МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Кафедра «Прикладная математика и информатика»
(наименование)

09.03.03 Прикладная информатика
(код и наименование направления подготовки, специальности)

Бизнес информатика
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
(БАКАЛАВРСКАЯ РАБОТА)

на тему «Разработка проекта мобильного приложения для заказа еды с использованием геолокации и системы рекомендаций»

Студент В.В. Шлейзе
(И.О. Фамилия) (личная подпись)

Руководитель к.т.н., Д.Г. Токарев
(ученая степень, звание, И.О. Фамилия)

Тольятти 2024

Аннотация

Тема: «Разработка проекта мобильного приложения для заказа еды с использованием геолокации и системы рекомендаций»

Объектом работы являются принципы функционирования и разработки мобильных приложений, предметом – разработка мобильного приложения по доставке еды.

Структура работы представлена введением, тремя главами основной части, заключением, списком использованных источников, и тремя приложениями.

В первой главе работы проводится описание предметной области, моделирование бизнес-процессов предметной области. Также проводится обзор аналогов и выбор средств проектирования и разработки, в результате чего для разработки приложения был выбран фреймворк Expo, а в качестве СУБД MongoDB. Также проведена постановка задачи на разработку системы, описаны ее основные функции.

Вторая глава посвящена разработке приложения. Проведено UML моделирование работы системы, позволившее определить основные функции и состав пользователей, после чего проведена разработка мобильного приложения и личного кабинета администратора.

В последней главе работы проведено тестирование разработанного приложения и описаны его основные функции.

Данная бакалаврская работа состоит из 32 рисунков, 2 таблиц, списка используемых источников из 18 источников на русском языке, 3 источников на иностранном языке, и 3 приложений.

Содержание

Введение

В настоящее время информационные технологии развиваются очень стремительно. Каждый год быстро меняются технологии, появляются новые изобретения и стандарты. Одной из самых динамически развивающихся техногенных инфраструктур являются смартфоны и рынок мобильных приложений. Мобильные телефоны давно перестали быть средством только для связи. Сейчас они являются неотъемлемой частью жизни человека, принимая участие в большинство сфер его деятельности.

Гиганты в производстве смартфонов, такие как Samsung, HTC, Apple, и другие, в стремлении превзойти друг друга регулярно совершенствуют свою продукцию, предоставляя разработчикам приложений большие технические возможности и простор для полета фантазии. Поэтому рынок мобильных программ огромен и разнообразен, и в нем пользователи могут найти то, что подходит именно под их нужды, будь то игры, новостные порталы, офисные инструменты, путеводители или средства для развлечения и досуга.

Именно поэтому сфера разработки мобильных приложений очень активно развивается, и разработчики подобных приложений получают огромный доход, все это указывает на актуальность выбранной темы работы.

В сегодняшнем быстро меняющемся цифровом ландшафте разработка мобильных приложений стала краеугольным камнем как для бизнеса, так и для разработчиков. Тем не менее, создание мобильных приложений, которые без проблем работают на нескольких платформах, сохраняя при этом высокий уровень эффективности, всегда представляло собой серьезную проблему.

Целью данной работы является проектирование структуры и разработка приложения доставки еды с используем геолокации и системы рекомендаций.

Объектом работы являются принципы функционирования и разработки мобильных приложений.

Предметом работы является разработка проекта мобильного приложения по доставке еды с использованием геолокации и системы рекомендаций.

Задачи работы:

- описать предметную область разработки мобильных приложений и основные требования к ним;
- провести моделирование бизнес-процессов предметной области;
- провести обзор аналогичных систем и выявить их основные недостатки;
- выбрать технологии разработки;
- описать основные требования к приложению;
- провести разработку мобильного приложения и его администраторской части с помощью выбранных средств и технологий;
- провести тестирование и документирование разработанной информационной системы.

При написании работы будут использованы труды отечественных и зарубежных авторов, а также интернет-источники по теме мобильной разработки.

1 Аналитическая часть

1.1 Описание предметной области

В эпоху распространения мобильных технологий наблюдается снижение популярности персональных компьютеров в пользу смартфонов и планшетов. Поэтому, стремясь быть ближе к своим клиентам, многие компании разрабатывают мобильные приложения или адаптируют свои веб-сайты для мобильных устройств.

Мобильное приложение можно определить, как программное обеспечение, предназначенное специально для мобильных устройств, таких как смартфоны и планшеты [20]. Каждое приложение выполняет определенные задачи и функции. Важно, чтобы мобильное приложение обладало функциональностью, но при этом было интуитивно понятным и удобным в использовании, иначе пользователи не станут его устанавливать.

Мобильное приложение и адаптивная версия сайта - это не одно и то же. Мобильная версия сайта представляет собой уменьшенную и слегка переработанную версию сайта для ПК. Эта версия особо не меняется в дизайне, лишь ее элементы становятся больше, так как экран телефонов и планшетов намного меньше экрана компьютера. Мобильное приложение же - это абсолютно новый продукт, созданный с учетом всех особенностей смартфонов, функционал его «заточен» под возможности телефона [19]. Чаще всего в них оставляют фирменные цвета, логотипы, стили и так далее, но структура становится совершенно иной.

Мобильные приложения имеют следующие характеристики:

- интерфейс создан именно для работы на мобильном устройстве без мыши и курсора;
- удобное и понятное для пользователей оформление;
- в отличие от адаптивной версии сайта, приложение может выполнять

различные функции в фоновом режиме, а некоторые даже оффлайн, например, присылать сообщения, уведомления, напоминания, отслеживать геопозицию и так далее;

- хранение персональных данных пользователя, например, паспортные данные, номер телефона, адрес, номер медицинского полиса и другое;
- различные приложения могут отслеживать геопозицию, биологические ритмы и многие другие данные в зависимости от того, какое это приложение [19].

Мобильные приложения делятся на разрабатываемые для:

- клиентов,
- внутреннего использования.

Приложения для клиентов необходимы для реализации какого-либо бизнеса, например, интернет-банки, бронирование билетов, интернет-магазины. Сюда же входят и программы лояльности (бонусные, скидочные карты, кэшбэк и тому подобное) [7].

Приложения для внутреннего использования обычно помогают синхронизировать и оптимизировать работу команды.

По типу мобильные приложения делятся на:

- мобильные приложения на основе веб сайта,
- гибридные приложения,
- нативные приложения.

Мобильные приложения на основе веб сайта проще в создании, обновлении и поддержке, но имеют низкую функциональность.

Гибридные приложения имеют более широкий функционал, нежели мобильные приложения и веб-сайты, например, пуш-уведомления, а также могут скачиваться на телефон пользователя через магазины приложений.

Разработка нативного приложения намного трудозатратнее, так как требует разработку под каждую операционную систему отдельно (iOS, Android, Windows Phone). Но отличается самой высокой функциональностью

среди других типов приложений, например, может использовать контакты, камеру, микрофон и так далее.

Программное обеспечение для смартфонов и цифровых помощников, созданное для операционных систем Android и iOS, может быть установлено на устройства уже на этапе производства, скачано из специализированных магазинов или доступно через интернет-браузеры. При создании таких приложений разработчики используются язы программирования и разметки, среди которых Java, Swift, C# и HTML5 [11].

1.2 Моделирование бизнес-процессов

Нотация IDEF0 представляет собой методологию функционального моделирования и графическую нотацию, предназначенную для формализации и описания бизнес-процессов [4].

Проведем моделирование бизнес-процессов, которые будут происходить при взаимодействии с приложением доставки еды, то есть составим диаграмму «КАК ДОЛЖНО БЫТЬ».

Контекстная диаграмма работы приложения показана на рисунке 1.

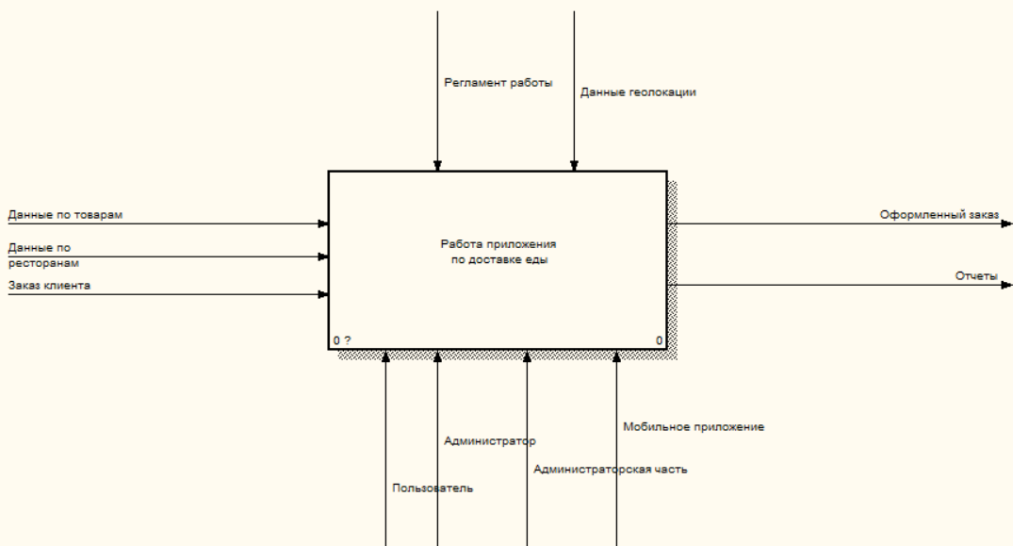

Рисунок 1 – Контекстная диаграмма работы приложения

Согласно контекстной диаграмме, входными данными процесса являются следующие:

- данные по товарам (блюдам),
- данные по ресторанам,
- заказ клиента.

Механизмы:

- пользователь,
- администратор,
- администраторская часть,
- мобильное приложение.

Управляющие воздействия:

- регламент работы,
- данные геолокации.

Выходные данные:

- данные по заказам,
- отчеты.

Декомпозиция контекстной диаграммы приведена на рисунке 2.

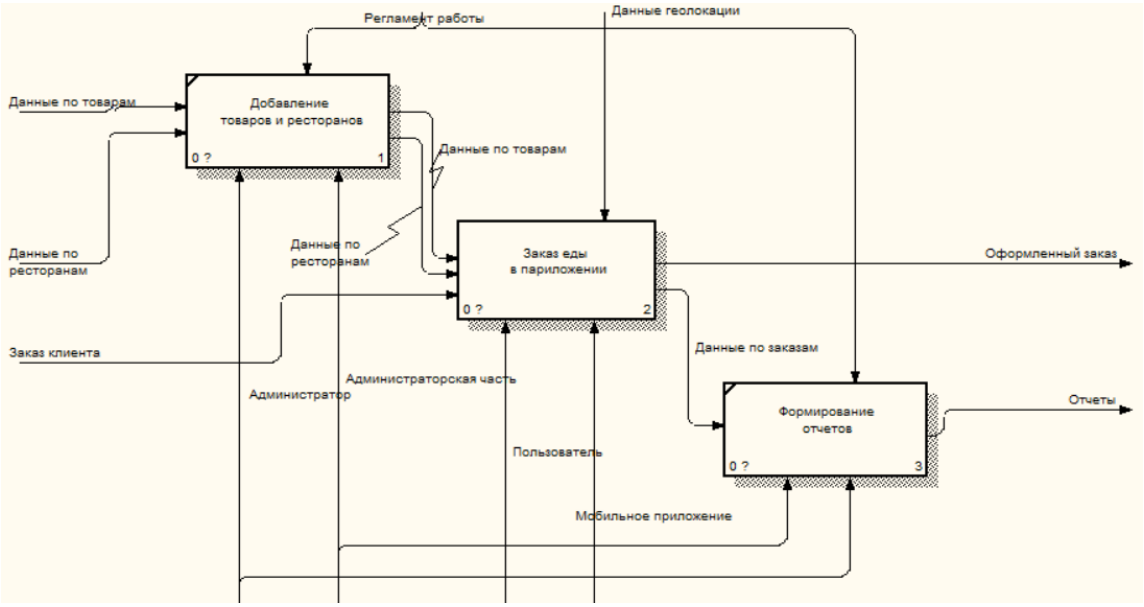

Рисунок 2 – Декомпозиция контекстной диаграммы

Согласно диаграмме декомпозиции, администратор получает данные по товарам и ресторанам, и добавляет их в систему, с помощью администраторской части.

После этого, основываясь на данных геолокации и рекомендаций, пользователь формирует заказ.

На основе данных по заказам формируются аналитические отчеты.

Диаграмма декомпозиции процесса «Заказ еды в приложении» показана на рисунке 3.

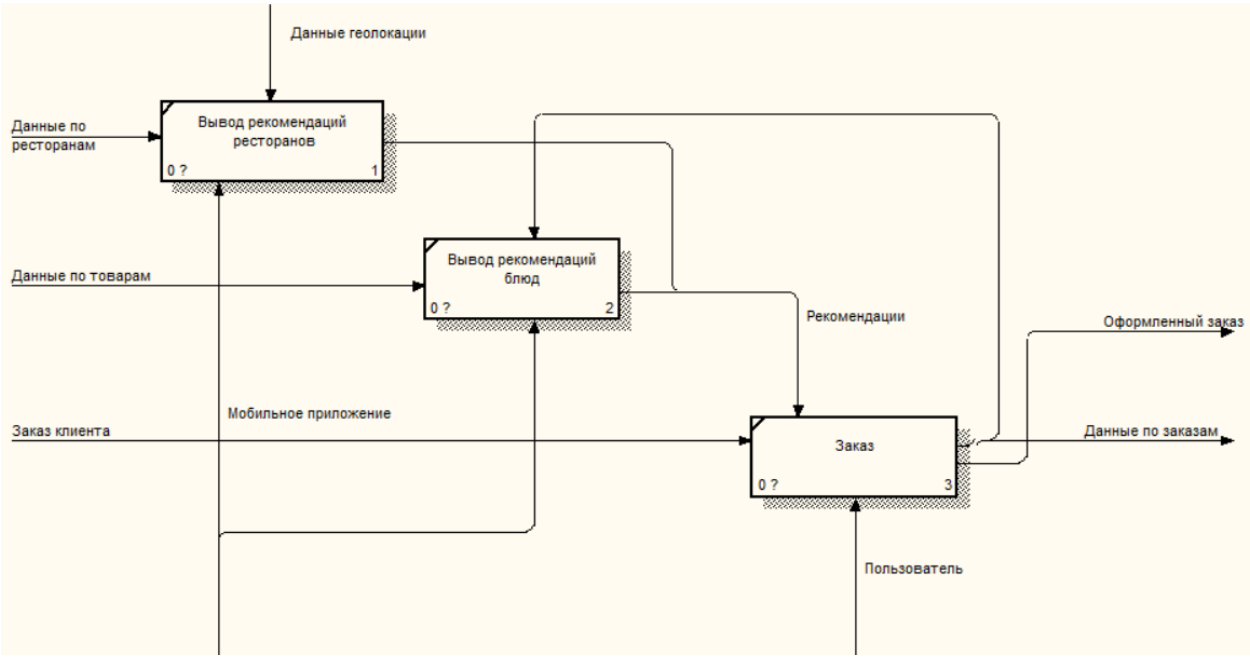

Рисунок 3 – Диаграмма декомпозиции процесса заказа

Согласно представленной декомпозиции, пользователь сначала выбирает необходимый ресторан, основываясь на данных геолокации.

После этого происходит выбор блюда на основании рекомендаций, в которых используются данные по заказам.

Затем происходит заказ блюда.

DFD диаграмма функционирования приложения показана на рисунке 4.

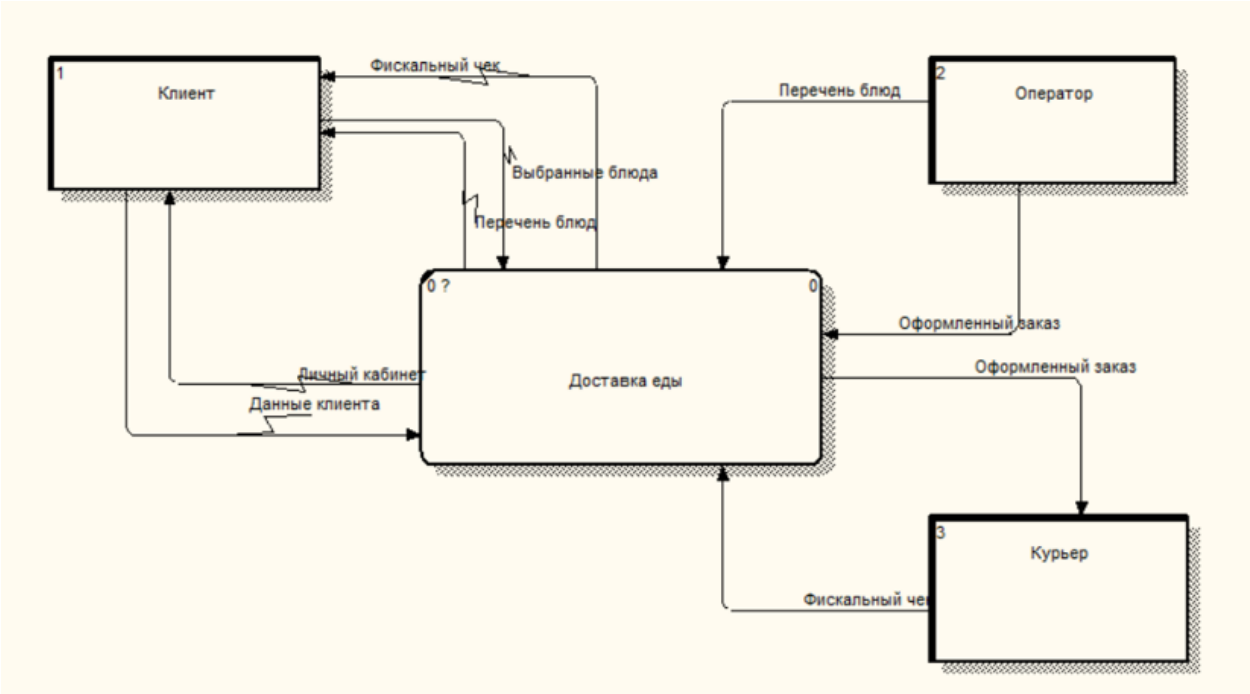

Рисунок 4 – DFD диаграмма функционирования системы

1.3 Обзор аналогичных систем

На рынке существует достаточно большое количество приложений, близких по функционалу к разрабатываемому.

Проведем обзор аналогичных разрабатываемому приложению программ, а именно:

- «Delivery Club»,
- «Самокат»,
- «Dostaевский».

В приложении Delivery Club, после указания адреса доставки (вручную или с использованием геолокации), доступен выбор из различных категорий ресторанов и магазинов (рисунок 5).

Рисунок 5 – Главная страница приложения «Delivery Club»

После выбора необходимой категории, выводятся наиболее популярные заведения в ней (рисунок 6).

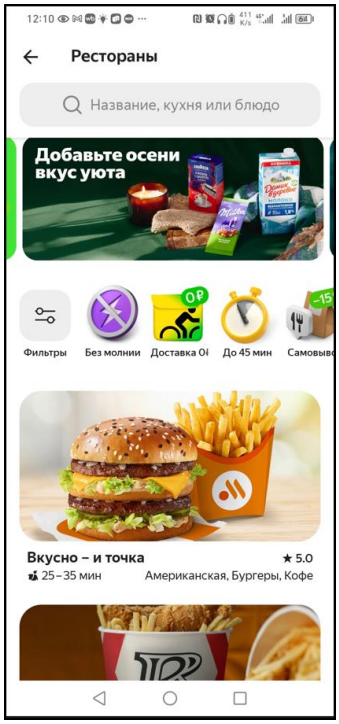

Рисунок 6 – Выбор заведения

После выбора ресторана, выводится список блюд доступных для заказа в данном заведении (рисунок 7).

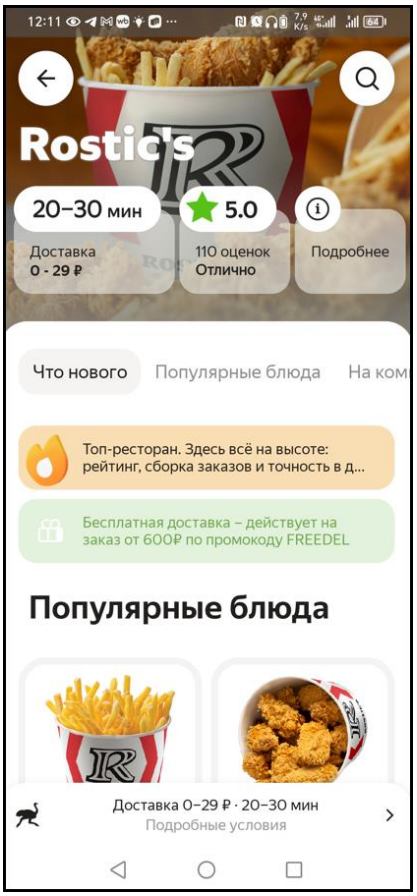

Рисунок 7 – Список блюд

Пользователь имеет возможность выбрать блюдо, посмотреть его состав, положить блюдо в корзину, и произвести заказ. Для оформления заказа обязательна регистрация в приложении.

Достоинствами приложения являются:

- возможность фильтрации блюд по различным параметрам;
- качественная система рекомендаций.

К недостаткам можно отнести:

- сложность навигации;
- перегруженность интерфейса большим количеством элементов.

Далее рассмотрим одно из самых популярных приложений в этой нише, приложение «Самокат».

После запуска, приложение также предлагает выбрать или определить адрес по геолокации.

На главной странице приложения выводятся рекомендации, и список наиболее популярных позиций для заказа (рисунок 8).

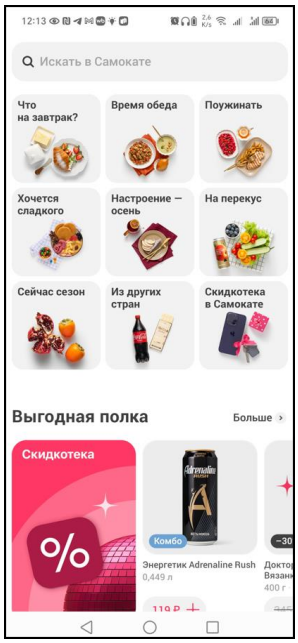

Рисунок 8 – Главная страница приложения «Самокат»

После выбора категории блюда, блюда в ней распределены еще на несколько подкатегорий (рисунок 9).

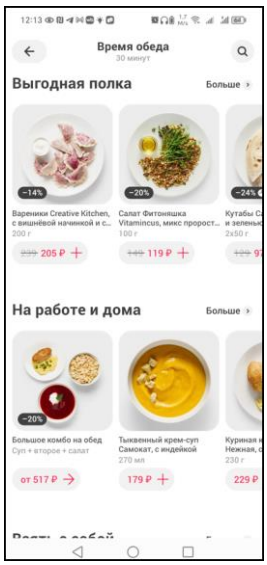

Рисунок 9 – Подкатегории

После формирования корзины, можно проверить состав заказа, а также применить к заказу различные бонусы, такие как промокод, или оплата бонусами «Спасибо» (рисунок 10).

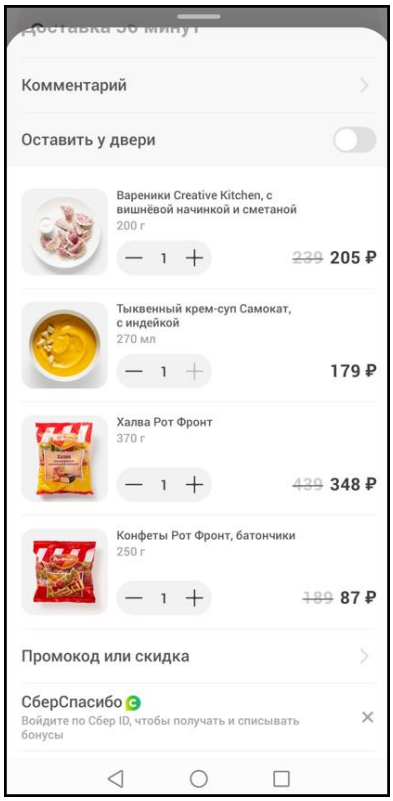

Рисунок 10 – Форма корзины

Для оформления заказа так же требуется регистрация в приложении.

К достоинствам приложения можно отнести:

- удобный функционал главной страницы;
- поиск в приложении.

К недостаткам приложения можно отнести:

- неудобная система категорий;
- отсутствие фильтрации блюд по параметрам.

Приложение «Dostaевский» - это тоже достаточно популярное приложение для заказа еды.

После указания адреса доставки, на главной странице приложения отображаются категории блюд и список рекомендаций (рисунок 11).

Здесь же можно перейти к меню, просмотреть корзину, или открыть чат со службой поддержки.

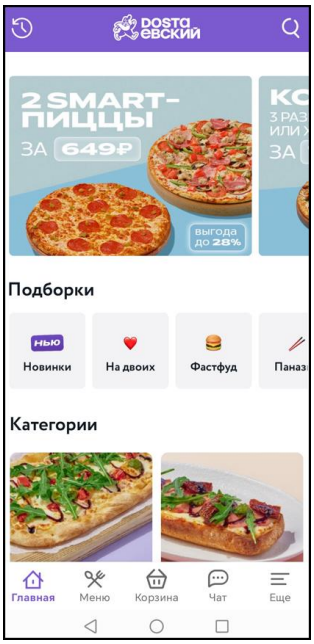

Рисунок 11 – Главное окно приложения «Dostaевский»

Функционал корзины стандартен для подобных приложений (рисунок 12).

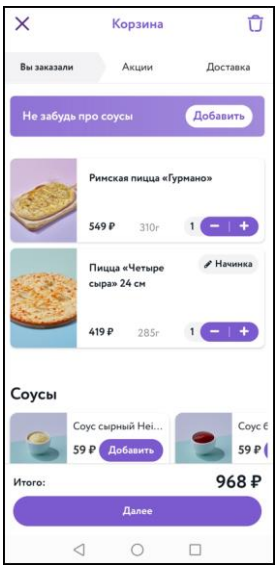

Рисунок 12 - Корзина

К достоинствам приложения можно отнести следующие:

- качественный дизайн;
- простота интерфейса и навигации.

К недостаткам можно отнести отсутствие фильтрации блюд по параметрам.

Сравнение рассмотренных приложений представлено в таблице 1.

Таблица 1 – Сравнение приложений доставки еды

Функции	Delivery Club	Самокат	Dostaевский	Разр. система
Возможность заказа из нескольких магазинов	+	-	-	+
Поиск и выбор ближайших магазинов	+	+	-	+
Заказ без регистрации	-	-	-	+
Размер	162	180	210	67

На основании представленного обзора, мы определили основные функции, присущие аналогичным приложениям. К таким функция можно отнести наличие рекомендательной системы, использование геолокации для определения местонахождения пользователя, разнесение блюд и заведений по категориям, функции поиска и фильтрации.

При разработке приложения следует ориентироваться на реализацию в нем именно представленных функций, при этом старая избежать описанных недостатков. Также, разрабатываемое приложение, в отличие от аналогов, будет обладать возможностью заказа без регистрации, что позволит привлечь дополнительных пользователей.

1.4 Выбор средств проектирования и разработки

Expo представляет собой комплексный инструментарий, облегчающий разработку приложений на базе нативного React. Он выполняет функции аналогичные тем, что предоставляют Laravel или Symphony для разработчиков PHP, а также Ruby on Rails для программистов, использующих Ruby. Возможность функционирования без использования специализированных средств разработки (например, Android Studio), является одной из важнейших функций Expo. Это особенно ценно при разработке React Native приложений, поскольку разработчики не сталкиваются с проблемами, связанными с настройкой этих инструментов [10].

Expo также предлагает разработчикам удобный слой для работы с React Native API, что существенно облегчает использование и управление данными интерфейсами. Кроме того, в комплектации Expo имеются инструменты, которые значительно упрощают первичное создание и последующее тестирование приложений React Native. В дополнение, Expo предлагает доступ к компонентам пользовательского интерфейса и различным сервисам, которые обычно доступны только после установки сторонних нативных компонентов

для React. Эти возможности предоставляются через Expo SDK, что делает разработку более гибкой и доступной.

Таким образом, Expo оказывает неоценимую помощь разработчикам, стремящимся создавать эффективные и функциональные нативные приложения на платформе React.

Тем не менее, важно ознакомиться с некоторыми ограничениями Expo.

Expo приложения ограничены нативными API, которые поддерживает Expo SDK. Это означает, что если ваше приложение имеет весьма специфические задачи, единственный вариант для реализации такой функциональности - это использование простого React Native, или написание собственного кода с помощью библиотеки ExpoKit [6].

Expo привязывает пользователя к использованию его инструментов. Это означает, что можно просто установить и использовать большую часть отличных средств, доступных для разработки React Native, таких как инструменты командной строки, вспомогательные библиотеки и фреймворки пользовательского интерфейса. Но хорошая новость в том, что Expo SDK совместим с простыми приложениями React Native, поэтому вы не будете иметь никаких проблем, если извлечете приложение из Expo.

Автономные исполняемые файлы приложений Expo могут быть построены только при наличии подключения к сети. Expo предоставляет инструменты командной строки под названием Exp. Это позволяет разработчикам запускать процесс сборки проекта на серверах Expo. После завершения сборки будет доступна URL-ссыка для скачивания файла .apk или .ipa.

Даже с учетом этих ограничений важно иметь в виду, что Expo является полностью функциональным фреймворком с огромной поддержкой часто используемых Android и iOS интерфейсов API. Это значит, что он обеспечит вас основным функционалом, который обычно нужен приложениям. Так что для реализации нативного функционала зачастую хватит возможностей Expo.

Сравнение Expo с другими популярными фреймворками для разработки мобильных приложений приведено в таблице 2.

Таблица 2 – Сравнение популярных фреймворков мобильной разработки

Функции	**Expo**	**Flutter**	**NativeScript**
Используемый язык	Javascript, Typescript	Dart	Javascript
Наличие дополнительных библиотек	Много	Средне	Средне
Скорость работы	Средняя	Высокая	Высокая
Удобство разработки	Высокое	Среднее	Низкое
Поддержка	Хорошая	Хорошая	Средняя
Кроссплатформенность	Полная	Частично	Частично

Основными критериями Expo стали следующие:

- Наличие большого количества дополнительных библиотек.
- Высокое удобство разработки.
- Хорошая поддержка;
- Полная кроссплатформенность.

WebStorm (рисунок 13) — это передовая интегрированная среда разработки, поддерживающая JavaScript, HTML и CSS, включая современные инновации, такие как TypeScript, React, Angular, Vue.js и Node.js. Среда IDE известна своим удобным интерфейсом, интеллектуальной помощью в написании кода и мощными функциями отладки.

WebStorm представляет собой редактор кода, предназначенный для разработчиков, работающих с JavaScript. Этот инструмент широко используется в индустрии и известен своей функциональностью.

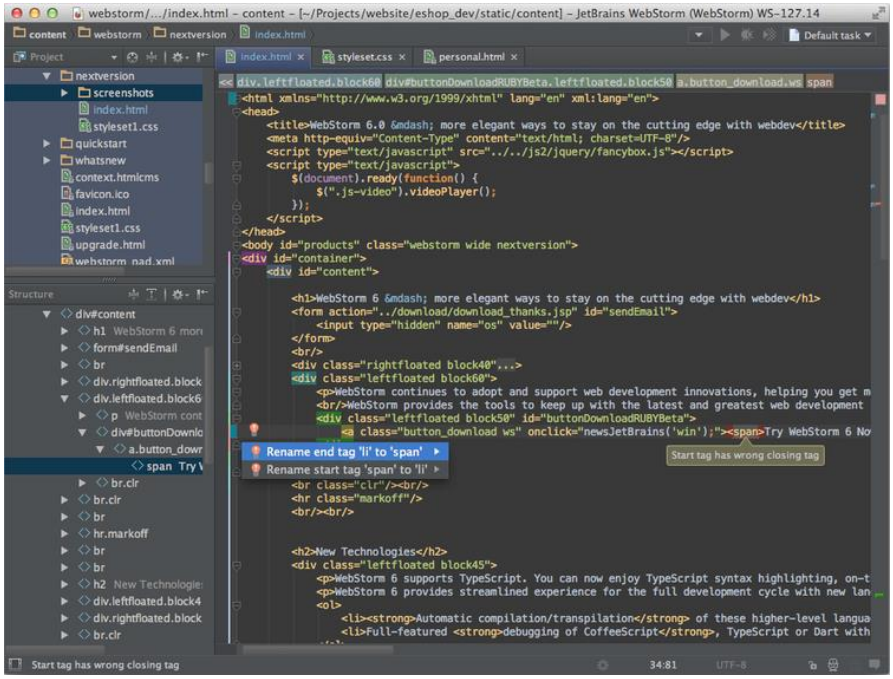

Рисунок 13 – Интерфейс WebStorm

WebStorm предлагает пользователям возможность начать работу сразу после установки, так как все необходимые инструменты для фронтенд-разработки на JavaScript включены по умолчанию. Среди них — популярные плагины и фреймворки, мощный анализатор, удобный движок для тестирования кода и современный отладчик приложений.

Кроме того, WebStorm является кроссплатформенным, что позволяет писать код в любой операционной системе и проверять результаты в различных браузерах.

В случае необходимости срочных изменений в проекте, WebStorm позволяет быстро скачать и установить редактор на любом устройстве, что обеспечивает оперативное внесение правок. Это особенно важно при работе над серьезными проектами, где требуется наличие всех необходимых плагинов и инструментов.

Интерфейс WebStorm можно настроить под индивидуальные предпочтения пользователя. Это включает возможность изменения интерфейса рабочего пространства, добавления или отключения плагинов и

организации инструментов в удобном порядке. Настройки могут быть заданы как для одного проекта, так и для всех новых файлов.

Для тех, кто предпочитает минималистичный подход, в WebStorm предусмотрен zen-режим, который убирает все элементы меню и позволяет сосредоточиться на коде, вызывая необходимые функции через горячие клавиши. Также редактор поддерживает работу с несколькими параллельными задачами, позволяя разметить рабочее пространство на несколько зон и использовать дополнительные мониторы для удобства.

MongoDB является системой управления базами данных с открытым исходным кодом, реализующей подход NoSQL, который представляет собой достойную альтернативу традиционным реляционным системам. Такие базы данных, как MongoDB, оптимально подходят для обработки больших масштабов распределённых данных и управления ими. Основной особенностьюMongoDB является её архитектура, которая организована в виде коллекций и документов, где коллекции выполняют роль таблиц в SQL-системах, а документы, содержащие пары «ключ-значение», функционируют как основные единицы данных [21].

MongoDB поддерживает широкий спектр языков программирования, включая C, C++, C#, Go, Java, Python, Ruby и Swift, что делает её доступной для разработчиков в различных технологических стеках. Система предоставляет серверы для создания и управления базами данных, что обеспечивает удобство и эффективность в работе с данной СУБД.

Документы в MongoDB могут хранить разнообразные типы данных, включая текст, массивы и даже другие документы, что делает их структуру гибкой и масштабируемой. Эти документы структурированы по принципу, схожему с JavaScript Object Notation (JSON), однако используют формат Binary JSON (BSON). Преимущество BSON заключается в его расширенных возможностях по вмещению различных типов данных. Отличительной чертой документов в MongoDB является наличие уникального идентификатора в виде первичного ключа, который позволяет эффективно управлять данными.

Пользователи могут манипулировать структурой документов, добавляя или удаляя поля в них. В MongoDB используется подход, основанный на создании и хранении данных в документах, которые являются ключевыми элементами данной базы данных. Каждый документ состоит из различных данных, структурированных в пары «поле-значение». Эти пары схожи с тем, как информация организована в столбцах реляционных баз. Однако MongoDB предпочитает формат Binary JSON вместо обычного JSON, поддерживая таким образом большее разнообразие типов данных. Например, в значениях полей могут присутствовать другие документы, массивы и даже массивы документов [21].

Каждый документ обязательно включает уникальный идентификатор, или первичный ключ, который гарантирует его уникальность в коллекции.

Интерфейс MongoDB предоставляет возможности для изменения структуры документа, позволяя добавлять или удалять поля в существующих документах. Наборы документов в Mongo называются коллекциями, которые функционируют как эквивалент таблиц реляционной базы данных. Коллекции могут содержать любой тип данных, но ограничение заключается в том, что данные в коллекции не могут быть распределены по разным базам данных. Пользователи MongoDB могут создавать несколько баз данных с несколькими коллекциями.

Оболочка Mongo является стандартным компонентом дистрибутивов MongoDB с открытым исходным кодом. После установки MongoDB пользователи подключают оболочку Mongo к своим работающим экземплярам MongoDB. Оболочка mongo действует как интерактивный интерфейс JavaScript для MongoDB, который позволяет пользователям запрашивать или обновлять данные и выполнять административные операции.

Двоичное представление документов, подобных JSON, обеспечивается форматом хранения документов BSON и обмена данными. Автоматическое сегментирование — еще одна ключевая функция, которая позволяет распределять данные в коллекции MongoDB по нескольким системам для

горизонтальной масштабируемости по мере увеличения объемов данных и требований к пропускной способности.

СУБД NoSQL использует единую главную архитектуру для обеспечения согласованности данных с базами данных-получателями, которые поддерживают копии основной базы данных. Операции автоматически реплицируются в эти базы данных-получатели для автоматического перехода на другой ресурс.

Основными критериями выбора данной СУБД послужили следующие преимущества:

- быстрая загрузка информации. Поскольку данные не нужно преобразовывать в таблицы, они помещаются в БД почти в неизменном виде, что гораздо быстрее;
- гибкость и масштабируемость. Если требуется добавить новое поле или тип файла, не нужно пересобирать всю базу заново — его достаточно просто добавить. Это позволяет легко расширять базу и хранить в ней даже файлы, изначально не предусмотренные структурой, что будет удобно в случае необходимости;
- надежность. Благодаря реплицированию данные всегда в большей сохранности, а база сохраняет доступность даже при обновлении или поломках.

1.5 Постановка задачи на разработку

Основные функциональные требования к приложению следующие:

- наличие мобильного приложения, доступного для загрузки потенциальными клиентами;
- возможность заказа блюд через мобильное приложение;
- возможность заказа без регистрации;
- ранжирование мест доставки в зависимости от геолокации пользователей;

- возможность ранжирования блюд по количеству заказов для вывода рекомендаций пользователю;
- наличие администраторской части;
- возможность ведения основных справочников в администраторской части;
- возможность просмотра и добавления блюд в администраторской части;
- возможность добавления ресторанов в администраторской части;
- возможность просмотра заказов в администраторской части.

После описания предметной области, выбора средств разработки, описания аналогичных систем и выбора функций, которые необходимо реализовать в приложении, опишем процесс разработки мобильного приложения.

2 Реализация программного обеспечения

2.1 UML моделирование системы

Проведем моделирование разрабатываемой системы с использованием нотации UML.

Диаграмма вариантов использования (use-case diagram) позволяет понять, какие основные функции будет реализованы в системе, определить круг пользователей, доступные им функции, и зависимости между функциями [2].

На рисунке 14 приведена диаграмма вариантов использования (прецедентов) для разрабатываемого мобильного приложения по заказу еды.

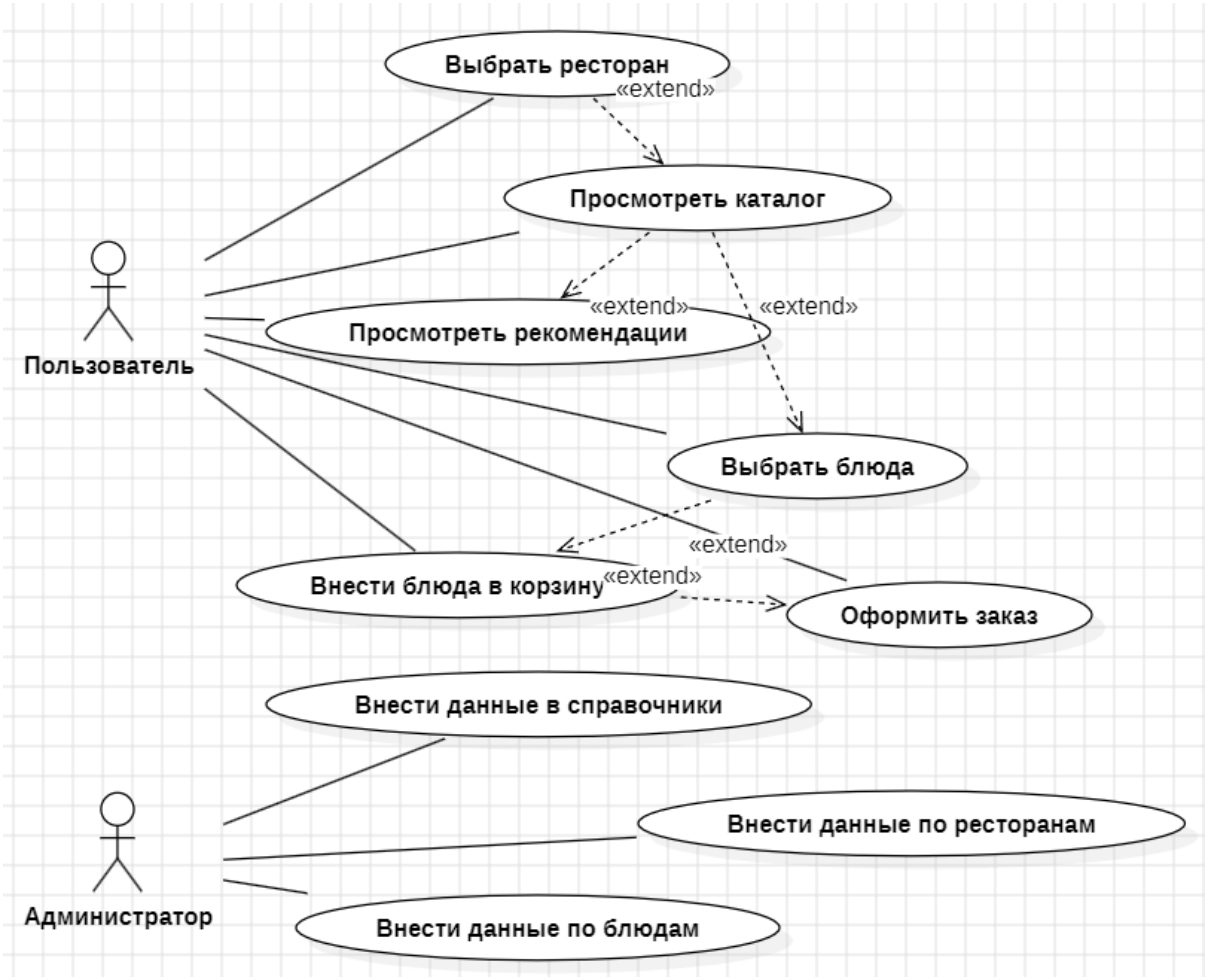

Рисунок 14 – Диаграмма вариантов использования (прецедентов)

Диаграмма включает в себя две роли: пользователь и администратор.

Пользователь, после входа в приложение, просматривает доступные для него ближайшие рестораны и выбирает нужный.

После этого пользователю становится доступен каталог блюд в данном ресторане, из которого он имеет возможность выбрать необходимые ему блюда. Также, при выборе пользователь может ориентироваться на рекомендации, выданные приложением.

На основе своих предпочтений, пользователь формирует корзину из необходимых ему блюд.

После внесения всех необходимых блюд в корзину, пользователь имеет возможность оформить заказ.

Диаграмма последовательности процесса оформления заказа показана на рисунке 15.

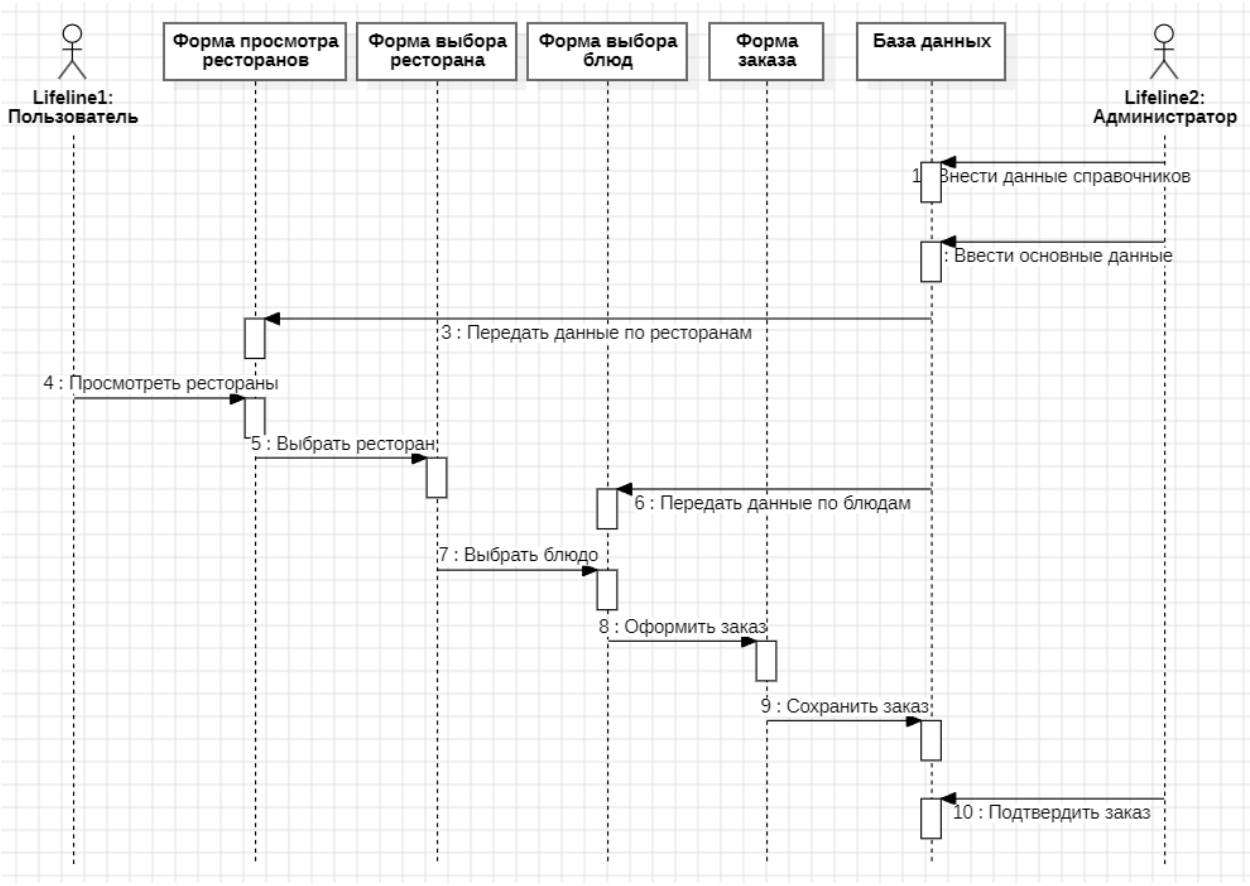

Рисунок 15 – Диаграмма последовательности

Администратор системы (менеджер) вводит данные справочников и основные данные (категории, рестораны, и блюда) в систему.

Пользователю при входе в приложение выводится список ресторанов, переданный из БД.

Пользователь выбирает ресторан.

После этого из БД передаются данные по блюдам, и пользователь имеет возможность выбрать и заказать необходимое блюдо.

Заказ сохраняется в системе, после чего подтверждается менеджером.

После определения основных функций системы и ролей пользователей перейдем к реализации проекта.

2.2 Реализация мобильного приложения

Общая архитектура реализуемого мобильного приложения схематично представлена ниже.

```
app
├── .expo
├── assets
├── node_modules
├── src
│   ├── api
│   ├── app
│   ├── assets
│   ├── components
│   ├── features
│   ├── hooks
│   ├── store
│   ├── types
│   └── utils
├── .env
```

├── .gitignore
├── app.config.ts
├── babel.config.js
├── eas.json
├── expo-env.d.ts
├── package.json
├── tsconfig.json
└── yarn.lock

Кратко опишем состав разрабатываемого приложения.

Папки верхнего уровня:

- .expo и assets: стандартные для проектов с Expo, содержат метаданные и ресурсные файлы;
- node_modules: библиотеки и зависимости Node.js.

Папка src: основная папка с исходным кодом приложения.

- api: код для работы с API, включая запросы и обработку данных;
- app: основные модули приложения;
- assets: ресурсы, такие как изображения и стили;
- components: переиспользуемые компоненты пользовательского интерфейса;
- features: модули, отвечающие за конкретные функциональные возможности;
- hooks: пользовательские хуки React;
- store: управление состоянием приложения, например, с Redux;
- types: TypeScript типы и интерфейсы;
- utils: вспомогательные утилитарные функции.

3. Конфигурационные файлы:

- .env: переменные окружения;
- .gitignore: файлы и папки, которые не будут добавлены в репозиторий;
- app.config.ts: конфигурация Expo (используемого фреймворка);

- babel.config.js: конфигурация Babel;
- eas.json: конфигурация Expo Application Services;
- expo-env.d.ts: определения типов для Expo;
- package.json: описание проекта и его зависимостей;
- tsconfig.json: конфигурация TypeScript;
- yarn.lock: файл блокировки зависимостей для Yarn.

Далее опишем основные ключевые функции разработанного приложения.

В Приложении А представлен листинг части программы, которая реализует алгоритм для проверки, находится ли заданная точка внутри многоугольника на плоскости. Функция использует метод, известный как алгоритм «Ray Casting» или «Ray Crossing».

Данная часть программы работает следующим образом:

Инициализация: Переменная inside инициализируется как false. Она будет использоваться для отслеживания, находится ли точка внутри многоугольника.

Цикл по ребрам многоугольника: Цикл проходит по всем вершинам многоугольника. Для каждой пары соседних вершин (i, j) вычисляются их координаты: (xi, yi) и (xj, yj).

Проверка пересечения: Для каждой пары соседних вершин выполняется проверка на пересечение горизонтального луча, исходящего из точки point, с ребром многоугольника:

- yi > point.latitude !== yj > point.latitude проверяет, пересекает ли ребро горизонтальную линию, проходящую через точку;
- point.longitude < ((xj - xi) * (point.latitude - yi)) / (yj - yi) + xi вычисляет, находится ли точка левее пересечения луча с ребром.

Инверсия состояния: Если обнаружено пересечение (intersect равно true), переменная inside инвертируется (меняется с true на false или наоборот). Это связано с тем, что каждый раз, когда луч пересекает границу многоугольника, он меняет свое положение относительно внутренней области.

Возврат результата: После завершения цикла функция возвращает значение переменной inside, которое будет true, если количество пересечений было нечетным (точка внутри), и false, если четным (точка снаружи).

В Приложении Б представлен листинг части программы, которая отвечает за поиск ближайшего магазина из списка потенциальных местоположений, а также ближайшего магазина, который находится в зоне доставки пользователя.

Код содержит следующие переменные:

- nearestCity, nearestStore, nearestStoreInZone — переменные для хранения ближайшего города, ближайшего магазина и ближайшего магазина в зоне доставки соответственно. Изначально они инициализированы как null;
- minCityDistance, minStoreDistance, minStoreInZoneDistance — переменные для хранения минимальных расстояний до города, магазина и магазина в зоне доставки соответственно. Изначально установлены в Infinity, чтобы любое реальное расстояние было меньше.

Цикл forEach проходит по массиву distances, содержащему расстояния до каждого из потенциальных местоположений, представленных в массиве candidateLocations.

Далее происходит определение типа местоположения, а именно.

Проверяется наличие в объекте location присутствует свойство deliveryZone, то это магазин (CityStore).

Проверяется, находится ли пользователь в зоне доставки магазина с помощью функции isPointInPolygon.

Если текущее расстояние меньше минимального расстояния до магазина (minStoreDistance), обновляются переменные minStoreDistance и nearestStore.

Если пользователь находится в зоне доставки и текущее расстояние меньше минимального расстояния до магазина в зоне

(minStoreInZoneDistance), обновляются переменные minStoreInZoneDistance и nearestStoreInZone.

Если свойство deliveryZone отсутствует, то это город (City).

Если текущее расстояние меньше минимального расстояния до города (minCityDistance), обновляются переменные minCityDistance и nearestCity.

Таким образом, в результате выполнения процедуры поисска ближайшего города из списка, мы получаем три результата:

- ближайший город из списка;
- ближайший магазин из списка;
- ближайший магазин, который находится в зоне доставки пользователя.

2.3 Реализация личного кабинета администратора

Ниже схематично показана структура администраторской части разработанного приложения.

```
src
│
├── app (payload)
│   └── admin
│       ├── [[...segments]]
│       └── importMap.js
├── api
│   ├── [...slug]
│   └── graphql
│       ├── graphql-playground
│       └── route.ts
│   └── route.ts
├── custom.scss
```

```
├── layout.tsx
├── collections
│   ├── Categories.ts
│   ├── Cities.ts
│   ├── Media.ts
│   ├── Orders.ts
│   ├── Products.ts
│   ├── Stores.ts
│   └── Users.ts
├── utils
├── payload-types.ts
├── payload.config.ts
├── .env.example
├── .eslintrc.cjs
├── .gitignore
├── .prettierrc.json
├── .yarnrc
└── docker-compose.yml
```

Структура проекта включает в себя следующие основные компоненты.

Каталог admin содержит административные страницы и компоненты, связанные с панелью управления контентом.

Каталог ...segments предназначен для динамических маршрутов и сегментов URL.

Каталог api содержит интерфейс взаимодействия с приложением, а именно:

- подкаталог ...slug – динамическая обработка API-запросов;
- каталог graphql – папка GraphQL API;
- файл route.ts определяет маршруты для GraphQL;
- каталог graphql-playground – инструмент для тестирования и отладки GraphQL;

- файл route.ts - сновной файл маршрутов API.

Стиль и макеты

- файл custom.scss: пользовательские стили проекта;
- файл layout.tsx: компонент для оформления макета приложения;

Каталог collections предназначен для схем данных в CMS, содержит в себе файлы Categories.ts, Cities.ts, Media.ts, Orders.ts, Products.ts, Stores.ts, Users.ts, которые определяют коллекции и их типы.

Каталог utils: утилитарные функции и вспомогательные скрипты.

Общие онфигурационные файлы

- файл payload-types.ts: Типизации для данных Payload CMS;
- файл payload.config.ts: Основной конфигурационный файл для Payload.

За конфигурацию среды и инструментов отвечают следующие файлы:

- eslintrc.cjs: конфигурация для ESLint (анализатор кода);
- .gitignore: исключения для контроля версий Git;
- prettierrc.json: конфигурация для Prettier (форматирование кода);
- .yarnrc: конфигурации для Yarn (менеджер пакетов).

docker-compose.yml: Конфигурация для запуска проекта в контейнерах Docker.

В Приложении В представлен листинг асинхронной функции, которая обрабатывает данные заказа и выполняет следующие операции.

Инициализация.

Функция принимает объект с двумя свойствами: data и operation, data предполагается объектом, представляющим заказ, и приводится к типу Order.

Хук для добавления дополнительных неизменяемых полей в заказ, с подсчетом общей суммы.

В первую очередь происходит ввод аргументов: функция принимает объект с параметрами data и operation. data представляет собой данные, которые подлежат изменению, а operation указывает тип операции (например, создание или обновление).

После этого происходит копирование данных: Создается копия объекта data, который приводится к типу Order и сохраняется в переменной order.

Затем происходит получение полезной нагрузки: Вызывается асинхронная функция getPayload, которая принимает объект конфигурации config. Результат выполнения этой функции сохраняется в переменной payload.

После этого происходит создание нового объекта заказа: Создается копия объекта order, которая сохраняется в переменной nextOrder.

И наконец происходит генерация уникального номера заказа: Если операция – это создание (operation === «create»), то создается экземпляр ShortUniqueId с использованием словаря символов, состоящего из заглавных букв и цифр («alphanum_upper»). Этот объект используется для генерации уникального идентификатора, который затем присваивается свойству orderNumber в объекте nextOrder.

Копирование заказа (создается копия заказа order в переменной nextOrder).

Обработка операции «create».

Если операция — «create», генерируется уникальный идентификатор для заказа с использованием библиотеки ShortUniqueId.

Этот идентификатор используется для присвоения номера заказа (orderNumber) новому заказу, который сохраняется в nextOrder.

С помощью метода payload.find производится поиск продуктов в коллекции «products»..

Вычисление общей стоимости заказа. На этом шаге выполняется итерация по элементам заказа (order.items) для вычисления общей стоимости, после чего для каждого элемента находится соответствующий продукт из полученных данных и его цена.

Общая стоимость вычисляется как сумма произведений цены продукта на количество. Если цена или количество не определены, они принимаются равными нулю.

3 Тестирование ИС

3.1 Описание процесса тестирования

Исследование качества программного продукта осуществляется через тестирование программного обеспечения. В рамках этой деятельности специалисты стремятся установить соответствие продукта заявленным требованиям. Для верификации кода и определения его соответствия используются различные методики [13].

Чтобы оценить качество программного обеспечения, принято учитывать ряд ключевых характеристик: практичность, производительность, надежность, функциональность, доступность, мобильность и удобство обслуживания. Объединение этих элементов позволяет получить полноценное представление о качестве анализируемого продукта.

Для оценки разработанных систем используют различные методы тестирования. Модульное тестирование, например, включает проверку отдельных наименьших тестируемых единиц, таких как функция или класс, которое проводится, чтобы удостовериться в корректности работы отдельных частей программы. Разработчики прибегают к нему довольно часто. После внесения обновлений в программный продукт следует регрессионное тестирование, которое задачей имеет подтверждение стабильности функций приложения, несмотря на изменения. Цель регрессионных тестов — обеспечение непрерывной работоспособности системы [6].

Интеграционное тестирование предусматривает анализ взаимодействий между разными компонентами и подсистемами, а при достаточном времени процесс может быть итерационным, что предполагает постепенное включение новых подсистем в процесс тестирования. Системное тестирование же направлено на подтверждение соответствия интегрированной системы всем установленным требованиям.

Альфа-тестирование применяется на ранних стадиях создания программного обеспечения и включает в себя проверку как разработчиками, так и потенциальными пользователями. Его задачей является имитация использования продукта или его реальное применение для обнаружения ошибок и недочетов, а также иногда это тестирование может проводиться в форме внутреннего приемочного теста для уже завершенных приложений [14].

Напротив, бета-тестирование часто проводится с версией продукта, у которой ограничены некоторые функции или время работы. Этот этап тестирования предоставляется небольшой выборке пользователей с целью подтверждения минимального количества ошибок и сбора обратной связи от потенциальных клиентов, что касается улучшений функциональности и возможностей программы [10]..

При разработке программного обеспечения особое внимание уделяется альфа и бета тестированию, так как они являются неотъемлемой частью проверки всех модулей и системы в целом. Процесс включает в себя работу команды специалистов по контролю качества, которые используют специально разработанные сценарии пользователя. Этот метод доказал свою эффективность и является предпочтительным при работе с комплексными программными решениями

Прохождение альфа и бета тестирования обеспечивает не только проверку соответствия системы установленным бизнес-правилам, но и возможность разработки новых функций на основе отзывов участников тестов для будущих обновлений проекта.

3.2 Представление и тестирование работы приложения

Панель администратора реализована в системе в виде веб-приложения.

После входа в личный кабинет, пользователь видит следующий дашбоард, в котором перечислены все справочники и документы, имеющиеся в системе (рисунок 16).

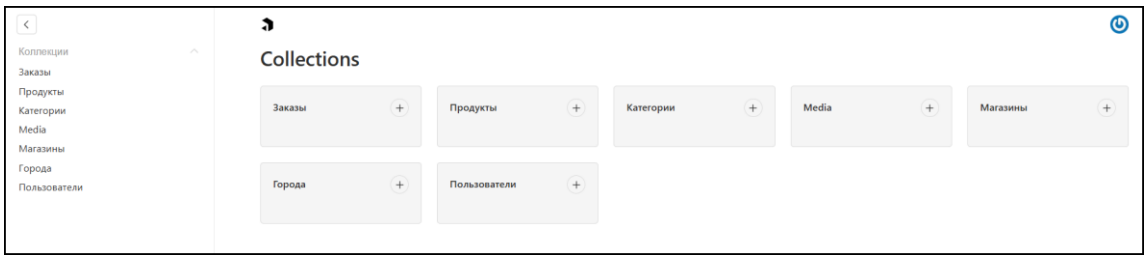

Рисунок 16 – Главная страница веб-приложения

В системе реализован функционал добавления блюд, с указанием ресторанов/магазинов в которых они доступны.

Администратор имеет возможность просматривать список имеющихся продуктов (рисунок17).

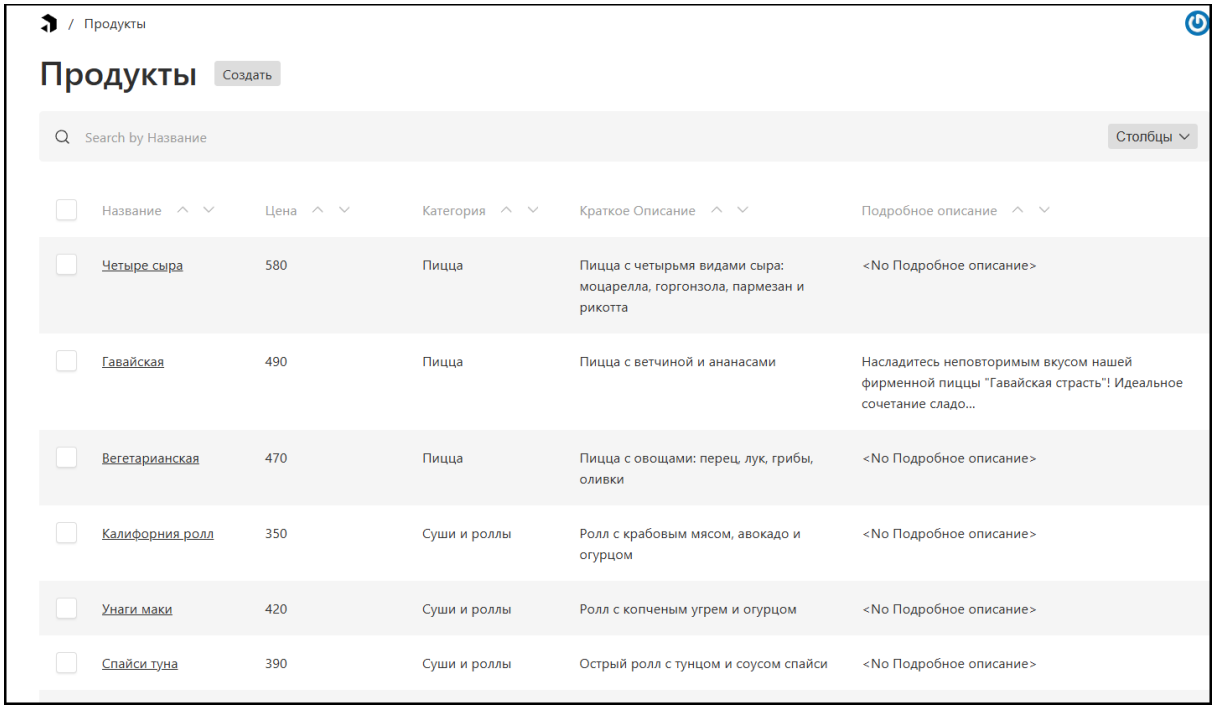

Рисунок 17 – Список блюд в наличии

При нажатии на позицию – можно посмотреть более подробную информацию о ней, либо отредактировать (рисунок 18).

Рисунок 18 – Просмотр/редактирование блюда

Для редактирования доступны:

- Название блюда.
- Краткое описание.
- Цена.
- Категория.
- Магазины, в которых доступно это блюдо.
- Изображение блюда.

В информационной системе реализована возможность добавления магазинов. При переходе на вкладку «Магазины» отображается список всех магазинов, доступных в приложении (рисунок 19).

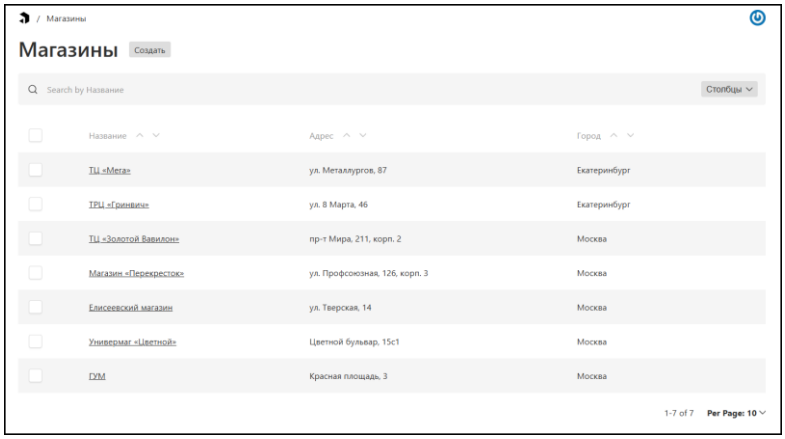

Рисунок 19 – Список магазинов

Можно добавить новый магазин, либо отредактировать уже внесённый в систему (рисунок 20).

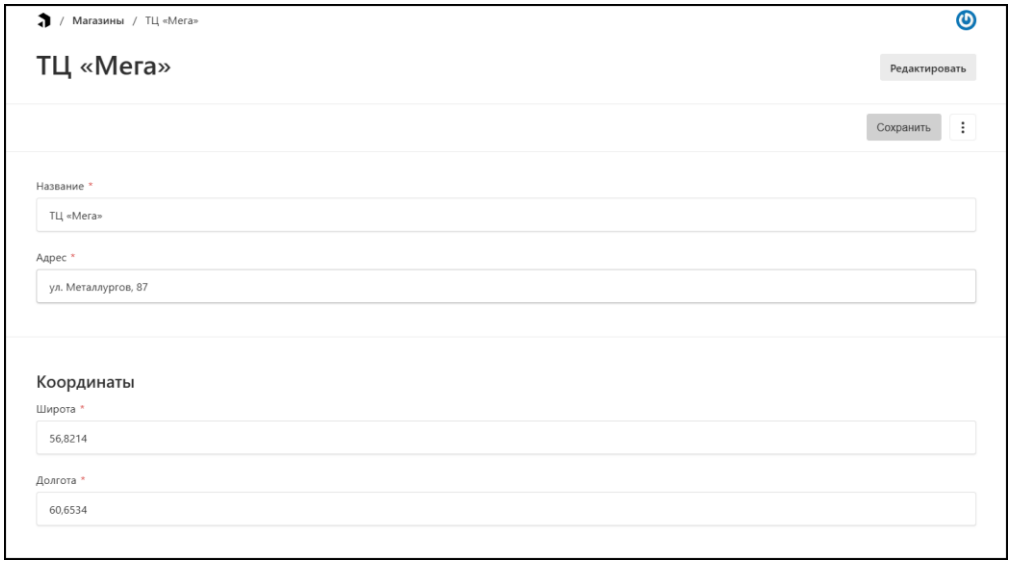

Рисунок 20 – Просмотр/редактирование магазина

Для каждого магазина указывается название, адрес, город, в котором он расположен, географические координаты, а также координаты зоны доставки.

Все заказы, поступающие от пользователей, отображаются в системе на вкладке «Заказы» (рисунок 21).

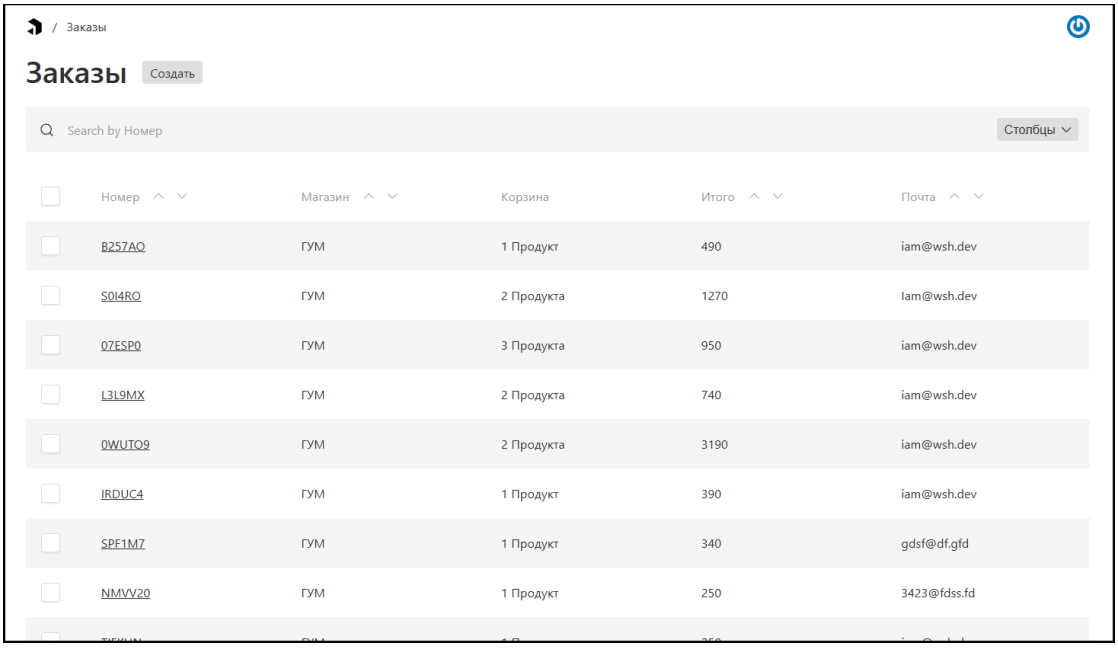

Рисунок 21 – Список заказов

Просмотр заказа осуществляется нажатием на номер заказа, интерфейс форму показан на рисунке 22.

Для каждого заказа отображаются:

- Магазин, в котором произошел заказ;
- Содержимое заказа с указанием каждого блюда, и их общей стоимости;
- Электронная почта клиента.
- Статус заказа.
- Адрес доставки.
- Номер телефона клиента.
- Способ оплаты.
- Статус оплаты.
- Комментарий клиента.
- Комментарий администратора.

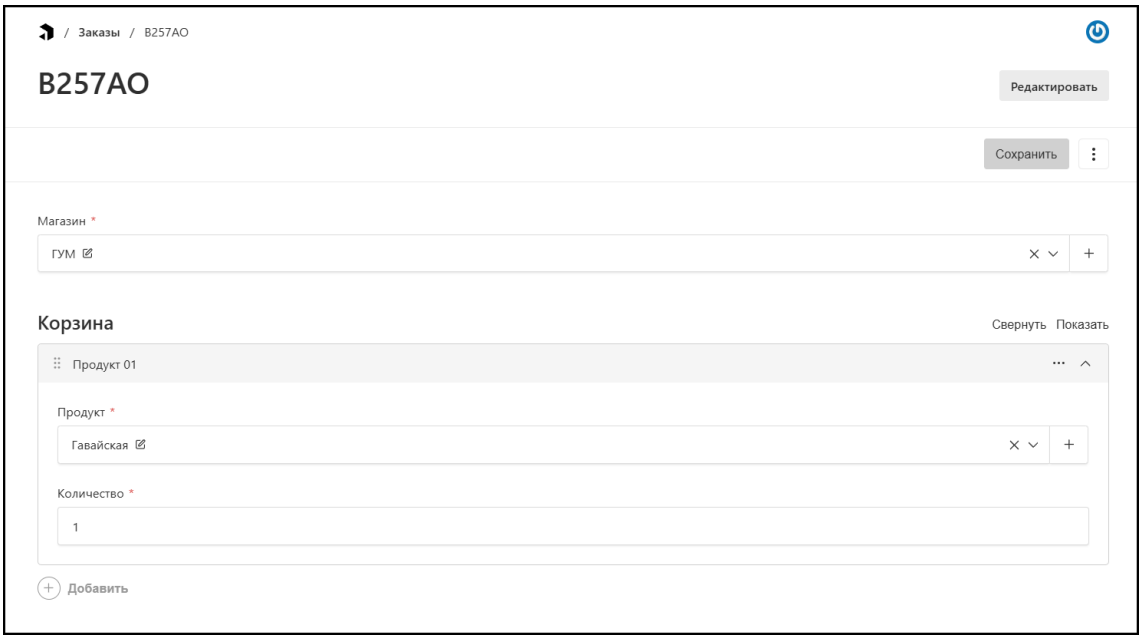

Рисунок 22 – Просмотр/редактирование заказа

Также, реализована возможность ведения справочника городов (рисунок 23).

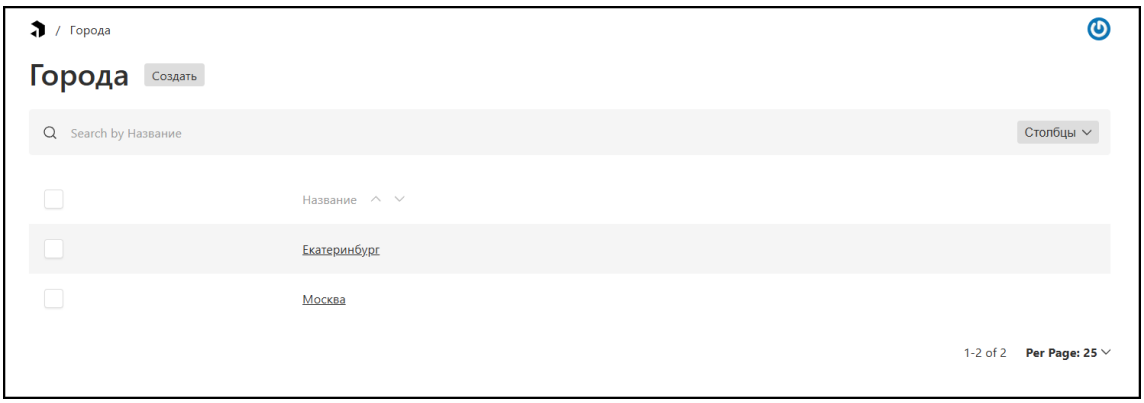

Рисунок 23 – Список городов

Форма редактирования города показана на рисунке 24.

Рисунок 24 – Форма просмотра/редактирования города

Для каждого города в систему заносится название и его координаты.

Все функции личного кабинета администратора работают корректно, можно перейти к тестированию мобильного приложения.

После запуска приложения пользователем, на экране отображается список магазинов на карте, при этом сразу указывается, какой магазин является ближайшим (рисунок 25).

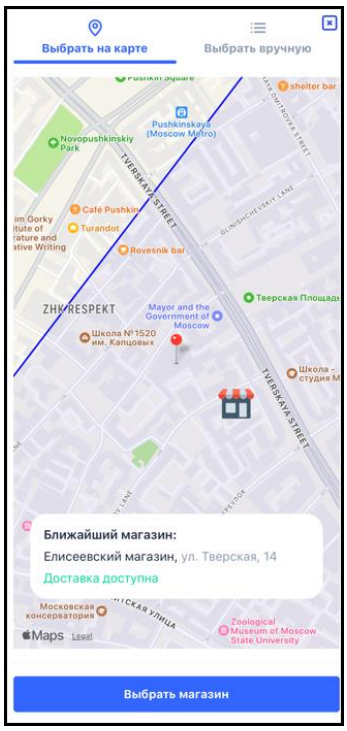

Рисунок 25 – Начальный экран приложения

Пользователь имеет возможность выбрать необходимый магазин/ресторан на карте, либо перейти на вкладку «Выбрать вручную», и выбрать необходимый ресторан из списка (рисунок 26).

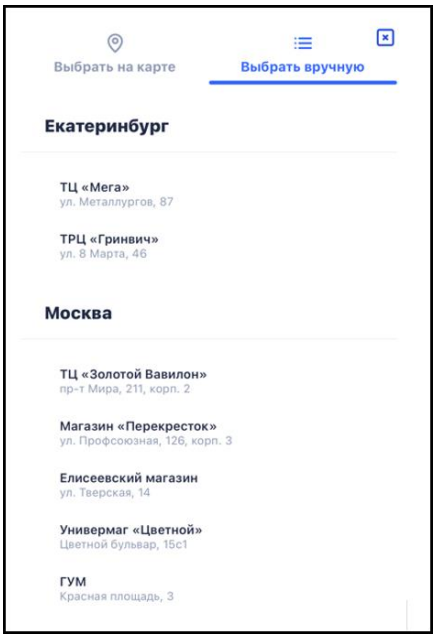

Рисунок 26 – Выбор магазинов/ресторанов из списка

После выбора ресторана/магазина на экране отображается список блюд, при этом наиболее популярные блюда в этом ресторане отображаются сверху (рисунок 27).

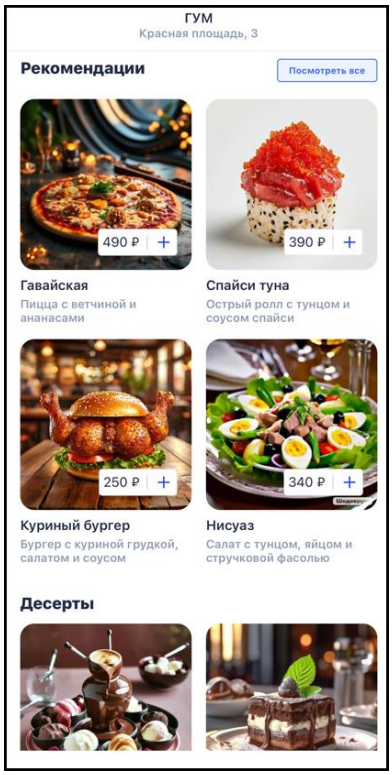

Рисунок 27 – Список блюд с рекомендациями

После того, как хотя бы одно блюдо добавлено в корзину – в нижней правой части экрана отображается кнопка для перехода к корзине (рисунок 28).

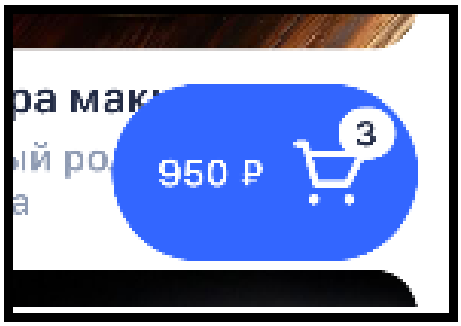

Рисунок 28 – Кнопка перехода в корзину

Внесенные в корзину блюда отображаются на экране, при этом пользователь может при необходимости изменить состав заказа (рисунок 29).

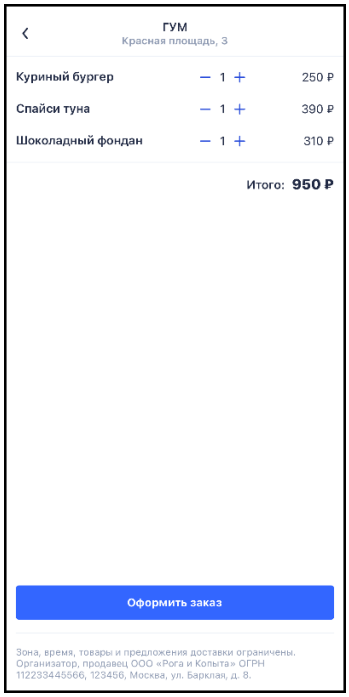

Рисунок 29 – Форма корзины

Форма оформления заказа показана на рисунке 30.

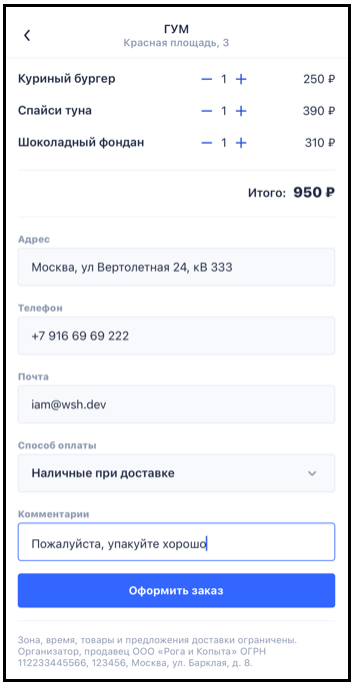

Рисунок 30 – Форма оформления заказа

Для оформления заказа в этой форме необходимо указать адрес, телефон, электронную почту, и способ оплаты. Все поля в данной форме являются обязательными.

После оформления заказа, на электронную почту пользователя высылается уведомление об этом (рисунок 31).

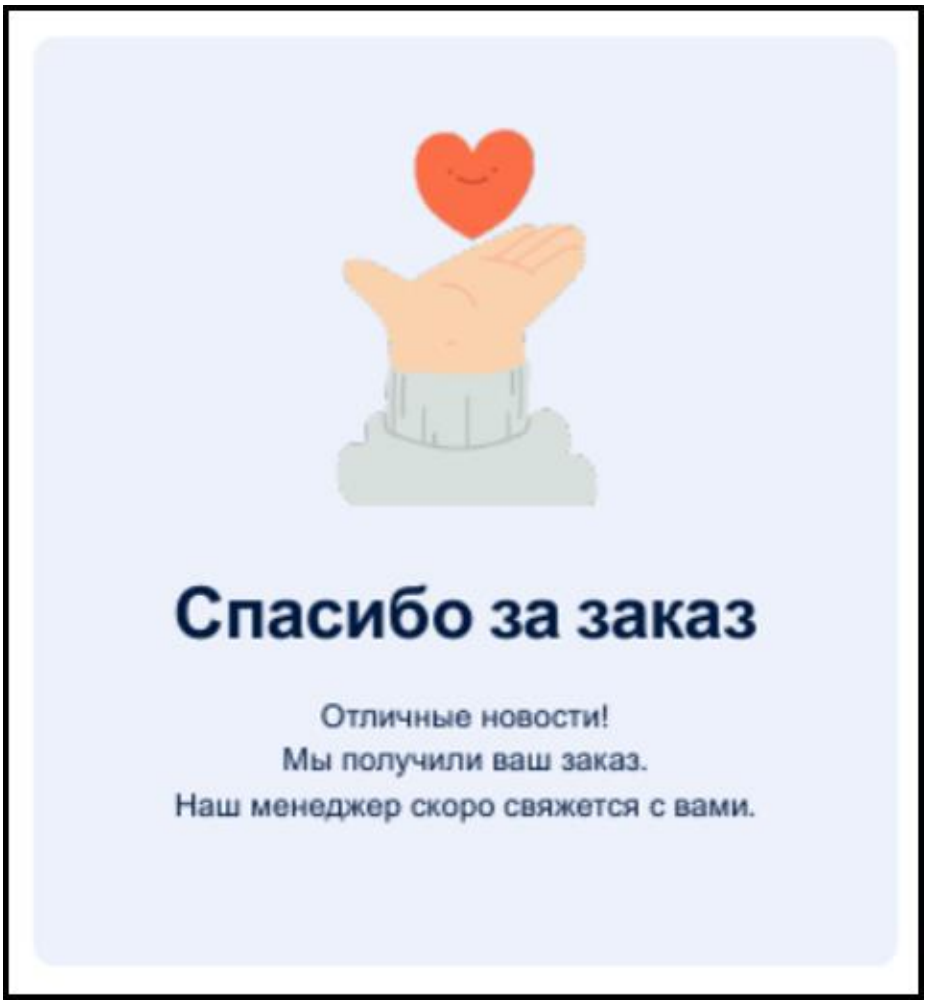

Рисунок 31 – Сообщение о формировании заказа

После подтверждения заказа менеджером, на электронную почту пользователя высылается уведомление с указанием состава и стоимости заказа.

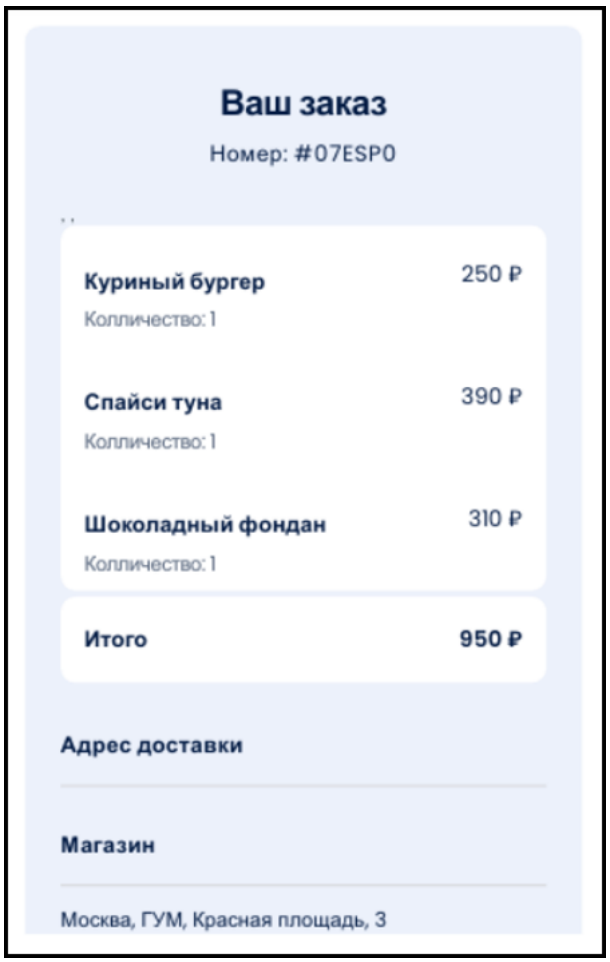

Рисунок 32 – Уведомление о заказе

В процессе написания данной главы были протестированы все основные функции личного кабинета администратора, и работа мобильного приложения. Существенных ошибок выявлено не было, все протестированные функции приложения работают корректно.

Заключение

В процессе разработки проекта мобильного приложения для заказа еды с использованием геолокации и системы рекомендаций были получены следующие результаты.

Было проведено исследование и описание предметной области, и требований к разрабатываемому приложению. Была дана общая характеристика мобильных приложений, проведена их классификация, и описание основных сфер применения.

Далее было проведено моделирование бизнес-процесса функционирования мобильного приложения для заказа еды, выявлены основные операции и участники процесса.

После этого был проведен обзор нескольких аналогичных систем, описаны их достоинства и недостатки, а также проведено сравнение с разрабатываемой информационной системой.

Далее проведен выбор средств проектирования и разработки. В качестве фреймворка для разработки был выбран Expo, в качестве СУБД – MongoDB, в качестве интегрированной среды разработки – WebStorm Jetbrains. Выполнена постановка задачи на разработку системы.

После этого было проведено UML моделирование разрабатываемой ИС, выделены основные роли пользователей, функции, связанные с ними, и описан процесс разработки мобильного приложения.

В заключительной части проведено описание процесса тестирования разработанного приложения.

В сфере разработки мобильных приложений наблюдается очень стремительный прогресс. Важность мобильных решений возрастает во всех секторах экономики – от розничной продажи и телекоммуникаций до страхования, здравоохранения и управления на государственном уровне. Современные мобильные устройства и соответствующие приложения, помогающие раскрыть их потенциал, являются важными как для отдельных

лиц, так и для компаний. Для поддержания своей конкурентоспособности, оперативности и успешности, организациям требуется разрабатывать приложения, отвечающие потребностям их клиентов, партнеров и сотрудников.

Разработанное приложение предоставляет пользователям возможность заказа блюд в различных ресторанах, с использованием геолокации и сервиса рекомендаций, а администраторам магазинов – размещать в системе реализуемые блюда, добавлять заведения, принимать и выполнять заказы пользователей.

Таким образом, задачи, поставленные в начале работы, можно считать выполненными.

Список используемых источников

1. Амблер С. Гибкие технологии: экстремальное программирование и унифицированный процесс разработки. Библиотека программиста [Текст]: - Спб.: Питер, 2015. - 412 с.
2. Буч Г., Якобсон А., Рамбо Дж. UML. Классика CS. 2-е изд. / Пер. с англ.; Под общей редакцией проф. С. Орлова — СПб.: Питер, 2006. — 736 с.
3. Буч, Г. UML: Руководство пользователя. / Г. Буч, Джекобсон И. и др. - М.: ДМК, 2008 г. – 356 с.
4. Варзунов А. В., Торосян Е. К., Сажнева Л. П., Анализ и управление бизнес-процессами // Учебное пособие. – СПб: Университет ИТМО, 2016. –112 с.
5. Введение в реальный ITSM / Роб Ингланд; Пер. с англ. – М.: Лайвбук, 2010. – 132 с.
6. Вигерс К., Битти Д. Разработка требований к программному обеспечению. 3-е изд., дополненное / Пер. с англ. — М. : Издательство «Русская редакция» ; СПб. : БХВ-Петербург, 2014. – 452 с.
7. Дональд Н. - Дизайн привычных вещей - 2020, 4-е изд. – 512 с.
8. Ларман К. Применение UML 2.0 и шаблонов проектирования. Практическое руководство. 3-е издание. [Текст]: Пер с англ. - М.: ООО «И.Д. Вильямс», 2009. - 736 с.: ил. - Парал. тит. англ.
9. Леоненков А. В. Самоучитель UML 2. - БХВ-Петербург, 2007 — 576 стр. - 3000 экз. - ISBN 978-5-94157-878-8.
10. Мартин Р. Чистая архитектура. Искусство разработки программного обеспечения.: Пер. с англ. — СПб: Питер, 2019 — 410 стр. — ISBN: 978-5-4461-0772-8.
11. Мартин Р. Чистый код: создание, анализ и рефакторинг. : Пер. с англ. — СПб: Питер, 2018 — 464 стр. — ISBN: 978-5-496-00487-9.
12. Моделирование бизнес-процессов: учебник и практикум для академического бакалавриата / О. И. Долганова, Е. В. Виноградова, А. М.

Лобанова ; под ред. О. И. Долгановой. — М. : Издательство Юрайт, 2016. — 289 с. — Серия : Бакалавр.

13. Орлов С. А. Технологии разработки программного обеспечения: Учебник. — СПб: Питер, 2002. — 464 с. — ISBN 5-94723145-X.
14. Орлов, С. А. Теория и практика языков программирования. — СПб: Питер, 2017 — 688 стр. — ISBN: 978-5-4461-0491-8.
15. Розенберг Д., Скотт К. Применение объектного моделирования с использованием UML и анализа прецедентов [Текст]: Пер. с англ. - М.: ДМК Пресс, 2002. - 160 с.: ил. (Серия «Объектно-ориентированные технологии программирования»)
16. Фаулер, М. UML в кратком изложении. / М. Фаулер. - М.: Мир, 2019 г. – 204 с.
17. Использование диаграмм вариантов использования UML при проектировании программного обеспечения. [Электронный ресурс]. https://habr.com/ru/articles/566218/, дата обращения 15.10.2024
18. 10 принципов разработки мобильных интерфейсов. - CMS magazine [Электронный ресурс] - URL: https://cmsmagazine.ru/journal/items-10-principles- mobile-interface-design/, дата обращения: 10.10.2024
19. Mobile App Lifecycle. [Электронный источник]. Режим доступа: https://buildfire.com/understanding-mobile-app-development-lifecycle/, дата обращения: 12.10.2024
20. Mobile Application Development. [Электронный источник]. Режим доступа: https://www.ibm.com/cloud/learn/mobile-application-development-explained, дата обращения: 12.10.2024
21. What is MongoDB?. [Электронный ресурс]. Режим доступа: https://www.techtarget.com/searchdatamanagement/definition/MongoDB, дата обращения: 12.10.2024

Приложение А

Реализация Алгорима «Ray Casting»

```
/**
 * Проверяет, находится ли точка внутри многоугольника.
 * @param {Coordinates} point - Координаты проверяемой точки.
 * @param {Coordinates[]} polygon - Массив координат, определяющих многоугольник.
 * @returns {boolean} true, если точка находится внутри многоугольника, иначе false.
 */
function isPointInPolygon(point: Coordinates, polygon: Coordinates[]): boolean {
  let inside = false;
  for (let i = 0, j = polygon.length - 1; i < polygon.length; j = i++) {
    const xi = polygon[i].longitude,
      yi = polygon[i].latitude;
    const xj = polygon[j].longitude,
      yj = polygon[j].latitude;
    const intersect =
      yi > point.latitude !== yj > point.latitude &&
      point.longitude < ((xj - xi) * (point.latitude - yi)) / (yj - yi) + xi;
    if (intersect) inside = !inside;
  }
  return inside;
}
```

Приложение Б

Функция поиска ближайших магазинов на карте

```
let nearestCity: City | null = null;
  let nearestStore: CityStore | null = null;
  let nearestStoreInZone: CityStore | null = null;
  let minCityDistance = Infinity;
  let minStoreDistance = Infinity;
  let minStoreInZoneDistance = Infinity;
  distances.forEach((distance, index) => {
    const location = candidateLocations[index];
    if («deliveryZone» in location) {
      // Это магазин
      const store = location as CityStore;
      const isInZone = isPointInPolygon(
        userLocation,
        store.deliveryZone.polygon,
      );
      if (distance < minStoreDistance) {
        minStoreDistance = distance;
        nearestStore = store;
      }
      if (isInZone && distance < minStoreInZoneDistance) {
        minStoreInZoneDistance = distance;
        nearestStoreInZone = store;
      }
    } else {
      // Это город
      if (distance < minCityDistance) {
        minCityDistance = distance;
        nearestCity = location as City;
      }
    }
```

Приложение В

Функция обработки заказов

```
async ({ data, operation }) => {
    const order = { ...data } as Order
    const payload = await getPayload({ config })
    let nextOrder = { ...order }
    if (operation === «create») {
      const uid = new ShortUniqueId({
        dictionary: «alphanum_upper»,
      })
      nextOrder = {
        ...order,
        orderNumber: uid.rnd(),
      }
    }
    const products = await payload.find({
      collection: «products»,
      where: {
        id: {
          in: order?.items?.map((item) => item.product).join(«,»),
        },
      },
      limit: 100,
    })
    return {
      ...nextOrder,
      total: order?.items?.reduce((acc, item) => {
        const price = products?.docs.find((doc) => doc.id === item.product)?.price  0
        const quantity = item?.quantity  0
        acc += price * quantity
        return acc
      }, 0),
    }
  },
```