техническую поддержку на российском рынке, что может быть критически важно для бизнеса при обеспечении бесперебойной работы веб-проектов.

5. Широкое распространение и опыт использования. 1С-Битрикс имеет обширное присутствие и опыт использования в российском бизнес-сообществе, что способствует наличию специалистов и разработчиков, знакомых с платформой.

6. Наличие ORM. Использование ORM в 1С-Bitrix способствует улучшению производительности разработки, облегчает поддержку и расширение проекта, а также повышает уровень безопасности при работе с базой данных.

Хостинг

Хостинг – это важный компонент для любого веб-проекта, предоставляющий возможность размещения сайта или приложения в Интернете. Выбор надежного хостинг-провайдера с адекватными техническими характеристиками и поддержкой – ключевой аспект успешной работы онлайн-проекта.

Для размещения проекта был выбран хостинг от компании Reddock [12]. Так как в проекте используется CMS 1C-Bitrix, то нужно учитывать требования данной системы. Данный хостинг имеет различные подходящие тарифные планы для размещения выбранной нами CMS.

Для нашего приложения потребуется 1 сайт, 1 база данных, возможность быстрой установки бесплатного SSL сертификата Let’s Encrypt.

Помимо подходящих тарифных планов этот хостинг имеет ряд преимуществ:

1) удобная панель управления ISPmanager 6;
2) неограниченное количество поддоменов;
3) неограниченное количество почтовых ящиков;
4) ежедневное резервное копирование;
5) круглосуточная тех. поддержка.

Онлайн-оплата

В настоящее время важность возможности онлайн-оплаты в мобильных приложениях становится все более очевидной. Введение такой функциональности является неотъемлемой частью создания удобного и эффективного сервиса для пользователей. Разрешение онлайн-оплаты в приложении обеспечивает клиентам быстрый и безопасный способ совершения покупок или оплаты услуг, повышая удобство пользовательского опыта. Этот функционал открывает новые возможности для бизнеса, помогая расширить аудиторию, увеличить конверсию и улучшить взаимодействие с клиентами.

В результате, возможность онлайн-оплаты в приложении становится ключевым инструментом для улучшения пользовательского опыта, увеличения доходов и повышения конкурентоспособности компании.

В качестве системы онлайн-оплаты был выбран сервис Yookassa [10], который предлагает широкий спектр услуг и имеет ряд преимуществ.

1. Обеспечивает высокий уровень безопасности транзакций, защищая данные платежей, что особенно важно для обработки чувствительной финансовой информации в мобильных приложениях.
2. Интеграция с Yookassa относительно проста благодаря хорошей документации и обширной поддержке, что упрощает процесс разработки и поддержки платежной системы в мобильном приложении.
3. Предоставляет широкий спектр методов оплаты, включая банковские карты, электронные кошельки, мобильные платежи, а также другие популярные методы, удовлетворяя потребности широкой аудитории.
4. Предлагает оптимизированные для мобильных устройств интерфейсы, что позволяет повысить результативность и удобство процесса оплаты для пользователей мобильных приложений.
5. Предлагает дополнительные функции, такие как работа с подписками, разделение платежей, возврат средств, автоматическое списание и другие, обогащая опыт оплаты в мобильном приложении.
6. Имеет тестовый доступ для разработки, который не требует подписания договора.

3.2. Программная реализация

ORM

Реализация ORM в 1С-Bitrix в ядре D7 призвана абстрагировать разработчика от механики работы с таблицами на уровне запросов к базе данных, введя следующие понятия:

1) сущность – таблица, в терминах битрикса;
2) поля сущности – столбцы или ссылки на другие сущности;

3) датаменеджер – система управления данными.

Работа в ORM возможна только если есть класс, описывающий таблицу, с которой нужно работать. Класс может быть сгенерирован и находиться в ядре, а может и отсутствовать. Все зависит от таблицы данных, с которой нужно работать.

В классе обязательно должны быть два метода:

1) `getMap()` – перечисляются все поля таблицы;

2) `getTableName()` – указывается название базы данных.

Класс, описывающий таблицу с продуктами представлен в листинге 1.

Листинг 1 – Класс `ProductsTable`

```
final class ProductsTable extends DataManager
{
    public static function getTableName(): string
    {
        return 'hlb_goods';
    }
    public static function getMap(): array
    {
        return [
            (new IntegerField('ID'))->configurePrimary()->configureAutocom-
plete(),
            (new TextField('UF_NAME'))->configureRequired(),
            (new IntegerField('UF_ACTIVE')),
            (new IntegerField('UF_CATEGORY')),
            (new TextField('UF_DESCR')),
            (new IntegerField('UF_IMAGE')),
            (new FloatField('UF_PRICE')),
            (new FloatField('UF_WEIGHT')),
            (new Reference(
                'CATEGORY',
                ProductCategoriesTable::class,
                Join::on('this.UF_CATEGORY', 'ref.ID')
            ))->configureJoinType('inner'),
        ];
    }
}
```

Имея описанную сущность, можно начать использовать объекты сущности, имеющие множество методов по манипуляции с собственными данными и отношениями между сущностями. Класс, описывающий объект сущности, представлен в листинге 2.

Листинг 2 – Класс `Product`

```
class Product extends ModelAbstract
{
    static function getTableEntity(): string
```

```
    {
        return ProductsTable::class;
    }
    public function getName(): string
    {
        return $this->getData()->get('UF_NAME');
    }
    public function getPrice(): float
    {
        return $this->getData()->get('UF_PRICE');
    }
    public function getDescription(): string
    {
        return $this->getData()->get('UF_DESCR');
    }
    public function getWeight(): float
    {
        return $this->getData()->get('UF_WEIGHT');
    }
    public function getImageId(): int
    {
        return $this->getData()->get('UF_IMAGE');
    }
    public function getCategory(): string
    {
        return $this->getData()->get('UF_CATEGORY');
    }
    public function getImage(): array
    {
        if (!$this->getImageId()) {
            return [];
        }
        $image = \CFile::ResizeImageGet($this->getImageId(), ['width' =>
1920, 'height' => 1080], bInitSizes: true);
        $image['src'] = 'https://' . $_SERVER['SERVER_NAME'] . $im-
age['src'];
        return $image;
    }
    public function getImageLink(): string
    {
        $img = $this->getImage();
        if (!$img) return '';
        return $img['src'];
    }
    public function data(): array
    {
        return $this->getData()->collectValues();
    }
}
```

Имея описанный объект сущности и саму сущность, можно начать использовать коллекции объектов определенного класса, имеющие методы для групповых манипуляций с ними. Класс коллекции представлен в листинге 3.

Листинг 3 – Класс `ProductsCollection`

```
class ProductsCollection extends CollectionAbstract
{
```

```
    public function push($entity): void
    {
        if($entity instanceof Product)
        {
            $this->items[] = $entity;
        }
        else
        {
            throw new ArgumentException("Коллекция может содержать только
экземпляры " . Product::class);
        }
    }
    /**
     * @throws ArgumentException
     */
    public function bindOrmCollection(Collection $collection): void
    {
        foreach($collection as $item)
        {
            $this->push(new Product($item));
        }
    }
}
```

Паттерн «Репозиторий»

Паттерн «Репозиторий» [13] хорошо подходит при работе с коллекциями, взаимодействия с объектами и данными.

Основные принципы и применение этого паттерна.

1. Репозиторий изолирует работу с данными от остальной части приложения, таким образом, упрощая разделение ответственности и поддержку кода.

2. Репозиторий представляет собой слой между бизнес-логикой приложения и источником данных (например, базой данных). Он инкапсулирует логику доступа к данным и предоставляет унифицированный интерфейс для работы с данными. Репозиторий использует интерфейс похожий на коллекции, он может фильтровать данные и возвращать результат обратно в зависимости от требований к приложению. Где и как он хранит эти объекты является деталью реализации.

Преимущества паттерна «Репозиторий»:

1) упрощает тестирование: легко мокировать или заменять реализацию репозитория для тестов;

2) повышает модульность: разделяет бизнес-логику и доступ к данным;

3) облегчает сопровождение: позволяет изменять источник данных без изменения бизнес-логики.

В PHP репозиторий может быть реализован как класс, предоставляющий методы для работы с данными, например: `getList()`, `getById()`, `add()`, `delete()` и т.д.

Предоставляя интерфейс Repository (листинг 4), разработчик сам решает, как хранить и взаимодействовать с данными.

Листинг 4 – Интерфейс `Repository`

```
interface Repository
{
    public function getList(array $filter);

    public function getById(int $id);

    public function add($item);

    public function delete(int $id);
}
```

Класс хранилища для работы с данными товаров, реализующий интерфейс `Repository`, представлен в листинге 1 приложения А.

Обработка адресов

Для регистрации обработчика по определенному адресу используется метод с определенным разрешенным методом HTTP. В листинге 5 используется пример для регистрации обработчика по адресу `/api/catalog/`, в параметр `callback` передается функция, которая будет обрабатывать запрос по указанному эндпоинту, `needAuth` используется для указания доступа к операции неавторизованному пользователю, `exceptionMessage` возвращает в ответе в случае ошибки указанный текст.

Листинг 5 – Регистрация функции-обработчика для определенного адреса

```
$api->handlerGet("/api/catalog/", [
    'callback' => \Project\Api\Catalog\CatalogController::getCatalog(...),
    'needAuth' => true,
    'exceptionMessage' => $api->exceptionMessage,
]);
```

Для получения информации о каталоге товаров используется функция-обработчик `getCatalog`, являющаяся статическим методом контроллера `CatalogController`, которая запускается при получении запроса по определенному адресу. Функция-обработчик представлена в листинге 6.

Листинг 6 – Функция-обработчик для получения каталога

```
public static function getCatalog($urlParams = []): array
{
    $result = [];
    $categoriesCollection = \Project\Main\Repositories\ProductCategoriesRe-
pository::getList(['UF_ACTIVE' => 1]);
    foreach ($categoriesCollection as $category) {
        $items = [];
        $itemsCollection = \Project\Main\Repositories\ProductReposi-
tory::getList(['UF_ACTIVE' => 1, 'UF_CATEGORY' => $category->getId()]);
        foreach ($itemsCollection as $item) {
            $image = $item->getImage();
            $items[] = [
                'id' => $item->getId(),
                'name' => $item->getName(),
                'price' => $item->getPrice(),
                'compound' => $item->getDescription(),
                'weights' => $item->getWeight(),
                'image' => $image['src'] ?? '',
                'isLunch' => false,
            ];
        }

        if (!$items) {
            $dinnersCollection = \Project\Main\Repositories\DinnerReposi-
tory::getList(['UF_ACTIVE' => 1, 'UF_FK_CATEGORY' => $category->getId()]);
            foreach ($dinnersCollection as $item) {
                $image = $item->getImage();
                $items[] = [
                    'id' => $item->getId(),
                    'name' => $item->getName(),
                    'price' => $item->getPrice(),
                    'compound' => $item->getDescription(),
                    'weights' => $item->getWeight(),
                    'image' => $image['src'] ?? '',
                    'isLunch' => true,
                ];
            }
        }

        if (!$items) continue;
        $result[] = [
            'id' => $category->getId(),
            'name' => $category->getName(),
            'items' => $items,
        ];
    }
    return $result;
}
```

Интеграция с Yookassa API

Сервис онлайн-оплаты Yookassa предлагает REST API для работы с онлайн-платежами. С помощью данного API можно отправлять запросы на оплату, получит информацию о конкретном платеже, смотреть историю платежей и многое другое.

API в качестве основного протокола использует HTTP. Поддерживает POST, GET и DELETE-запросы. Запросы и ответы используют формат JSON, независимо от типа запроса. Класс `PaymentService` для взаимодействия с Yookassa API представлен в листинге 7.

Листинг 7 – Класс `PaymentService`

```
class PaymentService
{
    const SHOP_ID = '337939';
    const SECRET_KEY = 'test__bIHzHeboUXOXBUIThRIFjO8AcAt2GfBw6w2R72ZN5A';
    const CLIENT_APP_KEY = 'test_MzM3OTM5x_TnCdIQMsizUkm450wrFwH4TfRbfpM-
Abs';
    const CLIENT_ID = '9bb2vtu0ch2arr7b3bhfhicmhq7505rg';
    private \YooKassa\Client $client;

    private string $idempotencePrefix = 'dragon';
    public function __construct()
    {
        $this->client = new \YooKassa\Client();
        $this->client->setAuth(self::SHOP_ID, self::SECRET_KEY);
    }

    public function getClient(): \YooKassa\Client
    {
        return $this->client;
    }

    public function createPayment(array $paymentData): CreatePaymentRe-
sponse
    {
        $idempotenceKey = uniqid($this->idempotencePrefix, true);
        return $this->getClient()->createPayment($paymentData, $idempo-
tenceKey);
    }

    public function getPayments(): null | PaymentsResponse
    {
        return $this->getClient()->getPayments();
    }

    public function getPaymentInfo(string $paymentId): null | PaymentInter-
face
    {
        return $this->getClient()->getPaymentInfo($paymentId);
    }

}
```

Для аутентификации запросов необходимо использовать HTTP Basic Auth. В заголовках запросов в качестве имени пользователя необходимо передать идентификатор магазина (константа `SHOP_ID`), в качестве пароля – секретный ключ (константа `SECRET_KEY`). Константы `CLIENT_APP_KEY` и `CLIENT_ID` нужны для создания токена на клиентской части, необходимого для создания платежа. Метод `createPayment` – создает платеж в сервисе. В данном методе для обеспечения корректного поведения и избегания нежелательного повторения транзакций используется ключ идемпотентности `$idempotenceKey`. Метод `getPayments` – запрашивает информацию о платежах. Метод `getPaymentInfo` – запрашивает информацию о конкретно платеже.

Yookassa API предоставляет возможность использования вебхуков для отслеживания статуса оплаты без задержек и опросов, так как интерфейс взаимодействия с запросом строится на стороне клиента. Обработка вебхуков представлена в листинге 8.

Листинг 8 – Обработка вебхуков

```
require_once($_SERVER['DOCUMENT_ROOT'] .
"/bitrix/modules/main/include/prolog_before.php");
// приходит json объект от yooKassa
$input = file_get_contents( 'php://input' );
// объект перекодируется в массив
$requestBody = json_decode( $input, true );
if (empty($requestBody)) {
    exit;
}
if ($requestBody['type'] !== 'notification') exit;
$paymentObj = $requestBody['object'];
if (empty($paymentObj)) {
    exit;
}
$status = $paymentObj['status'];
$metadata = $paymentObj['metadata'];
if (empty($metadata['order_id'])) exit;
$payment = PaymentRepository::getItemByFilter([
    'UF_YOOKASSA_ID' => $paymentObj['id'],
]);
$fields = [
    'UF_YOOKASSA_ID' => $paymentObj['id'],
    'UF_NAME' => $paymentObj['description'],
    'UF_STATUS' => $status
];
if (!$payment) {
    $addedPaymentId = PaymentRepository::add($fields);
} else {
    $updatedPaymentId = PaymentRepository::update($payment->getId(),
```

```
$fields);
}
if (!empty($addedPaymentId)) {
    $updatedOrderId = OrderRepository::update($metadata['order_id'], [
       'UF_PAYMENT_ID' => $addedPaymentId
    ]);
}
header("HTTP/1.0 200 OK");
exit;
```

В коде выше принимаются и обрабатываются данные о платеже, затем создается запись о новом платеже и обновляется статус платежа в заказе. Общий процесс оплаты в приложении представлен в диаграмме последовательности на рисунке 5.

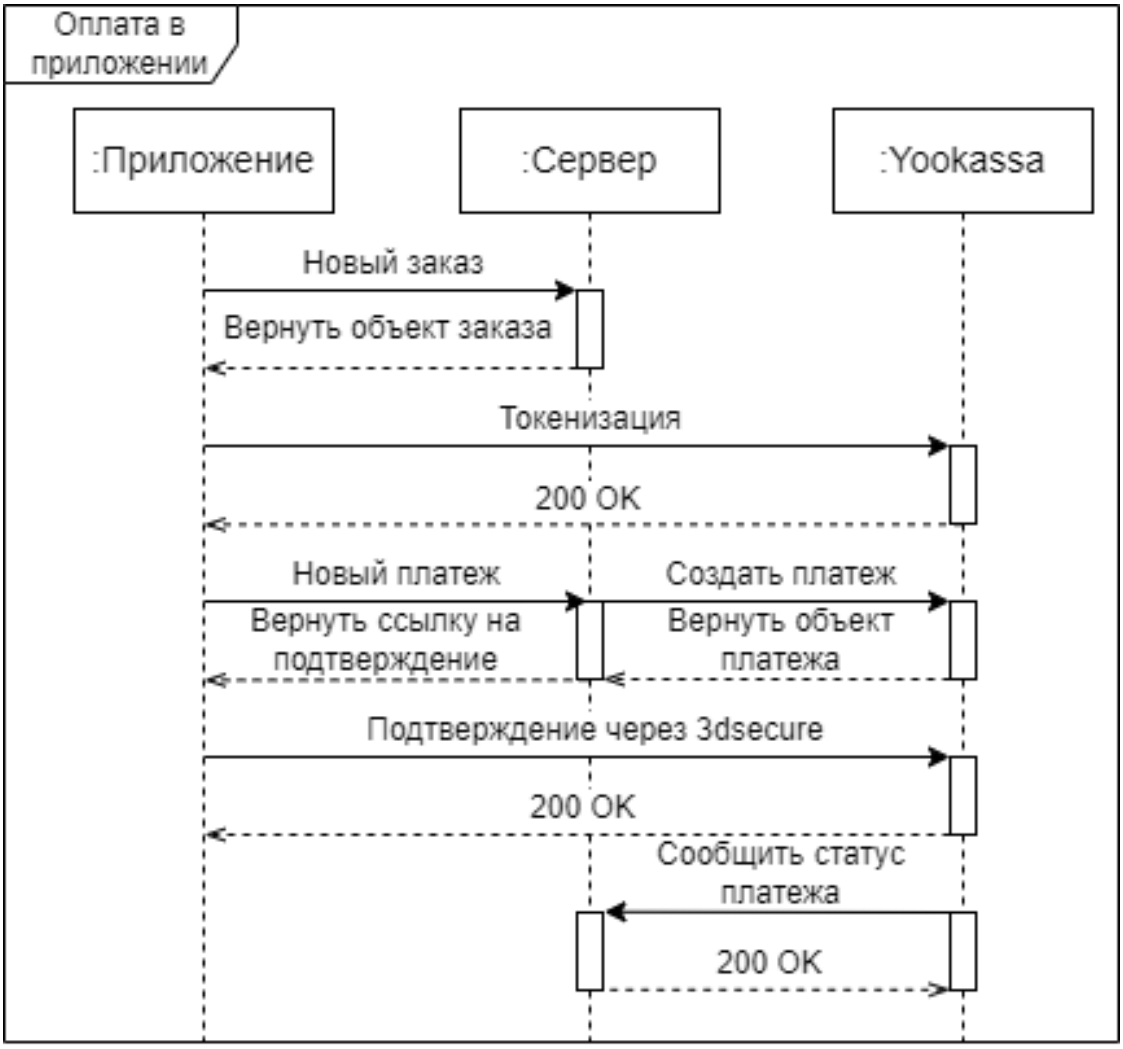

Рисунок 5 – Диаграмма последовательности оплаты в приложении

Интеграция с Firebase API

Push-уведомления представляют собой эффективный инструмент для оповещения пользователей и установления контакта с ними, поддержания их интереса к продукту или услуге. Сервис Firebase [14] предлагает API для отправки сообщений без дополнительной оплаты.

Для отправки уведомлений на одно устройство или группу устройств необходимо использовать файл, сгенерированный в личном кабинете в формате JSON и содержащий информацию об учетных данных, который должен храниться в безопасности. Каждый отправитель идентифицируется уникальным числовым значением, созданным при создании проекта в Firebase Console. Класс `PushService`, предназначенный для взаимодействия с Firebase API, представлен в листинге 2 приложения Б.

Константа `URL_SEND` содержит URL-адрес для отправки уведомлений через FCM (Firebase Cloud Message). Константа `CREDENTIALS_PATH` содержит путь к файлу учетных данных FCM. Метод `getAccessToken` – получает и возвращает токен доступа для отправки уведомлений. Метод `send` – отправляет уведомление с помощью переданного объекта `Message`. Он использует HTTP-клиент для выполнения запроса к FCM, а затем обрабатывает результат. Вспомогательный метод `sendPush` – принимает массив полей и создает из него объект `Message`, а затем вызывает метод `send`. Метод `sendPushes` – отправляет уведомления нескольким устройствам. Он циклически отправляет уведомления каждому устройству и подсчитывает успешные отправки. Общий процесс отправки push-уведомлений представлен в диаграмме последовательности на рисунке 6.

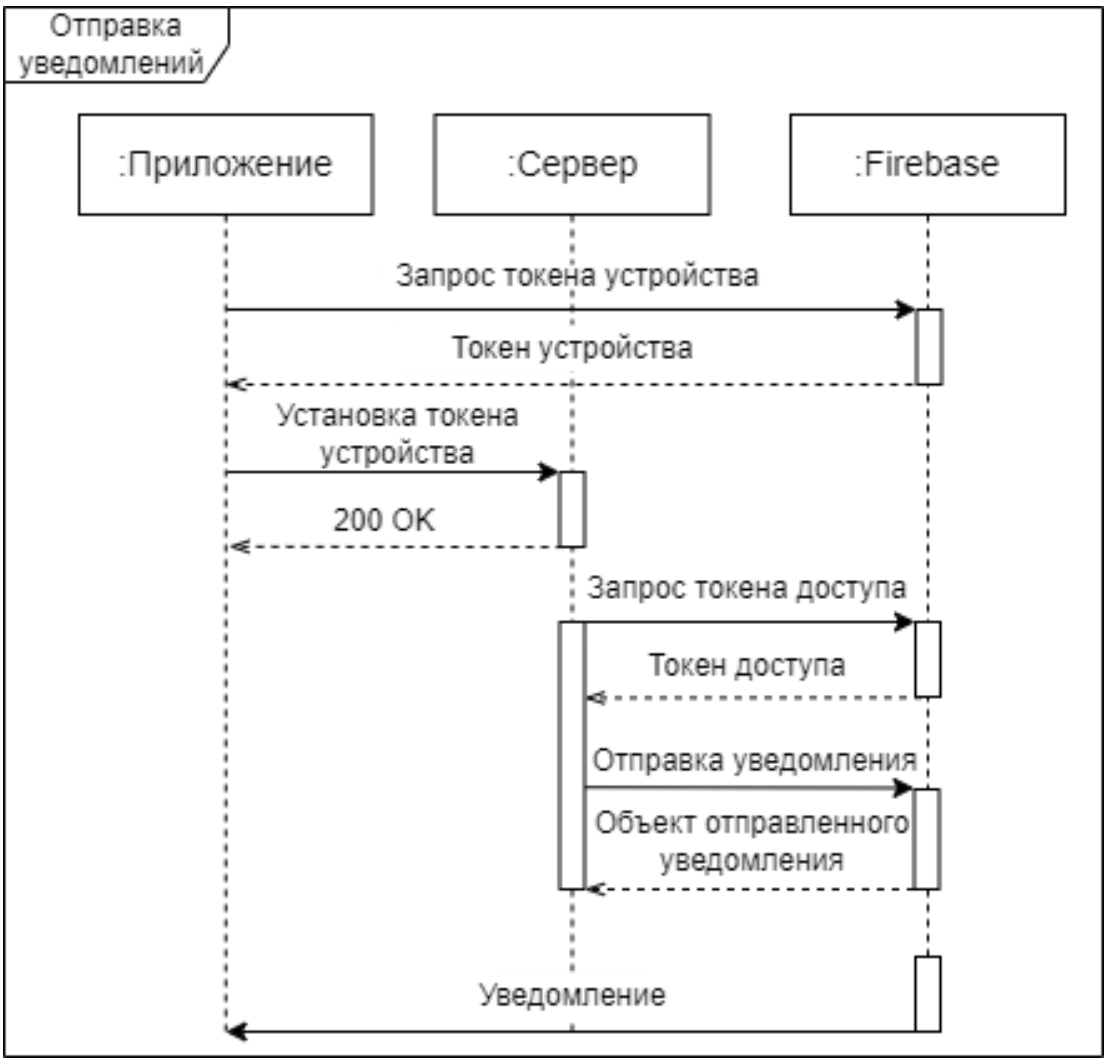

Рисунок 6 – Диаграмма последовательности отправки уведомлений

4. ТЕСТИРОВАНИЕ

Тестирование API является важной частью процесса разработки. Postman [18] предоставляет удобную платформу для отправки запросов к API, анализа ответов, создания тестовых скриптов и автоматизации тестирования.

В Postman можно создавать коллекции тестов, которые содержат набор запросов к API и связанные с ними тесты для автоматизированного выполнения. Данный сервис предоставляет возможность создания автоматизированных тестовых сценариев с помощью JavaScript. Эти сценарии могут проверять ответы от сервера на соответствие ожидаемым данным, а также выполнять различные проверки и валидации.

Пример теста с проверкой структуры ответа GET запроса по адресу `{{API_URL}}/catalog/` представлен в листинге 9.

Листинг 9 – Тест проверки структуры тела ответа на запрос каталога

```
pm.test("Validate the structure of the data array", function () {
    const responseData = pm.response.json();

    pm.expect(responseData).to.be.an('object');
    pm.expect(responseData.data).to.be.an('array');

    responseData.data.forEach(function(item) {
        pm.expect(item).to.have.property('id').that.is.a('number');
        pm.expect(item).to.have.property('name').that.is.a('string');
        item.items.forEach(function(subItem) {
            pm.expect(subItem).to.have.property('id').that.is.a('number');
            pm.expect(subItem).to.have.prop-
erty('name').that.is.a('string');
            pm.expect(subItem).to.have.property('price').that.is.a('num-
ber');
            pm.expect(subItem).to.have.property('com-
pound').that.is.a('string');
            pm.expect(subItem).to.have.property('weights').that.sat-
isfy((value) => {
               return Array.isArray(value) || typeof value === 'number';
            });
            pm.expect(subItem).to.have.property('im-
age').that.is.a('string');
            pm.expect(subItem).to.have.property('isLunch').that.is.a('bool-
ean');
        });
    });
});
```

Всего было реализовано 119 тестов, проверяющих код ответа, формат ответа, структуру ответа, обязательные поля в ответе и т.д. В среднем каждый запрос содержит 4–5 тестов.

Все тесты были пройдены и результаты представлены в отчете Postman. Отчет представлен на рисунке 7.

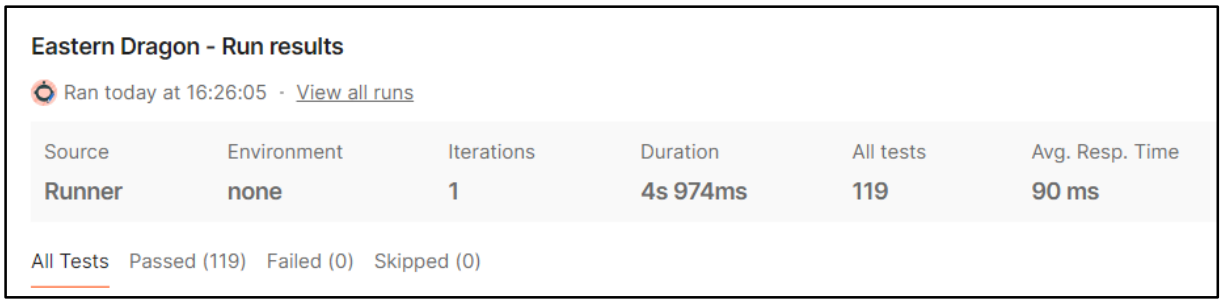

Рисунок 7 – Отчет тестирования

Каждый запрос в отчете содержит информацию о пройденных или не пройденных тестах. Пример запроса с пройденными тестами представлен на рисунке 8.

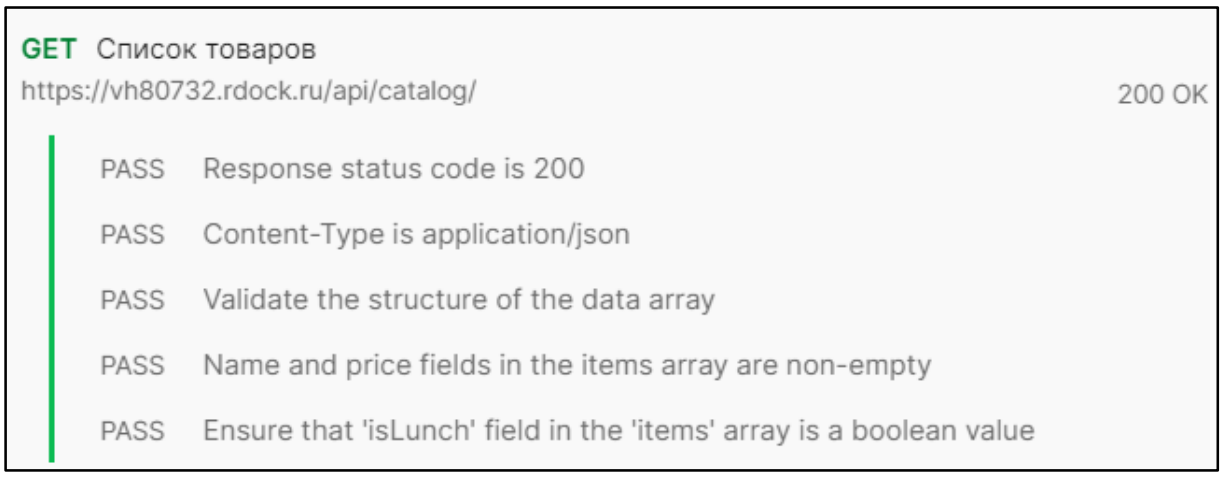

Рисунок 8 – Пример запроса с пройденными тестами

ЗАКЛЮЧЕНИЕ

В результате выполнения выпускной квалификационной работы была разработана серверная часть приложения для столовой «Восточный дракон».

1. Произведен анализ предметной области и аналогичных проектов.
2. Определены варианты использования и компоненты системы.
3. Спроектирована база данных и REST API.
4. Реализовано API.
5. Реализована интеграция с сервисом онлайн-оплаты Yookassa.
6. Реализована интеграция с сервисом отправки push-уведомлений Firebase Cloud Message.
7. Проведено тестирование API с помощью сервиса Postman.

Разработанную серверную часть можно улучшить следующими путями:

1) добавление нового функционала (внедрить программу лояльности);

2) улучшение текущего функционала (сменить способ авторизации на более быстрый и удобный);

3) интеграция с системами заведения;

4) оптимизация запросов к базе данных;

5) автоматизация процессов (автоматическая отправка push-уведомлений о смене статуса заказа);

6) большее покрытие кода тестами для лучшей его поддержки в дальнейшем.

ЛИТЕРАТУРА

1. Что такое CMS. [Электронный ресурс] URL: https://help.reg.ru/support/hosting/cms/chto-takoye-cms (дата обращения: 30.01.2024 г.).

2. Что такое API. [Электронный ресурс] URL: https://practicum.yandex.ru/blog/chto-takoe-api/ (дата обращения: 05.02.2024 г.).

3. Что такое SSL-сертификат – определение и описание. [Электронный ресурс] URL: https://www.kaspersky.ru/resource-center/definitions/what-is-a-ssl-certificate (дата обращения: 10.02.2024 г.).

4. Операции CRUD – Что такое CRUD? [Электронный ресурс] URL: https://appmaster.io/ru/blog/grubye-operatsii-chto-takoe-grubye-operatsii (дата обращения: 10.02.2024 г.).

5. Сообщения HTTP. [Электронный ресурс] URL: https://developer.mozilla.org/ru/docs/Web/HTTP/Messages (дата обращения: 10.02.2024 г.).

6. JSON. [Электронный ресурс] URL: https://www.json.org/json-ru.html (дата обращения: 15.02.2024 г.).

7. ORM в 1C-Bitrix. [Электронный ресурс] URL: https://dev.1c-bitrix.ru/learning/course/index.php?COURSE_ID=43&CHAPTER_ID=05748&LESSON_PATH=3913.3516.5748 (дата обращения: 10.03.2024 г.).

8. Коллекции в 1C-Bitrix. [Электронный ресурс] URL: https://dev.1c-bitrix.ru/learning/course/index.php?COURSE_ID=43&CHAPTER_ID=011743&LESSON_PATH=3913.3516.5748.11743 (дата обращения: 20.02.2024 г.).

9. Документация API Yookassa. [Электронный ресурс] URL: https://yookassa.ru/developers (дата обращения: 10.03.2024 г.).

10. Webhooks Atlassian. [Электронный ресурс] URL: https://developer.atlassian.com/server/jira/platform/webhooks/ (дата обращения: 10.03.2024 г.).

11. CMS 1C-Bitrix. [Электронный ресурс] URL: https://www.1c-bitrix.ru/ (дата обращения: 20.02.2024 г.).

12. Справочный центр хостинга Reddock. [Электронный ресурс] URL: https://reddock.ru/help/ (дата обращения: 20.02.2024 г.).

13. Паттерн проектирования «Репозиторий». [Электронный ресурс] URL: https://designpatternsphp.readthedocs.io/en/latest/More/Repository/README.html (дата обращения: 10.03.2024 г.).

14. Firebase REST API Documentation. [Электронный ресурс] URL: https://firebase.google.com/docs/reference/fcm/rest/v1/projects.messages (дата обращения: 21.03.2024 г.).

15. Понимание REST API: Основные понятия и принципы работы. [Электронный ресурс] URL: https://cloud.yandex.ru/docs/glossary/rest-api (дата обращения: 25.01.2024 г.).

16. Arpaci-Dusseau R.H., Arpaci-Dusseau A.C. Operating Systems: Three Easy Pieces. – «Arpaci-Dusseau Books», 2015. – С. 551–556.

17. GraphQL. A query language for your API. [Электронный ресурс] URL: https://graphql.org/ (дата обращения: 10.02.2024 г.).

18. Документация Postman. [Электронный ресурс] URL: https://learning.postman.com/docs/introduction/overview/ (дата обращения: 15.04.2024 г.).

19. Арлоу Д., Нейштадт И. UML 2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование, 2-е издание. – Пер. с англ. – СПб: СимволПлюс, 2007. – 624 с.

20. Зандстра М. PHP: объекты, шаблоны и методики программирования, 3-е издание. – Москва: «Вильямс», 2010. – 560 с.

21. Веллинг Л, Томсон Л. Разработка веб-приложений с помощью PHP и MySQL. – 5-е издание. – СПб.: «Диалектика», 2019. – 768 с.

22. Дронов В. PHP и MySQL. 25 уроков для начинающих. // Издательство БХВ, 2021. – С. 432.

ПРИЛОЖЕНИЯ

Приложение А. Реализация класса ProductRepository

В листинге 1 представлена реализация класса `ProductRepository`.

Листинг 1 – Класс `ProductRepository`

```
final class ProductRepository implements Repository
{
    public static function getList(array $filter): ProductsCollection |
null
    {
        $items = Product::getTableEntity()::getList([
           'filter' => $filter,
           'select' => [
               'ID',
               'UF_NAME',
               'UF_ACTIVE',
               'UF_CATEGORY',
               'UF_PRICE',
               'UF_DESCR',
               'UF_WEIGHT',
               'UF_IMAGE',
               'CATEGORY',
           ]
        ])->fetchCollection();

        $collection = new ProductsCollection();
        foreach($items as $item)
        {
            $collection->push(new Product($item));
        }

        return $collection;
    }

    public static function getById(int $id): Product | null
    {
        $item = Product::getTableEntity()::getList([
            'filter' => [
                'ID' => $id,
                'UF_ACTIVE' => 1,
            ],
            'select' => [
                'ID',
                'UF_NAME',
                'UF_ACTIVE',
                'UF_CATEGORY',
                'UF_PRICE',
                'UF_DESCR',
                'UF_WEIGHT',
                'UF_IMAGE',
                'CATEGORY',
            ]
        ])->fetchObject();

        return $item ? new Product($item) : null;
    }

    public static function add(array $fields): int | null
    {
       $result = Product::getTableEntity()::add($fields);
```

Окончание листинга 1 приложения А

```
        if ($result->isSuccess()) {
            return $result->getId();
        } else {
            return null;
        }
    }

    public static function delete(int $id): bool
    {
        $result = Product::getTableEntity()::delete($id);
        return $result->isSuccess();
    }
}
```

Приложение Б. Реализация класса PushService

В листинге 2 представлена реализация класса `PushService`.

Листинг 2 – Класс `PushService`

```
class PushService
{
    const URL_SEND = 'https://fcm.googleapis.com/v1/projects/eastern-
dragon-app/messages:send';
    const CREDENTIALS_PATH = '/firebase/firebase_credentials.json';

    private \Google\Client $client;


    public function __construct()
    {
       $this->client = new \Google\Client();
       $this->client->setAuthConfig($this->getCredentialsPath());
       $this->client->addScope('https://www.googleapis.com/auth/fire-
base.messaging');
    }

    public function getClient(): \Google\Client
    {
       return $this->client;
    }
    public function getCredentialsPath(): string
    {
       return $_SERVER['DOCUMENT_ROOT'] . self::CREDENTIALS_PATH;
    }

    public function getAccessToken(): array
    {
       $this->client->fetchAccessTokenWithAssertion();
       return $this->client->getAccessToken();
    }

    public function send(Message $message): bool
    {
       $accessToken = $this->getAccessToken();
       $fields = $message->collectData();
       $requestHeaders = [
          'Content-Type'  => 'application/json',
          'Authorization' => 'Bearer ' . $accessToken['access_token'],
       ];

       $httpClient = new HttpClient();
       $httpClient->setHeaders($requestHeaders);

       $result = $httpClient->post(self::URL_SEND, json_encode($fields,
JSON_UNESCAPED_UNICODE));
       $decodedResult = json_decode($result, true);

       if ($decodedResult['error'])
       {
          Tools::log([$fields, $decodedResult, '$result' => $result, '$re-
questHeaders' => $requestHeaders, 'fields' => json_encode($fields, JSON_UN-
ESCAPED_UNICODE)], 'sendFirebaseError', 'pushes');
          return false;
       }
       else
       {
```

```
            return true;
        }
    }

    public function sendPush(array $push): bool
    {
        $message = Message::fromArray([
            'token'      => $push['device_token'],
            'title'      => $push['title'],
            'text'       => $push['text'],
            'data'       => $push['data']
        ]);
        $sendRes = $this->send($message);
        if (!$sendRes)
        {
            Tools::log([
                'token'      => $push['device_token'],
                'title'      => $push['title'],
                'text'       => $push['text'], $push], 'sendErrors', 'push-
es');
            return false;
        }
        return true;
    }

    public function sendPushes(array $deviceTokens, Message $message): int
    {
        if (!$deviceTokens) {
            throw new Exception("Нет девайс токенов");
        }
        $count = 0;
        foreach ($deviceTokens as $deviceToken) {
            $message->setToken($deviceToken);
            $isSuccess = $this->send($message);
            if ($isSuccess) {
                $count++;
            }
        }
        return $count;
    }
}
```

Приложение В. Диаграмма вариантов использования

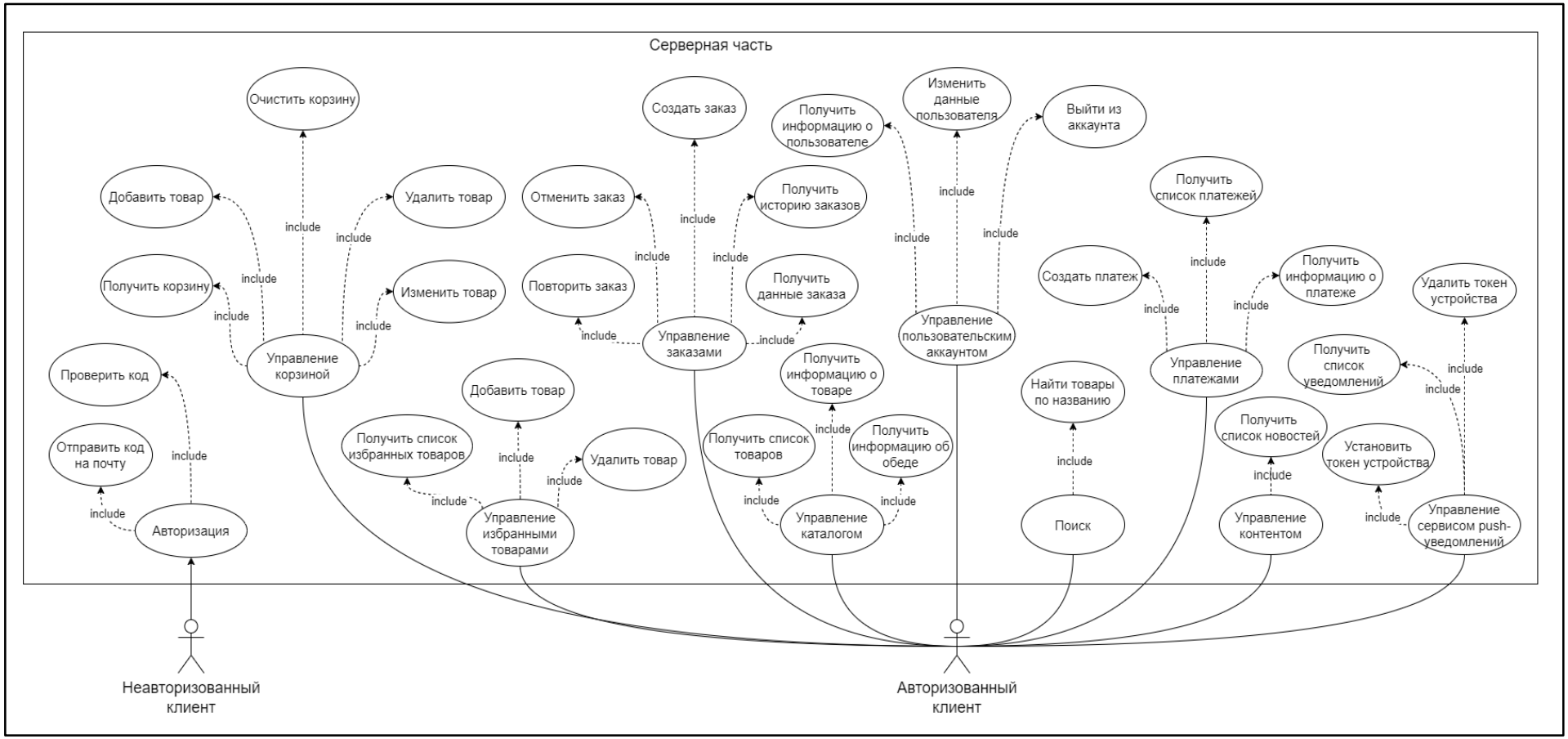

Рисунок 1 – Диаграмма вариантов использования

Приложение Г. Диаграмма компонентов

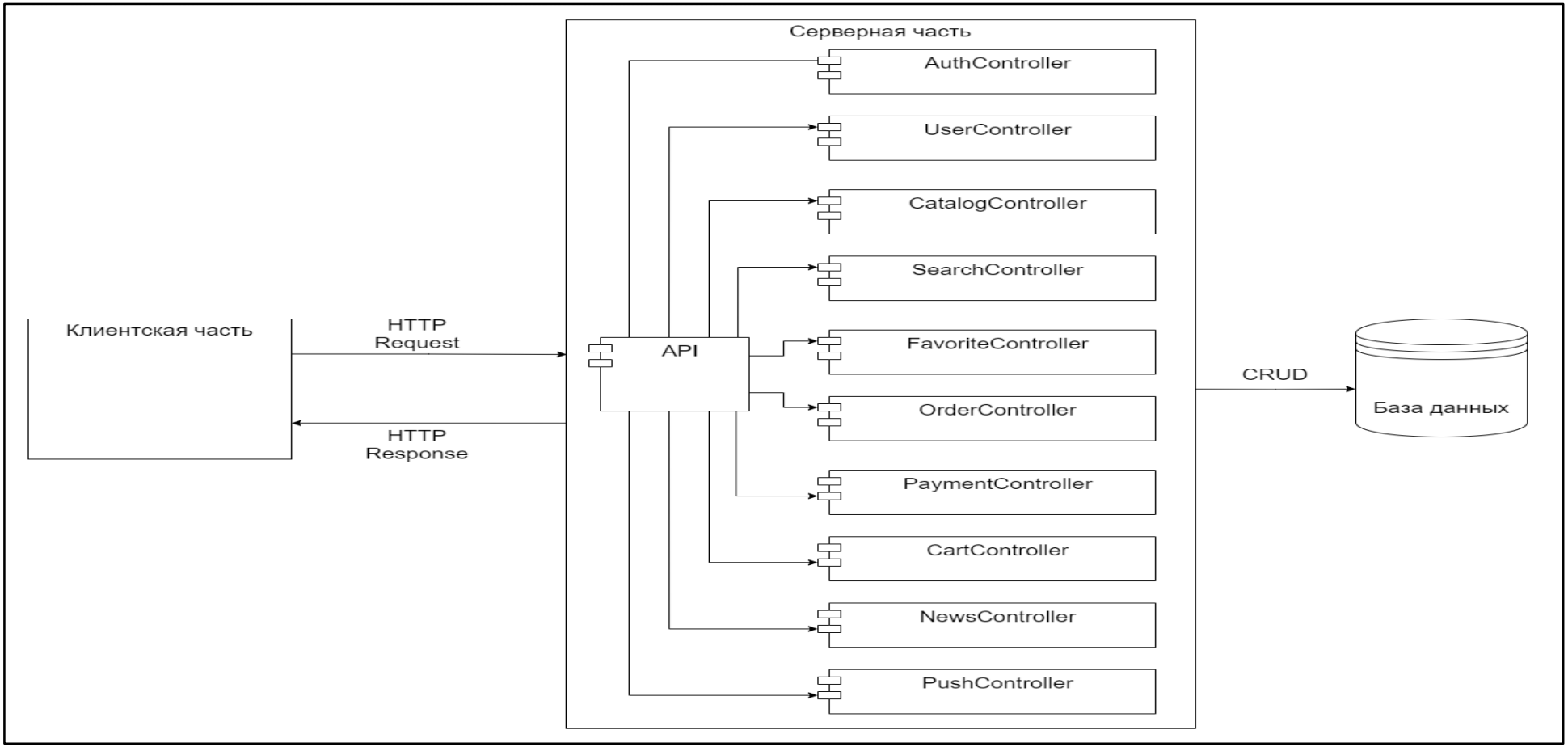

Рисунок 2 – Диаграмма компонентов

Приложение Д. Схема базы данных

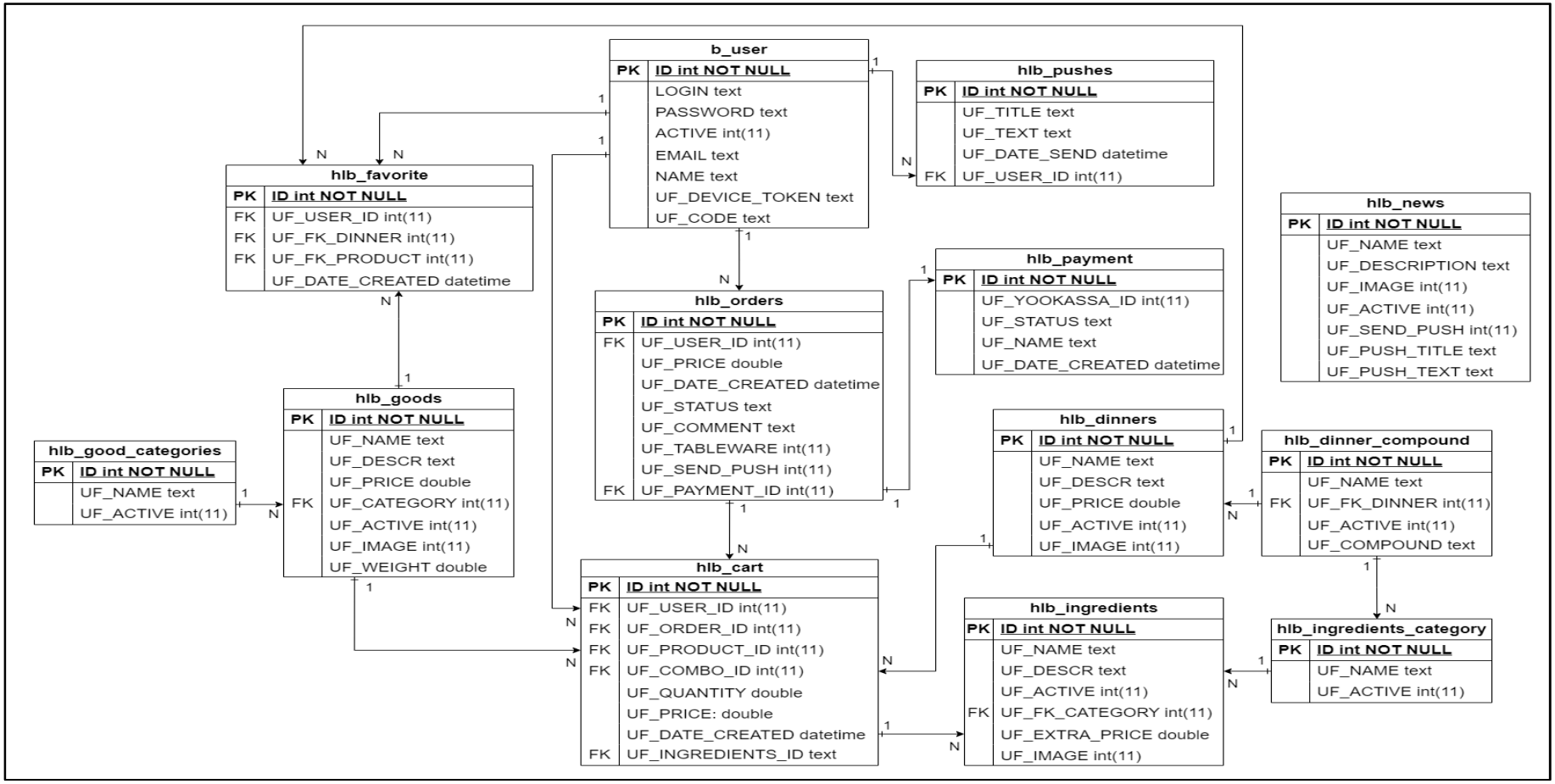

Рисунок 3 – Схема базы данных