# 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 прейскуранта, 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.

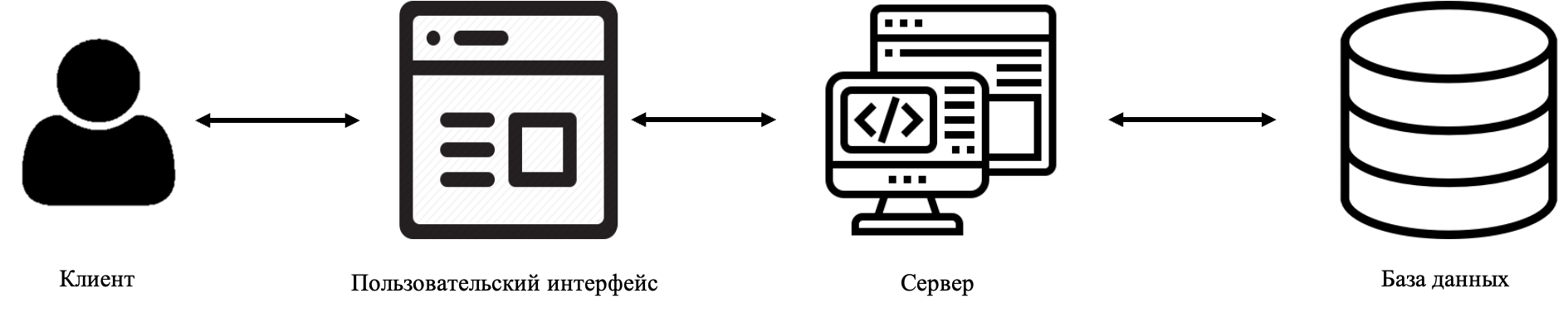


Рисунок 8 – Классическая схема работы клиент-серверного приложения

# 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 их 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 информацию 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

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

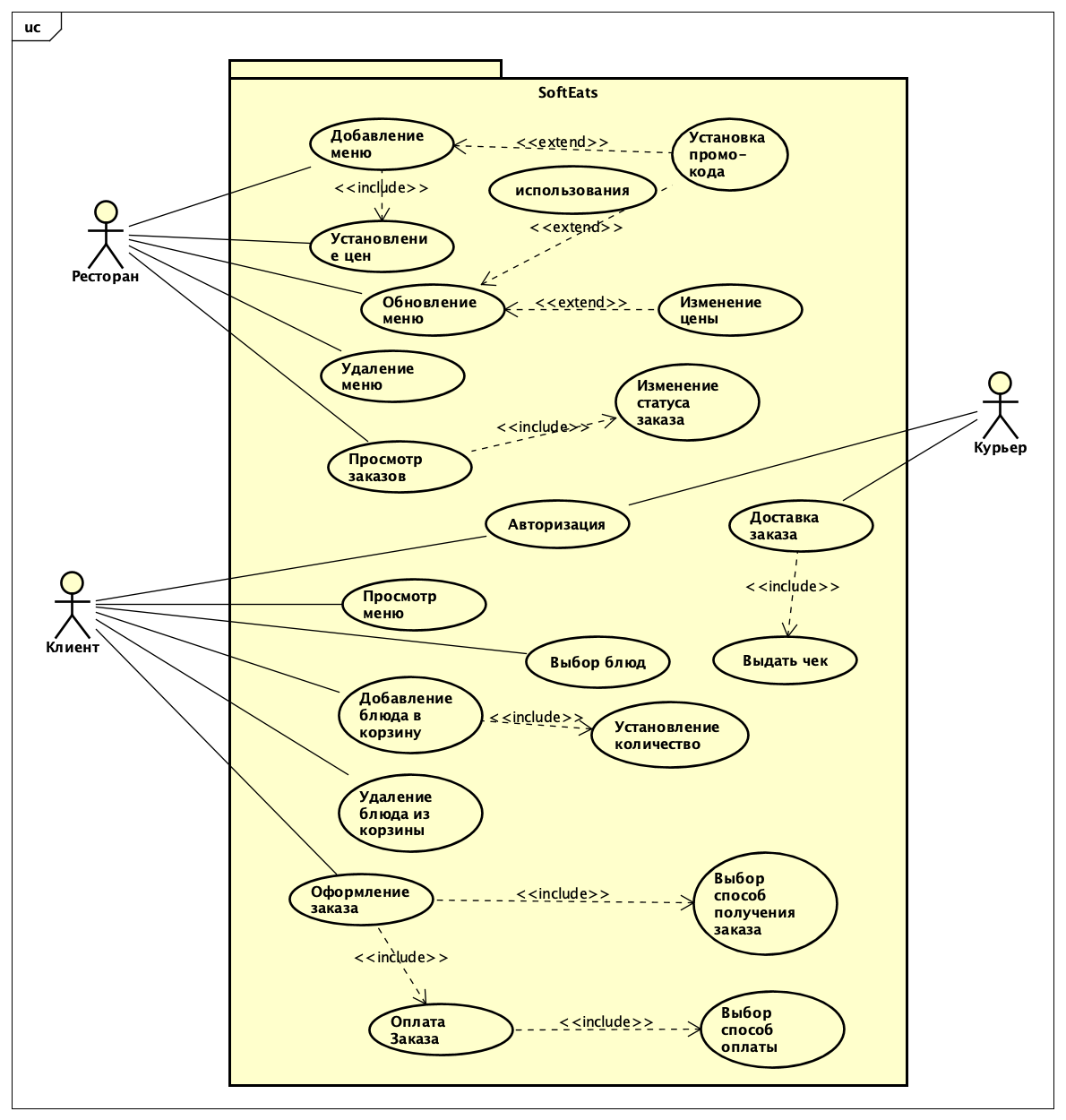


Рисунок 10 – Функциональные требования к приложению.

# 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.

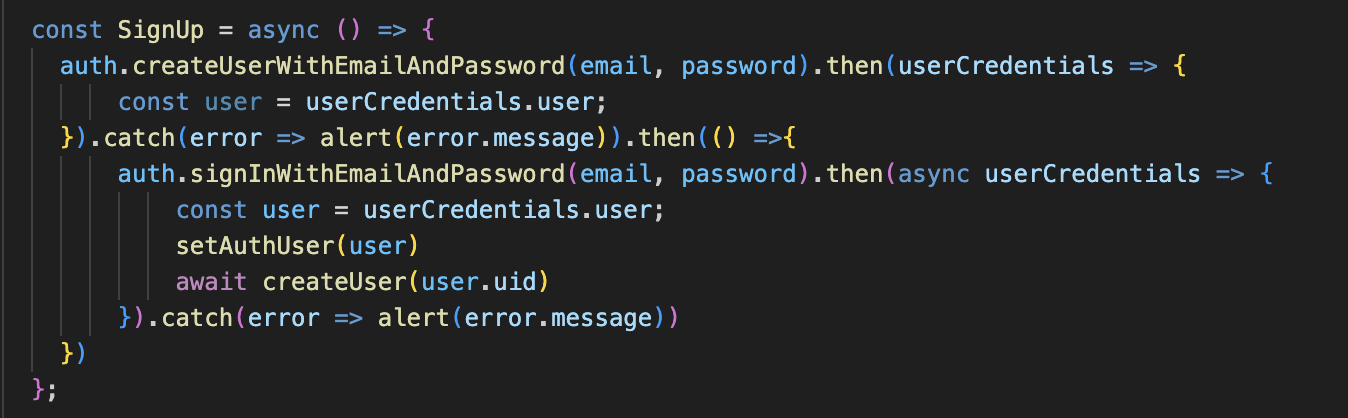


Рисунок 11 – Функция SignUp для создания нового пользователя

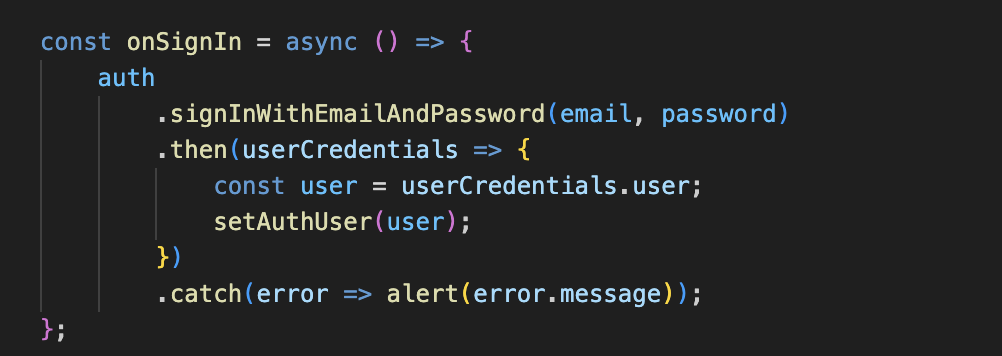


Рисунок 12 – Функция onSignIn для входа пользователя в систему



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 сеансами.

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.

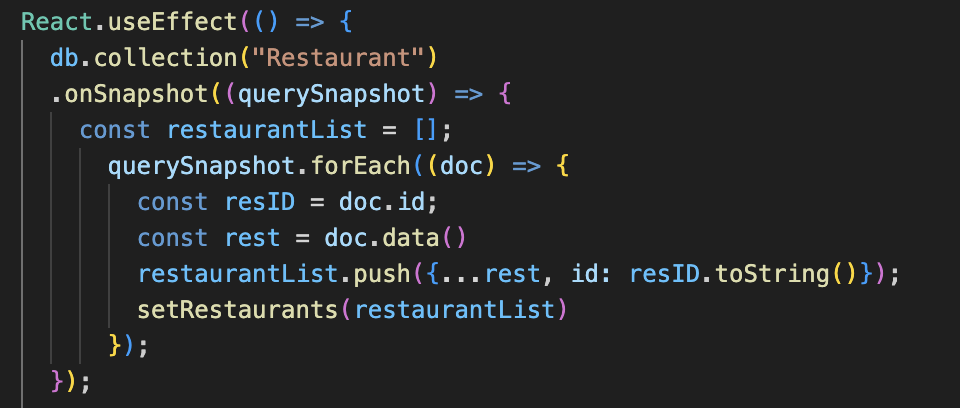


Рисунок 14 – Функция для извлечения ресторанов из базы данных

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.

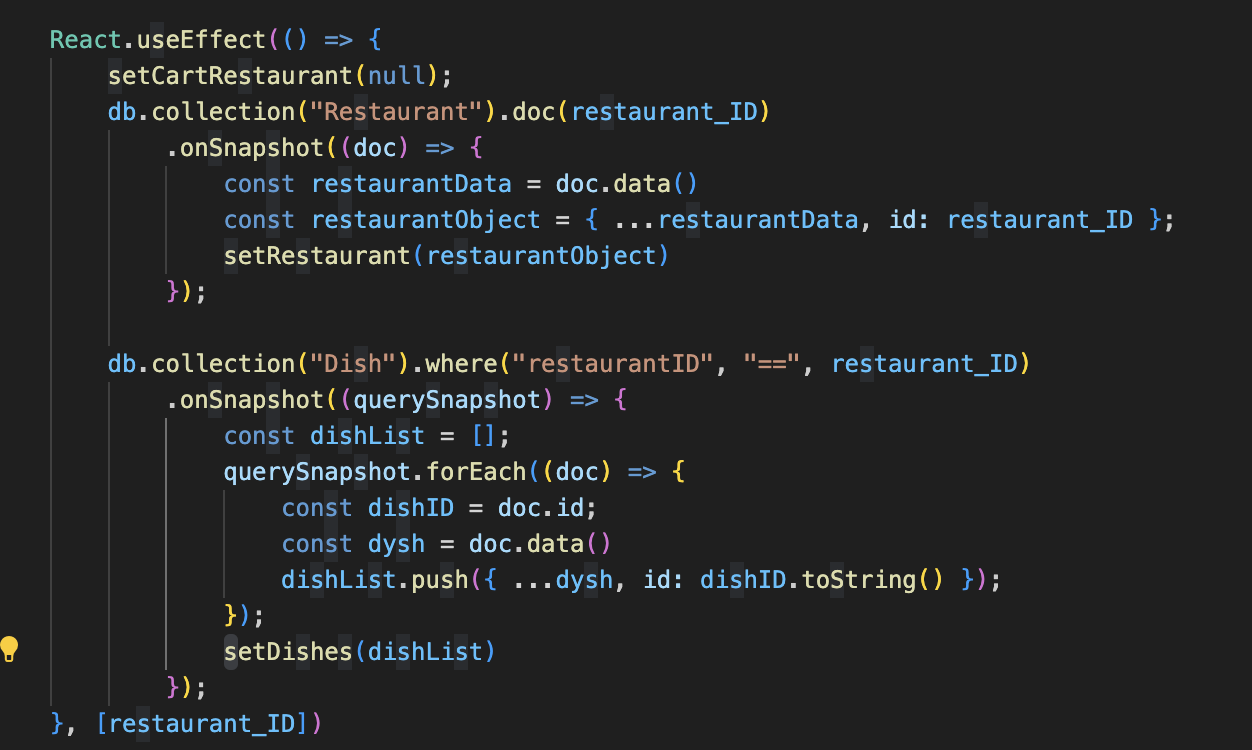


Рисунок 15 – Функция для извлечения информации о ресторане и его блюдах из базы данных ресторанов

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.

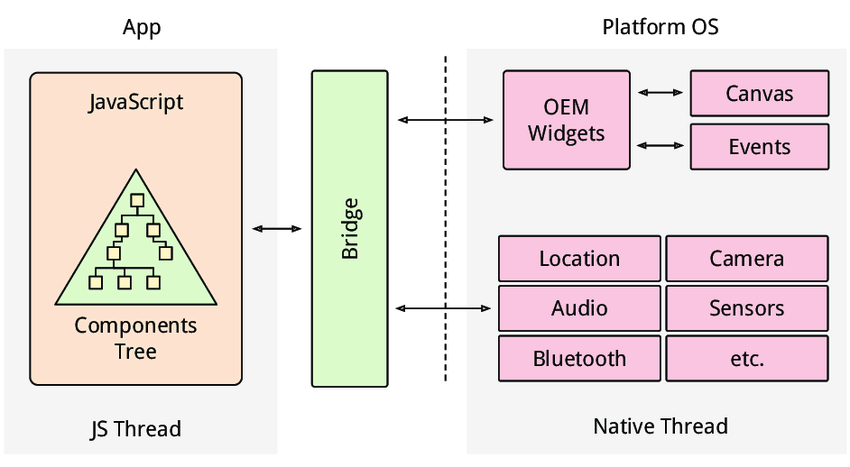


Рисунок 16 – Обзор архитектуры React Native.

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.

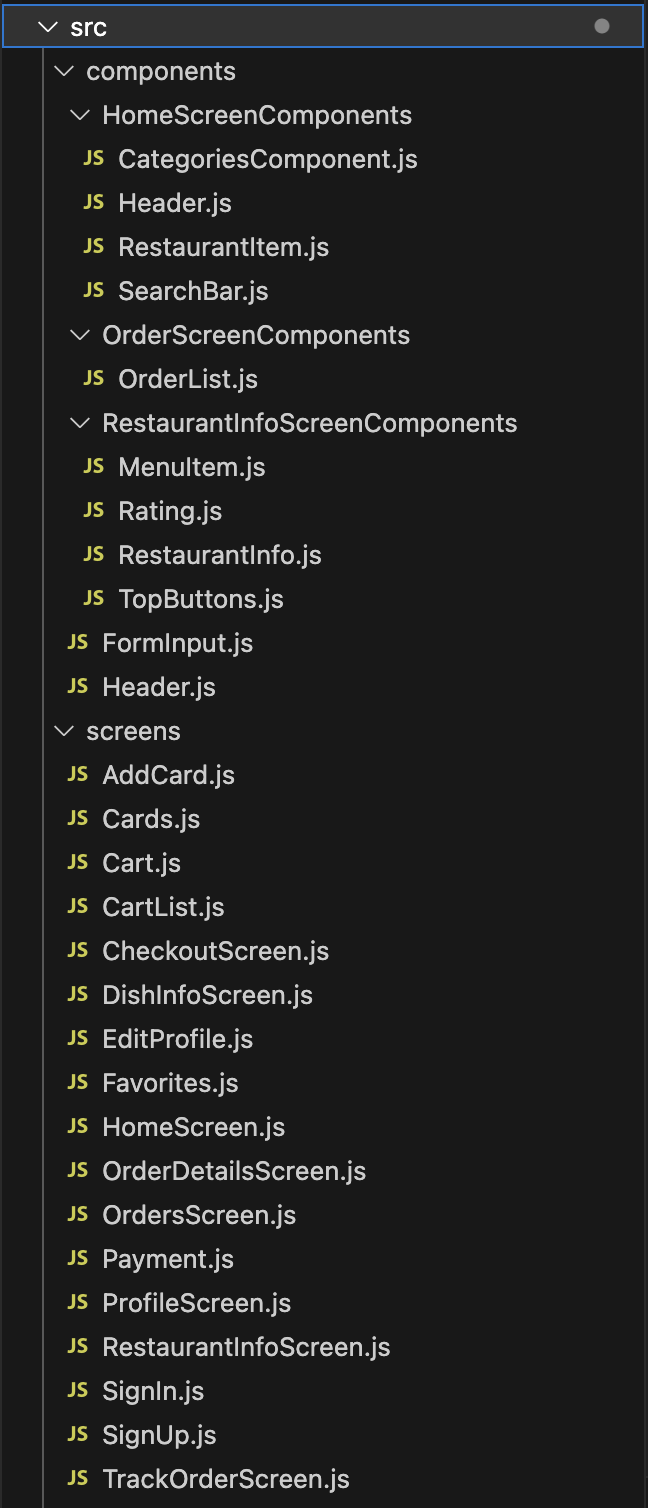


Рисунок 17 – Организация файлов пользовательского интерфейса React Native.

# 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.

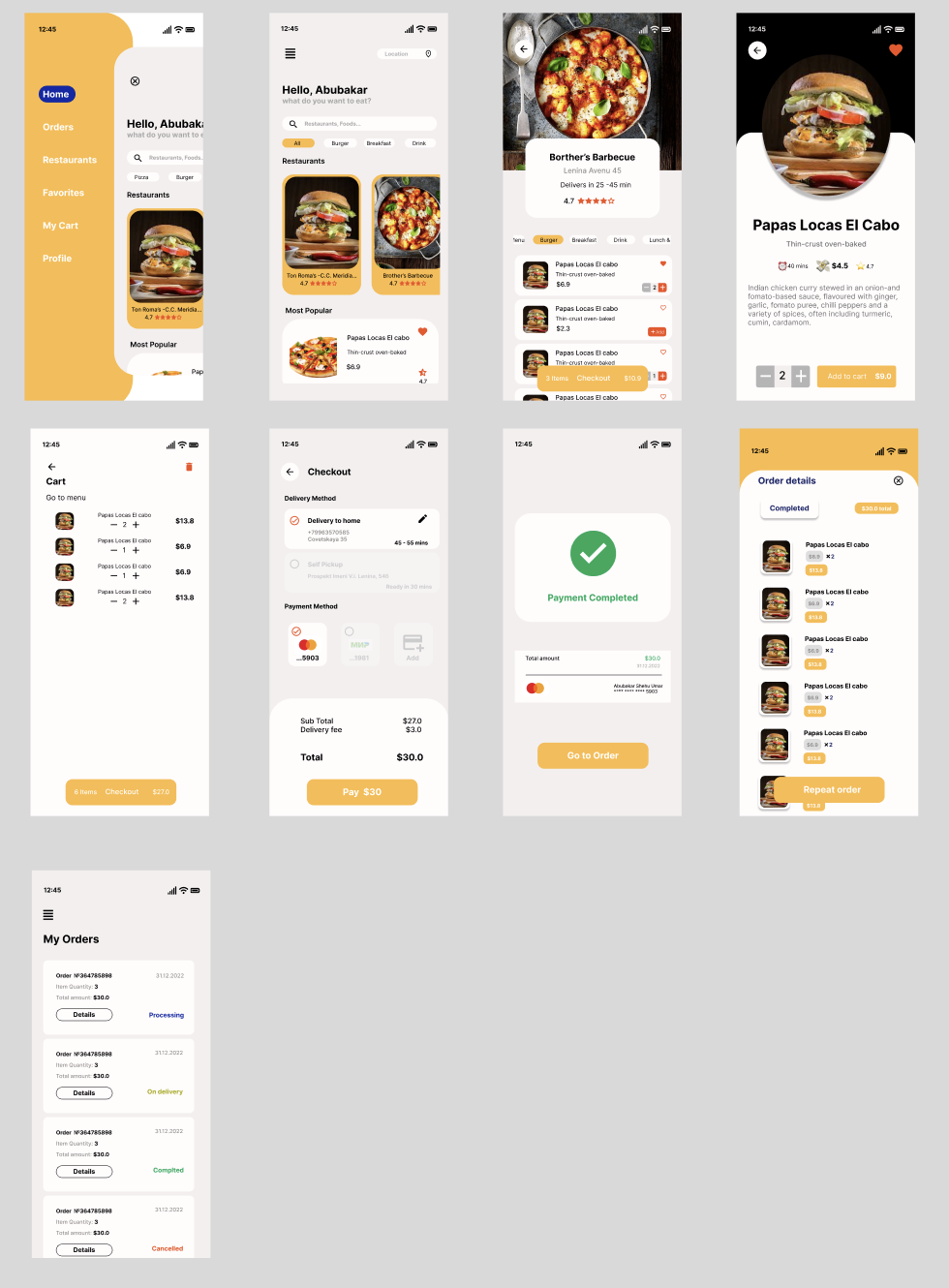


Рисунок 19 – Стилизация мобильного интерфейса.

# Выводы к третьей главе

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.