2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

Изучив теоретические аспекты, связанные с проектированием и разработкой системы, и выработав список требований, можно разбить систему на приложения. Каждое приложение представляет из себя отдельный программный продукт, который можно разбить на функциональные блоки. Каждый функциональный блок отвечает за ту или иную функцию приложения.

2.1 Мобильное приложение

2.1.1 Блок пользовательского интерфейса

Блок пользовательского интерфейса обеспечивает отображение данных на экране устройства и взаимодействие пользователя с приложением. В построении пользовательского интерфейса для приложения применяются UI компоненты фреймворка React Native, в составе которых различные кнопки, текст и прочие компоненты систем iOS и Android. Также он предоставляет возможность изменения параметров элементов и обработки событий, возникающих при взаимодействии пользователя с элементами интерфейса.

Система расположения UI элементов React Native работает таким образом, что при грамотном построении дерева компонентов пользовательский интерфейс получается адаптивным, то есть изменяют свой размер и положение в зависимости от размера и ориентации экрана.

2.1.2 Блок мобильной бизнес-логики

Блок мобильной бизнес-логики является связующим звеном между блоком интерфейса и блоками работы с хранилищем и сервером. Его задача заключается в обработке событий, генерируемых пользовательским интерфейсом (нажатия на кнопки, движение карты и т.д.), в генерировании запросов и обработке ответов сервера, а также в сохранении этих данных.

В мобильном приложении данная функциональность реализована с помощью библиотеки Redux Saga, которая позволяет проводить синхронные и асинхронные операции вне жизненного цикла компонента, из которого было получено событие. Также в ней присутствуют различные вспомогательные функции для упорядочивания обработки входящих событий, их сортировки и т.д.

2.1.3 Блок работы с хранилищем

Блок работы с хранилищем инкапсулирует в себе всю логику хранения и восстановления данных. Пользовательский интерфейс подписывается на

изменение данного хранилища, и при каждом его изменении компонент, зависящий от измененного значения хранилища, будет перерисован. Это гарантирует отображение на экране только актуальных данных.

Блок хранилища разделен на две части – хранилища в постоянной и оперативной памяти. За хранение данных в оперативной памяти отвечает библиотека Redux, легкая и быстрая, позволяющая модульно разделять хранилище на логические блоки. За хранение данных в постоянной памяти используется библиотека MMKV – хранилище типа ключ-значение, написанное на C++, она обеспечивает высокую скорость чтения и записи и не блокирует поток отрисовки графического интерфейса.

2.1.4 Блок работы с сервисами

Блок работы с сервером обеспечивает соединение между клиентом и сервером. Блок работает как с одиночными запросами на REST эндпоинты, так и с постоянным подключением через WebSocket к сервису подключения клиентов. Также он ответствен за авторизацию клиента, так как к каждому запросу прикрепляется авторизационный ключ, который однозначно идентифицирует пользователя на сервере.

Блок реализован с помощью библиотек Axios и WebSocket. Axios позволяет полностью контролировать содержимое REST запроса, например устанавливать заголовки и тело запроса, время ожидания ответа и так далее. WebSocket позволяет поддерживать постоянное подключение с сервером с помощью одноименного протокола, эта библиотека написана с использованием языков Java (Android) и Objective-C (iOS). Благодаря ей сервер может отправлять водителям уведомление с предложением взять новый заказ, в то же время постоянно принимая поток данных об обновленном местоположении пользователей с их устройств.

2.1.5 Блок вспомогательных утилит

Блок утилит представляет из себя набор различных классов и функций, в которых инкапсулирована работа со сторонними библиотеками, о которых не упоминалось ранее. Данный блок является дополнительным слоем абстракции, он позволяет уменьшить связанность кода и зависимость от определенных библиотек во всей кодовой базе. Например, если бы мы не использовали обертки над такими библиотеками, то при обновлении ее версии нам бы пришлось менять логику ее использования во всех местах программы. Такое решение является неоптимальным ввиду больших затрат по времени. В случае же использования утилиты-обертки мы можем сменить вызовы библиотечных функций только в этой утилите, и вся остальная программа продолжит корректное исполнение.

2.2 Сервис обработки данных о пользователе

2.2.1 Блок получения запроса клиента

Блок получения запроса клиента является слоем абстракции между веб-сервером (например Apache, Nginx) и бизнес-логикой. В данном месте заключены настройки REST эндпоинтов, проверка доступов пользователя и проверка передаваемых параметров. В данном блоке работа с зависимостями серверного фреймворка должна быть сосредоточена так, чтобы блок бизнес-логики был максимально переносимым. Это значит, что блок бизнес-логики не должен ничего знать о том, кто к нему обращается и в каком фреймворке этот блок используется. Например, бизнес-логика по генерированию авторизационного ключа не должна ничего знать о том, что этот код выполняется в контексте запроса пользователя к серверу, написанному на Spring. После выполнения всех проверок выполнение передается блоку бизнес-логики, либо же, при возникновении ошибки в проверяемых данных, данный блок вернет клиенту ошибку без последующей передачи управления блоку с логикой.

Для построения блока используются различные аннотации и классы библиотеки Spring Web. Например, для создания класса, инкапсулирующего в себе различное количество REST эндпоинтов, используется аннотация `@RestController`, а для создания пути, обрабатывающего запрос типа GET, используется аннотация `@GetMapping`.

2.2.2 Блок бизнес-логики

Блок бизнес-логики содержит в себе логику по созданию, изменению и удалению данных пользователя. Блок должен быть абстрагирован от зависимостей фреймворка и может использовать сторонние библиотеки, подключаемые через пакетный менеджер и не имеющие отношения к фреймворку. Обязанности данного блока:

- обмен данными с другими сервисами через блок работы с ними;
- проведение операций с базой данных через блок работы с ней;
- обработка запросов на вход и регистрацию;
- обработка запросов на получение данных о пользователе и истории его поездок;
- обработка запросов на создание экземпляра поездки в базе данных;
- обработка запросов на добавление новых платежных средств.

Блок реализован на языке Kotlin преимущественно без использования сторонних библиотек, исключением является библиотека JWT для генерации ключей авторизации пользователей.

2.2.3 Блок работы с базой данных

Блок работы с базой данных сосредотачивает в себе работу с конкретной реализацией СУБД, что позволяет абстрагировать бизнес-логику от той или иной СУБД, благодаря этому подходу в блоках с логикой мы можем использовать различные системы хранения данных, предоставляя использующему классу лишь унифицированный интерфейс взаимодействия.

Блок построен с использованием библиотеки Hibernate и драйвера для работы с PostgreSQL. Hibernate – библиотека для виртуальной машины JVM, предназначенная для решения задач объектно-реляционного отображения (ORM), самая популярная реализация спецификации Java Persistence Adapter. Объектно-реляционного отображение – технология программирования, суть которой заключается в создании «виртуальной объектной базы данных». Благодаря этой технологии разработчики могут использовать язык программирования, с которым им удобно работать с базой данных, вместо написания операторов SQL или хранимых процедур. Это может значительно ускорить разработку приложений. ORM также позволяет переключать приложение между различными реляционными базами данных. Например, приложение может быть переключено с MySQL на PostgreSQL с минимальными изменениями кода.

2.2.4 Блок работы с сервисами системы

Блок работы с сервисами системы обеспечивает связь с другими микросервисами проекта. Например, при поиске водителя для клиента этот блок выполнит запрос на сервис подключения клиентов в реальном времени и запросит список ближайших к клиенту водителей.

Блок использует для связи с сервисами спецификацию RPC. RPC (Remote Procedure Call), или же удаленный вызов процедур – технология, позволяющий программам вызывать функции и процедуры в другом адресном пространстве (на удаленном узле, либо же на данном узле в другом процессе). Реализация данной технологии включает в себя протокол для обмена сообщениями между узлами и язык сериализации данных в сообщение. В качестве реализации технологии RPC был выбран вариант от компании Google – gRPC, ввиду открытости его исходного кода, а также совместимостью практически со всеми популярными на данный момент платформами и технологиями. В качестве транспорта между устройствами gRPC использует HTTP/2, являющийся улучшенной версией наиболее распространенного HTTP/1.1. gRPC предоставляет такие функции как аутентификация, двунаправленная потоковая передача и управление потоком, блокирующие или неблокирующие привязки (stubs), а также отмена и тайм-ауты. Генерирует кроссплатформенные привязки клиента и сервера для многих языков.

2.3 Сервис оплаты

2.3.1 Блок получения запроса клиента

Логика работы данного блока идентична этому же блоку сервиса обработки данных о пользователе с тем лишь исключением, что для построения блока используются встроенные во фреймворк Nest утилиты. Например, вместо `@RestController` Nest предоставляет аннотацию `@Controller`, а вместо `@GetMapping` – аннотацию `@Get`.

2.3.2 Блок бизнес-логики

Блок бизнес-логики получает данные о типе платежа, его размере и платежном средстве, используя их для проведения транзакции в сети Stripe.

Stripe – американская технологическая компания, разрабатывающая решения для приёма и обработки электронных платежей. Предоставляет утилиты для интеграции с различными языками программирования, в том числе и TypeScript. В числе поддерживаемых способов оплаты находятся банковские карты (Visa, Mastercard, American Express), мобильные платежные средства (Apple Pay, Google Pay), а также различные виды расчета с отсроченным платежом (Klarna, AfterPay).

Мобильный клиент будет запрашивать у данного микросервиса создание сессии оплаты Stripe, а при получении данной сессии будет подтверждать оплату напрямую на серверах платежного шлюза Stripe.

2.3.3 Блок работы с сервисом обработки данных

Логика работы данного блока идентична этому же блоку сервиса обработки данных о пользователе с тем лишь исключением, что для реализации используются библиотеки @grpc/grpc-js и @grpc/proto-loader, которые являются официальными утилитами от Google. Фреймворк Nest имеет встроенную поддержку данных библиотек “из коробки”.

2.4 Сервис подключения клиентов в реальном времени

2.4.1 Блок подключения клиента

Блок подключения клиента представляет из себя шлюз для подключения клиентов мобильного приложения к серверу по протоколу WebSocket. Это необходимо для двухстороннего обмена сообщения с клиентами в режиме реального времени ввиду того, что у клиентов отсутствуют публичные IP адреса, и сервера не могут напрямую посылать запрос на мобильный клиент. Также передача данных через протокол WebSocket значительно упрощает и

ускоряет работу сервера в сравнении с часто повторяющейся генерацией REST запросов на определенный маршрут (long-polling).

Такой тип подключения будет использоваться для сбора данных о местоположении машин и клиентов в реальном времени, а также оповещении водителя и клиента об изменении статуса заказа. Реализация WebSocket соединения написана с использованием библиотеки socket.io, являющейся самой популярной и эффективной для платформы NodeJS.

2.4.2 Блок бизнес-логики

Блок бизнес-логики будет отвечать за сбор данных о текущей геопозиции пользователя и трансформации географических координат в уникальные координаты секторов, получаемые из библиотеки S2.

S2 – библиотека от Google, проецирующая географические координаты на геоид и вычисляющая ID сектора исходя из переданных координат. При вычислении идентификатора сектора указывается размер сектора. Размер сектора – длина стороны квадрата, которыми будет разбит геоид. Например, при размере сектора в 1 км, геоид будет разбит на множество квадратов, имеющих размер 1км в ширину и 1 км в длину. Соответственно люди, находящиеся рядом и имеющие схожие географические координаты, будут иметь одинаковые либо близкие друг к другу идентификаторы секторов.

Данная библиотека упрощает работу по поиску ближайших к пользователю водителей. Поиск без этой библиотеки предполагает сравнение координат, учет ширины и долготы, учет размера сектора в градусах (а не в километрах). Все это происходило бы прямо во время поиска водителя, что значительно замедляет этот процесс. Однако, используя библиотеку S2, задача поиска ближайшего водителя сводится к поиску в массиве подключенных водителей объект, в котором модуль разницы между ID сектора водителя и ID сектора пользователя минимален или равен нулю.

2.4.3 Блок работы с сервисами системы

Логика работы данного блока идентична этому же блоку сервиса оплаты.

2.5 Сервис просчета стоимости и маршрута

2.5.1 Блок получения запроса клиента

Логика работы данного блока идентична этому же блоку сервиса оплаты.

2.5.2 Блок бизнес-логики

Блок бизнес-логики, используя выбранные пользователем местоположения начала и конца поездки, просчитывает маршрут и время в пути используя блок работы с сервисами Google. Исходя из времени в пути и загруженности дорог блок просчитывает стоимость поездки и возвращает результат в виде объекта, содержащего путь на картах Google, стоимость поездки и время в пути.

Блок написан на языке TypeScript без использования сторонних библиотек.

2.5.3 Блок работы с сервисами Google

Блок работы с сервисами Google представляет из себя набор классов и утилит, которые инкапсулируют в себе работу с Google Maps API. Данный блок позволяет:

- получать кратчайший маршрут используя Directions API;
- получать данные об адресе по географическим координатам используя Geocoding API;
- получать расстояние между точками используя Distance Matrix API.

Блок написан на языке TypeScript с использованием официальной библиотеки от Google – @googlemaps/google-maps-services-js. Данная библиотека полностью типизирована и предоставляет все необходимые обертки и параметры для получения необходимого результата.