

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Волгоградский государственный технический университет»

Факультет Электроники и вычислительной техники

Кафедра Программное обеспечение автоматизированных систем

Согласовано

Утверждаю
Зав. кафедрой

(должность гл. специалиста предприятия)

Ю. А. Орлова

(подпись)

(инициалы, фамилия)

(подпись)

(инициалы, фамилия)

« ____ » _____ 20 ____ г.

« ____ » _____ 20 ____ г.

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к выпускной квалификационной работе бакалавра на тему
(наименование вида работы)

Разработка мобильного приложения для заказа и доставки еды

Автор _____
(подпись и дата подписания) Шеху Абубакар Умар
(фамилия, имя, отчество)

Обозначение ВКРБ–09.03.04–10.19–16–23
(код документа)

Группа ПрИн-467
(шифр группы)

Направление 09.03.04 – Программная инженерия,
Разработка программно-информационных систем
(код и наименование направления, наименование программы (профиля))

Руководитель работы _____
(подпись и дата подписания) Гилка В.В.
(инициалы и фамилия)

Консультанты по разделам:

(краткое наименование раздела) _____
(подпись и дата подписания) _____
(инициалы и фамилия)

(краткое наименование раздела) _____
(подпись и дата подписания) _____
(инициалы и фамилия)

Нормоконтролер: _____
(подпись и дата подписания) Кузнецова А.С.
(инициалы и фамилия)

Волгоград 2023 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Волгоградский государственный технический университет»

Кафедра Программное обеспечение автоматизированных систем

Утверждаю

Зав. кафедрой

Ю. А. Орлова

(подпись)

(инициалы, фамилия)

« » 20 г.

ЗАДАНИЕ

на выпускную квалификационную работу бакалавра

(наименование вида работы)

Студент Шеху Абубакар Умар

(фамилия, имя, отчество)

Код кафедры 10.19 Группа ПрИн-467

Тема Разработка мобильного приложения для заказа и доставки еды

Утверждена приказом по университету «06» сентября 20 22 г. № 1235-ст

Срок представления готовой работы (проекта)

(дата, подпись студента)

Исходные данные для выполнения работы (проекта)

Задание, выданное научным руководителем кафедры «ПОАС»

Содержание основной части пояснительной записки

Введение

Цель работы и задачи исследования

1 Аналитическая часть

2 Выбор средств и методов для создания приложения

Выводы к второй главе

3 Проектирование мобильного приложения

Выводы к третьей главе

4 Тестирование мобильного приложения

Выводы к четвертой главе

Список использованных источников

Приложение А — Справка о результатах проверки выпускной
квалификационной работы на наличие заимствований

Приложение Б — Техническое задание

Приложение В — Руководство системного программиста

Перечень графического материала

- 1) _____
 - 2) _____
 - 3) _____
 - 4) _____
 - 5) _____
 - 6) _____
 - 7) _____
 - 8) _____
 - 9) _____
 - 10) _____
 - 11) _____
 - 12) _____
-

Руководитель работы (проекта)

(подпись и дата подписания)

Гилка В.В.

(инициалы и фамилия)

Консультанты по разделам:

(краткое наименование раздела)

(подпись и дата подписания)

(инициалы и фамилия)

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Волгоградский государственный технический университет»

Кафедра «Программное обеспечение автоматизированных систем»

УТВЕРЖДАЮ:

Зав. кафедрой ПОАС

_____ Ю.А. Орлова

«____» _____ 2023 г.

Разработка мобильного приложения для заказа и доставки еды

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

ВКРБ–09.03.04–10.19–16–23–81

Листов 73

Руководитель работы

_____ Гилка В.В.

«____» _____ 2023 г.

Нормоконтролер

Исполнитель

студент группы ПрИн-467

_____ Кузнецова А.С.

_____ Шеху А.У.

«____» _____ 2023 г.

«____» _____ 2023 г.

Волгоград, 2023 г.

Аннотация

Документ представляет собой пояснительную записку к дипломной работе бакалавра на тему: «Разработка мобильного приложения для заказа и доставки еды».

В первом разделе проводится исследование выбранной предметной области. Проведен анализ существующих аналогов.

Во втором разделе рассматриваются основные технологии и инструменты, используемые при разработке мобильных приложений.

В третьем разделе рассматривается реализация программы: структура программы, реализация базы данных и пользовательского интерфейса.

В четвертом разделе анализируются применяемые методы тестирования.

Документ включает в себя страниц - 74, рисунков - 25, приложений -3.

Ключевые слова: React Native, Firebase, Firestore, NoSQL, Food delivery, UI, Expo CLI

Annotation

This document is an explanatory note to the bachelor's thesis on the topic "Development of a Mobile Application for Food Ordering and Delivery."

In the first section, a study of the selected subject area is carried out. The analysis of existing analogues is carried out.

The second section discusses the main technologies and tools used in the development of mobile applications.

The third section discusses the implementation of the program: program structure, database implementation and user interface.

The fourth section analyzes the testing methods used.

The document consists of 74 pages, including 23 figures and 3 appendices.

Keywords: React Native, Firebase, Firestore, NoSQL, Food delivery, UI, Expo CLI.

Содержание

Введение	9
1 Аналитическая часть	11
1.1 Анализ предметной области	11
1.2 Обоснование выбора платформы	12
1.3 Назначение мобильного приложения	14
1.4 Обзор аналогов разработки	16
1.4.1 Delivery Club	16
1.4.2 Яндекс.Еды	18
2 Выбор средств и методов для создания приложения	20
2.1 React Native	20
2.2 Firebase Cloud Firestore	24
2.3 Expo CLI	27
2.4 Visual Studio Code	30
Выводы	34
3 Проектирование мобильного приложения	34
3.1 Проектирование приложения	34
3.1.1 Структура приложения	35
3.1.2 Определение основных функций и возможностей	36
3.1.3 Разработка информационной архитектуры	37
3.2 Реализация базы данных	38
3.2.1 Определение структуры базы данных	38
3.2.2 Создание коллекций и документов	40
3.3 Формализация функциональных требований к приложению	43
3.4 Разработка серверного программного обеспечения	44
3.4.1 Разработка модуля авторизации и аутентификации	46
3.4.2 Реализация функции поиска и просмотра ресторанов и меню	48
3.5 Разработка пользовательского интерфейса	51
3.5.1 Разработка дизайна и UI-элементов	51
3.5.2 Создание макетов экранов	54
3.5.3 Архитектура программы средства	57
Выводы	58

4 Тестирование мобильного приложения	58
4.1 Пользовательское тестирование	58
4.2 А/В тестирование	60
4.3 Опросы и анкеты	63
Выводы	65
Заключение	66
Список использованных источников	68
Приложение А – Справка о результатах проверки выпускной квалификационной работы на наличие заимствований	71
Приложение Б – Техническое задание	72
Приложение В – Руководство системного программиста	73

Введение

С повсеместной доступностью мобильного интернета мобильные приложения приобрели значительную популярность, особенно для заказа товаров и услуг через смартфоны. Мобильные приложения предлагают удобный и доступный способ взаимодействия с различными сервисами, позволяя пользователям размещать заказы в удобное для них время, независимо от того, находятся ли они на работе, в школе или во время неспешных прогулок.

Традиционно для заказа и доставки еды клиенты звонили в предпочитаемое ими заведение и ждали курьера, который доставит их заказ по указанному адресу. Однако современные технологии предоставляют безграничные возможности, и предприятия должны адаптироваться к новым тенденциям, интегрируя цифровые методы и быстро реагируя на ожидания клиентов. Сегодня люди, привыкшие к онлайн-покупкам, ожидают аналогичных ощущений при заказе еды. Ключевыми аспектами, которые делают онлайн-заказы привлекательными, являются удобство, быстрая доставка и разумные цены.

Заведения, предлагающие услуги на вынос, должны использовать технологические достижения и внедрять мобильные приложения, чтобы сделать свои услуги легкодоступными и удержать клиентов. Заказ еды через мобильное приложение обеспечивает быструю доставку без необходимости активного взаимодействия с человеком.

Спрос на доставку продуктов питания, особенно готовых к употреблению блюд, значительно вырос во время первой волны пандемии. Пандемия COVID-19 расширила использование мобильных приложений, что привело к увеличению оборота, увеличению объемов транзакций и увеличению средней стоимости заказов в секторе доставки продуктов питания. В течение нескольких месяцев рынок развивался таким образом, на

что обычно уходило не менее года. Даже в летние месяцы, когда люди обычно обедали вне дома, объем заказов продолжал расти, поскольку люди предпочитали оставаться дома и заказывать еду [1].

Целью работы является – создание удобного и функционального приложения, которое позволит пользователям быстро и легко выбирать блюда и оформлять заказы.

Задачи:

- произвести анализ предметной области;
- произвести обзор существующих аналогов и выявить их преимущества и недостатки;
- определить требования к разрабатываемому мобильному приложению;
- произвести проектирование базы данных исходя из требований;
- разработка мобильного приложения;
- протестировать разработанное программное средство и доказать его работоспособности и эффективность.

Объектом исследования в работе является процесс разработки мобильного приложения для заказа и доставки еды.

Предметом исследования являются технологии, методы и инструменты, используемые при разработке приложения, а также его функциональность и интерфейс, который должен удовлетворять потребностям пользователей.

Методы исследований. Для решения поставленных задач были использованы методы математического моделирования, системного анализа, программной инженерии, объектно-ориентированного программирования, технологии проектирования человеко-машинного взаимодействия.

Практическая ценность работы заключается в том, что разработка такого приложения может быть полезна как для потенциальных пользователей, которые смогут воспользоваться его удобными функциями, так и для компаний, которые могут использовать его для увеличения продаж и улучшения сервиса доставки.

1 Аналитическая часть

1.1 Анализ предметной области

Разработка мобильного приложения для заказа и доставки еды - сложная задача, которая включает в себя разработку удобного интерфейса, интеграцию платежных систем и обеспечение своевременной доставки еды.

Целью проекта является создание платформы доставки еды, которая свяжет клиентов с широким спектром местных ресторанов, позволяя им в любое время заказать свои любимые блюда из полного меню. Приложение будет предоставлять услуги доставки еды потребителям, независимо от того, подается ли она в специальных местах общественного питания или ресторанах самообслуживания. Платформа будет включать заведения общественного питания с обслуживанием на вынос и обеспечивать приготовление и доставку блюд для непосредственного потребления.

Предлагаемая платформа "ЗаказIT" позволит клиентам оформить заказ, открыв приложение, пройдя регистрацию или авторизацию (указав свое имя, адрес и номер мобильного телефона), добавив товары в корзину и оставив заказ оператору. У каждого ресторана на платформе будет менеджер, который сможет принять и подтвердить заказ, изменить меню и прайс-лист, а также добавить различные акции и промо-коды.

Чтобы обеспечить своевременную доставку, у ЗаказIT будет сеть курьеров, которые будут доставлять еду клиентам из определенного ресторана по нужному адресу. Курьер может доставить еду пешком, на велосипеде, скутере или личном автомобиле. Платформа будет интегрировать платежную систему для обеспечения безопасных и простых транзакций клиентов.

Приложение будет разработано с использованием React Native, популярного фреймворка с открытым исходным кодом для создания собственных мобильных приложений для iOS и Android. React Native

позволяет ускорить разработку, повысить производительность и упростить обслуживание мобильных приложений, что делает его идеальным выбором для разработки платформы доставки еды.

1.2 Обоснование выбора платформы

Согласно отчету Statista, по состоянию на 2021 год в мире насчитывается около 5,2 миллиарда пользователей мобильных телефонов. Ожидается, что к 2023 году это число достигнет 5,4 миллиарда. Утверждается, что средний человек ежедневно проводит на своем смартфоне около 3 часов 15 минут, большую часть этого времени проводя в социальных сетях, приложениях для обмена сообщениями и платформах потоковой передачи видео. Ожидается, что использование мобильных устройств вскоре продолжит расти, поскольку все больше людей полагаются на них для общения, развлечений и доступа в Интернет. Ожидается, что растущая популярность технологии 5G также будет способствовать росту использования мобильных устройств, поскольку она обеспечивает более высокие скорости и более эффективные соединения [2].

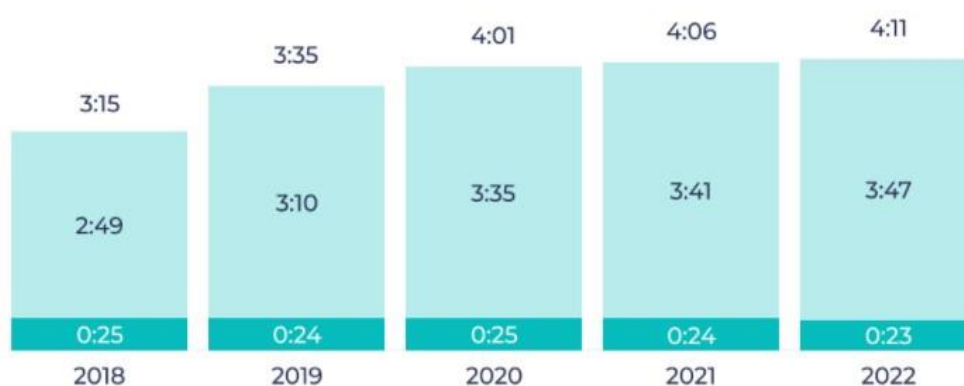


Рисунок 1 – Средний показатель ежедневного использования приложений и браузеров

Важно продумать, на какие платформы следует ориентировать приложение. В 2021 году двумя доминирующими платформами были iOS и Android, совокупная доля рынка которых составляла более 99%. По данным Sensor Tower, в 2021 году в App Store было доступно более 2 миллионов приложений с более чем 140 миллиардами загрузок. Это означает увеличение доступности приложений на 10% и увеличение количества загрузок на 25% по сравнению с предыдущим годом [5].

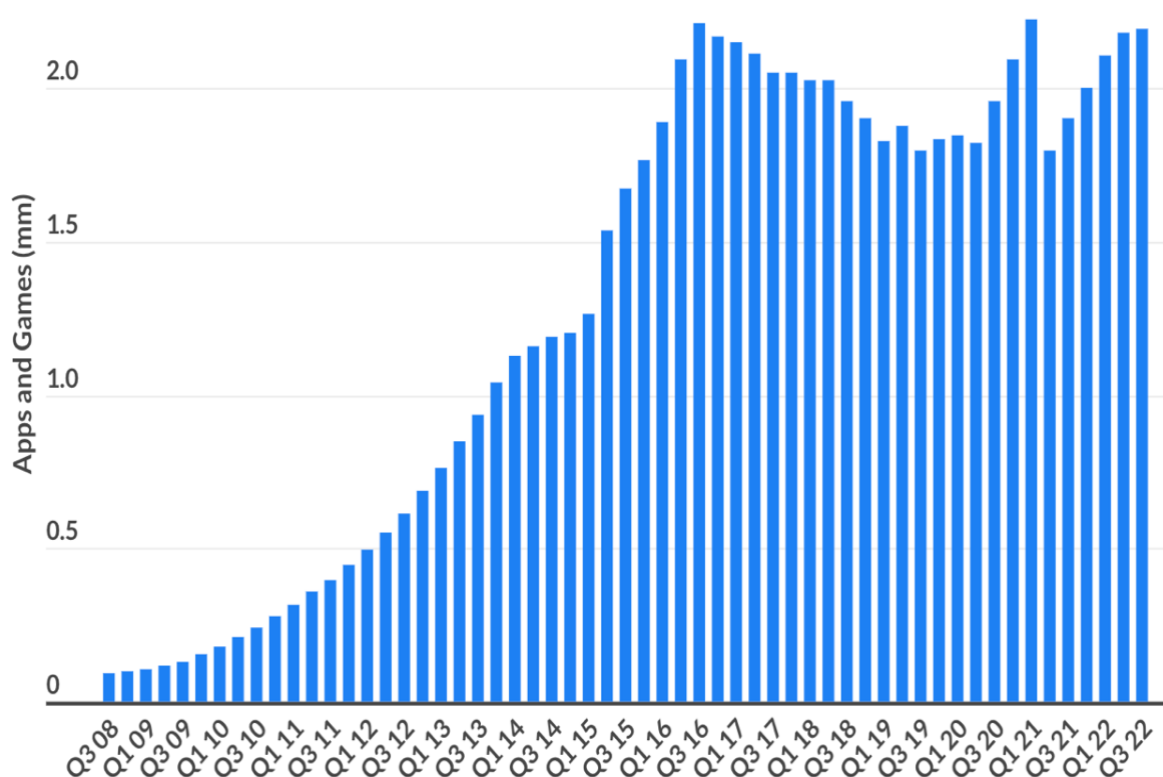


Рисунок 2 – Количество приложений на платформе Apple App Store

В платформе Google Play также наблюдался значительный рост: в 2021 году было доступно более 3,5 миллионов приложений и более 240 миллиардов загрузок. Это означает увеличение доступности приложений на 20% и увеличение количества загрузок на 30% по сравнению с предыдущим годом [5].

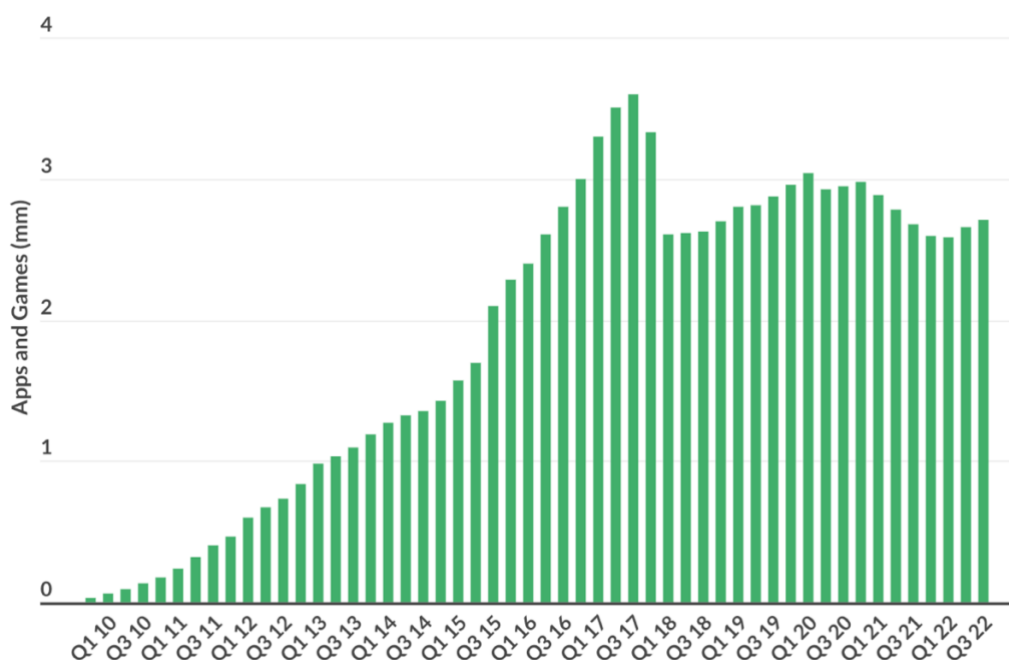


Рисунок 3 – Количество приложений на платформе Google Play

Основываясь на обзоре платформы и статистике загрузки мобильных приложений, выбор платформ iOS и Android для разработки может помочь охватить более широкую аудиторию и потенциально повысить успех приложения. Таким образом, будет разработано кроссплатформенное приложение, а следовательно, и платформа для размещения приложений - App Store и Google Play

1.3 Назначение мобильного приложения

Мобильные приложения для платформы онлайн-заказа и доставки еды ЗаказIT должны решать следующие задачи:

1. Возможность пользователей регистрироваться и входить в систему на платформе.
2. Возможность клиентов просматривать и выбирать нужный им ресторан и пункты меню с помощью интерфейса приложения.

3. Приложение должно предоставлять возможность фильтровать рестораны по различным критериям, таким как тип кухни, рейтинг, расстояние и цена.

4. Приложение должно позволять клиентам добавлять товары в корзину, изменять количество товаров и удалять их из корзины.

5. Возможность клиентов размещать свои заказы и управлять ими через приложение. Приложение должно предоставлять понятный и простой в использовании интерфейс для выбора вариантов доставки, отслеживания статуса заказа и обращения в ресторан или службу поддержки в случае возникновения каких-либо проблем.

6. Приложение должно интегрировать различные способы оплаты, такие как кредитные / дебетовые карты, онлайн-платежные системы и наложенный платеж.

7. Возможность клиентов оставлять отзывы и оценки о ресторане, качестве блюд, службе доставки и общем впечатлении.

В целом, приложение должно обеспечивать клиентам беспрепятственный и удобный процесс заказа и получения блюд из желаемых ресторанов.

Создание данного программного продукта преследует следующие цели:

1. Ускорение и упрощение выполнения заказа.
2. Увеличение потока новых клиентов.
3. Быстрый поиск нужных позиций блюд.
4. Возможность введения новых выгодных рекламных акций.

1.4 Обзор аналогов разработки

Разработка мобильного приложения для доставки и заказа еды не является новой концепцией, и на рынке уже существует множество аналогов. Однако конкуренция в индустрии доставки еды растет, и спрос на мобильные приложения с расширенными функциями и удобными интерфейсами растет. В этом разделе будет представлен

- обзор некоторых существующих аналогов,
- подчеркнуты их сильные и слабые стороны,
- потенциал для улучшения при разработке нового приложения с использованием React Native.

Понимание конкуренции на рынке и доступных альтернатив имеет решающее значение для разработки успешного и конкурентоспособного продукта.

При анализе современных средств разработки были выбраны 2 основных кандидата:

- Delivery Club
- Яндекс.Еды

1.4.1 Delivery Club

Delivery Club — это мобильный клиент для электронных гаджетов, позволяющий заказывать еду из множества ресторанов и кафе в вашем населенном пункте. Приложение позволяет в считанные секунды оформить заказ при этом, не отвлекаясь от основной деятельности, очень удобно, когда отправляетесь домой с работы, а вас уже поджидает курьер с любимым блюдом [8]. На рисунке 1.4, показана основное меню мобильного приложения.

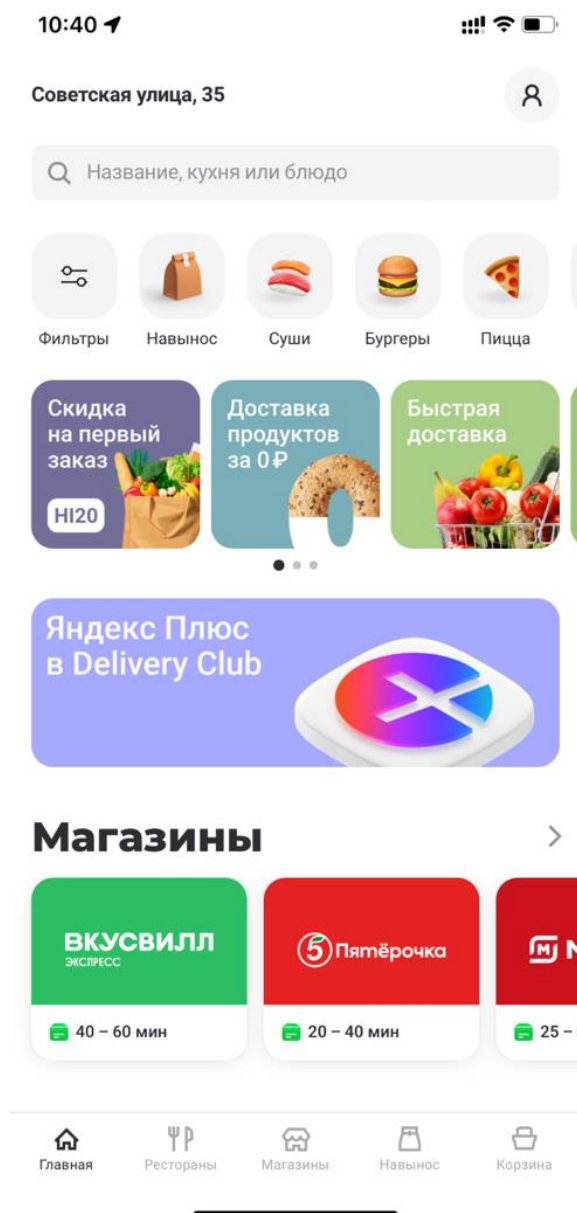


Рисунок 4 – Мобильное приложение Delivery Club

Заказ через сервис Delivery Club осуществляется с помощью приложения Delivery Club (iOS или Android) или на веб-сайте.

При выборе ресторана или магазина пользователь может ориентироваться на время доставки, рейтинг заведения, наличие специальных предложений. После добавления блюда или продуктов в корзину выбирается способ оплаты и оформляется заказ. После оформления заказа его статус можно отследить в мобильном приложении.

Интерфейс интуитивно понятный. Программа многофункциональна. Хотелось бы отметить следующие преимущества:

- информативность;
- функция повтора заказа;
- возможность привязки банковской карты;

Недостатки:

- отсутствует раздел «Избранное».
- отсутствует возможность удалить аккаунт;

1.4.2 Яндекс.Еды

Яндекс.Еды — Это мобильное приложение для Android и IOS, которое предназначено для заказа еды, а также продукты с доставкой на дом. Основная форма мобильного приложения показана на рисунке 1.5.



Рисунок 5 – Мобильное приложение Яндекс.Еды

Заказ еды осуществляется с помощью приложения Yandex.Eda (iOS или Android) или через веб-сайт, а также через Yandex Go и Яндекс.

Чтобы сделать заказ через сайт или приложение, пользователь выбирает ресторан или магазин из тех, что доступны по указанному адресу. Вы также можете отфильтровать доставку заказов из продуктовых магазинов. Пользователь выбирает блюда или продукты, добавляет их в корзину, выбирает способ оплаты и размещает заказ на доставку на дом или самовывоз. Если ресторан работает со службой доставки Яндекс.Еда, пользователь получает уведомления, когда ресторан начинает готовить заказ. Когда заказ забирается из ресторана курьерской службой, сотрудничающей с Яндекс.Еда, пользователь получает уведомления об изменениях статуса заказа, а также может отслеживать свой заказ на карте в режиме реального времени [9].

- малый вес;
- информативность;
- функция повтора заказа;
- возможность привязки банковской карты;

Недостатки мобильного приложения Яндекс.Еда:

- отсутствует раздел «Избранное».
- отсутствует возможность удалить аккаунт;

Сравнение выбранных мобильных приложений представлено в таблице 1 по следующим критериям: информативность, интерфейс, дизайн, удобство для пользователя.

Таблица 1.1 – Сравнение мобильных приложений

Критерий	Delivery Club	Яндекс.Еда
Информативность	Информативно	Информативно
Дизайн	Максимально удобный	Интуитивно понятный
Интерфейс	Хороший	Хороший
Удобство для пользователя	Проста в использовании	Удобно

2 Выбор средств и методов для создания приложения

Выбор средств и методов является важнейшим аспектом процесса разработки, поскольку он напрямую влияет на эффективность, масштабируемость и общий успех проекта. Для этой дипломной работы на степень бакалавра мы тщательно оценили и выбрали набор мощных и универсальных инструментов, чтобы обеспечить бесперебойную и надежную разработку.

Выбранные инструменты для этого проекта включают React Native, Firebase Cloud Firestore, Expo и Visual Studio Code (VS Code) в качестве редактора кода. Каждый инструмент обладает своими уникальными возможностями и преимуществами, позволяя разработчикам создавать высококачественное мобильное приложение, соответствующее требованиям и целям проекта.

2.1 React Native

React Native — это JavaScript-фреймворк для написания реальных мобильных приложений с собственным рендерингом для iOS и Android [11]. Он основан на React, JavaScript-библиотеке Facebook для создания пользовательских интерфейсов, но ориентирован не на браузер, а на мобильные платформы. Другими словами: веб-разработчики теперь могут писать мобильные приложения, которые выглядят и ощущаются по-настоящему “родными”, и все это с комфортом, используя библиотеку JavaScript, которую мы уже знаем и любим. Кроме того, поскольку большая часть кода, который вы пишете, может быть разделена между платформами, React Native упрощает одновременную разработку как для Android, так и для iOS.

Подобно React for the Web, приложения React Native написаны с использованием смеси JavaScript и разметки в стиле XML, известной как JSX. Затем, под капотом, React Native “bridge” вызывает собственные API рендеринга в Objective-C (для iOS) или Java (для Android). Таким образом, ваше приложение будет отображаться с использованием реальных компонентов мобильного пользовательского интерфейса, а не веб-просмотров, и будет выглядеть и чувствовать себя как любое другое мобильное приложение. React Native также предоставляет интерфейсы JavaScript для API-интерфейсов платформы, поэтому ваши приложения React Native могут получать доступ к функциям платформы, таким как камера телефона или местоположение пользователя.

React Native в настоящее время поддерживает как iOS, так и Android и имеет потенциал для расширения и на будущие платформы. В этой книге мы рассмотрим как iOS, так и Android. Подавляющее большинство кода, который мы напишем, будет кроссплатформенным. И да: вы действительно можете использовать React Native для создания готовых к работе мобильных приложений! Немного анекдота: Facebook, Palantir и TaskRabbit уже используют его в производстве для пользовательских приложений.

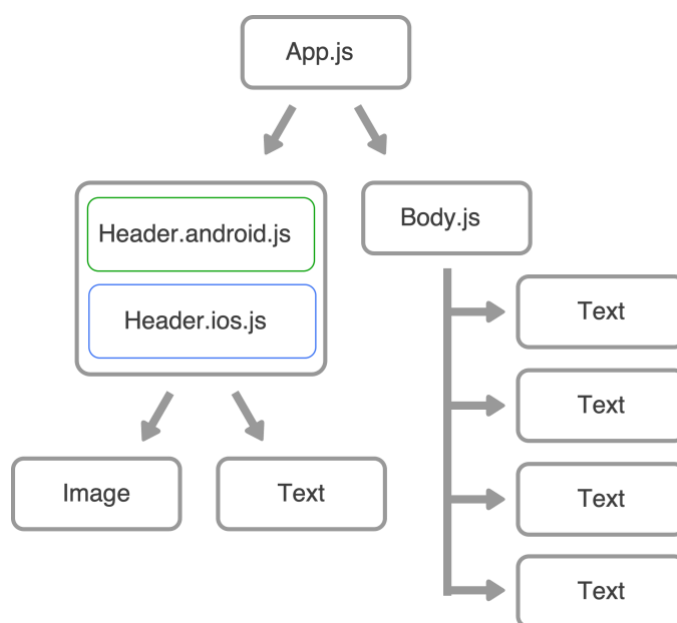


Рисунок 6 – Кроссплатформенная разработка с использованием React Native

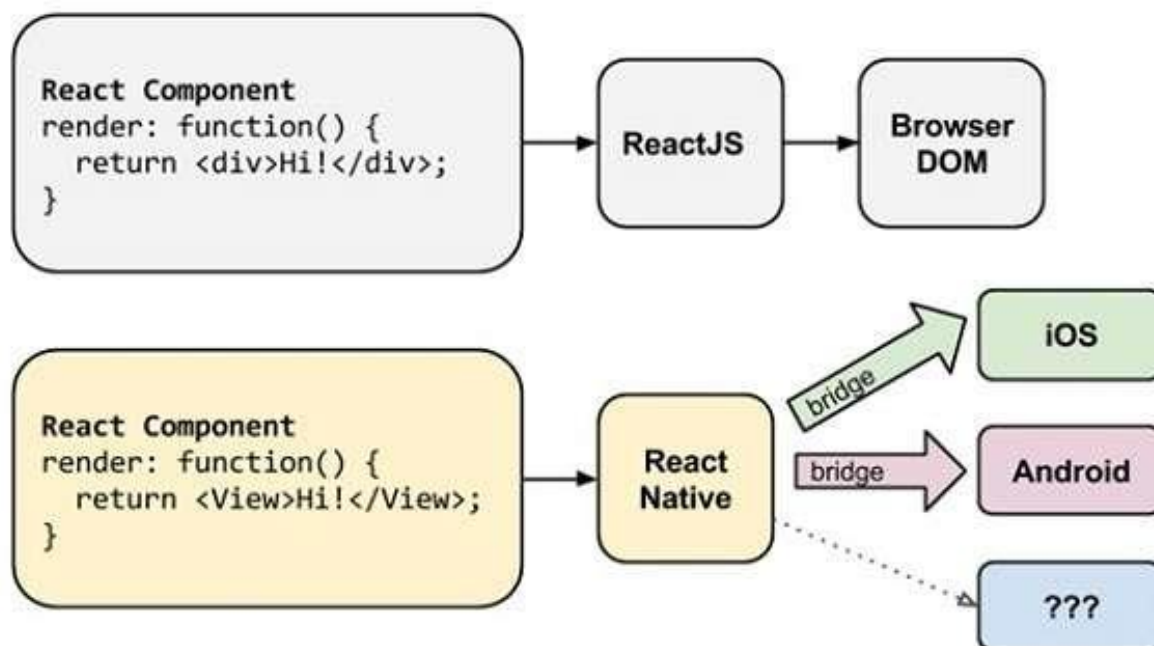


Рисунок 7 – Рендеринг компонентов в React Native

Решение использовать React Native для этого проекта было основано на нескольких ключевых факторах:

1. Кроссплатформенная совместимость: Одним из существенных преимуществ React Native является его способность создавать приложения, которые без проблем работают как на платформах iOS, так и на Android. Благодаря общей кодовой базе разработчики могут писать один раз и развертывать на нескольких платформах, экономя время и усилия. Такая кроссплатформенная совместимость обеспечивает более быстрые циклы разработки и более широкий охват аудитории.

2. Производительность, подобная нативной: React Native устраняет разрыв между нативной и веб-разработкой, используя нативные компоненты и рендера их с помощью JavaScript. Такой подход позволяет приложению обеспечивать производительность, близкую к нативной, обеспечивая плавный и отзывчивый пользовательский интерфейс. React Native достигает этого за счет оптимизации для конкретной платформы и рендеринга компонентов пользовательского интерфейса с использованием базовых собственных API.

3. Возможность повторного использования кода: Одним из ключевых преимуществ React Native является его способность повторно использовать код на разных платформах. Большинство бизнес-логики и компонентов пользовательского интерфейса могут быть разделены между iOS и Android, что сокращает дублирование и улучшает удобство обслуживания. Эта возможность совместного использования кода не только ускоряет разработку, но и упрощает процесс обслуживания, поскольку обновления и исправления ошибок могут применяться единообразно к обеим платформам.

4. Богатая экосистема и поддержка сообщества: React Native извлекает выгоду из динамичного и активного сообщества разработчиков и надежной экосистемы библиотек, инструментов и ресурсов. Эта обширная сеть поддержки обеспечивает доступ к широкому спектру готовых компонентов, сторонних интеграций и удобных для разработчиков утилит, которые могут значительно ускорить разработку и расширить функциональность приложения.

5. Быстрая итерация и скорость разработки: Функция горячей перезагрузки React Native позволяет разработчикам видеть изменения, внесенные в код, практически мгновенно, не требуя полной перестройки приложения. Этот быстрый цикл итераций значительно повышает скорость разработки, позволяя разработчикам быстро выполнять итерации, тестировать и совершенствовать функции, макеты и функциональность приложения.

6. Эффективность разработчика: React Native предлагает декларативную модель программирования, использующую возможности архитектуры React, основанной на компонентах. Такой подход упрощает разработку пользовательского интерфейса, поскольку разработчики могут создавать повторно используемые компоненты пользовательского интерфейса и эффективно управлять их состоянием. Знакомый и интуитивно понятный характер модели программирования React в сочетании с обширными возможностями инструментария и отладки React Native повышает производительность разработчиков и сокращает время обучения.

7. Широкое внедрение в отрасли: React Native завоевал значительную популярность в отрасли, и многие известные компании и организации используют его для своих нужд в разработке мобильных приложений. Такое широкое внедрение обеспечивает зрелую и стабильную платформу с постоянной поддержкой и обновлениями, что делает ее надежным выбором для создания готовых к работе приложений.

2.2 Firebase Cloud Firestore

Cloud Firestore — это гибкая, масштабируемая база данных для разработки мобильных устройств, веб-приложений и серверов от Firebase и Google Cloud. Как и Firebase Realtime Database, она синхронизирует ваши данные между клиентскими приложениями с помощью прослушивателей реального времени и предлагает автономную поддержку для мобильных устройств и Интернета, так что вы можете создавать адаптивные приложения, которые работают независимо от задержки в сети или подключения к Интернету. Cloud Firestore также предлагает бесшовную интеграцию с другими продуктами Firebase и Google Cloud, включая облачные функции.

Cloud Firestore — это размещенная в облаке база данных NoSQL, доступ к которой ваши приложения Apple, Android и веб-приложения могут получить напрямую через собственные SDK. Облачный Firestore также доступен на родном языке Node.js, Java, Python, Unity, C++ и Go SDK, в дополнение к REST и RPC API.

В соответствии с моделью данных NoSQL Cloud Firestore данные хранятся в документах, содержащих поля, соответствующие значениям. Эти документы хранятся в коллекциях, которые являются контейнерами для документов, которые можно использовать для организации данных и построения запросов. Документы поддерживают множество различных типов

данных, от простых строк и чисел до сложных вложенных объектов. Можно также создавать вложенные коллекции внутри документов и создавать иерархические структуры данных, которые будут изменяться по мере роста базы данных. Модель данных Cloud Firestore поддерживает любую структуру данных, которая лучше всего подходит для вашего приложения.

Кроме того, запросы в Cloud Firestore являются выразительными, эффективными и гибкими. Создавайте неглубокие запросы для извлечения данных на уровне документа без необходимости извлекать всю коллекцию или какие-либо вложенные полколлекции. Добавьте сортировку, фильтрацию и ограничения к вашим запросам или курсорам для разбивки результатов на страницы. Чтобы сохранить данные в актуальном состоянии вашего приложения, не извлекая всю базу данных при каждом обновлении, добавьте прослушиватели в реальном времени. Добавление прослушивателей в реальном времени в ваше приложение уведомляет вас моментальным снимком данных всякий раз, когда данные, которые прослушивают ваши клиентские приложения, изменяются, извлекая только новые изменения.

Защитите доступ к своим данным в Cloud Firestore с помощью аутентификации Firebase и правил безопасности Cloud Firestore для платформ Android, Apple и JavaScript или управления идентификацией и доступом (IAM) для серверных языков.

Решение о включении Firebase Cloud Firestore в проект было основано на его многочисленных ключевых функциях и преимуществах, которые способствуют общей эффективности и функциональности приложения.

1. Гибкость - Cloud Firestore предоставляет гибкую модель данных, которая поддерживает иерархические структуры данных. Вы можете хранить свои данные в документах, организованных в коллекции. Документы могут содержать сложные вложенные объекты в дополнение к вложенным коллекциям. Такая гибкость позволяет вам организовать ваши данные таким образом, который наилучшим образом соответствует требованиям вашего приложения. Независимо от того, есть ли у вас простые пары ключ-значение

или сложные вложенные структуры данных, Cloud Firestore может адаптировать вашу модель данных.

2. Выразительный запрос - Облачный Firestore позволяет использовать запросы для извлечения определенных документов или для извлечения всех документов в коллекции, соответствующих определенным критериям. Запросы могут включать в себя несколько связанных фильтров, сочетающих операции фильтрации и сортировки. По умолчанию Cloud Firestore индексирует запросы, обеспечивая эффективную производительность, пропорциональную размеру результирующего набора, а не всего набора данных. Это делает запрос и извлечение данных из базы данных очень выразительными, эффективными и масштабируемыми.

3. Обновления в реальном времени - Cloud Firestore, подобно базе данных реального времени, использует синхронизацию данных для предоставления обновлений в режиме реального времени на любом подключенном устройстве. Он эффективно выполняет одноразовые запросы для извлечения данных и обеспечивает синхронизацию данных между всеми подключенными клиентами. Всякий раз, когда в данных, которые прослушивают ваши клиентские приложения, происходят изменения, Cloud Firestore немедленно уведомляет их, гарантируя, что ваши пользователи всегда будут иметь актуальную информацию. Эта возможность синхронизации в режиме реального времени позволяет разрабатывать динамичные и отзывчивые приложения.

4. Автономная поддержка - Cloud Firestore предлагает встроенную автономную поддержку, позволяющую вашему приложению бесперебойно функционировать, даже если устройство отключено от сети или имеет плохое сетевое подключение. Он кэширует активно используемые данные на устройстве, позволяя пользователям читать, записывать, прослушивать и запрашивать данные локально. Как только устройство повторно подключается к сети, Cloud Firestore автоматически синхронизирует локальные изменения с удаленной базой данных, обеспечивая согласованность данных. Эта

автономная поддержка улучшает пользовательский опыт, обеспечивая бесперебойное использование приложения независимо от доступности сети.

5. Cloud Firestore предназначен для работы с приложениями любого масштаба. Он использует мощную инфраструктуру Google Cloud, обеспечивая автоматическую репликацию данных в нескольких регионах, надежные гарантии согласованности, атомарные пакетные операции и поддержку реальных транзакций. Независимо от того, имеет ли ваше приложение небольшую базу пользователей или сталкивается с большим трафиком и объемами данных, Cloud Firestore справится с требованиями сложности и масштабируемости. Он был разработан в соответствии с требованиями крупнейших глобальных приложений, гарантируя, что ваше приложение сможет беспрепятственно развиваться по мере расширения вашей пользовательской базы.

2.3 Expo CLI

Expo — это фреймворк и платформа для создания мобильных приложений с использованием React Native. Он предоставляет набор инструментов и сервисов, которые упрощают процесс разработки и позволяют разработчикам быстро создавать, тестировать и развертывать мобильные приложения на различных платформах, таких как iOS и Android.

Expo строится поверх React Native и расширяет его возможности, предлагая дополнительные функции и инструменты. Некоторые ключевые особенности выставки включают в себя:

1. Единая среда разработки: Expo предлагает единую среду разработки, которая позволяет разработчикам создавать приложения как для платформ iOS, так и для Android, используя единую кодовую базу. Этот подход, известный как кроссплатформенная разработка, значительно сокращает время

и усилия разработки за счет устранения необходимости поддерживать отдельные базы кода для каждой платформы. С помощью Ехро разработчики могут написать логику своего приложения один раз и без особых усилий развернуть ее на нескольких платформах, обеспечивая согласованность работы пользователей на разных устройствах.

2. Простая настройка: Ехро упрощает первоначальную настройку среды разработки. Он предоставляет инструмент интерфейса командной строки (CLI), который автоматизирует процесс создания нового проекта, управления зависимостями и настройки необходимых параметров сборки. Интуитивно понятный интерфейс CLI помогает разработчикам в процессе настройки, делая его доступным даже для тех, кто новичок в разработке мобильных приложений. Такая простота настройки позволяет разработчикам быстро приступить к разработке, не увязая в сложных задачах настройки.

3. Обширные встроенные API: Ехро предлагает богатый набор готовых встроенных API, которые обеспечивают доступ к различным функциям устройств и датчикам. Эти API охватывают широкий спектр возможностей, включая доступ к камере, геолокацию, акселерометр, push-уведомления, доступ к файловой системе и многое другое. Используя собственные API-интерфейсы Ехро, разработчики могут легко внедрять функции, зависящие от устройства, в свои приложения без необходимости писать код, зависящий от платформы. Это значительно упрощает процесс разработки и позволяет разработчикам сосредоточиться на создании основной функциональности своих приложений, а не тратить время на детали реализации, зависящие от конкретной платформы.

4. Обновления в режиме реального времени (OTA): Одной из выдающихся особенностей Ехро является ее способность предоставлять обновления в режиме реального времени (OTA) для развернутых приложений. Это означает, что разработчики могут вносить обновления и исправления ошибок в свои приложения, не требуя от пользователей скачивать и устанавливать новую версию из app Store. OTA-обновления Ехро позволяют

разработчикам беспрепятственно загружать обновления на устройства своих пользователей, гарантируя, что у них всегда будет последняя версия приложения. Эта функция особенно полезна для быстрого выполнения итераций и развертывания обновлений для пользовательской базы приложения, поскольку избавляет пользователей от необходимости проходить процесс обновления вручную.

5. Инструменты тестирования и отладки: Ехро предоставляет набор инструментов тестирования и отладки, которые упрощают процесс выявления и устранения проблем в приложении. Среда разработки Ехро включает встроенный симулятор и функцию перезагрузки в режиме реального времени, которая позволяет разработчикам мгновенно видеть изменения, которые они вносят в свой код, не перестраивая все приложение целиком. Это облегчает быструю итерацию и тестирование, повышая производительность разработчика. Кроме того, Ехро предлагает интеграцию с популярными инструментами отладки, такими как React Native Debugger и Flipper, что позволяет разработчикам отлаживать свои приложения непосредственно из среды разработки.

6. Сообщество и экосистема: у Ехро есть динамичное и активное сообщество разработчиков, которые вносят свой вклад в ее экосистему. Сообщество Ехро предоставляет множество ресурсов, включая учебные пособия, документацию и примеры кода, которые могут помочь разработчикам ускорить процесс обучения и разработки. Сообщество активно поддерживает Ехро и предлагает помощь разработчикам, сталкивающимся с проблемами или нуждающимся в руководстве. Кроме того, Ехро располагает обширной коллекцией созданных сообществом библиотек и компонентов, которые расширяют его функциональность и предлагают готовые решения для общих задач разработки. Эта обширная экосистема гарантирует разработчикам доступ к постоянной поддержке, возможность использовать коллективные знания и опыт других пользователей Ехро и ускорить

разработку за счет повторного использования существующих компонентов и решений.

2.4 Visual Studio Code

Visual Studio Code, также обычно называемый VS Code, представляет собой редактор исходного кода, созданный Microsoft на платформе Electron Framework для Windows, Linux и macOS. Функции включают поддержку отладки, подсветку синтаксиса, интеллектуальное завершение кода, фрагменты, рефакторинг кода и встроенный Git.

VS Code разработан как легкий, но многофункциональный редактор кода, который предлагает широкий спектр функциональных возможностей для разработчиков на различных языках программирования и платформах. Он обеспечивает интуитивно понятный и настраиваемый пользовательский интерфейс, что делает его легко адаптируемым к индивидуальным предпочтениям и рабочим процессам.

Решение использовать VS Code для этого проекта было основано на нескольких ключевых факторах:

1. Легкий и быстрый: Одним из заметных преимуществ VS Code является его легкий характер. Он разработан для обеспечения плавного и отзывчивого кодирования даже при работе с большими базами кода. В отличие от более тяжелых интегрированных сред разработки (IDE), VS Code предлагает минималистичную, но мощную среду, которая фокусируется на основных инструментах, необходимых для эффективного кодирования. Его эффективная архитектура обеспечивает быстрое время запуска, минимальное потребление ресурсов и плавную производительность, позволяя разработчикам сосредоточиться на написании кода без каких-либо отвлекающих факторов или задержек.

2. Кроссплатформенная совместимость: VS Code создан для работы на нескольких платформах, включая Windows, macOS и Linux. Такая кроссплатформенная совместимость позволяет разработчикам работать в предпочитаемой ими среде операционной системы без ущерба для функциональности или производительности. Разработчики могут легко переключаться между различными платформами, сохраняя при этом один и тот же набор функций и рабочих процессов, обеспечивая согласованность процесса разработки на разных компьютерах. Такая гибкость позволяет разработчикам беспрепятственно сотрудничать с членами команды, которые могут использовать разные операционные системы.

3. Богатая экосистема расширений: VS Code может похвастаться обширной и постоянно растущей экосистемой расширений, которые представляют собой плагины, созданные сообществом и расширяющие его функциональность. Эти расширения охватывают широкий спектр областей, включая языковую поддержку, форматирование кода, отладку, интеграцию с системой контроля версий, управление проектами и многое другое. Разработчики могут легко настроить свой редактор, установив расширения, которые наилучшим образом соответствуют их потребностям, что позволяет им адаптировать среду программирования к своим предпочтениям и работать более эффективно. VS Code Marketplace предоставляет централизованную платформу, где разработчики могут находить и устанавливать расширения всего несколькими щелчками мыши, что еще больше повышает их производительность и расширяет возможности редактора.

4. Надежные инструменты редактирования и рефакторинга: VS Code предлагает полный набор инструментов редактирования и рефакторинга, которые упрощают модификацию кода и обслуживание. Эти инструменты включают расширенные возможности поиска и замены, функции навигации по коду, такие как перейти к определению и найти все ссылки, и мощные операции рефакторинга, такие как переименование переменных и извлечение методов. Имея в своем распоряжении эти инструменты, разработчики могут

эффективно управлять своей кодовой базой, проводить рефакторинг кода для улучшения удобства сопровождения и с легкостью ориентироваться в сложных проектах. Интуитивно понятный и удобный интерфейс VS Code в сочетании с функциями редактирования и рефакторинга позволяет разработчикам писать чистый, хорошо структурированный код и поддерживать его качество на протяжении всего процесса разработки.

5. Интегрированный контроль версий: VS Code легко интегрируется с популярными системами контроля версий, такими как Git, предоставляя разработчикам плавный и интегрированный контроль версий. Встроенная интеграция с Git позволяет разработчикам просматривать различия в файлах, поэтапные изменения, фиксировать код и разрешать конфликты слияния непосредственно в редакторе. Такая тесная интеграция упрощает рабочий процесс разработки, упрощая отслеживание изменений, совместную работу с членами команды и обеспечивая целостность кода на протяжении всего процесса разработки. Разработчики могут получать доступ к общим операциям управления версиями, не переключаясь на внешние инструменты или терминалы, что позволяет им поддерживать бесперебойный рабочий процесс в редакторе.

6. Возможности отладки: VS Code предлагает надежные возможности отладки, которые помогают разработчикам эффективно выявлять и устранять проблемы в их коде. Он предоставляет интуитивно понятный интерфейс для установки точек останова, проверки переменных, пошагового выполнения кода и анализа стеков вызовов. VS Code поддерживает отладку для широкого спектра языков программирования и фреймворков, что делает его универсальным инструментом для отладки как серверного, так и интерфейсного кода. Возможность отладки кода непосредственно в редакторе значительно сокращает время и усилия, необходимые для диагностики и исправления ошибок, улучшая общий опыт разработки и ускоряя цикл разработки.

7. Встроенный терминал: VS Code включает в себя встроенный терминал, который позволяет разработчикам запускать инструменты командной строки и скрипты, не выходя из редактора. Эта функция устраняет необходимость переключаться между редактором и отдельным окном терминала, обеспечивая оптимизированную и унифицированную среду разработки. Встроенный терминал поддерживает различные оболочки и интерфейсы командной строки, позволяя разработчикам выполнять процессы сборки, запускать тесты или выполнять другие операции командной строки непосредственно в редакторе. Такая бесшовная интеграция повышает производительность разработчика и устраняет необходимость во внешних терминалах или дополнительных инструментах для выполнения рутинных задач командной строки.

8. Широкие возможности настройки: VS Code предлагает высокую степень настройки, позволяя разработчикам адаптировать редактор к своим конкретным предпочтениям и рабочим процессам. Разработчики могут настроить внешний вид редактора, выбирая из широкого спектра тем, цветовых схем и наборов значков. Они также могут настраивать сочетания клавиш, правила форматирования кода и поведение редактора в соответствии со своим стилем кодирования. Настройки VS Code легко настраиваются и могут быть персонализированы как на уровне пользователя, так и на уровне рабочей области, обеспечивая гибкость и адаптируемость к индивидуальным потребностям и потребностям команды.

9. Активное сообщество и поддержка: VS Code извлекает выгоду из динамичного сообщества разработчиков, которые вносят свой вклад в его постоянное совершенствование и делятся своими знаниями и расширениями. Ориентированный на сообщество характер VS Code гарантирует разработчикам доступ к множеству ресурсов, руководств и обсуждений для устранения неполадок, открытия новых функций и изучения передового опыта. Кроме того, корпорация Майкрософт предоставляет регулярные

обновления и поддержку, гарантируя, что VS Code остается надежным и современным редактором.

Выводы

В данной главе подробно рассматриваются основные технологии и инструменты, используемые при разработке мобильных приложений. В частности, в нем рассматривается React Native, широко распространенный фреймворк, позволяющий разработчикам создавать кроссплатформенные мобильные приложения с использованием JavaScript и React. Также объясняется обоснование использования Firebase Cloud Firestore, подчеркиваются его достоинства как гибкого и масштабируемого решения для баз данных NoSQL, которое облегчает синхронизацию данных в режиме реального времени и надежные возможности выполнения запросов. Кроме того, в главе рассматривается Expo - комплексный набор инструментов и сервисов, которые упрощают разработку и развертывание приложений React Native. Наконец, в нем рассказывается об использовании Visual Studio Code (VS Code), универсального и настраиваемого редактора исходного кода, известного тем, что он предлагает легкую, но мощную среду, способствующую эффективному кодированию.

3 Проектирование мобильного приложения

3.1 Проектирование приложения

В этом разделе будет рассмотрен процесс разработки мобильного приложения для заказа и доставки еды. Дизайн - важный этап разработки,

поскольку в этом разделе определяется структура приложения, его основные функции и возможности, информационная архитектура и схема навигации.

3.1.1 Структура приложения

Разрабатываемое приложение будет состоять в целом из трех основных компонентов: пользовательского интерфейса, серверной системы и базы данных.

Пользовательский интерфейс — это та часть приложения, с которой взаимодействуют пользователи. Он служит связующим звеном между пользователем и базовой функциональностью приложения. Пользовательский интерфейс будет включать в себя экраны для просмотра вариантов питания, выбора товаров, размещения заказов, осуществления платежей и отслеживания доставки. Кроме того, там будут экраны для регистрации пользователей, входа в систему и управления учетной записью.

Серверная система будет обрабатывать обработку и логику за кулисами. Он будет отвечать за управление учетными записями пользователей, обработку размещения и выполнения заказов, координацию с партнерами по доставке и генерирование уведомлений для пользователей о статусе их заказов.

База данных будет хранить и систематизировать данные, необходимые для приложения. Сюда входит информация о пользователях, пунктах преysкуранта, заказах, адресах доставки и деталях транзакции. База данных обеспечит надежное хранение данных и легкий доступ к ним для эффективного поиска и обновления.

Процесс обмена данными между клиентом и сервером состоит из последовательности шагов. Первоначально клиент отправляет запрос на сервер. После получения запроса сервер проверяет и идентифицирует характер запроса. При необходимости сервер обращается к базе данных для

извлечения соответствующей информации, связанной с запросом. Затем сервер обрабатывает эти данные и преобразует их в формат JSON, который является требуемым форматом для клиента. Наконец, сервер передает данные в формате JSON обратно клиенту, завершая цикл обмена данными.

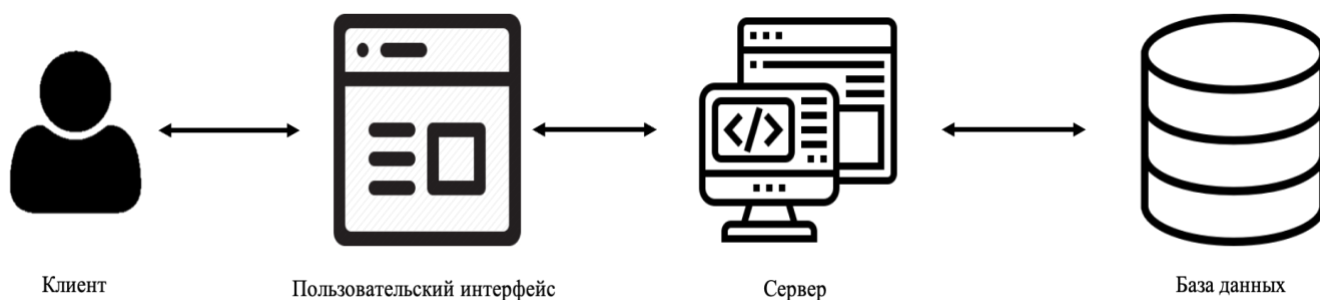


Рисунок 8 – Классическая схема работы клиент-серверного приложения

3.1.2 Определение основных функций и возможностей

Разрабатываемое приложение будет предоставлять несколько ключевых функций и опций, которые позволят пользователям быстро и легко выбирать блюда и оформлять заказы.

Основные функции приложения будут включать в себя:

- Выбор блюд: Пользователи смогут просматривать разнообразные пункты меню различных ресторанов, организованные по категориям и типам блюд. Они могут просматривать такие сведения, как названия товаров, описания, цены.

- Размещение заказа: Пользователи могут добавлять выбранные товары в свою корзину, настраивать свои заказы (например, указывать количество, способ получения заказа) и просматривать общую стоимость перед размещением заказа.

- Обработка платежей: Приложение будет поддерживать безопасные способы онлайн-оплаты, такие как кредитные карты или наложенный платеж. Пользователи могут выбрать предпочитаемый ими способ оплаты и беспрепятственно совершать транзакции.

- Отслеживание доставки: Пользователи смогут отслеживать статус своих заказов в режиме реального времени. Они будут получать уведомления и обновленную информацию о ходе доставки, включая предполагаемое время доставки и имя назначенного партнера по доставке.

- Отслеживание доставки: Пользователи смогут отслеживать статус своих заказов в режиме реального времени. Они будут получать уведомления и обновленную информацию о ходе доставки, включая предполагаемое время доставки и имя назначенного партнера по доставке.

- Отзывы пользователей: после получения своих заказов пользователи могут оставлять отзывы и оценки о качестве блюд, службе доставки и общем впечатлении. Это поможет поддерживать качество сервиса и поможет другим пользователям принимать обоснованные решения.

3.1.3 Разработка информационной архитектуры

Информационная архитектура разрабатываемого приложения включает в себя организацию и структурирование данных, которые будут храниться и извлекаться внутри системы.

К основным компонентам информационной архитектуры относятся:

- Профили пользователей: у каждого пользователя будет профиль, содержащий его личную информацию, контактные данные и историю заказов. Это обеспечивает персонализированный интерфейс и простое управление пользовательскими предпочтениями и предыдущими заказами.

- Профили ресторанов: у каждого ресторана будет профиль, содержащий 2.ПЗ со всеми приложениями личную информацию, контактные данные, пункты меню, стоимость доставки и историю заказов. Это обеспечивает персонализированный подход и бесперебойное управление меню и предыдущими заказами для ресторанов.

- Пункты меню: Информация о доступных продуктах питания будет организована структурированным образом. Сюда входят такие сведения, как названия товаров, описания, цены и любые специальные акции или скидки, связанные с ними.

- Заказы: Каждый заказ будет содержать важные сведения, такие как пользователь, который его разместил, ресторан, из которого сделан заказ, выбранные товары, адрес доставки, статус платежа и временные метки. Эта информация надежно хранится и легкодоступна для эффективной обработки заказов и их отслеживания.

- Информация о доставке: Сведения, относящиеся к процессу доставки, включая назначенного партнера по доставке, предполагаемое время доставки и текущий статус, сохраняются и обновляются в режиме реального времени. Это позволяет пользователям отслеживать свои заказы и получать точную информацию о ходе доставки.

Структурируя информационную архитектуру таким образом, приложение обеспечивает плавное управление данными, бесперебойный пользовательский интерфейс, а также эффективную обработку заказов и отслеживание доставки.

3.2 Реализация базы данных

3.2.1 Определение структуры базы данных

Cloud Firestore — это база данных NoSQL, ориентированная на документы. В отличие от базы данных SQL, здесь нет таблиц или строк. Вместо этого данные хранятся в документах, которые организованы в коллекции.

Каждый документ содержит набор пар ключ-значение. Облачный Firestore оптимизирован для хранения больших коллекций небольших документов.

Все документы должны храниться в коллекциях. Документы могут содержать вложенные коллекции и вложенные объекты, оба из которых могут включать примитивные поля, такие как строки, или сложные объекты, такие как списки.

Коллекции и документы создаются неявно в Cloud Firestore. Просто назначьте данные документу в коллекции. Если коллекция или документ не существуют, Cloud Firestore создает их.

Документ — это облегченная запись, содержащая поля, которые представляют собой пары ключ-значение, представляющие данные. Каждый документ в Firebase Cloud Firestore уникально идентифицируется по имени или идентификатору документа.

Например, давайте рассмотрим пример, в котором у нас есть документ, представляющий пользователя. Этот пользовательский документ может содержать различные поля, такие как "имя", "электронная почта" и "адрес", где каждое поле содержит определенное значение, связанное с этим пользователем. Имя или идентификатор документа служит уникальным идентификатором для данных этого конкретного пользователя в базе данных Firestore.

Объединяя данные в документы, Firebase Cloud Firestore обеспечивает эффективное хранение и извлечение информации. Каждый документ действует как автономная сущность, которая содержит соответствующие данные для конкретной сущности или объекта в вашем приложении, например пользователя, ресторана или заказа.

Коллекция — это место, где хранится документ, который является просто контейнерами для документов. Например, коллекция `users` содержит различных пользователей, каждый из которых представлен документом. Это похоже на наличие разных папок на вашем компьютере для упорядочивания файлов.

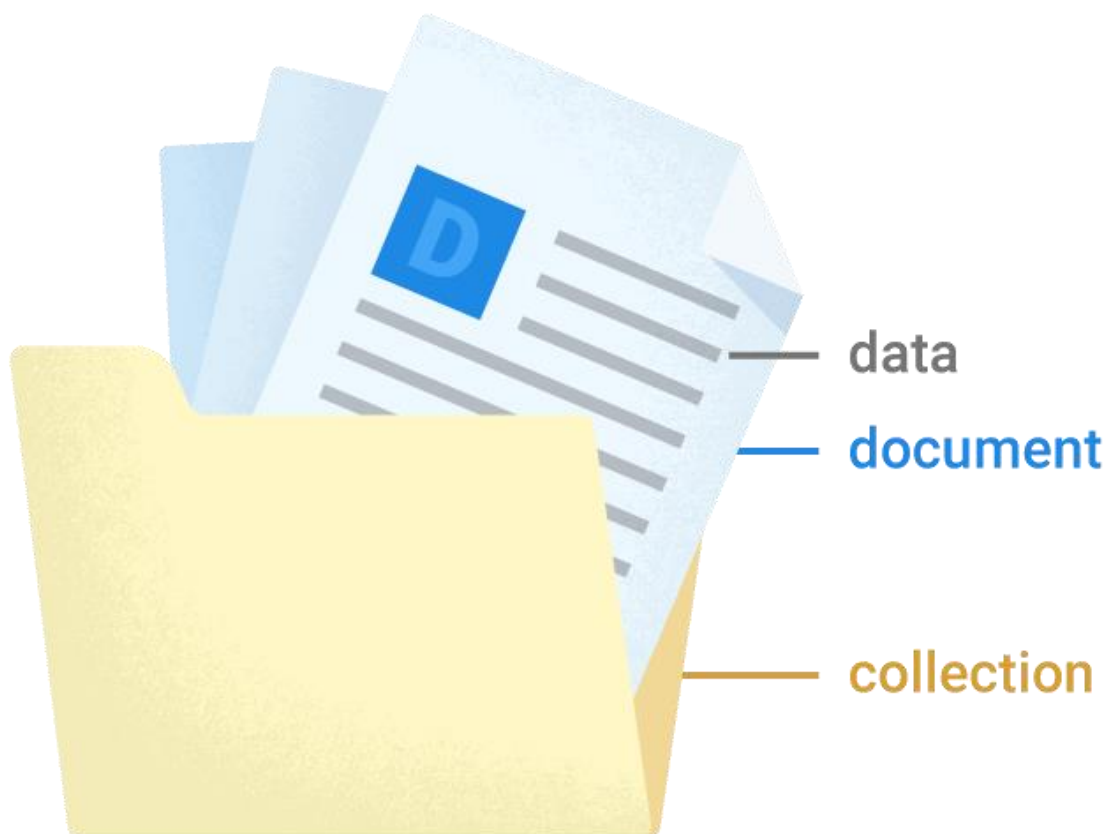


Рисунок 9 – Структура данных базы данных Firebase Cloud Firestore

3.2.2 Создание коллекций и документов

Создание коллекций и документов в Firebase Cloud Firestore упрощается за счет использования Firebase API. С помощью интерфейса прикладного

программирования мы можем использовать специальные методы для создания новых коллекций и включения в них документов. Чтобы проиллюстрировать это, создание коллекции с именем "Рестораны" может быть выполнено путем реализации метода "collection()" и указания желаемого имени для коллекции. Впоследствии, используя метод "add()", к этой коллекции может быть добавлен новый документ, позволяющий присваивать значения соответствующим полям. Для этого проекта созданы следующие коллекции:

- Коллекция "User": хранит информацию об отдельных пользователях. Он включает в себя такие поля, как "Name" и "Address", содержащие подробную информацию об имени и адресе пользователя соответственно. Эта коллекция позволяет идентифицировать и извлекать информацию, относящуюся к конкретному пользователю.

- Коллекция "Restaurant": содержит информацию о различных ресторанах. Он включает в себя такие поля, как "Name", "Address", "deliveryFee", "Image", "maxDeliveryTime" и "minDeliveryTime". Поле "Name" представляет название ресторана, в то время как поле "Address" хранит его физическое местоположение. В поле "deliveryFee" указана плата, взимаемая за услугу доставки. В поле "Image" хранится изображение, связанное с рестораном. В полях "maxDeliveryTime" и "minDeliveryTime" указаны максимальные и минимальные интервалы времени, необходимые для доставки заказов из соответствующего ресторана.

- Коллекция "Categories": хранится информация о различных категориях блюд. Он включает в себя такие поля, как "Image" и "Name", которые предоставляют подробную информацию о категориях, связанных с изображением и названием, соответственно. Эта коллекция позволяет организовать и классифицировать блюда на основе их соответствующих категорий.

- Коллекция "Dish": содержит информацию об отдельных блюдах. Он содержит такие поля, как "name", "image", "description" и "restaurant". Поле "name" представляет название блюда, в то время как поле

"image" хранит соответствующее изображение. Поле "description" содержит краткое описание или подробные сведения о блюде, а поле "restaurant" связывает блюдо с соответствующим рестораном.

— Коллекция "Orders": хранится информация об отдельных заказах. Он включает в себя такие поля, как "Restaurant", "userID", "Status", "subTotal" и "Total". Поле "Restaurant" представляет ресторан, связанный с заказом, в то время как поле "userID" идентифицирует пользователя, который разместил заказ. Поле "Status" указывает текущий статус заказа, как определено коллекцией "OrderStatus". В поле "subTotal" хранится промежуточная сумма заказа без учета налогов или дополнительных сборов, в то время как поле "Total" представляет общую сумму, включая все применимые сборы.

— Коллекция "OrderItems": содержит информацию о каждом отдельном блюде, включенном в заказ. Он включает в себя такие поля, как "Dish", "Quantity", "Price", "OrderID" и "Size". Поле "Dish" указывает на конкретное блюдо, включенное в заказ, в то время как поле "Quantity" представляет количество порций или единиц этого блюда. В поле "Price" хранится цена отдельного блюда, а поле "OrderID" связывает элемент заказа с конкретным заказом. Кроме того, поле "Size" содержит информацию о размере или порции блюда.

— Коллекция "OrderStatus": служит для перечисления различных статусов заказов. Он включает в себя такие значения, как "Accepted", "Preparing", "Arriving", "Ready_for_pickup" и "Delivered". Эта коллекция позволяет отслеживать и обновлять статус заказов, предоставляя информацию о ходе выполнения или текущем состоянии каждого заказа.

— Коллекция "Sizes": содержит информацию о различных размерах, доступных для посуды. Он включает в себя такие поля, как "Dish", "name" и "price". Поле "Dish" указывает на конкретное блюдо, к которому применяются параметры размера. Поле "name" представляет название или метку размера, в то время как в поле "Price" хранится соответствующая цена для этого размера.

3.3 Формализация функциональных требований к приложению

Диаграммы вариантов использования необходимы для сбора и визуализации требований к программной системе или приложению. Они описывают функциональные возможности и взаимодействия между различными компонентами, включая вовлеченных пользователей или участников. Каждый вариант использования представлен эллипсом на диаграмме и содержит описание конкретной операции или действия, которое он представляет. Описав варианты использования в предметной области, мы можем создать всеобъемлющую диаграмму, которая представляет функциональные возможности системы.[2]

Функциональные требования к разрабатываемому приложению представлены в виде диаграммы вариантов использования в соответствии с рисунком 3.3

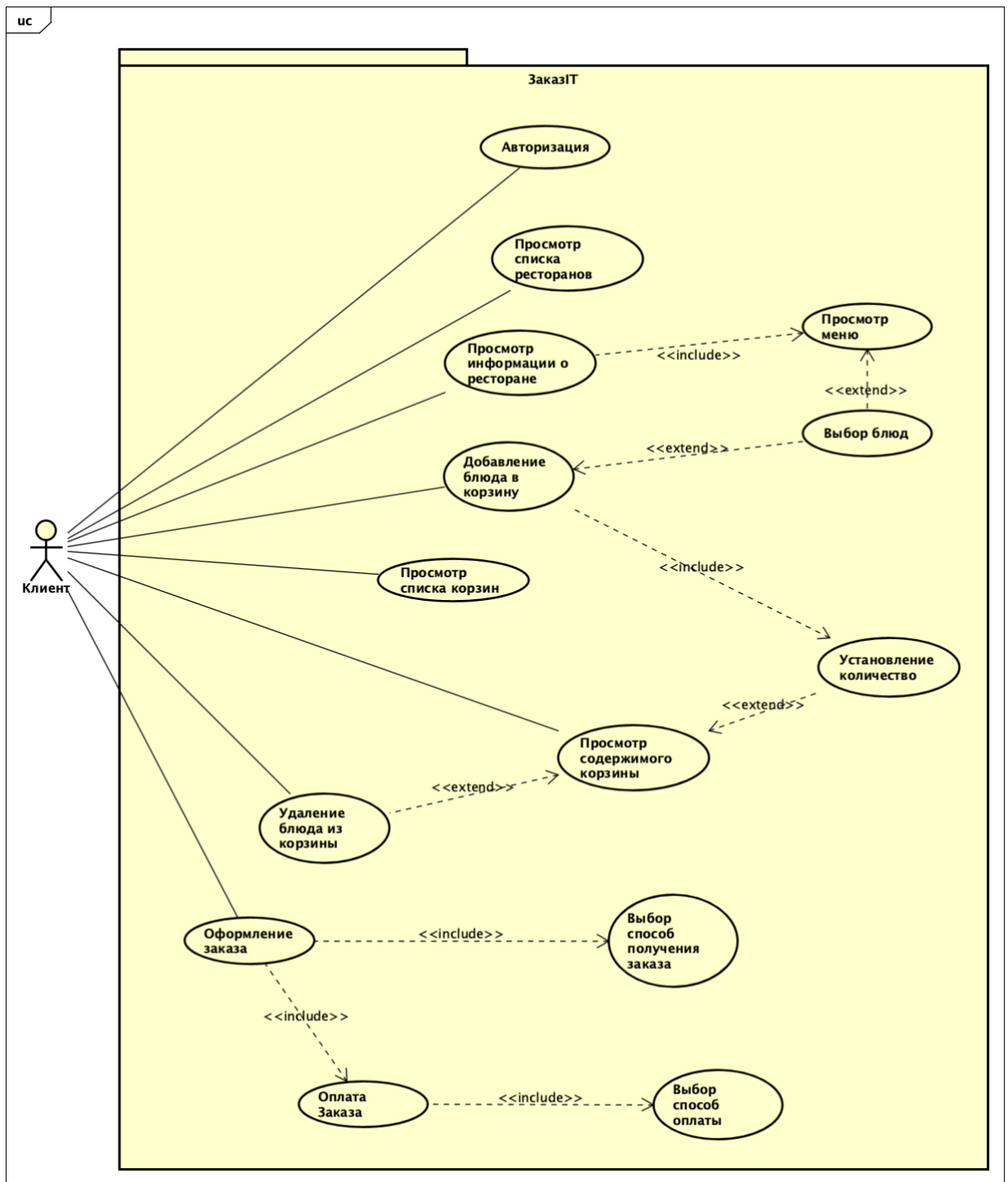


Рисунок 10 – Функциональные требования к приложению.

3.4 Разработка серверного программного обеспечения

Основная функциональность серверного программного обеспечения заключается в облегчении передачи и извлечения данных для клиентского приложения, в частности, связанных с объектными моделями. Сервер обрабатывает различные задачи и команды для выполнения этой цели. Вот обзор основных задач и команд:

1. Авторизация: обработка аутентификации пользователя и обеспечение безопасного доступа к приложению.
2. Регистрация: Управление процессом регистрации новых пользователей и надежное хранение информации об их учетной записи.
3. Выгрузка истории заказов клиента: Извлечение и отображение истории заказов для конкретного клиента.
4. Получение информации о категориях блюд: Извлечение данных о различных категориях блюд, доступных в приложении.
5. Получение информации о блюде: Предоставление подробной информации о конкретном блюде.
6. Подтверждение заказа: Обработка и подтверждение заказов клиентов.
7. Оплата: Обработка платежных транзакций по заказам.
8. Выбор типа оплаты: позволяет клиентам выбрать предпочитаемый ими способ оплаты.
9. Изменения цен: Управление обновлениями цен на блюда и их отражение в приложении.
10. Выбор адреса доставки: позволяет клиентам выбрать желаемый адрес доставки в процессе оформления заказа.
11. Добавление блюда: позволяет авторизованным пользователям добавлять новые блюда в каталог приложения.
12. Удаление блюда: предоставляет функциональность для удаления блюд из каталога приложения.
13. Добавление категории блюд: позволяет авторизованным пользователям создавать новые категории для блюд.

14. Удаление категории блюд: позволяет авторизованным пользователям удалять существующие категории блюд.
15. Получение информации о заказе: Предоставление подробной информации о конкретном заказе.
16. Формирование квитанций: Создание квитанций или счетов-фактур для выполненных заказов.
17. Добавление изображений блюд и категорий: позволяет авторизованным пользователям загружать изображения для блюд и категорий блюд.
18. Удаление изображений блюд и категорий: предоставляет функциональность для удаления изображений, связанных с блюдами и категориями блюд.

Реализуя эти задачи и команды в серверном программном обеспечении, клиентское приложение может беспрепятственно передавать и получать данные, обеспечивая плавный и эффективный пользовательский интерфейс.

3.4.1 Разработка модуля авторизации и аутентификации

Для реализации модуля авторизации и аутентификации в серверном программном обеспечении вашего мобильного приложения используется аутентификация Firebase. Firebase Authentication предлагает серверные службы и удобные SDK для обеспечения аутентификации пользователя в вашем приложении. Он поддерживает различные методы аутентификации, включая пароли, телефонные номера и популярных поставщиков федеративных удостоверений, таких как Google, Facebook и Twitter.

Для этого проекта мы будем использовать метод входа Email/password sign-in, который является распространенным подходом для аутентификации пользователей в приложениях. При использовании этого метода пользователи должны указать свой адрес электронной почты и безопасный пароль. Они

могут зарегистрироваться для новой учетной записи, используя метод `createUserWithEmailAndPassword`, или войти в существующую учетную запись, используя метод `signInWithEmailAndPassword`. Мы также используем метод `signOut`, чтобы вывести пользователя из текущего состояния аутентификации.

```
const SignUp = async () => {
  auth.createUserWithEmailAndPassword(email, password).then(userCredentials => {
    const user = userCredentials.user;
  }).catch(error => alert(error.message)).then(() =>{
    auth.signInWithEmailAndPassword(email, password).then(async userCredentials => {
      const user = userCredentials.user;
      setAuthUser(user)
      await createUser(user.uid)
    }).catch(error => alert(error.message))
  })
};
```

Рисунок 11 – Функция `SignUp` для создания нового пользователя

```
const onSignIn = async () => {
  auth
    .signInWithEmailAndPassword(email, password)
    .then(userCredentials => {
      const user = userCredentials.user;
      setAuthUser(user);
    })
    .catch(error => alert(error.message));
};
```

Рисунок 12 – Функция `onSignIn` для входа пользователя в систему

```
const signOut = () => {
  auth
    .signOut()
    .then(function () {
      setAuthUser(null)
      setDbUser(null)
    })
    .catch(error => alert(error.message));
};
```

Рисунок 13 – Функция signOut для выхода пользователя из системы

В ходе разработки модуля авторизации и аутентификации мы создадим экранные компоненты с использованием React Native, чтобы обеспечить пользовательский интерфейс для регистрации, входа в систему и функций управления учетными записями. При взаимодействии с Firebase Firestore мы будем использовать Firebase Authentication API для обработки регистрации пользователей, проверки подлинности учетных данных пользователей и управления пользовательскими сессиями.

Используя аутентификацию Firebase, вы можете обеспечить безопасную и оптимизированную аутентификацию пользователей для вашего мобильного приложения, позволяя пользователям безопасно регистрироваться, входить в систему и управлять своими учетными записями.

3.4.2 Реализация функции поиска и просмотра ресторанов и меню

Первым шагом в реализации функциональности просмотра ресторанов является получение списка всех доступных ресторанов из базы данных Firestore. Это позволяет пользователям просматривать доступные варианты и делать осознанный выбор.

Чтобы достичь этого, мы можем использовать запрос Firestore для извлечения всех документов из коллекции "Рестораны". Этот запрос извлекает всю коллекцию, возвращая моментальный снимок всех документов ресторана. Выполнив этот запрос, мы получим список ресторанов с их соответствующими полями и значениями. Это происходит, как только пользователь входит в систему, чтобы избежать отправки нескольких запросов к базе данных.

```
React.useEffect(() => {  
  db.collection("Restaurant")  
    .onSnapshot((querySnapshot) => {  
      const restaurantList = [];  
      querySnapshot.forEach((doc) => {  
        const resID = doc.id;  
        const rest = doc.data()  
        restaurantList.push({...rest, id: resID.toString()});  
        setRestaurants(restaurantList)  
      });  
    });  
});
```

Рисунок 14 – Функция для извлечения ресторанов из базы данных

Как только список ресторанов получен, он может быть представлен в пользовательском интерфейсе приложения, позволяя пользователям просматривать и выбирать конкретный ресторан для получения дополнительной информации.

После того, как пользователи выбрали конкретный ресторан, они могут просмотреть меню ресторана, которое состоит из списка блюд, предлагаемых заведением. Эта функциональность требует извлечения и отображения соответствующих пунктов меню из коллекции "Блюдо" в Firestore.

Чтобы реализовать эту функцию, создается запрос Firestore для извлечения блюд, связанных с идентификатором выбранного ресторана. Выполнив этот запрос в коллекции "Блюдо", можно получить все блюда,

связанные с выбранным рестораном. Извлеченные данные, включая такие детали, как название блюда, изображение, описание и доступные размеры, могут быть представлены в пользовательском интерфейсе приложения, предоставляя пользователям полный обзор меню.

```

React.useEffect(() => {
  setCartRestaurant(null);
  db.collection("Restaurant").doc(restaurant_ID)
    .onSnapshot((doc) => {
      const restaurantData = doc.data()
      const restaurantObject = { ...restaurantData, id: restaurant_ID };
      setRestaurant(restaurantObject)
    });

  db.collection("Dish").where("restaurantID", "==", restaurant_ID)
    .onSnapshot((querySnapshot) => {
      const dishList = [];
      querySnapshot.forEach((doc) => {
        const dishID = doc.id;
        const dish = doc.data()
        dishList.push({ ...dish, id: dishID.toString() });
      });
      setDishes(dishList)
    });
}, [restaurant_ID])

```

Рисунок 15 – Функция для извлечения информации о ресторане и его блюдах из базы данных ресторанов

Чтобы обеспечить пользователям доступ к актуальной информации о ресторанах и меню, Firestore предоставляет возможности синхронизации в режиме реального времени. Используя прослушиватели Firestore или обновления Firestore в режиме реального времени, любые изменения, внесенные в данные ресторана или меню, автоматически отражаются в приложении в режиме реального времени. Эта функция гарантирует, что

пользователи всегда будут иметь самую свежую информацию, даже если изменения будут внесены во время активного использования приложения.

3.5 Разработка пользовательского интерфейса

3.5.1 Разработка дизайна и UI-элементов

При разработке пользовательского интерфейса (UI) этого проекта мы используем фреймворк React Native, мощный инструмент, который позволяет создавать кроссплатформенные приложения с использованием JavaScript. Используя React Native, мы можем гарантировать, что наше приложение будет беспрепятственно функционировать на нескольких платформах, таких как iOS и Android, при максимальном повторном использовании кода и эффективности разработки.

Приступая к разработке пользовательского интерфейса, мы тщательно учитываем различные факторы, включая эстетику, удобство использования и соответствие лучшим отраслевым практикам. Объединяя эти соображения, мы стремимся создать интуитивно понятный и визуально привлекательный интерфейс, который улучшает общий пользовательский опыт.

Чтобы заложить основу для нашего дизайна пользовательского интерфейса, мы проводим тщательный анализ требований приложения, ожиданий пользователей и целевой аудитории. Это исследование позволяет нам получить четкое представление о желаемом потоке пользователей, приоритизации функций и ключевых функциональных возможностях, которые необходимо включить в пользовательский интерфейс.

Опираясь на полученные знания, мы приступаем к определению общего стиля и элементов дизайна, которые соответствуют целям проекта. Это включает в себя выбор подходящих цветовых палитр, типографики и

визуальных компонентов, которые перекликаются с темой приложения и брендингом. Кроме того, мы учитываем отраслевые стандарты и рекомендации по созданию пользовательских интерфейсов, чтобы обеспечить пользователям единообразный и привычный интерфейс.

Чтобы создать отзывчивый и удобный пользовательский интерфейс, мы подчеркиваем важность внедрения интуитивно понятных шаблонов навигации, четкой иерархии информации и хорошо структурированных макетов. Эффективно организуя контент и предоставляя логичные точки взаимодействия, мы даем пользователям возможность без особых усилий просматривать доступные рестораны, получать доступ к соответствующей информации и делать осознанный выбор.

При разработке пользовательского интерфейса мы также учитываем разнообразие устройств и размеров экранов, на которых будет использоваться приложение. Это включает в себя внедрение принципов адаптивного дизайна, гарантирующих, что пользовательский интерфейс адаптируется и оптимизирует свой макет для различных разрешений экрана и ориентаций, обеспечивая согласованность работы на всех устройствах.

На протяжении всего процесса разработки пользовательского интерфейса мы уделяем особое внимание удобству использования и доступности. Мы придерживаемся рекомендаций по обеспечению доступности, включая такие функции, как соответствующий цветовой контраст, текстовые альтернативы для изображений и доступность клавиатуры. Такой инклюзивный подход гарантирует, что приложение может быть доступно и использоваться широким кругом пользователей, включая людей с ограниченными возможностями.

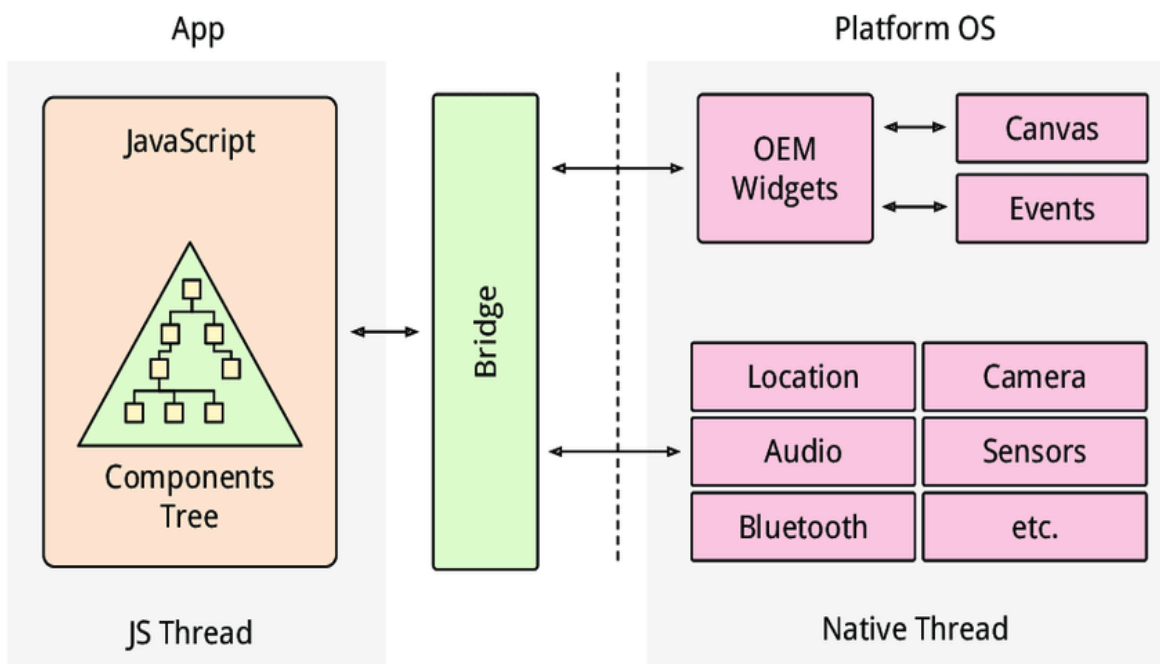


Рисунок 16 – Обзор архитектуры React Native.

Поскольку React Native по своей сути не основан на каком-либо конкретном архитектурном шаблоне, таком как MVC (Model-View-Controller) или MVP (Model-View-Presenter), для этого проекта мы придерживаемся архитектуры на основе компонентов, которая фокусируется на создании повторно используемых компонентов пользовательского интерфейса.

React Native использует модель однонаправленного потока данных, в которой данные передаются от родительских компонентов к дочерним компонентам через свойства (props). Эта модель потока данных аналогична принципу "нисходящего потока данных" в шаблоне MVP.



Рисунок 17 – Организация файлов пользовательского интерфейса React Native.

3.5.2 Создание макетов экранов

Для создания макетов экранов мы используем инструменты и библиотеки, предоставляемые React Native. Они позволяют нам разрабатывать

макеты, определять расположение и структуру компонентов на экране, а также определять стили и внешний вид элементов интерфейса.

Чтобы эффективно представить функциональность графического интерфейса, важно следовать четкому процессу. Выполняются следующие три шага:

- Создание прототипа интерфейса;
- Детальное прототипирование;
- Стилизация мобильного интерфейса.

Создание прототипа интерфейса: здесь создается прототип, который описывает внешний вид продукта, логику работы и ключевые функции. Этот прототип включает в себя базовые объектные модели, переходы, цвета оформления и основные функциональные элементы.



Рисунок 18 – Прототип интерфейса.

Детальное прототипирование: после создания первоначального макета следующим шагом является разработка подробного прототипа, который обеспечивает более конкретное представление пользовательского интерфейса. Этот подробный прототип демонстрирует элементы интерфейса и их предполагаемое взаимодействие.

Стилизация мобильного интерфейса: как только этап прототипирования завершен, следующим шагом является применение стилизации к мобильному интерфейсу. Это включает в себя доработку элементов визуального дизайна, таких как цвета, типографика, значки и общая эстетика, для создания целостного и визуально привлекательного пользовательского интерфейса.

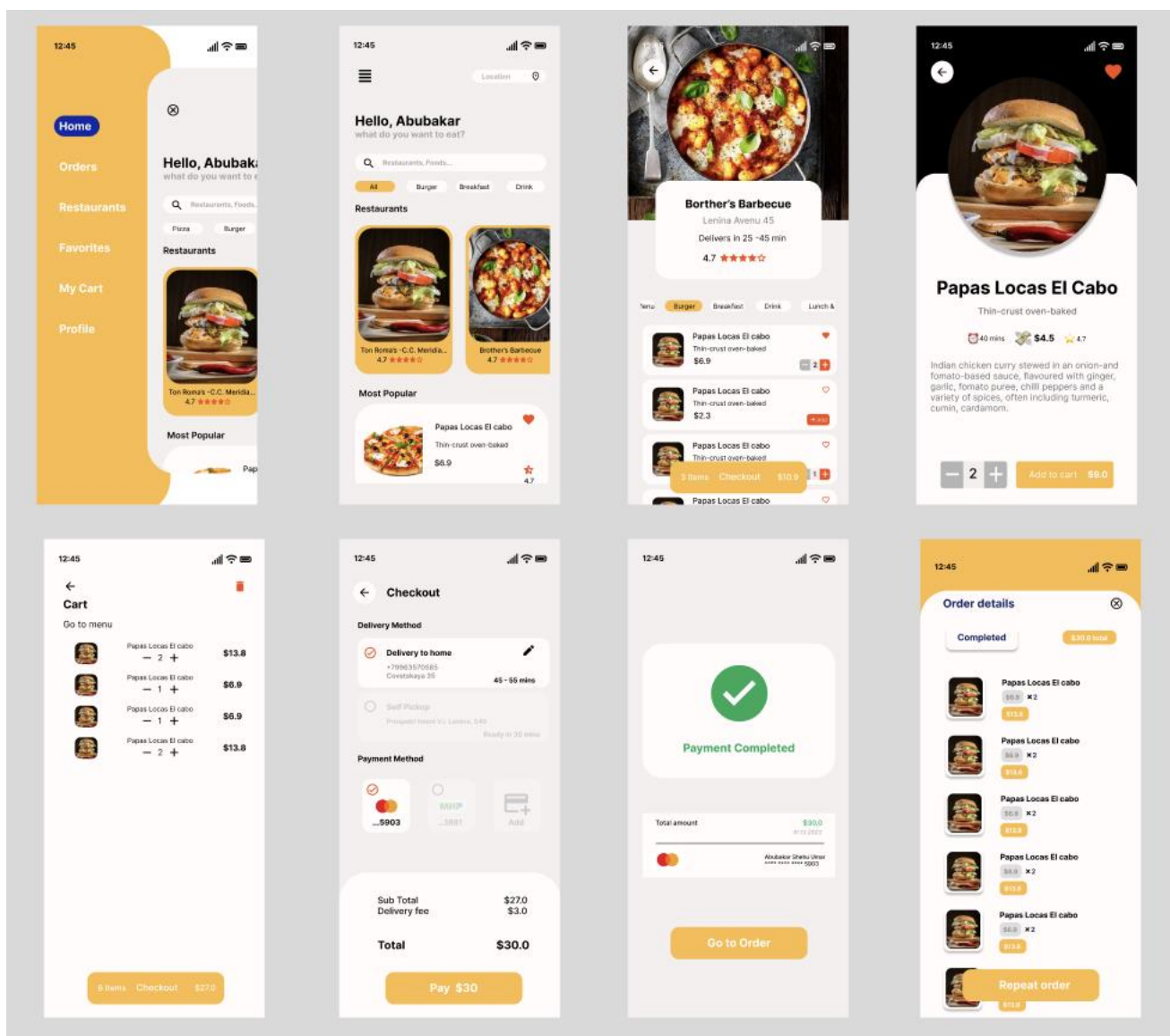


Рисунок 19 – Стилизация мобильного интерфейса.

3.5.3 Архитектура программы средства

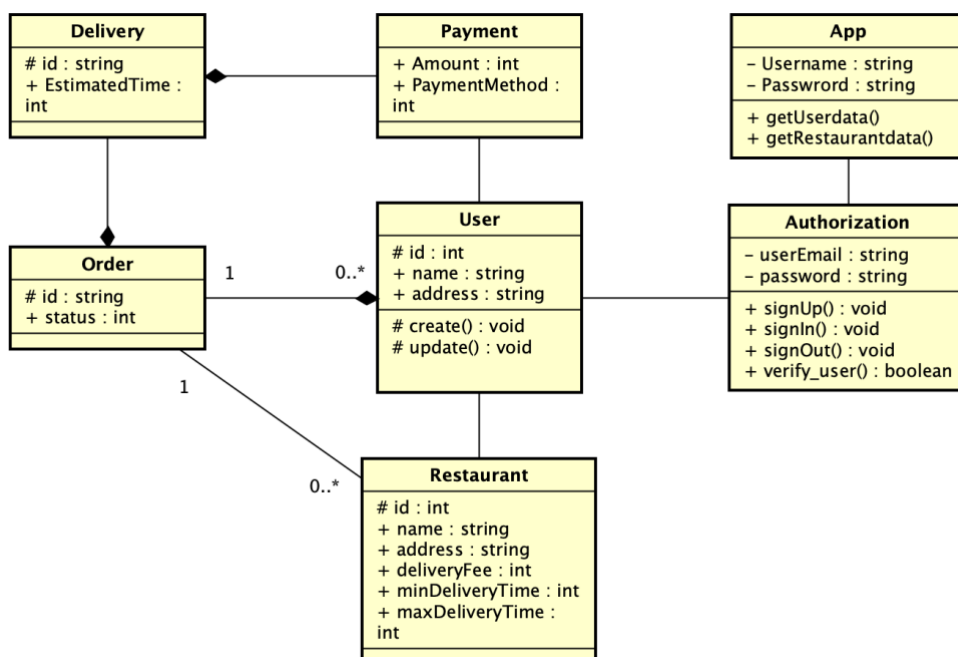


Рисунок 20 – Диаграмма классов приложения концептуального уровня в нотации UML.

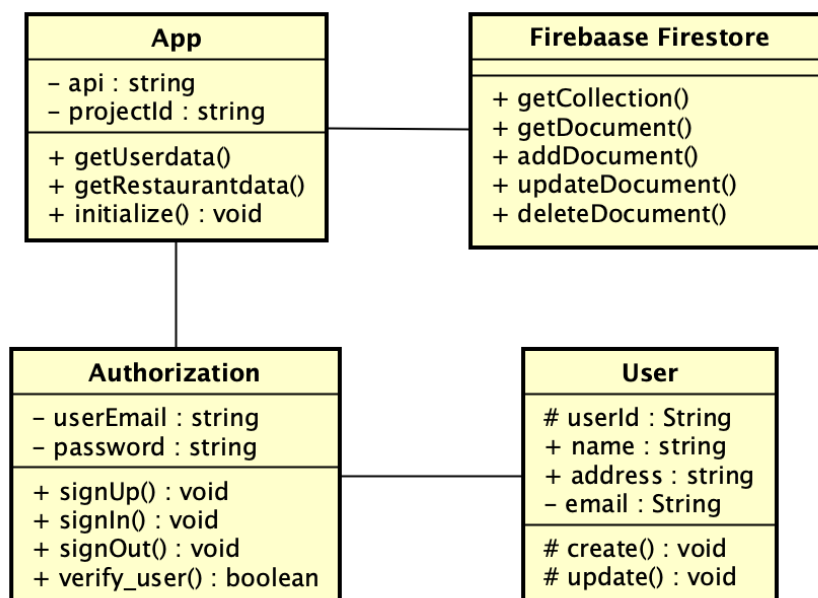


Рисунок 21 – Диаграмма классов сервера концептуального уровня в нотации UML.

Выводы

В этой главе обсуждается структура приложения, определение основных функций и возможностей, разработка информационной архитектуры. В нем также описывается внедрение базы данных, создание коллекций и документов, а также разработка серверного программного обеспечения, включая модуль авторизации и аутентификации, функцию поиска и просмотра ресторанов и меню. В конце главы мы рассмотрим разработку пользовательского интерфейса, включая дизайн и элементы пользовательского интерфейса, а также создание макетов экранов.

4 Тестирование мобильного приложения

Для оценки производительности и результативности разработанного мобильного приложения было использовано несколько методик тестирования. Для оценки различных аспектов приложения были проведены следующие тесты:

- Пользовательское тестирование (User Testing);
- A/B тестирование;
- Опросы и анкеты (Surveys and Questionnaires).

4.1 Пользовательское тестирование

Пользовательское тестирование (или тестирование юзабилити) является процессом, который идентифицирует проблемы продукта. Вы находите

реальных пользователей, желательно из вашей целевой аудитории, и просите их попробовать ваше детище. История этого вида тестирования началась во время второй мировой войны, когда тестировали удобство использования военной техники [26].

Результаты пользовательского тестирования этого мобильного приложения позволили нам получить ценную информацию о взаимодействии пользователя с интерфейсом и обнаружить некоторые аспекты, которые нуждаются в улучшении. Во время тестирования мы провели серию заданий, которые должны были выполнить пользователи, и собрали отзывы и наблюдения об их опыте использования приложения.

Одним из результатов пользовательского тестирования стало выявление трудностей в навигации по приложению. Пользователи отметили, что некоторые функциональные элементы не всегда понятны и интуитивно понятны в использовании. Это позволило нам понять, что нам необходимо улучшить визуальную привлекательность и понятность элементов интерфейса, а также оптимизировать маршрутизацию между экранами приложений.

Кроме того, пользовательское тестирование выявило определенные проблемы с производительностью и отзывчивостью приложения на некоторых устройствах. Это указывало на необходимость оптимизации кода и ресурсов приложения для повышения его производительности на различных платформах и устройствах.

Отзывы, которые мы получили от пользователей, также позволили нам улучшить удобство использования приложения. Они внесли ряд предложений по улучшению визуального оформления, удобства навигации и функциональности приложения. На основе этих рекомендаций мы внесли соответствующие изменения в интерфейс и добавили новые функции.

В целом, результаты пользовательского тестирования помогли нам лучше понять потребности и ожидания наших пользователей, а также выявить слабые места и возможности для улучшения нашего мобильного приложения,

разработанного на базе React Native. Мы учли отзывы и рекомендации пользователей, что позволило нам создать более удобный, интуитивно понятный и производительный интерфейс приложения.

4.2 A/B тестирование

Чтобы улучшить пользовательский интерфейс и оптимизировать процесс добавления блюда в корзину в нашем мобильном приложении, мы провели A/B-тест для сравнения двух разных интерфейсов. Цель состояла в том, чтобы определить, какая версия обеспечит более плавное и интуитивно понятное взаимодействие с пользователем, что в конечном итоге приведет к более высокой вовлеченности пользователей и коэффициенту конверсии.

Для A/B-теста мы тщательно разработали и внедрили две версии интерфейса: версию А и версию В. Обе версии были направлены на упрощение процесса выбора блюд и их добавления, но они предлагали различные элементы дизайна и пользовательские потоки.

На этапе тестирования мы разделили нашу целевую аудиторию на две группы: группу А, которой была представлена версия интерфейса А, и группу В, которая испытала версию В. Каждая группа была распределена случайным образом, чтобы обеспечить беспристрастное распределение.

На протяжении всего тестирования мы собирали ряд данных, чтобы оценить производительность и пользовательский опыт каждой версии. Они включали взаимодействие с пользователями, время, затраченное на выполнение задачи, и качественную обратную связь, полученную в ходе опросов и интервью с пользователями. Кроме того, мы отслеживали ключевые показатели, такие как коэффициент конверсии, который измеряет количество успешных добавлений блюд в корзину.

Результаты А/В-теста выявили убедительные выводы. Версия В продемонстрировала более высокий коэффициент конверсии по сравнению с версией А. Пользователи сочли версию В более интуитивно понятной, с ее упрощенными шагами и четкими призывами к действию, направляющими их на протяжении всего процесса. Улучшенный пользовательский интерфейс и улучшенный визуальный дизайн версии В положительно повлияли на вовлеченность пользователей, что привело к увеличению количества блюд, добавляемых в корзину.

Основываясь на этих выводах, мы приняли стратегическое решение внедрить версию В в качестве интерфейса по умолчанию в нашем мобильном приложении. Поступая таким образом, мы стремились обеспечить нашим пользователям более плавный и приятный процесс добавления блюд в их корзины. Ожидалось, что это изменение приведет к повышению удовлетворенности пользователей и коэффициенту конверсии, что в конечном итоге приведет к росту бизнеса.

Методология А/В тестирования позволила нам объективно сравнить две версии интерфейса и принимать решения, основанные на данных. Это дало ценную информацию о предпочтениях, поведении и ожиданиях пользователей. Выбрав версию В, мы уверены, что оптимизировали процесс добавления блюд в корзину, приведя его в соответствие с потребностями и предпочтениями пользователя.

Этот процесс А/В тестирования не только улучшил наше понимание предпочтений пользователей, но и подчеркнул важность итеративного проектирования и постоянного совершенствования. Мы продолжим использовать А/В тестирование в будущих обновлениях и выпусках функций, чтобы гарантировать, что наше мобильное приложение неизменно соответствует меняющимся потребностям и ожиданиям наших пользователей.

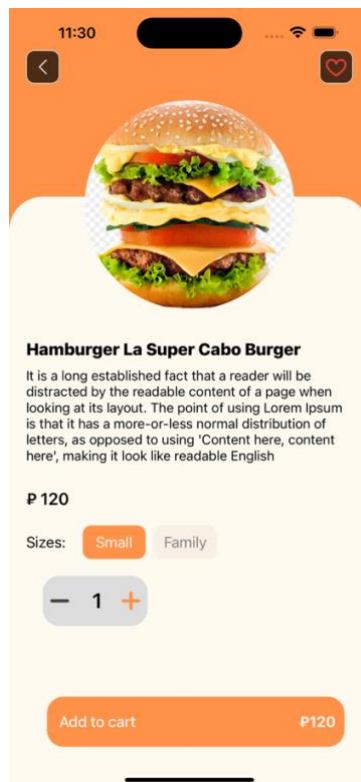


Рисунок 22 – Версия А интерфейса для добавления блюда в корзину.

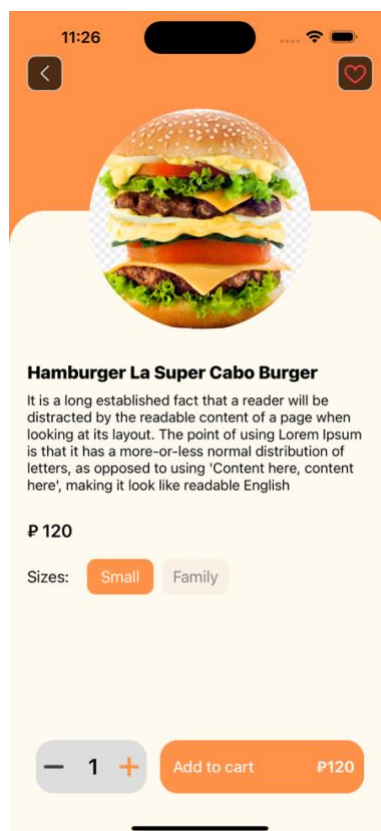



Рисунок 23 – Версия В интерфейса для добавления блюда в корзину


4.3 Опросы и анкеты

Для того чтобы проверить, эффективно разработанное приложение или нет, была составлена анкета, которая показана на рисунке 23.

The image shows a survey form for a mobile application. The title is 'Мобильное приложение для заказа и доставки еды'. Below the title, there is a user profile section showing the email 'shehuumara@gmail.com' with a 'Switch account' link and a cloud icon. Below that, it says 'Not shared' with an envelope icon. A red asterisk indicates a required question. The first question is 'Name *' with a text input field labeled 'Your answer'. The second question is 'Удовлетворены ли вы функциональностью разработанного приложения? *' with four radio button options: 'Да, полностью', 'Да, частично', 'Нет, неполностью', and 'Нет, неудовлетворен'. At the bottom, there is a purple 'Submit' button and a 'Clear form' link.

Мобильное приложение для заказа и доставки еды

shehuumara@gmail.com [Switch account](#) 

 Not shared

* Indicates required question

Name *

Your answer

Удовлетворены ли вы функциональностью разработанного приложения? *

☐ Да, полностью

☐ Да, частично

☐ Нет, неполностью

☐ Нет, неудовлетворен

Submit Clear form

Рисунок 24 – Форма опроса

Затем было собрано 50 человек, которым было поручено заказать еду с помощью приложения и на основании этого ответить на вопрос из предоставленной формы. Исходя из этого, были получены следующие результаты, как показано на рисунке 24.

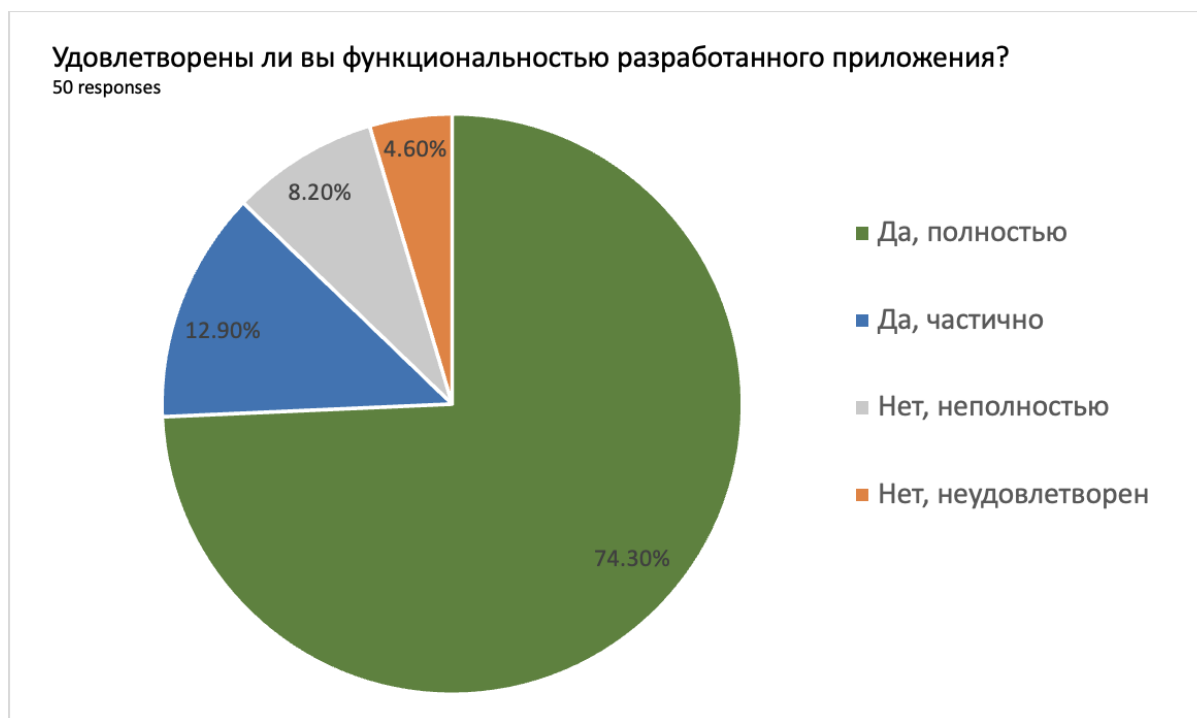


Рисунок 25 – Результаты опроса

Анализ полученных данных показывает, что разработанный виртуальный тур был весьма эффективен в предоставлении необходимой информации пользователям. Из общего числа респондентов 74,3% выразили полное удовлетворение виртуальным туром, заявив, что они получили всю необходимую информацию. Еще 12,9% сообщили о частичном удовлетворении, указав, что они получили некоторые, но не все необходимые детали. Однако 8,2% выразили недовольство, поскольку почувствовали, что их потребности были удовлетворены не полностью, в то время как 4,6% выразили полное недовольство полученными результатами.

Основываясь на результатах этого опроса, можно сделать вывод, что разработанное мобильное приложение для доставки и заказа еды успешно

решает ранее выявленные проблемы. Приложение значительно упрощает процесс выбора блюда и позволяет пользователям более эффективно заказывать еду, что приводит к повышению удобства и экономии времени. Высокий уровень удовлетворенности пользователей демонстрирует эффективность приложения в плане удовлетворения их ожиданий и улучшения их общего опыта.

Выводы

В этой главе этап тестирования процесса разработки мобильного приложения сыграл решающую роль в обеспечении его эффективности, удовлетворенности пользователей и общего успеха. Благодаря внедрению пользовательского тестирования, А/В-тестирования и опросов/анкетирований были получены ценные сведения и обратная связь, позволяющие вносить улучшения и оптимизацию.

В целом, этап тестирования сыграл важную роль в доработке мобильного приложения, обеспечении его удобства использования, функциональности и соответствия потребностям пользователей. Информация, полученная в результате различных методов тестирования, легла в основу процесса разработки, что привело к улучшению пользовательского интерфейса и созданию более надежного приложения. Отзывы, полученные от пользователей, были ценными при формировании конечного продукта и подтверждении его эффективности в решении целевых проблем.

Заключение

В заключение отметим, что этот проект был сосредоточен на разработке мобильного приложения для заказа и доставки еды с использованием React Native и Firebase Firestore. На протяжении всего проекта мы успешно разрабатывали и внедряли удобное в использовании и эффективное приложение, отвечающее потребностям современных служб доставки еды.

Анализ домена выявил растущий спрос на услуги доставки еды и потребность в удобном мобильном приложении для облегчения процесса заказа и доставки. Выбрав React Native в качестве платформы разработки, мы обеспечили кроссплатформенную совместимость и возможность таргетирования как на устройства iOS, так и на Android.

Структура приложения была тщательно спланирована с учетом различных компонентов, таких как аутентификация пользователя, управление рестораном и меню, обработка заказов и отслеживание заказов. Эти компоненты были легко интегрированы, чтобы обеспечить единый и интуитивно понятный пользовательский интерфейс.

Процесс разработки включал в себя проектирование пользовательского интерфейса, создание макетов экранов, реализацию навигации и разработку необходимой функциональности. React Native позволил нам использовать JavaScript для создания отзывчивого и динамичного пользовательского интерфейса, в то время как Firebase Firestore служила серверной базой данных для эффективного хранения и извлечения данных.

Было проведено тщательное тестирование, чтобы убедиться в функциональности, производительности и стабильности приложения. Для выявления и устранения любых проблем или багов были использованы различные методы тестирования, включая модульное тестирование, интеграционное тестирование и тестирование производительности. Этот тщательный процесс тестирования гарантирует создание надежного приложения, способного работать в реальных сценариях.

Интеграция внешних сервисов была еще одним важным аспектом проекта. Платежные системы были легко интегрированы для облегчения безопасных и удобных платежных операций, в то время как службы доставки были интегрированы для обеспечения точной и своевременной доставки обновлений. Кроме того, взаимодействие приложения с API-интерфейсами ресторанов и поставщиков обеспечило бесперебойный обмен данными и оптимизацию операций.

В заключение, в рамках этого проекта было успешно разработано мобильное приложение, которое позволяет пользователям удобно заказывать и отслеживать доставку продуктов питания. Сочетание React Native и Firebase Firestore обеспечило мощную и эффективную среду разработки, обеспечивающую быструю разработку и развертывание. Удобный интерфейс приложения, надежная функциональность и плавная интеграция с внешними сервисами делают его ценным инструментом как для клиентов, так и для поставщиков услуг доставки еды.

В целом, успешное завершение этого проекта демонстрирует эффективность использования React Native и Firebase Firestore для разработки мобильных приложений в сфере доставки продуктов питания. Функции и возможности приложения соответствуют первоначальным целям и требованиям, обеспечивая превосходный пользовательский интерфейс и открывая новые возможности для будущих улучшений и расширений.

Список использованных источников

1. Исследование рынка доставки еды – Текст: электронный // Yookassa – 2022 – URL: <https://promo.yookassa.ru/delivery-research>(дата обращения: 20.12.2022).
2. Number of mobile phone users worldwide from 2018 to 2023 (in billions) – Текст: электронный // Statista – 2023 URL: <https://www.statista.com/statistics/271409/forecast-of-mobile-phone-users-worldwide/>(дата обращения: 20.12.2022).
3. Маркин, Е.И. Использование реактивного программирования при разработке мобильных приложений / Е.И. Маркин, К.М. Рябова // Computational nanotechnology. — 2016. — No 2. — С. 170-173. — ISSN 2313-223X. — Текст : электронный // Лань : электронно-библиотечная система. — URL: <https://e.lanbook.com/journal/issue/298484>(дата обращения: 12.01.2023).
4. Average daily time spent on mobile devices in the United States from 2016 to 2022 (in minutes per day) – Текст: электронный // Statista – 2022 URL: <https://www.statista.com/statistics/663905/average-daily-time-spent-on-mobile-devices-us/>(дата обращения: 20.12.2022).
5. Number of available apps in the Apple App Store from 2008 to July 2022 – Текст: электронный // Statista – 2022 URL: <https://www.statista.com/statistics/268251/number-of-apps-in-the-itunes-app-store-since-2008/>(дата обращения: 15.03.2023).
6. Number of available applications in the Google Play Store from December 2010 to September 2022 – Текст: электронный // Statista – 2022 URL: <https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/>(дата обращения: 15.03.2023).
7. Delivery Club - Сервис доставки готовой еды и продуктов в России: офиц. сайт. - URL: <https://press.delivery-club.ru/mediakit>(дата обращения: 12.02.2023).

8. Маркет Деливери / Википедии — свободной энциклопедии — URL: <https://shorturl.at/cBEJS>(дата обращения: 12.02.2023).
9. Что заказывают в Яндекс.Еде – Текст: электронный // Яндекс – 2020 – URL: <https://yandex.ru/company/researches/2020/food>(дата обращения: 15.03.2023).
10. Что такое Яндекс.Еда – Текст: электронный // ЯндексСправка – 2022 – URL: <https://yandex.ru/support/eda/common/about.html>(дата обращения: 15.03.2023).
11. React Native - a JavaScript framework for building natively rendering mobile applications for iOS and Android platforms: офиц. сайт. - URL: <https://reactnative.dev/> (дата обращения: 12.02.2023).
12. React Native versions: офиц. сайт. - URL: <https://reactnative.dev/versions> (дата обращения: 10.01.2023).
13. Заяц, А. М. Введение в гибридные технологии разработки мобильных приложений : учебное пособие для спо / А. М. Заяц, Н. П. Васильев. — Санкт- Петербург : Лань, 2021. — 160 с. (дата обращения: 10.01.2023).
14. Семенчук, В. Мобильное приложение как инструмент бизнеса / В. Семенчук. — Москва : Альпина Паблишер, 2017. — 240 с. (дата обращения: 10.01.2023).
15. Expo Documentation: офиц. сайт. - URL: <https://docs.expo.io/> (дата обращения: 15.03.2023).
16. Cloud Firestore: офиц. сайт. - URL: <https://firebase.google.com/docs/firestore> (дата обращения: 04.04.2023).
17. Develop a Mobile App with React Native // Medium. – 2022. – URL: <https://javascript.plainenglish.io/develop-a-mobile-app-with-react-native-85d7228e6d2b>(дата обращения: 20.12.2022).
18. Danielsson W. React Native application development: A comparison between native. Android and React Native / W. Danielsson // Linköpings University – 2016 – P. 1-70. (дата обращения: 20.12.2022).

19. React Native Firebase: офиц. сайт. - URL: <https://rnfirebase.io/>. (дата обращения: 04.04.2023).
20. Beyshir, A. Cross-platform development with React Native / A. Beyshir // Uppsala University – 2016 – P. 1-32. – URL: <https://uu.diva-portal.org/smash/get/diva2:971240/FULLTEXT01.pdf>
21. Kaushik, V. React Native Application Development / V. Kaushik, K. Gupta, D. Gupta. // International Journal of Advanced Studies of Scientific Research. – 2018 – Vol. 4, No. 1, 2019. – URL: <https://ssrn.com/abstract=3330011> (дата обращения: 05.04.2023).
22. Nikita S. Prospects for Using React Native for Developing Cross-platform Mobile Applications / S. Nikita, S. Dmitriy, K. Nadezda // Central Ukrainian Scientific Bulletin. Technical Sciences – 2019 – P. 208-213. (дата обращения: 05.04.2023).
23. Гилка В. Методические рекомендации по подготовке, оформление выпускной квалификационной работы и преддипломной практики: учебно-методическое пособие / В. Гилка, А.С. Кузнецова; ВолГТУ – Волгоград: ВолГТУ, 2022. – 76 с.
24. 7 шагов к отличному результату – пользовательское тестирование // Spark – 2017 – URL: <https://shorturl.at/aktQT> (дата обращения: 05.05.2023).
25. Eisenman, B. Learning React Native / B. Eisenman. — California ; O'Reilly Media, 2017. — 432 p. (дата обращения: 10.01.2023). — ISBN: 9781491989142.
26. Daniel, K. Hands-On Mobile App Testing: A Guide for Mobile Testers and Anyone Involved in the Mobile App Business / K. Daniel. — Boston ; Addison-Wesley Professional, 2015. — 256 p. (дата обращения: 20.06.2023). — ISBN: 978-0134191713.

Приложение А – Справка о результатах проверки выпускной
квалификационной работы на наличие заимствований

Приложение Б – Техническое задание

Приложение В – Руководство системного программиста