Техническое задание: MVP платформы доставки еды с маркетинговой аналитикой

Цели

- **Запуск собственной платформы доставки:** Создать минимально жизнеспособный продукт (MVP) цифровой платформы для компании **APPETIT**, включающей брендированный веб-сайт и мобильные приложения (Android и iOS). Платформа должна позволить принимать онлайн-заказы напрямую, без участия сторонних агрегаторов, что обеспечит поступление всей выручки напрямую компании и усилит бренд. Решение должно предоставить возможность управлять всеми ключевыми процессами доставки в единой системе – **меню, заказами, оплатами и маркетинговыми акциями**.
- **Маркетинговая аналитика с первого дня:** MVP должен содержать базовые средства аналитики, позволяющие оценивать эффективность работы и маркетинга. В системе необходимо собирать и отображать основные метрики: источники заказов (например, по UTM-меткам рекламных кампаний), общую выручку, средний чек заказа, процент повторных заказов постоянных клиентов и пр. Эти данные помогут **принимать обоснованные решения** по продвижению продукта и оптимизации затрат. Архитектура MVP должна быть спроектирована с учётом дальнейшего расширения аналитических возможностей и интеграции с внешними сервисами аналитики и рекламы (сквозная аналитика).

Пользователи

- **Клиент (покупатель):** Конечные пользователи платформы – клиенты APPETIT, заказывающие еду через мобильное приложение или веб-сайт. Для них важны удобство и скорость заказа, привлекательный интерфейс и надежность. Клиенты смогут просматривать меню, настраивать блюда, оформлять заказ (с выбором доставки или самовывоза) и использовать маркетинговые акции. Они ожидают получать подтверждение и информацию о статусе своего заказа, а также персональные предложения (push-уведомления, промо-акции).
- **Администратор (менеджер компании):** Сотрудники APPETIT, ответственные за управление платформой – например, менеджер ресторана, маркетолог или администратор. Через административную панель они будут **управлять содержанием** (меню, категориями, акциями), **обрабатывать заказы** и контролировать их выполнение, а также **отслеживать аналитику** и ключевые показатели. Администратор имеет доступ к разделам управления меню, заказами, маркетинговыми инструментами (промокоды, баннеры, рассылки) и отчетности. Предусмотрена защита админ-доступа (авторизация) для безопасности данных.

Функции

Функциональность для клиентов (мобильное приложение и сайт)

- **Просмотр меню:** Клиент может просматривать каталог блюд, организованный по категориям (разделы меню). Для каждого блюда отображается фотография, название, описание, цена и категория/раздел меню. Возможна фильтрация или поиск блюд по названию (для удобства навигации).
- **Детали блюд и модификации:** При выборе блюда пользователь видит страницу с подробным описанием и фото. Реализована поддержка **модификаторов** – дополнительных опций или вариантов для блюда (например, выбор размера порции, добавление топпингов или гарниров). Пользователь может настраивать блюдо, выбирая необходимые опции перед добавлением в корзину.
- **Корзина и промокоды:** Пользовательская корзина отображает добавленные позиции, их количество, цену и итоговую сумму. Клиент может изменять содержимое корзины (удалять или менять количество позиций). На этапе оформления заказа предусмотрено поле для ввода **промокода** – при вводе действующего кода рассчитывается скидка на заказ. Также могут отображаться текущие акции или баннеры с предложениями (например, на главной странице сайта или в разделе акций приложения).
- **Оформление заказа:** Клиент выбирает способ получения – **доставка** (курьером на указанный адрес) или **самовывоз** (из пункта выдачи/ресторана). Для доставки пользователь вводит адрес доставки (с возможностью указать ориентир, подъезд и т.п.; в будущем планируется автоподстановка адреса по геолокации или через API карт). Для самовывоза – выбирает филиал/адрес ресторана (если у компании несколько точек) и, возможно, желаемое время. Клиент вводит контактные данные (имя, телефон; при регистрации эти данные могут подтягиваться автоматически).
- **Оплата заказа:** На MVP этапе возможно несколько подходов к оплате:

    · **Оплата при получении:** наличными или картой курьеру/при самовывозе (основной вариант на старте, не требующий интеграции с онлайн-платежами).

- **Онлайн-оплата (опционально в MVP):** возможность оплатить заказ онлайн банковской картой через защищенный платежный шлюз. Если интеграция платежей не реализована в MVP, архитектура должна предусматривать её подключение в будущем (например, через API платежных систем). На этапе MVP достаточно обеспечить возможность отметить способ оплаты («оплата при получении»), с возможностью расширения до онлайн-оплаты позже.
- **Подтверждение и статус заказа:** После оформления заказа пользователь получает подтверждение (на экране приложения/сайта отображается, что заказ принят, с номером заказа). Предусматривается отправка уведомления (push и/или SMS/email) о создании заказа. В личном кабинете или разделе заказов пользователь может просматривать статус текущего заказа (например, «принят», «готовится», «в пути», «доставлен»). Обновление статуса может осуществляться в реальном времени, если реализована соответствующая

логика. *Для MVP минимально достаточно статуса «Заказ оформлен».* В будущем, при интеграции с курьерскими сервисами или при наличии интерфейса курьера, статусная модель будет расширена.

- **Регистрация и профиль (желательно):** Пользователю предлагается зарегистрироваться или войти в аккаунт для удобства повторных заказов. Регистрация может быть по номеру телефона (с SMS-кодом подтверждения), email + пароль либо через социальные сети. **Профиль пользователя** хранит контактные данные, историю заказов, предпочтения. Для MVP допустимо упрощенное хранение данных (например, привязка заказов к номеру телефона без полноценной авторизации), однако для отслеживания **повторных заказов** и персонализации в аналитике рекомендуется реализовать базовую форму регистрации. В профиле пользователь сможет просмотреть историю прошлых заказов и при необходимости быстро повторить заказ.
- **Получение уведомлений и акций:** Клиент может получать **push-уведомления** о специальных предложениях, персональных скидках или статусе заказа. Например, отправка push со скидкой ко дню рождения или о новых поступлениях в меню[6]. В приложении и на сайте также могут отображаться баннеры акций (например, сезонные предложения) в соответствующих разделах.

Функциональность для администратора (панель управления)

- **Управление меню и каталогом:** Администратор имеет раздел для работы с меню. Он может создавать, редактировать и удалять **категории** меню (разделы, например «Пицца», «Суши», «Напитки») и **позиции блюд** внутри категорий. Для каждого блюда админ заполняет: название, описание, цену, категорию, загружает фотографию. Система должна поддерживать добавление **модификаторов к блюдам** – администратор задаёт варианты дополнений/размеров и их влияние на цену (например, «Размер: Большой +200 тг» или «Добавить сыр +50 тг»). Все изменения меню (новые блюда, цены, наличие) должны оперативно отражаться на сайте и в приложении. *Опционально*, администратор может управлять наличием/остатками блюд (например, пометка «нет в наличии»), либо это может автоматически приходить из интегрированной POS-системы (в будущем).
- **Управление заказами:** Админ-панель предоставляет **список поступивших заказов** в реальном времени. Новый заказ появляется в системе с пометкой «Новый»; администратор (или операционист) может открыть детали заказа: содержимое (блюда, количество), адрес доставки или точку самовывоза, контакты клиента, комментарии. Предусмотрены инструменты для обработки: напр. отметить заказ как **подтверждённый/принятый**, передать на кухню (в реальном MVP это может быть внешним процессом), и после приготовления – отметить **отправленным/готовым к выдаче**, а затем **завершённым**. Смена статусов может вручную выполняться администратором; статусы синхронизируются с приложением клиента для информирования. Также админ может редактировать заказ (например, изменить статус, время доставки, добавить комментарий). Желательно реализовать **уведомления для администратора** о новых заказах (звуковой сигнал или всплывающее окно в панели) для своевременной реакции.

- **Аналитика и отчёты:** В админ-панели доступен раздел **«Аналитика»** или дашборд, где отображаются основные показатели работы службы доставки:

  - Количество заказов за выбранный период (день, неделя, месяц).
  - **Выручка** (суммарная) за период.
  - **Средний чек** (средняя сумма заказа).
  - **Источники заказов:** распределение заказов по каналам привлечения (например, сколько заказов пришло через прямой заход, сколько – по рекламе в Instagram, сколько – через поиск и т.д., на основе данных, собранных по UTM-меткам или реферальным ссылкам).
  - **Повторные заказы:** метрика лояльности – доля заказов от клиентов, которые заказывают не впервые. Если реализован механизм авторизации, можно точно считать повторные заказы; либо по номеру телефона/e-mail клиента.
  - *Дополнительно, необязательно для MVP:* количество новых пользователей, воронка оформления заказа (сколько людей добавило в корзину vs. сколько оформили заказ), наиболее популярные блюда, и т.д.

Аналитика может отображаться в виде цифр-индикаторов, графиков и таблиц. Должна быть возможность выбирать период анализа (дату «с... по...», пресеты «за месяц», «за неделю»). **На этапе MVP достаточно простого дашборда** с ключевыми цифрами. Архитектурно следует заложить возможность расширения аналитики (например, интеграции сторонних BI-инструментов или добавления новых метрик).

  - **Маркетинговые инструменты:** Администратор управляет встроенными инструментами продвижения:
  - **Промокоды и скидки:** Возможность создавать промо-акции – задавать код (набор символов) и параметры скидки (процент или фиксированная сумма, срок действия, ограничения по минимальной сумме или количеству использований и т.д.). Админ может активировать/деактивировать промокоды, отслеживать, сколько раз они использованы.
  - **Баннеры и акции:** В панели должна быть возможность добавлять рекламные баннеры/акции, которые будут отображаться пользователям на сайте или в приложении. Например, загрузить изображение баннера, указать текст акции, срок проведения, и разместить на главной странице сайта или в разделе предложений приложения.
  - **Push-уведомления и рассылки:** Инструмент для отправки push-уведомлений пользователям приложения. Администратор может составить сообщение (например, акция «2 по цене 1 на пиццу в эти выходные»), выбрать аудиторию (всем пользователям или определённому сегменту – например, только тем, кто не заказывал месяц, или только тем, у кого день рождения и т.п. – *такие сегменты можно добавить позднее*), и

отправить сразу либо запланировать рассылку на определённое время. На MVP этапе достаточно возможности отправить уведомление всем пользователям вручную.

- **Интеграция с аналитикой рекламы:** На старте платформа должна **отслеживать источник заказов** для оценки эффективности рекламы. Это требует учета UTM-меток: если клиент пришёл по рекламной ссылке с метками, система фиксирует эти метки при оформлении заказа (сохраняет их вместе с заказом). Также на сайт должен быть внедрён счетчик/код веб-аналитики (Google Analytics 4, Яндекс Метрика и пр.), а в мобильном приложении – настроены события аналитики (просмотр блюда, оформление заказа и др.). Эти данные будут использоваться маркетологами для анализа. На стороне админ-панели можно предусмотреть простейший отчет по UTM-меткам (например, разбиение заказов по utm_source). Глубокую сквозную аналитику планируется реализовать в будущем, но **MVP должен быть готов к интеграции** с инструментами аналитики (возможность добавить tracking ID, передавать нужные события и пр.).
- **Разграничение прав (опционально):** В MVP достаточно одной роли администратора с полными правами. В дальнейшем, возможно, появится необходимость в различных ролях (например, «Контент-менеджер» только для меню, «Маркетолог» только для акций, «Оператор» только для обработки заказов). Архитектура должна допускать добавление ролей и прав доступа. На начальном этапе, если требуется, можно реализовать хотя бы разделение «Администратор» и «Оператор», где оператор может менять статусы заказов, но не редактировать меню и промо. Однако, это не строго обязательно для MVP.

Архитектура MVP

Архитектура решения строится по многоуровневому клиент-серверному принципу, обеспечивая модульность, масштабируемость и готовность к интеграциям. Основные компоненты системы:

- **Мобильные приложения (Android и iOS):** Клиентские приложения, устанавливаемые пользователями. Реализуют интерфейс для заказа: отображают меню, корзину, выполняют оформление заказа и взаимодействуют с сервером через сеть. Приложения могут быть реализованы нативно (Kotlin/Java для Android, Swift/Obj-C для iOS) либо с использованием кроссплатформенного фреймворка (Flutter, React Native и др.) для ускорения разработки. Приложения получают данные (меню, статус заказа и пр.) из backend API и отправляют туда действия пользователя (новые заказы, обновление профиля и т.д.). Также приложения отвечают за получение push-уведомлений (через интеграцию с сервисом уведомлений, например Firebase Cloud Messaging).
- **Веб-сайт (Frontend веб-клиент):** Брендированный сайт доставки, доступный в браузере. Реализует тот же пользовательский функционал, что и приложение: просмотр меню, оформление заказа, личный кабинет. Сайт должен быть адаптивным (responsive), чтобы им было удобно пользоваться как с

компьютера, так и с телефона. Технологически сайт может быть реализован как одностраничное приложение (SPA) на современном фреймворке (React, Angular, Vue) либо на серверной шаблонной генерации – по усмотрению команды разработки, но в любом случае он взаимодействует с backend через API для получения/отправки данных. На сайт будет встроен код веб-аналитики (напр., GA) для отслеживания трафика и конверсий.

- **Серверная часть (Backend API):** Центральный компонент архитектуры, отвечающий за бизнес-логику и хранение данных. Backend реализуется как веб-сервер с RESTful API (или GraphQL) интерфейсом. Основные функции backend:

  · Обработка запросов от мобильного приложения и сайта (получение меню, регистрация заказа, авторизация пользователей, применение промокода и т.п.).

  · Управление данными в базе данных (создание/чтение/редактирование записей о блюдах, заказах, пользователях, промоакциях и т.д.).

  · Реализация бизнес-правил (например, проверка валидности промокода, расчет суммы заказа с учетом скидок, проверка времени работы ресторана для доставки).

  · Генерация аналитических агрегатов: backend может вычислять некоторые показатели (например, средний чек, считать количество заказов за день) либо отдавать сырые данные для отображения и передавать их во внешние системы анализа.

  · Обеспечение интеграции с внешними сервисами: отправка запросов в API сторонних сервисов при необходимости (например, вызов API геокодирования адреса, взаимодействие с платежным шлюзом, отправка уведомлений через Firebase API и др.).

Backend должен быть спроектирован модульно, чтобы облегчить добавление новых функций. Например, отдельные модули/сервисы: модуль заказа, модуль меню, модуль пользователей, модуль аналитики. Это позволит в будущем выделять их в отдельные микросервисы при росте нагрузки.

· **База данных:** Системе требуется централизованное хранилище данных. Рекомендуется использовать реляционную СУБД (PostgreSQL, MySQL/MariaDB или аналог) для надежности и чтобы выдержать сложные запросы (например, аналитические выборки). В БД хранятся:

· Справочники меню: таблицы категорий, блюд, модификаторов.

· Операционные данные: заказы (связанные с позицями блюд, статусами, клиентом), транзакции оплат (если онлайн-оплата), промокоды и их параметры, и т.д.

- Данные пользователей: аккаунты, хешированные пароли (если используются), контактная информация, история заказов.

- Логи и аналитические данные: например, таблица для хранения источников заказа (UTM-меток, реферера) связанная с заказом.

Проектирование базы должно обеспечить целостность (например, заказ связан с пользователем и позициями через внешние ключи). Использование ORM библиотеки возможно для ускорения разработки. Также, для некоторых нефункциональных данных (как логи действий или кеш сессий) можно применять NoSQL-хранилище или in-memory хранилище, но в MVP достаточно реляционной БД.

- **Административная панель:** Веб-интерфейс для администраторов, который фактически является клиентским приложением, работающим поверх Backend API (может быть реализован как отдельное SPA-приложение или как часть сайта, защищенная авторизацией). Панель предоставляет доступ к функциям администрирования (меню, заказы, промо, аналитика). Архитектурно админ-панель не контактирует напрямую с базой, а использует тот же слой API, что и мобильное приложение/сайт, но с привилегированными правами. Это упрощает логику (один backend для всех типов клиентов) и готовит систему к выделению сервисов. Админ-панель должна быть защищена – например, с помощью JWT-токенов или сессий с проверкой ролей на backend.
- **Сервис уведомлений:** Для реализации push-уведомлений используется внешняя облачная служба – например, **Firebase Cloud Messaging (FCM)** для кросс-платформенных push. Мобильные приложения регистрируются в FCM и получают device token; Backend хранит эти токены и при необходимости отправляет через FCM API сообщение нужным пользователям. Само уведомление может инициироваться администратором (маркетинговая рассылка) или автоматическими событиями (смена статуса заказа – тогда backend триггерит уведомление). FCM выбран за простоту и бесплатность, также он входит в инфраструктуру Firebase, которая может быть использована и для аналитики приложения. Для SMS-уведомлений (например, при оформлении заказа) можно интегрировать SMS-шлюз, но это опционально.
- **Интеграция с аналитическими сервисами:** На уровне архитектуры предусмотрено подключение внешних сервисов аналитики и маркетинга. Например, на веб-сайт вставлен код **Google Analytics 4** или Яндекс.Метрики для отслеживания визитов и конверсий. Мобильное приложение может использовать Firebase Analytics или аналогичный SDK для сбора событий (просмотр экрана, добавление в корзину, покупка и т.д.). Кроме того, backend может передавать сведения о совершённых заказах во внешние системы через API (при реализации сквозной аналитики). **UTM-метки** от рекламы, попадающие в ссылки, передаются на backend при оформлении заказа и сохраняются – это позволит сопоставлять заказы с рекламными кампаниями. В основе такого механизма идентификации источника лежат **client_id и UTM-метки** пользователя, которые связывают посещения с последующими заказами. Архитектура должна быть способна принимать эти идентификаторы и хранить их для дальнейшего анализа.

- **Безопасность и масштабируемость:** Все взаимодействие клиентов с сервером происходит по защищенному протоколу **HTTPS** (для защиты персональных данных и платежной информации). В системе реализована авторизация и аутентификация пользователей и администраторов (например, с использованием JWT токенов или OAuth2, особое внимание к защите административного доступа). Данные в базе (особенно чувствительные, как пароли, токены оплаты) хранятся в зашифрованном/хешированном виде. MVP развертывается вначале на ограниченной инфраструктуре, но с расчётом на рост нагрузки – рекомендуется использовать масштабируемые решения (например, контейнеризация Docker + Kubernetes, или облачные сервисы авто-масштабирования) по мере роста. Логирование и мониторинг сервера следует заложить с начала (писать логи действий, ошибок), чтобы облегчить отладку и сопровождение. При увеличении нагрузки архитектура может быть развёрнута на несколько экземпляров backend (stateless-сервис) за балансировщиком. Также возможен переход на разделение сервисов (автономный сервис аналитики, сервис уведомлений и т.п.) в будущем, без кардинальной перестройки системы, благодаря изначально модульному подходу.

Приоритеты

Высокий приоритет (функционал MVP): Основные функции, необходимые для демонстрации работоспособности платформы и ценности продукта: - Реализация полного цикла заказа: от просмотра меню и добавления блюд в корзину до оформления заказа с нужными параметрами (доставка/самовывоз, контакты) и сохранения его в системе. Это ядро MVP. - База данных с основными сущностями (меню, заказы, пользователи, промокоды) и базовая бизнес-логика (расчет суммы заказа, применение скидок, смена статуса заказа вручную). - Клиентские приложения (мобильное и/или веб) с пользовательским UI для меню и оформления заказа. Хотя желательно иметь и сайт, и мобильное приложение, в условиях ограничения времени хакатона команда может сначала сосредоточиться на одной платформе (например, веб-PWA или Android-приложении), но с возможностью расширения на вторую платформу. - Административный интерфейс для управления меню и просмотра поступающих заказов. Минимум – админ может видеть новые заказы и менять их статус (что уже позволит использовать MVP в тестовом режиме). Также обязательно – функция добавления/редактирования позиций меню через админку, так как это ключевая потребность бизнеса. - Маркетинговые основы: поддержка промокодов при оформлении заказа и отображение баннеров акций (статично заданных) – эти функции должны работать в MVP, чтобы продемонстрировать ценность маркетинговых инструментов. - Базовая аналитика: хотя бы статические показатели на админ-панели (число заказов, сумма выручки, средний чек за день/неделю). Они могут быть выведены заглушечно или считаться простым запросом к БД. Важно показать, что система уже **собирает данные** и визуализирует их для бизнеса. - Интеграция с Google Analytics (или аналогом) на базовом уровне: установка счетчика на сайт, интеграция SDK в приложение. Это не видно пользователю напрямую, но высокоприоритетно для последующей оценки использования MVP и маркетинга. - Отправка push-уведомлений: настроить инфраструктуру (например, Firebase) и убедиться, что можно отправить хотя

бы тестовое уведомление пользователю приложения. Маркетинговую рассылку с UI для администратора можно считать второстепенной, но сама техническая возможность push должна быть проверена в MVP.

Низкий приоритет (учесть архитектурно, можно отложить): Функциональность, которая обозначена как требование на будущее или улучшения, не обязательно полностью реализуемые в ходе хакатона, но которые следует иметь в виду при разработке: - **Интеграция с POS/ERP на этапе MVP не реализуется**, но архитектура должна позволять её добавить без переделки ядра. Например, заложить уникальные идентификаторы для блюд, соответствующие ID в POS-системе, предусмотреть, что после оформления заказ может передаваться во внешнюю систему. Функционал двусторонней синхронизации (обновление меню из POS, передача заказа в POS) пока не реализуется. - **Интеграция с рекламными кабинетами** (Facebook Ads, Google Ads): в MVP можно опустить автоматический импорт расходов из рекламных систем. Пока достаточно сбора UTM и базовой аналитики. Полноценная сквозная аналитика (ROI в реальном времени) – задача последующих этапов. Однако, кодовую базу можно сразу строить так, чтобы было возможно интегрировать API рекламных платформ (например, хранить в базе поля для cost, cpc и т.п. даже если они пока не заполняются). - **Продвинутая аналитика и отчетность:** такие возможности, как детальная сквозная аналитика, когортный анализ, LTV клиентов, визуализация в виде диаграмм или сложных дашбордов – всё это пока вторично. MVP должен сконцентрироваться на корректности основных данных. В будущем планируется подключение BI-инструментов, поэтому сейчас можно ограничиться экспортом данных (например, CSV выгрузка заказов) для внешнего анализа. Аналогично, **коллтрекинг** (отслеживание звонков) и офлайн-каналы – это не реализуется сейчас, но должна быть учтена возможность в будущем (например, хранить поле «номер звонка/call_id» в заказе, если он пришёл по телефону). - **Дополнительный функционал для удобства:** вещи как отзывы клиентов о заказах, рейтинг блюд, чат поддержки, трекинг курьера на карте, рекомендации блюд, реферальная программа – всё это может значительно усилить продукт, но не является критичным для MVP и может быть реализовано позже. Сейчас они не в приоритете. Команда может описать, как их можно добавить, но не тратить основное время разработки на них. - **Полировка UI/UX и производительности:** хотя удобство важно, на этапе MVP допустимы упрощения (например, отсутствие анимаций, минималистичный дизайн). Главный акцент – рабочий функционал. Оптимизация скорости, нагрузочное тестирование, кросс-браузерная отладка – это тоже более актуально на стадии выхода в продакшн. В ТЗ будущих этапов можно заложить улучшение этих аспектов.

Приоритизация таким образом позволит сконцентрироваться на **краеугольных возможностях** (заказ, меню, базовая аналитика), показывающих идею продукта, и заложить базу для последующей доработки второстепенных функций в более полном продукте.

Интерфейсы

Пользовательский интерфейс (мобильное приложение и веб-сайт)

Платформа предусматривает два основных канала для клиентов – мобильное приложение и веб-сайт. Оба канала должны предоставлять схожий опыт и функциональность, следуя единым принципам брендирования и UX.

Мобильное приложение: На главном экране приложения может располагаться приветствие и ключевые акции (баннеры). Далее – навигация по категориям меню (в виде списка или плиток с изображениями категорий). Пользователь просматривает список блюд в выбранной категории: карточки блюд содержат фото, название, короткое описание и цену. При нажатии на блюдо открывается экран подробностей, где пользователь может выбрать модификаторы (если есть) и добавить товар в корзину.

- В приложении должна быть удобно доступна **корзина** (например, значок корзины в шапке или отдельная вкладка). На экране корзины – список позиций, возможность изменить количество или удалить позиции, поле для ввода промокода и итоговая сумма заказа с учётом скидок.

- Экран оформления заказа запрашивает у пользователя детали доставки: адрес (можно реализовать ввод адреса с подсказками или простую текстовую форму на MVP), комментарий к заказу, способ оплаты, а также предлагает подтвердить заказ. В случае самовывоза – выбор филиала (либо подтягивается адрес ресторана по умолчанию).

- После подтверждения – отображается сообщение об успешном оформлении. Приложение может предложить пользователю зарегистрироваться или войти (если это не сделано) для отслеживания заказа.

- **Навигация приложения** может быть устроена через таб-бар или боковое меню: например, вкладки «Меню», «Акции», «Профиль/Заказы». В разделе «Акции» – список текущих промо (с баннерами и условиями). В «Профиле» – данные аккаунта, история заказов (если реализовано).

- Приложение должно корректно обрабатывать разные состояния: загрузка данных (spinner при загрузке меню), отсутствие интернет-соединения, пустой список (например, «В этой категории пока нет блюд»), подтверждение действий (диалог «Удалить из корзины?») и т.д., чтобы пользовательский опыт был цельным и понятным.

Веб-сайт: Интерфейс сайта во многом дублирует приложение, с той разницей, что он адаптируется под разные размеры экрана. На **десктопе** меню может отображаться в виде списков или плиток с категориями и товарами на одной странице (например, список категорий слева, блюда – справа). На мобильной версии сайта дизайн сближается с приложением (вертикальный скролл списка блюд, выпадающее меню категорий). Брендирование должно быть явным: сайт оформляется в цветах компании APPETIT, логотип размещается в шапке.

- На сайте, как и в приложении, должна быть заметна **кнопка доступа к корзине**. При оформлении заказа на сайте – форма может быть сразу на странице (тип доставки, адрес, контакты, оплата) с возможностью прокрутки.

- **Респонсивность:** сайт должен быть удобен на смартфонах, т.к. часть пользователей может не устанавливать приложение, а оформить заказ через мобильный браузер. Это значит крупные нажимные элементы, минимум скролла, адаптивные изображения.

- **Дополнительные страницы:** На сайте можно разместить и страницы информации (например, «О нас», «Контакты», «Доставка и оплата») – для полного бренд-сайта. В приложении эти сведения могут быть в разделе профиля или справки. Для MVP достаточно базовой контактной информации (телефон, адрес ресторана) где-нибудь на видном месте.

- **Поддержка нескольких языков:** Если целевая аудитория русскоязычная, MVP может быть только на русском. Но интерфейс (как сайта, так и приложения) следует спроектировать с учетом **возможности локализации** на другие языки (например, казахский, английский) в будущем – текстовые элементы отделены от кода, нет надписей, вписанных в изображения и пр.

Поведение и обратная связь: Пользовательский интерфейс должен быть отзывчивым: на нажатия кнопок происходит визуальная реакция (подсветка, загрузка). После оформления заказа пользователю явно показывается, что заказ принят (например, страница с номером заказа и сообщением благодарности). Если пользователь сделал что-то неправильно (не заполнил поле адреса или промокод недействителен) – интерфейс выдает понятное сообщение об ошибке рядом с соответствующим полем.

Административный интерфейс (панель управления)

Админ-панель – это веб-приложение (доступное, например, по защищенному URL типа admin.appetit.kz), которое используют менеджеры для управления системой. Основные элементы UI админ-панели:

- **Авторизация:** При переходе в панель администратор видит страницу входа (логин и пароль). Доступ имеют только уполномоченные сотрудники. После успешного входа – перенаправление на главную страницу панели (например, дашборд).

- **Дашборд (главная страница):** Здесь отображаются краткие сводки: количество заказов сегодня, выручка за месяц, и т.п. Могут быть выведены графики заказов по дням, топ-5 популярных блюд, последние отзывы (если есть) – для наглядности. В MVP можно ограничиться текстовыми показателями (цифрами) или простыми столбиками. Дашборд помогает админу быстро оценить состояние дел.

- **Меню (контент):** Раздел для управления меню ресторана. UI включает список категорий и список блюд. Администратор должен видеть структуру меню. Например, слева – список категорий (с возможностью добавить/удалить категорию, изменить название, порядок), справа – таблица блюд выбранной категории. В таблице отображаются имя блюда, цена, статус (активно/скрыто), кнопки «редактировать» и «удалить». Выше

списка – кнопка «Добавить блюдо». При добавлении или редактировании блюда открывается форма: поля названия, описания (многострочное), цены, загрузки изображения (через файловый диалог), выбор категории, настройка модификаторов (например, можно добавить варианты с доплатой). Если изображение уже загружено, оно отображается с возможностью заменить. Интерфейс должен проверять корректность ввода (например, цена – число, обязательные поля заполнены). После сохранения изменений блюда обновляются сразу в пользовательском приложении.

- **Управление заказами:** Раздел «Заказы» представляет собой список всех заказов. Табличное представление: номер заказа, дата/время, имя клиента, сумма, статус (цветовое выделение: новый – выделен, выполненный – серый и т.п.). В начале списка – самые новые заказы. Администратор может кликнуть на заказ для просмотра подробностей: открывается карточка заказа с полным составом, стоимостью, выбранным методом доставки/оплаты, контактами клиента. В этой карточке доступны действия: изменить статус (напр., выпадающий список статусов), распечатать чек/накладную (опционально, если принтер подключен), связаться с клиентом (например, отобразить телефон). После изменения статуса у клиента улетает уведомление (если предусмотрено). Админ-интерфейс должен автоматически обновлять список (через периодический опрос или WebSocket, опционально) чтобы новые заказы появлялись без перезагрузки страницы.

- **Промо и маркетинг:** В панели есть раздел **«Маркетинг»** или «Акции». Здесь администратор управляет промокодами, баннерами и рассылками:

- **Промокоды:** интерфейс для создания промокода – форма с полями: код, тип скидки (% или сумма), значение, дата начала/окончания, ограничения (минимальная сумма, кол-во применений, для каких категорий действует). Список промокодов отображается таблицей (код, скидка, срок, активность, число использований). Админ может отключить или удалить промокод.

- **Баннеры/акции:** возможность загрузить графический баннер (JPEG/PNG) и указать ссылку или описание акции, чтобы он отобразился пользователям. Можно задавать порядок баннеров, активность (например, прошедшие акции отключать). В простейшем случае достаточно загружать баннер и текст.

- **Push-уведомления:** инструмент рассылки – администратор пишет заголовок и текст уведомления, может выбрать сегмент (MVP: всем пользователям), нажимает «Отправить». Показывается подтверждение отправки и, возможно, статистика доставленных уведомлений. Интерфейс должен быть предельно простым (поле ввода текста + кнопка).

- **Настройки и интеграции:** В MVP можно иметь простой раздел настроек: часы работы (для блокировки оформления на нерабочее время), данные для интеграций (например, можно ввести ключ Google Analytics или Firebase в настройки, чтобы приложение его использовало). Здесь же могут быть

настройки доставки (стоимость доставки, радиус и т.п. – если нужно). В будущих итерациях – раздел интеграции с POS: ввод API-ключей POS-системы, отображение статуса синхронизации.

UX админ-панели: так как ею будут пользоваться сотрудники ресторана, важно сделать интерфейс понятным для пользователей с базовыми техническими навыками. Использовать знакомые элементы: выпадающие меню, таблицы, кнопки сохранения. Критичные действия (удаление блюда, промокода) должны подтверждаться диалогом. Цветовая схема может соответствовать фирменной, но главное – **читабельность** (напр., фон панели светлый, текст тёмный). Для графиков и показателей – использовать визуальные подсказки (зелёный цвет роста показателя, красный – падения, например). Панель должна быть оптимизирована под настольные компьютеры (основной сценарий – менеджер работает с ноутбука/ПК). Возможно, адаптивность под планшет тоже полезна, но мобильную версию админки можно не делать.

Внешние интерфейсы и интеграции

MVP должен быть готов к взаимодействию с внешними сервисами и предоставлять удобные точки интеграции: - **Интеграция с POS-системами:** Заложена возможность подключить POS/ERP систему ресторана (такие как iiko, Poster, r_keeper) через API. Это означает, что архитектура API нашего backend сможет как принимать, так и отдавать данные, необходимые для синхронизации меню и заказов. Например, у каждого блюда в базе может храниться внешний ID (ID в системе iiko). При интеграции приложение будет получать меню напрямую из POS или отправлять подтверждённые заказы через API POS. **Внешний интерфейс** для этого – модуль-коннектор, который обращается к API POS-системы. У iiko и r_keeper существуют публичные API для обмена данными. На старте взаимодействие будет односторонним (наша система выгружает данные или загружает), но архитектура контроллера заказов уже может предусмотреть вызов внешнего сервиса при изменении статуса заказа. *Например:* при интеграции с iiko, после оформления заказ сразу передается в iikoCloud через их REST API, минуя необходимость ручного перебивания заказа[14]. Это ускорит обработку и уменьшит ошибки, к чему и будем стремиться в дальнейших версиях. - **API для мобильных приложений:** Наш backend сам является внешним интерфейсом для приложений. API должен быть хорошо документирован (особенно, если хакатон-проект будет развиваться далее другими командами). Предусматривается использование стандартов REST (HTTP методы GET/POST/PUT/DELETE, ответы в формате JSON). В будущем, этот же API может быть использован для подключения новых клиентских платформ (например, приложение для курьеров или виджет для соцсетей). - **Интеграция с платежными шлюзами:** Если планируется онлайн-оплата, система интегрируется с платежным провайдером через его API. Внешний интерфейс тут – SDK или REST API банка/платёжной системы, куда будут отправляться данные транзакций. MVP архитектура уже учитывает это: например, после оформления заказа, если выбран онлайн-платёж, наш сервер создает через API платёжной системы ссылку/токен оплаты и возвращает клиенту, клиент оплачивает, а наш сервер получает callback об успешной оплате и меняет статус заказа. Такие интеграции отрабатываются с популярными провайдерами (Stripe, Braintree, либо локальные в KZ). В FoodTech решениях часто используются готовые модули оплаты – мы

обеспечим совместимость с ними архитектурно[15]. - **Службы карты/геолокации:** Для удобства ввода адреса доставки планируется интеграция сервиса карт (Google Maps, Яндекс.Карты или 2GIS API). Во внешних интерфейсах можно задействовать API геокодирования: при вводе адреса вызывать сервис для получения координат и более стандартизированного формата адреса. Это поможет в будущем (например, для расчета расстояния доставки, привязки заказа к ближайшему филиалу). - **Интеграция с рекламными платформами:** Для реализации сквозной аналитики система должна взаимодействовать с API рекламных кабинетов (Facebook/Meta Ads, Google Ads, etc.). Во внешних интерфейсах это выражается так: с определенной периодичностью или по запросу наша система будет запрашивать у этих платформ данные о расходах, кликах, показах по кампаниям. Например, через Facebook Graph API получать статистику по указанным кампаниям, через Google Ads API – расходы по каждому идентификатору кампании. Эти данные затем сопоставляются с внутренними данными заказов (по UTM_campaign идентификатору). MVP не требует реализации этого сбора, но **интерфейс** должен быть предусмотрен: например, хранение токенов доступа к API рекламных систем в настройках, таблицы для хранения полученных данных, базовая структура модулей, которые будут дергать внешние API. После хакатона можно будет развить эту часть, подключив сервисы типа Roistat, Garpun или собственные скрипты. - **Call-центр и телефония:** Интеграция с IP-телефонией позволит учитывать заказы, поступившие по телефону, в общей системе. Внешний интерфейс здесь – либо программная **АТС (виртуальная телефония)**, либо сервис коллтрекинга. Планируется подключение коллтрекинга с набором подменных телефонных номеров, чтобы отслеживать, с какого канала рекламы звонит клиент. При звонке система будет получать данные (например, номер клиента, источник номера) через Webhook от сервиса телефонии. Эти звонки могут автоматически регистрироваться как лиды или заказы (если оператор заполняет форму заказа вручную в админ-панели). В архитектуре MVP можно заложить API endpoint для приема таких webhook (например, для фиксации звонка), и связать звонок с созданием черновика заказа. Также возможно в будущем реализовать **клиент для оператора**: когда звонит телефон, на экране всплывает карта клиента (если номер узнан) и форма быстрого создания заказа. Пока это за рамками MVP, но возможности API (например, **Mango Office**, **Asterisk** или **Twilio**) нужно учесть. - **Прочие интеграции:** На будущее возможны подключения: модуль лояльности (например, iikoCard или собственная CRM), мессенджеры (приём заказов через Telegram-бота), интеграция с сервисом рассылок email/SMS, и т.д. Каждое из них требует продуманного API-интерфейса. Мы стремимся сделать backend универсальным ядром, взаимодействующим через чёткие интерфейсы с любыми внешними модулями.

Подводя итог, **MVP предоставляет открытые и документируемые точки интеграции** (API для приложений, вебхуки, готовность подключать внешние API), что облегчит расширение функционала и включение внешних сервисов без ломки существующего кода.

Требования к UX/UI

К пользовательскому опыту (UX) и интерфейсу (UI) платформы предъявляются высокие требования, так как от удобства работы с приложением/сайтом будет зависеть

вовлечённость клиентов и эффективность заказов. Ниже перечислены ключевые требования и рекомендации:

- **Единый фирменный стиль:** Дизайн мобильного приложения и сайта должен соответствовать бренду APPETIT. Используются фирменные цвета, логотип, стилистика, создающие узнаваемость. Все элементы интерфейса (кнопки, иконки, шрифты) – в единой визуальной теме. Это повышает доверие пользователей и формирует образ собственного цифрового продукта компании, отличного от агрегаторов.
- **Простота и интуитивность:** Интерфейс для клиента должен быть **понятным без обучения**. Пользователь в 2-3 касания должен достигать цели (найти блюдо и добавить в корзину). Навигация логична: категории – список блюд – детали блюда – корзина – оформление заказа. Элементы управления подписаны понятным текстом или общепринятыми значками (значок «корзины», «домой», «назад» и т.д.). Если действие пользователя требует ожидания (например, загрузка меню), это визуально отображается. Ошибки пользователя (незаполненный адрес, неверный формат телефона) сопровождаются ясными подсказками рядом с полем (например, красный текст «Введите номер телефона»). **Конечная цель UX** – минимизировать шаги на пути к заказу и устранить возможное замешательство пользователя.
- **Быстродействие и отзывчивость:** Приложение и сайт должны работать быстро. Это значит – оптимизировать изображения блюд (чтобы не было долгой загрузки), предусмотреть кэширование меню локально (чтобы при повторном открытии пользователь мгновенно видел список блюд). Анимации (если используются) должны быть легкими и не тормозить даже на бюджетных устройствах. Визуальные отклики на нажатия (highlight эффект на кнопках, скелетон-экраны при загрузке) улучшают восприятие скорости. Если связь медленная – приложение должно об этом сообщать («Проблемы со связью, пытаемся снова…»). Веб-сайт должен корректно работать во всех современных браузерах; для мобильного веба – оптимизация под WebView.
- **Адаптивность и кроссплатформенность:** UI разрабатывается с учетом разных устройств и экранов. Мобильное приложение должно корректно отображаться на разных разрешениях экранов (от небольших смартфонов до планшетов). Веб-сайт – адаптироваться под мобильные экраны (через media queries или динамические компоненты). Все функциональные возможности (заказ, регистрация, применение промо) должны быть доступны и на сайте, и в приложении, без предпочтения одной платформы. Пользователь, начавший оформление на сайте, и пользователь в приложении не должны испытывать различий в возможностях (UX-паттерны могут отличаться, но возможности – те же).
- **Фокус на визуальном контенте еды:** При разработке UI следует учитывать специфику продукта – доставка еды. **Фотографии блюд** должны быть в высоком качестве, аппетитные, занимать заметную область на экране деталей блюда. Цветовая палитра – «вкусная» (например, акцентные цвета, вызывающие аппетит). Шрифты для названий блюд – читабельные, достаточно крупные. Желательно избегать перегрузки текстом: краткость описаний, больше визуальных элементов (иконки «острое», «хит», «новинка» и т.п. могут сопровождать описание блюда). Пользовательский путь выбора еды должен

приносить удовольствие, поэтому UI/UX можно сравнить с листанием меню в ресторане – привлекательно и не утомительно.

- **Удобство оформления заказа:** Экран checkout (оформления) должен быть предельно прост: минимум обязательных полей. Лучше использовать маски для ввода телефона, автоматическое подставление города, выпадающий список улиц (если реализуемо). Кнопка подтверждения заказа – заметная, крупная, с ясным текстом («Оформить заказ»). После нажатия – не оставлять пользователя в неведении: показать индикатор обработки и затем экран результата. Если требуется подтверждение via SMS или email – продумать, как не фрустрировать пользователя задержками (например, прогресс-бар ожидания кода).
- **UX доверия и безопасности:** Поскольку речь о еде и деньгах, UI должен вызывать доверие. Это означает – наличие элементов безопасности (значок замка или надпись «Защищенное соединение» где нужно), логотипы известных платежных систем при оплате, ссылки на политику конфиденциальности и оферту (в футере сайта, в приложении – в разделе «О приложении»). Отзывы или рейтинги (если собираются) тоже повышают доверие. *Хотя сбор отзывов не обязателен в MVP*, оставить место для них – хорошая UX-практика.
- **Административный UX:** Интерфейс админ-панели должен быть **функциональным и лаконичным**. Предпочтение таблицам, спискам и формам без излишков. Важна группировка: например, на странице заказа администратор видит все данные заказа в одном месте, не нужно переходить по разным вкладкам. **Визуальные индикаторы:** новые заказы могут подсвечиваться, большие цифры на дашборде – выделяться цветом. Для аналитики – графики снабжены легендой, понятными подписями осей. Если данных мало (например, за выбранный день только 1 заказ) – интерфейс все равно правильно это показывает, не ломается. В админке крайне важно избежать случайных действий: например, кнопка «Удалить блюдо» не должна быть слишком легко нажать; лучше требовать подтверждения. Также, при внесении изменений – показывать уведомление «изменения сохранены». Это дает уверенность пользователю-администратору.
- **Многоязычность интерфейса:** Для админов, скорее всего, достаточно русского языка. Для клиентской части, как уже упомянуто, нужно учитывать возможность добавления языков. Переключение языка на сайте – через явный переключатель (флаг или надпись), в приложении – в настройках. Все надписи должны быть переведены. Шрифты и верстка должны поддерживать отображение кириллицы и латиницы корректно. Если планируется казахский язык, проверить отображение специфических символов. Визуальные элементы (иконки) – универсальны, без надписей внутри, чтобы не дублировать для каждого языка.
- **Доступность (Accessibility):** По возможности, соблюдать принципы доступности: достаточный контраст текста и фона (особенно для важной информации, например, цена, ошибки – контрастно выделены) для пользователей с нарушением зрения. Размер шрифта – не слишком мелкий (не менее 12px на мобильных). Кнопки и интерактивные элементы – достаточно крупные для нажатия пальцем (минимум ~44px по рекомендациям). Если время позволяет, можно обеспечить корректную работу приложения с сервисами озвучивания (TalkBack/VoiceOver) – как минимум, чтобы важные кнопки имели

content-description. Это не первоочередно в хакатоне, но хорошие практики UI стоит учитывать.

В целом, **UX/UI критерий успеха**: пользователь (клиент) легко совершает заказ, а пользователь (администратор) легко управляет процессами. Любые препятствия или двусмысленности в интерфейсе должны устраняться на этапе дизайна. Желательно провести краткое юзабили-тистирование внутри команды: пройти путь заказа самому и посмотреть, где могут быть затруднения, и улучшить эти моменты до подачи решения.

Технологии

Выбор технологий для разработки MVP должен обеспечивать быстрый выпуск продукта без ущерба для его качества и возможности масштабирования. Ниже предложены технологические стек и инструменты для различных компонентов платформы:

- **Мобильная разработка:** Рекомендуется использовать кроссплатформенный фреймворк, позволяющий разрабатывать единое приложение под Android и iOS одновременно. Варианты: **Flutter** (Dart) – позволит быстро создать нативно компилируемые приложения с единым UI; либо **React Native** (JavaScript) – интеграция нативных компонентов через React. Оба варианта поддерживают Firebase (для аналитики и push) и имеют большое сообщество. Кроссплатформенный подход экономит время (особенно важный фактор на хакатоне) и гарантирует одинаковый функционал на обеих платформах. Тем не менее, если у команды есть сильная экспертиза в нативной разработке, можно сделать MVP приложение для одной платформы (например, Android на Kotlin) – но тогда вторую платформу придется догонять позже. При любом подходе, код должен быть модульным, чтобы логику (работу с API, хранение данных) можно было переиспользовать и тестировать отдельно от UI.
- **Веб-фронтенд:** Для создания современного и динамичного веб-сайта лучше применить JavaScript-фреймворк. **React** + экосистема (Redux или Context для состояния, React Router для маршрутизации) – популярный выбор, обеспечивающий компонентный подход. Альтернативы – **Angular** или **Vue.js**, которые также подходят; выбор может зависеть от навыков участников. Возможно использование готовых UI-библиотек (Bootstrap, Ant Design, Material UI) для ускорения верстки компонентов. Сайт должен собираться в оптимизированный бандл (использование Webpack, Vite или Create React App). Для обеспечения SEO (если важно индексирование) – можно реализовать серверную рендеринг (Next.js для React), но в рамках MVP это не критично, так как основная аудитория – прямые пользователи, а не поисковый трафик. Главное – обеспечить адаптивность. Также можно рассмотреть **PWA (Progressive Web App)** технологии, чтобы сайт мог устанавливаться как приложение и работать офлайн – но это опционально. Важно подключить на сайт **Google Analytics 4** (через gtag.js или официальный пакет) для сбора статистики посещаемости и конверсий.
- **Backend-сервер:** На серверной стороне подойдут современные фреймворки, позволяющие быстро создать REST API. Варианты:

- **Node.js** – с фреймворком Express.js (или более структурированным NestJS) для написания контроллеров и маршрутов. Node хорош быстротой разработки на JS/TypeScript, большим числом библиотек (например, для интеграции с Firebase, для JWT, для работы с БД через ORM типа Prisma или TypeORM).

- **Python** – фреймворк Django (более монолитный, но «из коробки» много всего, включая админку) или Flask/FastAPI (легковесные и быстрые решения). Python имеет плюс в виде простой реализации аналитических расчетов, если потребуется.

- **Java** – фреймворк Spring Boot, гарантирующий надежность и масштабируемость, но разработка на Java может быть медленнее для MVP. Если в команде есть опыт, можно использовать.

- **PHP** – Laravel, если команда предпочитает PHP, может дать быстрый результат (Laravel имеет встроенные средства для аутентификации, ORM Eloquent для БД и пр.). Однако, Node или Python сейчас более популярны для подобных задач.

Независимо от языка, **важно реализовать слои**: маршруты API, логика (сервисный слой) и уровень доступа к данным (ORM или прямые запросы). Логика не должна быть жестко привязана к веб-фреймворку, чтобы ее можно было переиспользовать (например, вынести часть в serverless-функции позже или в cron-задачи).

Для хранения конфигураций (ключи API, секреты) – использовать безопасные хранилища (env-переменные, .env файл, либо настройки в Kubernetes Secrets, если облако).

Реализовать **документацию API** (можно с помощью Swagger/OpenAPI генерации прямо в коде), чтобы дальнейшая команда разработки или интеграторы POS понимали доступные endpoints.

- **База данных:** Как отмечалось, оптимальным выбором будет **реляционная СУБД**. **PostgreSQL** – предпочтительно из-за поддержки сложных типов (JSONB, гео-типов, что может пригодиться для адресов), надежности и соответствия стандартам. MySQL/MariaDB тоже подойдет. Использование ORM облегчит миграции схемы и построение запросов – для Node это TypeORM/Prisma, для Python – SQLAlchemy/Django ORM, для Java – Hibernate, для PHP – Eloquent и т.д. В начальной схеме нужно создать таблицы: Users, Dishes, Categories, Modifiers, Orders, OrderItems (состав заказа), PromoCodes, etc. Желательно продумать индексы (по основным полям запросов – например, по дате заказа, по промокоду, по пользователю). Для масштабирования БД предусмотреть возможность вертикального роста (достаточно выбрать проверенное решение, которое можно развернуть в облаке, например AWS RDS). Резервное копирование БД – на этапе MVP вручную/периодически (бакап раз в день). Если возникнет потребность в **pealtime** (например, обновление списка заказов у админа мгновенно), можно рассмотреть внедрение **Redis** (для PUB/SUB

уведомлений) или использовать WebSockets, но MVP может обойтись опросом раз в N секунд.
- **Интеграция и Backend-сервисы:** Для push-уведомлений выбираем **Firebase Cloud Messaging (FCM)** – это бесплатно и удобно. Подключаем Firebase SDK в мобильное приложение. На backend используем официальные библиотеки Firebase Admin SDK для отправки уведомлений на конкретные токены устройств. Для отправки email (например, подтверждение регистрации или уведомление) можно подключить SMTP (любой почтовый сервис, либо сервис вроде SendGrid). Для отправки SMS – при необходимости интегрируем API провайдера SMS (например, Twilio, либо локальные Kcell/Beeline SMS-шлюзы). Эти коммуникации не обязательны в MVP, но могут быть частью пользовательского опыта (SMS с подтверждением или кодом).

Для аналитики выбираем **Google Analytics 4**. На фронтенде подключаем GA-теги, в мобильном – Firebase Analytics (который может собирать события и пересылать в GA4, так как GA4 унифицирован для приложений и веба). Также, чтобы отслеживать установки приложения по рекламным кампаниям, можно использовать **Firebase Dynamic Links** или App Links с UTM, а затем Firebase Attribution.

Карты и гео: Google Maps API (Places Autocomplete) для подсказок адресов – если будем делать, то на фронтенде подключается JS API ключ, на мобильном – Google Places SDK. Стоимость использования надо учесть, но в небольших объемах Google предоставляет бесплатный квот.

- **Интеграция с POS:** На перспективу, если выбирается конкретная POS, можно сразу использовать их SDK/API. Например, для **iiko** – использовать iikoCloud API (REST), для **Poster** – их Web API, для r_keeper – возможно потребует промежуточный коннектор. Код интеграции выносим в отдельный модуль/сервис, который можно вызывать при определенных событиях (создании заказа, обновлении меню). Для тестирования на этапе MVP, можно сделать эмуляцию: например, написать модуль, который при новом заказе отправляет JSON-файл на локальный сервер, имитируя отправку в POS. Это покажет, что система готова к интеграции.
- **DevOps и развёртывание:** Для MVP достаточно развернуть сервер на любом VPS или облачном сервисе (например, Heroku, DigitalOcean, AWS EC2). Обязательно настроить **SSL** для домена, так как без HTTPS современные браузеры не разрешают работу сервисов геолокации, Push и др. Контейнеризация (Docker) приветствуется – можно собрать контейнер с backend и, например, PostgreSQL, чтобы развёртывание было воспроизводимым. Если команда умеет, можно настроить CI/CD (например, GitHub Actions, GitLab CI) для автоматического деплоя при обновлении кода – чтобы упростить демонстрацию обновлений жюри. Логирование – использовать стандартные лог-файлы или сервисы (например, Sentry для ошибок фронтенда, Papertrail для серверных логов) – поможет отслеживать проблемы.
- **Тестирование:** По возможности, написать unit-тесты для критичных частей логики (расчет цены с промокодом, авторизация). Использовать фреймворки тестирования: Jest/Mocha (JS), PyTest (Python), JUnit (Java). Также, end-to-end тесты (Selenium, Cypress) для основных сценариев UI могут пригодиться, но в

условиях хакатона ограничиваемся ручным тестированием критических путей. Впоследствии, наличие автотестов ускорит доработку продукта.

- **Версионность API и кодовой базы:** Вводим практику версионирования API (например, prefix /api/v1/ для всех endpoints) – это позволит в будущем вносить изменения без ломания старых клиентов. Код храним в системе контроля версий (**Git**). Репозиторий может быть разделён на фронтенд и бэкенд, либо монорепо – главное, чтобы вся команда могла параллельно работать. Будет плюсом настроить **код-ревью** и применение style guide (ESLint/Prettier для JS, PEP8 для Python и т.д.) для поддержания читаемости кода.

Подытоживая, стек технологий должен быть выбран исходя из компетенций команды, но с учётом долгосрочной поддержки. Решения на **Node.js + React + PostgreSQL + Firebase** выглядят наиболее сбалансированными для данной задачи, хотя другие стеки тоже приемлемы. В отчёте команда должна указать, какие технологии они выбрали и почему, описать архитектуру приложения, и как выбранные технологии помогут масштабировать и интегрировать продукт в будущем.

Перспективы расширения

MVP является основой, на которую в дальнейшем будут добавляться расширенные возможности. Ниже перечислены будущие требования и направления развития платформы, которые должны быть учтены архитектурно и реализованы на следующих этапах:

- **Интеграция с POS/ERP-системами ресторана:** После успешного запуска MVP планируется тесная связка с внутренними системами учета и автоматизации. Например, интеграция с популярными системами общественного питания – **iiko, Poster, r_keeper**. Это позволит:

- **Синхронизировать меню и остатки:** Все изменения в меню (цены, наличие блюд) будут автоматически передаваться из POS в приложение, и наоборот – заказы из приложения будут сразу появляться в интерфейсе POS на кухне[14]. Благодаря этому исключается разночтение данных и необходимость вручную переносить заказы.

- **Учет лояльности и скидок:** Если в POS ведется программа лояльности (например, iikoCard), интеграция позволит применять бонусы клиента при заказе онлайн.

- **Единый складской учет:** Списание ингредиентов при онлайн-заказе будет отражаться в ERP, что важно для аналитики продуктов.

Технически такая интеграция достигается за счет использования API POS-систем. У iiko, например, есть облачный API (iikoTransport) для получения меню и отправки заказов. Будет реализован модуль-коннектор, который периодически (или по webhook) обновляет меню из iiko и через вызов метода API создаёт заказ в системе при поступлении онлайн-заказа. Аналогично для Poster (REST API) или r_keeper (может

потребовать промежуточных решений, так как классический r_keeper менее открыт, но существуют интеграторы).

Архитектура MVP уже учитывает это: есть сущности ID внешних систем, API-клиенты и очереди задач на синхронизацию, чтобы в будущем просто дописать нужный код. После интеграции с POS мы ожидаем ускорение обслуживания заказов и снижение ошибки (человеческий фактор устраняется).

- **Интеграция с рекламными кабинетами и сквозная аналитика:** Одно из стратегических направлений – **сквозная (end-to-end) аналитика маркетинга**, которая позволит в реальном времени оценивать ROI каждого рекламного канала. Для этого платформа будет собирать данные не только о заказах, но и о расходах на привлечение:

- Планируется подключить API **Meta (Facebook/Instagram) Ads**, **Google Ads**, возможно, **Яндекс Директ**. Система, используя сохраненные токены доступа, сможет раз в N минут (или каждый час) обращаться к API и получать суммарные расходы, клики, показы по активным кампаниям.

- Одновременно, каждому заказу уже сейчас привязываются идентификаторы источника (UTM-метки, referrer). Объединяя эти данные, можно построить отчеты: например, за сегодня с Google Ads кампании X – 10 заказов на сумму Y, при затратах Z, ROI = (Y/Z*100)%.

- Система будет вычислять **CPA, ROI, ROMI** автоматически и показывать админу на дашборде вблизи реального времени. Например, утром менеджер видит, что вчера реклама в Instagram принесла 50 заказов на 500k тг при бюджете 100k тг (ROI 500%), а реклама в Google – 30 заказов на 300k тг при бюджете 200k (ROI 150%). Такие инсайты позволят быстро перераспределять рекламный бюджет. Без сквозной аналитики бизнес видит только общие продажи, не зная их источников.

- Будет реализовано хранение подробных данных по рекламным каналам, вплоть до ключевых слов и объявлений (если использовать API, который детализирует до UTM_term). Это объемные данные, поэтому, возможно, будет принято решение хранить их в отдельном хранилище или отправлять в системы анализа (например, BigQuery, PowerBI). Но на уровне архитектуры нужно быть готовым – обеспечить уникальную связь заказа с параметрами трафика и иметь возможность расширить модель данных заказа полями для маркеров кампаний.

Также, для полной картины, в сквозную аналитику будут включены данные **коллтрекинга** (телефонных звонков) и офлайн-продаж (если есть). Используя инструменты вроде Garpun или Roistat либо собственными разработками, можно объединить разные источники данных. На перспективу возможно интегрировать готовый сервис (например, Roistat) через их API, но владение своей системой даст больше гибкости.

- **Интеграция с IP-телефонией колл-центра:** Несмотря на рост онлайн-заказов, определенная доля клиентов предпочитает звонить. Чтобы учитывать их, мы планируем подключить телефонную систему:

- **Виртуальная АТС (IP-PBX):** Например, использовать сервис **Mango Office**, **RingCentral** или **Asterisk**. При звонке клиента система будет автоматически определять, откуда он узнал номер (с помощью подменных номеров, распознавания DTMF меню или вопроса оператора) – эта информация (канал) привяжется к заказу.

- **Рабочее место оператора:** Для колл-центра планируется интерфейс (в дальнейшем) на основе админ-панели, где оператор принимает звонок, заполняет заказ в той же системе (в специальном режиме «оформление звонка»). Таким образом, **все заказы, будь то с сайта, приложения или по телефону, окажутся в единой базе**. Оператор сможет пользоваться тем же каталогом блюд и акциями (например, применять промокод, если клиент продиктует). После оформления операторским способом, заказ пойдет в общую очередь на кухню как обычный.

- **Call-tracking:** Будет реализовано отслеживание эффективности звонков: сколько звонков конвертировалось в заказы, из каких источников эти звонки (например, отдельные номера для рекламы на билборде, на Instagram – и мы увидим, что больше звонят по номеру с билборда, например). Это достигается интеграцией с сервисом коллтрекинга, который предоставляет пул номеров и API для получения статистики. В системе появится модуль, периодически опрашивающий коллтрекинг-сервис и создающий «лиды звонков» или добавляющий данные в аналитику.

- **Запись разговоров и качество сервиса:** В перспективе интеграция позволит сохранять записи звонков привязанными к заказам, внедрить AI-расшифровку и оценку разговоров (это уже advanced-level, вне ближнего плана, но архитектурно возможно через API тех же телефоний или сторонних Speech-to-text сервисов). Цель – улучшать качество обслуживания на телефоне так же, как мы улучшаем UX приложения.
- **Программа лояльности и CRM:** Чтобы **удерживать клиентов и стимулировать повторные заказы**, платформа будет развиваться в направлении CRM:

- Внедрение **электронной программы лояльности**: накопительные бонусы, скидки за частые заказы, персональные предложения. Например, за каждый заказ начислять бонусы (процент от суммы), которые можно тратить на следующие заказы. Или ввод уровней (бронзовый, серебряный, золотой клиент) с растущими привилегиями. Система уже содержит профиль пользователя, поэтому добавить поля для балансов, уровней не сложно. Потребуется фронтенд-отображение (в профиле показывать баланс бонусов, историю их использования). Админ-панель получит раздел «Лояльность», где можно настроить правила начисления и списания, просматривать активность клиентов.

- **Интеграция с iikoCard или аналогом:** Если использовать готовые решения лояльности (например, у iiko есть iikoCard, у r_keeper – Points), то интеграция с POS автоматически откроет возможность использовать ту же карту лояльности и в нашем приложении[21]. В заказ будет достаточно передать ID клиента из нашей системы, а POS сама рассчитает скидку/бонус. Это потребует связать аккаунты: при регистрации пользователя в приложении можно заводить его и в POS CRM.

- **Персонализация маркетинга:** Имея данные CRM (дни рождения, частота заказов, любимые блюда), можно настраивать тонкие механики: автоматическая отправка купона ко дню рождения[22], рекомендация блюд на основе истории (например, «Вы часто заказываете роллы, у нас новый ролл в меню!»). Это направление тесно связано с аналитикой big data и будет развиваться по мере накопления данных.
- **Масштабирование бизнеса и мульти-ресторанность:** Если компания APPETIT планирует расширяться (новые филиалы, города) или предоставлять платформу партнерам, система должна быть готова:

  - **Поддержка нескольких ресторанов/кухонь:** Архитектура данных может быть расширена до мультитенантной – например, добавить сущность «Ресторан/Точка» и привязать к ней заказы и блюда. Тогда единая платформа сможет обслуживать сразу несколько заведений (каждое со своим меню и зоной доставки). Уже сейчас можно минимально предусмотреть поле restaurant_id в основных таблицах. В интерфейсе это проявится в фильтрах (админ выбирает, с каким филиалом работает). Пользовательская часть – определять по адресу, к какому филиалу отнести заказ (потребует реализации геозон доставки).

  - **Высокая нагрузка:** При росте числа пользователей и заказов потребуется масштабировать инфраструктуру. Будет реализован **кластеризация** backend (несколько экземпляров за балансировщиком), разделение службы БД (master-slave или sharding по географии, если будет нужно), использование CDN для статики (изображений блюд) – чтобы ускорить загрузку по всей стране. Возможно, стоит перейти на **микросервисную архитектуру**: выделить сервис авторизации, сервис заказов, сервис нотификаций и пр., которые будут независимо масштабироваться. MVP заложил основы (разделение логики), так что рефакторинг под микросервисы будет выполним без переписывания с нуля.

- **Кроссплатформенные каналы заказов:** В будущем, помимо собственного приложения и сайта, **возможна интеграция с внешними каналами** – модуль заказа через соцсети (например, чат-бот Telegram/Facebook), прием заказов через голосовые помощники. Наше API позволит этим каналам создавать заказы так же, как мобильное приложение. Таким образом, платформа APPETIT может стать **универсальным HUB**, куда стекутся заказы из разных точек (онлайн-приложение, телефон, соцсети).
- **Дополнительные сервисы и функции:**

- **Отзывы и рейтинги:** Добавить возможность клиентам оставлять отзыв о выполненном заказе (с оценкой), и отображать эти отзывы в админке, а также средний рейтинг блюд. Это поможет контролю качества. Можно даже внедрить авто-ответчика на отзывы на базе AI (как модуль, отвечающий шаблонно на отзывы)[23].

- **Расширенная аналитика товаров:** Отслеживать по каждому блюду статистику продаж, маржинальность, популярность среди новых vs. постоянных клиентов[24]. Эта информация будет полезна шеф-повару и маркетологу (какие позиции в промо продвигать, а какие возможно убрать).

- **Логистика доставки:** Если объем доставок большой, возможно стоит внедрить модуль логистики – трекинг курьеров, оптимизация маршрутов. Интеграция с сервисами типа Яндекс.Курьеры или своя карта для курьера, чтобы клиент видел, где его заказ. Это серьезный пласт работ (GPS-трекинг, отдельное приложение для курьера), поэтому на будущее, но благодаря тому, что заказы хранят координаты адреса, можно будет подключить API расчета маршрута и времени доставки. Foodpicasso, например, реализует функцию «автовызова курьера»[15] – наш проект в перспективе тоже сможет так делать, связавшись с системой логистики.

- **Монетизация платформы:** Если APPETIT решит предоставлять платформу другим ресторанам (B2B-модель, как SaaS), потребуется кабинет для новых партнеров, система биллинга (взимание комиссии или абонплаты с ресторанов), управление множеством брендов. Наша архитектура мульти-ресторанности и ролей пользователей может эволюционировать в эту сторону. Но это уже стратегическое расширение, меняющее модель бизнеса.

Заключение: MVP закладывает фундамент, а перечисленные перспективы очерчивают направление развития. Важно, что **каждый из будущих этапов уже предусмотрен архитектурно**: модульная структура, API-first подход, расширяемая схема данных. Это позволит добавлять новые возможности постепенно, без кардинальной переработки. Команде разработки рекомендуется в финальной документации описать, как именно их решение можно будет масштабировать под указанные перспективы. С таким видением платформа APPETIT сможет не только удовлетворить текущие потребности (запуск доставки с аналитикой), но и эволюционировать в мощную экосистему цифровых сервисов для ресторанного бизнеса.