# Проектирование интернет-магазина Marcus (Next.js + Strapi)

## Архитектура приложения

Интернет-магазин Marcus построен по принципу **headless CMS**: фронтенд-приложение на Next.js взаимодействует с CMS бекендом Strapi через API. Такая архитектура обеспечивает гибкость разработки и SEO-оптимизацию за счёт серверного рендеринга контента. Стек технологий включает Next.js (с поддержкой SSR, SSG и ISR), Strapi в роли headless CMS, поисковый движок Meilisearch для полнотекстового поиска, базу данных PostgreSQL, кеширование и очереди на Redis, а хранение медиафайлов в S3-совместимом хранилище с CDN для быстрого доставки изображений[[1]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=Frontend%3A%20Next.js%20%28SSR%2FSSG%2FISR%20%2B%20on,revalidate%2C%20next%2Fimage). Развёртывание всех компонентов планируется на выделенном VPS-сервере под управлением Ubuntu 24.04 LTS с использованием Nginx в качестве reverse-proxy и менеджера процессов PM2 для Node.js-приложений; деплой автоматизирован через CI/CD (GitHub Actions) с разделением сред на dev, stage и prod[[2]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=DB%3A%20PostgreSQL%3B%20%D0%BA%D0%B5%D1%88%2F%D0%BE%D1%87%D0%B5%D1%80%D0%B5%D0%B4%D0%B8%3A%20Redis%3B%20%D0%BC%D0%B5%D0%B4%D0%B8%D0%B0%3A,%D1%81%D0%BE%D0%B2%D0%BC%D0%B5%D1%81%D1%82%D0%B8%D0%BC%D0%BE%D0%B5%20%D1%85%D1%80%D0%B0%D0%BD%D0%B8%D0%BB%D0%B8%D1%89%D0%B5%20%2B%20CDN)[[3]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=Ubuntu%20LTS%2C%20users%2FSSH%2F%D1%84%D0%B0%D0%B5%D1%80%D0%B2%D0%BE%D0%BB%3B%20Nginx%20,js%20%2B%20PM2%20cluster%2C%20logrotate).

**Взаимодействие между компонентами:** Запросы пользователей по HTTPS поступают на Nginx, который проксирует их на Next.js-приложение. Next.js выполняет серверный рендеринг страниц (SSR/ISR) и при необходимости запрашивает данные через REST/API у Strapi. Strapi обращается к базе PostgreSQL для получения контента каталога. Для страниц каталога и поиска фронтенд взаимодействует с Meilisearch, используя его API для выдачи результатов с фасетными фильтрами. Redis используется для кеширования (например, результатов API, сессий пользователя) и для фоновых задач (очереди). Статические файлы и изображения не хранятся на VPS напрямую: Strapi загружает медиа-ресурсы в S3-совместимое облачное хранилище, откуда они раздаются через CDN. При обновлении контента Strapi посылает вебхук уведомления – это запускает **on-demand ISR** у Next.js (выполняется регенерация затронутых страниц) и пересобирает поисковый индекс Meilisearch[[4]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%98%D1%81%D1%82%D0%BE%D1%87%D0%BD%D0%B8%D0%BA%3A%20Oasis%20Catalog%20API%20,%D1%81%20%D1%86%D0%B5%D0%BD%D0%B0%D0%BC%D0%B8%2C%20%D0%BE%D1%81%D1%82%D0%B0%D1%82%D0%BA%D0%B0%D0%BC%D0%B8%20%D0%B8%20%D0%BC%D0%B5%D0%B4%D0%B8%D0%B0)[[5]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=lifecycle%2FIA), чтобы витрина и поиск всегда отображали актуальные данные.

*Примечание:* На этапе расширения функциональности (Этап 2) фронтенд интегрируется с внешними сервисами: API платёжной системы ЮKassa для онлайн-оплаты заказов, а также виджетами и API служб доставки (например, СДЭК, Деловые Линии, Яндекс.Доставка) для расчёта стоимости и выбора пунктов выдачи заказов[[6]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%94%D0%BE%D1%81%D1%82%D0%B0%D0%B2%D0%BA%D0%B0). Эти интеграции выполняются на уровне фронтенда (через SDK/виджеты) или через backend API, обеспечивая бесшовный пользовательский опыт оплаты и доставки.

**SSR/SSG/ISR стратегия:** Для обеспечения быстрого времени отклика и SEO показателей, Next.js будет использовать комбинацию режимов рендеринга: - **SSR (Server-Side Rendering)** – для страниц, требующих актуальных данных при каждом запросе или зависящих от параметров сессии. Например, серверный рендеринг применяется для страницы поиска (первичный вывод результатов по запросу), страницы корзины/оформления заказа (с учётом состояния пользователя) и потенциально для страниц, требующих актуальной информации в реальном времени. - **SSG/ISR (Static Generation с Incremental Static Regeneration)** – для большинства контентных страниц каталога, которые обновляются относительно нечасто. Карточки товаров и страницы категорий генерируются статически для быстрого ответа и SEO (страницы товаров и категорий будут в индексируемом HTML). Витрина будет настроена на **ISR**: страницы генерируются разово и хранятся как статические, но при обновлении данных (например, изменение цены или наличия товара) соответствующая страница автоматически перечёрствляется по вебхуку Strapi[[5]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=lifecycle%2FIA). Также можно настроить периодическую регенерацию (revalidate) через Next.js для гарантированного обновления раз в N секунд, однако предпочтительнее on-demand подход по событиям из CMS. - **SSG (Static Site Generation)** без частой регенерации – для страниц с редко меняющимся контентом: информационные страницы («О компании», «Помощь», политика и оферта) и, например, главная страница (если на ней выводятся промо-блоки, управляемые через Strapi). Эти страницы генерируются при деплое или при изменении соответствующего контента в CMS.

Такой подход комбинированного рендеринга позволит достичь оптимального **SEO** и производительности: поисковые краулеры будут получать полностью сформированный контент с мета-тегами, заголовками и микроразметкой, а пользователи – быстрый первый отклик (TTFB ~800 мс или меньше на 95 перцентиле[[7]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%9F%D1%80%D0%BE%D0%B8%D0%B7%D0%B2%D0%BE%D0%B4%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE%D1%81%D1%82%D1%8C%3A%20API%20p95%20,Web%20Vitals%20%E2%80%94%20%C2%AB%D0%B7%D0%B5%D0%BB%D1%91%D0%BD%D0%B0%D1%8F%C2%BB%20%D0%B7%D0%BE%D0%BD%D0%B0)). Страницы товаров и категорий попадут в поисковые индексы (они перечислены в **sitemap.xml**), тогда как результаты комбинаций фильтров будут помечены noindex и содержать канонический URL базовой категории, чтобы избежать дублирования контента[[8]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=SEO%20%D0%B8%20%D0%BA%D0%BE%D0%BD%D1%82%D0%B5%D0%BD%D1%82). Это значит, что при фильтрации на сайте SEO не пострадает – поисковикам будут видны только основные страницы каталога.

## Модель данных и структура БД (Strapi)

В проекте используется PostgreSQL для хранения данных каталога, а Strapi выступает надстройкой для управления этой базой. Контент в Strapi моделируется через **типы контента** (Collection Types и др.), соответствующие предметной области интернет-магазина Marcus. Ниже перечислены основные сущности и их поля (связи между ними):

* **Category (Категория)** – древовидная категория товаров. Содержит поля: name (название категории), slug (URL-идентификатор), parent (ссылка на родительскую категорию, что позволяет организовать произвольную вложенность категорий). Категория может иметь список дочерних категорий (self-relation). Продукты привязываются к конечным категориям. В Strapi это реализуемо через Component или прямую само-связь, что позволит отразить иерархию «каталог → подкатегория → ...» любой глубины[[9]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%9A%D0%B0%D1%82%D0%B0%D0%BB%D0%BE%D0%B3%2C%20%D0%BA%D0%B0%D1%80%D1%82%D0%BE%D1%87%D0%BA%D0%B0%2C%20%D0%BC%D0%B5%D0%B4%D0%B8%D0%B0).
* **Product (Товар-модель)** – основная сущность товара, описывающая товарную модель (например, футболка определённой модели, имеющая несколько вариантов/цветов). Поля продукта: title (название модели), slug (уникальный URL-идентификатор), baseCategory (ссылка на основную категорию товара), subCategories (массив дополнительных категорий или тегов, если товар относится к нескольким категориям одновременно), attributes (динамическая зона или JSON, описывающая набор атрибутов модели – например, доступные цвета, размеры, материал и т.д.), priceType (тип цены: фиксированная или ступенчатая), prices (массив ценовых опций – например, оптовые цены при разных объёмах заказа), MOQ (минимальный размер заказа, если применимо), hasLogoPrinting (булево – доступна ли печать логотипа на этом товаре), logoMethods (список доступных методов нанесения логотипа – ссылки на справочник PrintingMethod), logoArea (описание области нанесения, напр. размеры площадки под логотип), images (массив изображений товара), SEO-поля (мета-заголовок, описание для поисковиков) и системное поле publishedAt (дата публикации)[[10]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=Product%20,SEO%2C%20publishedAt). **Связи:** Product связан с одной основной категорией (baseCategory, тип связи many-to-one) и может быть связан с несколькими категориями (subCategories, связь many-to-many для дополнительных категорий). Также Product может содержать вложенные компоненты или JSON для атрибутов, которые унаследуют варианты.
* **ProductVariant (Вариация товара)** – отдельный вариант/артикул товара, принадлежащий определённой модели Product. Пример: футболка модель X, вариант – красного цвета, размер L. Поля: title (название вариации, если нужно, иначе может совпадать с моделью + атрибут), slug (URL вариации; важно, что у вариаций будут собственные страницы/ссылки по ТЗ[[9]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%9A%D0%B0%D1%82%D0%B0%D0%BB%D0%BE%D0%B3%2C%20%D0%BA%D0%B0%D1%80%D1%82%D0%BE%D1%87%D0%BA%D0%B0%2C%20%D0%BC%D0%B5%D0%B4%D0%B8%D0%B0)), sku (артикул/код товара), price (цена для этой вариации, если отличается), stock (количество на складе), attributes (JSON с конкретными значениями атрибутов для этой вариации, например {цвет: красный, размер: L}), images (изображения, специфичные для вариации, если есть), isDefault (флаг "основная вариация" – например, вариант по умолчанию, который отображается первым)[[11]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=ProductVariant%20,isDefault). **Связи:** каждая вариация ссылается на родительский Product (связь many-to-one). Атрибуты вариации хранятся денормализованно в JSON для быстрого фильтра в поиске.
* **Attribute (Атрибут)** – справочник доступных атрибутов товаров, которые используются для фильтрации (фасеты) и описания. Содержит: название атрибута, тип (например, строка, число, опция) и, возможно, набор возможных значений (если тип ограничен). Примеры атрибутов: Цвет, Объём (мл), Материал и пр. Атрибуты применяются динамически через поле attributes товара. В Meilisearch каждый атрибут будет зарегистрирован как фасетный фильтр[[12]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%9F%D0%BE%D0%B8%D1%81%D0%BA%20%D0%B8%20%D1%84%D0%B8%D0%BB%D1%8C%D1%82%D1%80%D1%8B%20).
* **PrintingMethod (Метод нанесения)** – справочник методов брендирования/нанесения логотипа. Поля: название метода (например, Тампопечать, УФ-печать), описание, возможно, характеристики (цветность, ограничения). Используется для отображения информации на странице товара (если hasLogoPrinting = true, то показываем блок "Возможны методы нанесения: ...") и для фильтрации по наличию нанесения логотипа.
* **TieredPricing (Диапазоны цен)** – сущность, описывающая ценовые градации для оптовых заказов. Поля: минимальное количество (MOQ), цена за единицу при таком количестве. Связь: принадлежит конкретному Product (или, возможно, хранится внутри Product.prices как компонент). В MVP достаточно хранить эти данные внутри продукта, но вынесение в отдельный тип поможет расширяемости.
* **Order (Заказ)** – сущность заказа (в Strapi можно создать Collection Type "Order"). Поля: данные покупателя (имя, email, телефон), список позиций (массив вложенных компонентов OrderItem с товаром, вариацией, ценой, количеством), статус заказа (напр. new, pending, confirmed, shipped – статусы жизненного цикла заказа)[[13]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%9A%D0%BE%D1%80%D0%B7%D0%B8%D0%BD%D0%B0%20%D0%B8%20%D0%BE%D1%84%D0%BE%D1%80%D0%BC%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5%20%28%D0%B1%D0%B5%D0%B7%20%D0%BE%D0%BD%D0%BB%D0%B0%D0%B9%D0%BD), сумма, выбранный способ доставки, адрес доставки, комментарий и т.п. **Связи:** возможно связь на пользователя (если реализован личный кабинет, см. ниже).
* **Статические страницы** – информацию из разделов "Помощь", "Компания", "Контакты", "Политика конфиденциальности" и др. планируется хранить в Strapi, чтобы контент можно было редактировать. Для этого можно завести тип контента **Page** с полями: title, slug и содержимое (например, Rich Text или Markdown). Альтернативно, для каждого раздела сделать отдельные Single Types (например, single type "Оферта", "Политика", "О компании" и т.д.), но единый тип Page более универсален. В Strapi можно разграничить эти страницы по какому-то полю типа category (например, "help" или "legal") или папкам, но на старте достаточно хранить их как записи с определёнными slug ("/help/delivery", "/policy", и т.п.).

**Денормализация под поиск:** Для эффективной работы поиска и фильтров часть данных будет продублирована или агрегирована: при индексировании товаров в Meilisearch в индекс каждой вариации или модели будут включены поля категории, названия и значения атрибутов, чтобы Meilisearch мог сразу отдать фасетные подсчёты без дополнительных запросов к БД[[4]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%98%D1%81%D1%82%D0%BE%D1%87%D0%BD%D0%B8%D0%BA%3A%20Oasis%20Catalog%20API%20,%D1%81%20%D1%86%D0%B5%D0%BD%D0%B0%D0%BC%D0%B8%2C%20%D0%BE%D1%81%D1%82%D0%B0%D1%82%D0%BA%D0%B0%D0%BC%D0%B8%20%D0%B8%20%D0%BC%D0%B5%D0%B4%D0%B8%D0%B0). Кроме того, поля, по которым осуществляется поиск (название товара, артикул, категория, бренды, возможно, описания) – все будут храниться в индексе. Таким образом, поиск по сайту будет осуществляться полностью через Meilisearch, без прямого обращения к PostgreSQL, что значительно ускорит ответы на поисковые запросы. Strapi будет выступать лишь точкой ввода данных и настройки параметров поиска (например, релевантность, синонимы).

Стоит отметить, что **связи Product-Category** и **Product-ProductVariant** реализованы как типичные отношения one-to-many. В PostgreSQL будут соответствующие таблицы (например, products, product\_variants, categories и таблица связей many-to-many для продуктов и дополнительных категорий). Эти связи может управлять сам Strapi (он создаст необходимые foreign key). Благодаря этому мы сможем выполнять сложные выборки, хотя в рантайме основная нагрузка будет ложиться на Meilisearch.

## Импорт товаров из Oasis Catalog

Источник данных для каталога – **Oasis Catalog API**, официальный API поставщика, откуда Marcus будет получать информацию о товарах. Импорт должен покрывать загрузку **категорий, товарных моделей и вариаций**, включая цены, остатки на складах и изображения[[4]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%98%D1%81%D1%82%D0%BE%D1%87%D0%BD%D0%B8%D0%BA%3A%20Oasis%20Catalog%20API%20,%D1%81%20%D1%86%D0%B5%D0%BD%D0%B0%D0%BC%D0%B8%2C%20%D0%BE%D1%81%D1%82%D0%B0%D1%82%D0%BA%D0%B0%D0%BC%D0%B8%20%D0%B8%20%D0%BC%D0%B5%D0%B4%D0%B8%D0%B0).

**Алгоритм импорта:** процесс будет происходить автоматически один раз в сутки (например, ночью по cron-задаче). Реализуем скрипт-импортёр (на Node.js или с использованием Strapi Cron Tasks), который: 1. Обращается к API Oasis Catalog и запрашивает изменения или весь каталог. Если API предоставляет возможность инкрементальной выборки (например, "товары, обновлённые после даты X"), то используется она. Иначе, реализуется разбиение на **батчи по 1000-5000 товаров**[[14]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%A7%D0%B0%D1%81%D1%82%D0%BE%D1%82%D0%B0%3A%20%D0%B5%D0%B6%D0%B5%D0%B4%D0%BD%D0%B5%D0%B2%D0%BD%D0%BE%20%D0%BD%D0%BE%D1%87%D1%8C%D1%8E%20%28cron%29,demand%20ISR), чтобы не перегружать систему за раз. 2. Обрабатывает полученные данные: для новых или изменённых товаров обновляет записи в Strapi (через его сервисы или прямой SQL). **Дедупликация**: идентифицируем товары по уникальному SKU или slug – если товар с таким SKU уже существует, обновляем его, иначе создаём новый[[14]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%A7%D0%B0%D1%81%D1%82%D0%BE%D1%82%D0%B0%3A%20%D0%B5%D0%B6%D0%B5%D0%B4%D0%BD%D0%B5%D0%B2%D0%BD%D0%BE%20%D0%BD%D0%BE%D1%87%D1%8C%D1%8E%20%28cron%29,demand%20ISR). Это предотвращает дублирование записей при повторном импорте. 3. Обновляет или создаёт соответствующие **категории** (в том числе привязки товаров к категориям). При импорте учитывается вложенность категорий из Oasis: скрипт будет проверять, есть ли такая категория в БД, и создавать недостающие, связывая с родительскими. 4. **Атрибуты** товаров сопоставляются со справочником атрибутов. Если в данных Oasis приходят параметры (например, "цвет: красный"), скрипт проверит, есть ли уже атрибут "Цвет" в Strapi; при отсутствии создаст новую запись Attribute. Значение атрибута (красный) может храниться просто как строка внутри JSON вариации, но для фильтрации важно иметь сам атрибут как фасет. 5. Загрузка **изображений**: API, вероятно, предоставляет URL картинок товаров. Скрипт скачивает новые изображения или берёт ссылки и сохраняет их через Strapi Upload Provider в S3-хранилище. Нужно учесть, что на каждый товар 1–4 ракурса, в формате JPEG/WebP с минимальным размером ~1000×1000[[15]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=Tiered,%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B1%D0%BB%D0%BE%D0%BA%20%D0%BD%D0%B0%20MVP). Скачанные изображения заливаются на S3; Strapi хранит ссылки. 6. Логирование: все операции (количество обработанных товаров, новых, обновлённых, ошибки) фиксируются в лог-файле или таблице логов. В случае сбоев или ошибок импорта (например, API не отвечает) – реализовать **ретраи** (повторные попытки) и уведомление ответственных (например, письмо или сообщение). После успешного импорта можно формировать отчёт (количество новых товаров, обновлений, ошибок).

**После импорта** выполняются действия по актуализации витрины: - По завершении каждого батча (или всего импорта) скрипт будет отправлять запросы к соответствующим **вебхукам Strapi**. Strapi, получив обновлённые записи, сгенерирует события обновления контента. Эти вебхуки настроены так, чтобы вызывать: - **Обновление поискового индекса** Meilisearch – можно интегрировать через Strapi lifecycle hooks или же вызывать отдельный скрипт, который берёт изменённые товары и отправляет их в индекс Meilisearch (через Meilisearch API). Таким образом, новые товары или изменения цен сразу отражаются в поиске[[14]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%A7%D0%B0%D1%81%D1%82%D0%BE%D1%82%D0%B0%3A%20%D0%B5%D0%B6%D0%B5%D0%B4%D0%BD%D0%B5%D0%B2%D0%BD%D0%BE%20%D0%BD%D0%BE%D1%87%D1%8C%D1%8E%20%28cron%29,demand%20ISR). - **On-demand ISR** в Next.js – Next.js предоставляет API-роут для программной регенерации страниц (revalidate). После импорта скрипт делает HTTP-запрос к специальному защищённому endpoint Next.js, передавая, какие страницы необходимо обновить (например, URL изменённого товара или категории). Next.js, получив запрос, сбрасывает кеш этой страницы и при следующем заходе пользователем срендерит её заново и сохранит. Это гарантирует, что контент обновится без полного redeploy сайта[[14]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%A7%D0%B0%D1%81%D1%82%D0%BE%D1%82%D0%B0%3A%20%D0%B5%D0%B6%D0%B5%D0%B4%D0%BD%D0%B5%D0%B2%D0%BD%D0%BE%20%D0%BD%D0%BE%D1%87%D1%8C%D1%8E%20%28cron%29,demand%20ISR)[[5]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=lifecycle%2FIA). - Также после ночного импорта можно автоматически пересобирать **карту сайта (sitemap.xml)**, если используется статический генератор. Однако Next.js и Strapi могут делать это динамически: например, Strapi может хранить обновлённый sitemap в базе, или Next.js при ISR регенерирует sitemap (в ТЗ указано, что sitemap автообновляется при импорте[[16]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%9F%D0%B0%D0%B3%D0%B8%D0%BD%D0%B0%D1%86%D0%B8%D1%8F%20%E2%80%94%20rel%3D)).

Таким образом, каждое обновление в данных Oasis Catalog будет максимум через сутки отражено на сайте. Если нужна более оперативная синхронизация (например, остатки/цены), можно увеличить частоту импорта или использовать вебхуки Oasis (если имеются) для событий в реальном времени. Но MVP предусматривает ежедневный ночной импорт как достаточный вариант.

## Поиск и фильтрация (Meilisearch)

Для поиска по каталогу Marcus внедрён **Meilisearch** – высокопроизводительный open-source поисковый движок, который отлично подходит для e-commerce. Он будет развернут на том же сервере (в контейнере) или на отдельном инстансе, и хранить индекс товаров.

**Индексирование:** В Meilisearch создаётся индекс (например, products) для товаров. В него помещаются документы, соответствующие либо товарным моделям, либо вариациям. Оптимально индексировать именно вариации как отдельные документы, чтобы фильтрация по атрибутам (например, цвет = красный) могла сразу возвращать нужные варианты. Однако, чтобы на выдаче не показывать одну и ту же модель многократно, можно индексировать модели, но с агрегированными атрибутами. В MVP решаем так: **индексируем товарные модели**, но к полям модели добавляем объединённые значения атрибутов всех вариаций (например, модель "Футболка X" будет иметь список доступных цветов ["красный","синий"], размеров ["M","L"] прямо в поисковом документе). Это позволит Meilisearch отвечать на запрос "футболка красная L" находя модель, и фасеты посчитает корректно. На этапе 2 можно усложнить логику (индексировать и модели и вариации для разных задач).

**Фасеты и фильтры:** При настройке Meilisearch указываем набор полей для фасетной фильтрации – это **атрибуты товаров, категория, наличие (stock), цена** и другие необходимые поля[[12]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%9F%D0%BE%D0%B8%D1%81%D0%BA%20%D0%B8%20%D1%84%D0%B8%D0%BB%D1%8C%D1%82%D1%80%D1%8B%20). Например, атрибут "Цвет" и "Объём" будут фасетами: Meilisearch по каждому поисковому запросу будет возвращать не только список товаров, но и агрегаты – сколько товаров каждого цвета, объёма и т.д. Это позволит на сайте отображать фильтры с количеством результатов. Если число уникальных значений какого-то атрибута велико (так называемый "длинный хвост"), будем показывать топ-5 значений, а остальные скрывать под поиск по значению (функциональность *facet search*, например пользователь может начать вводить цвет, чтобы отфильтровать список значений)[[17]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%9E%D0%BF%D0%B5%D1%87%D0%B0%D1%82%D0%BA%D0%B8%3A%201%20%D0%B4%D0%BB%D1%8F%20%D0%BA%D0%BE%D1%80%D0%BE%D1%82%D0%BA%D0%B8%D1%85%20,%D0%9D%D0%B0%D1%81%D1%82%D1%80%D0%B0%D0%B8%D0%B2%D0%B0%D0%B5%D1%82%D1%81%D1%8F%20%D0%BF%D0%B0%D1%80%D0%B0%D0%BC%D0%B5%D1%82%D1%80%D0%B0%D0%BC%D0%B8%20Meilisearch). Также предусмотрено, что если значений больше 10, включается поиск по значениям прямо в меню фильтра (это и есть facet-search)[[17]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%9E%D0%BF%D0%B5%D1%87%D0%B0%D1%82%D0%BA%D0%B8%3A%201%20%D0%B4%D0%BB%D1%8F%20%D0%BA%D0%BE%D1%80%D0%BE%D1%82%D0%BA%D0%B8%D1%85%20,%D0%9D%D0%B0%D1%81%D1%82%D1%80%D0%B0%D0%B8%D0%B2%D0%B0%D0%B5%D1%82%D1%81%D1%8F%20%D0%BF%D0%B0%D1%80%D0%B0%D0%BC%D0%B5%D1%82%D1%80%D0%B0%D0%BC%D0%B8%20Meilisearch).

**Сортировка результатов:** Meilisearch позволит задать кастомные поля сортировки. По ТЗ нужно реализовать сортировки по *популярности, цене и новизне*[[18]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%9F%D0%BE%D0%B8%D1%81%D0%BA%20%D0%B8%20%D1%84%D0%B8%D0%BB%D1%8C%D1%82%D1%80%D1%8B%20). Поле популярности может определяться либо вручную (свойство товара) либо по статистике просмотров/продаж (можно обновлять отдельным процессом). В индекс добавим поле popularity (число) и createdAt (для новизны), а также price. Тогда фронтенд сможет запрашивать сортировку: по умолчанию, скажем, по популярности, а по выбору пользователя – по цене (возр/убыв) или по дате добавления (новые выше).

**Толерантность к опечаткам:** Meilisearch поддерживает fuzzy search. По требованиям, допускаем 1 опечатку для коротких запросов (до 6 символов) и до 2 – для длинных (≥7-9 символов)[[19]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%A4%D0%B0%D1%81%D0%B5%D1%82%D1%8B%20%D0%BF%D0%BE%20%D0%B0%D1%82%D1%80%D0%B8%D0%B1%D1%83%D1%82%D0%B0%D0%BC%3B%20%D1%81%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B8%20%28%D0%BF%D0%BE%D0%BF%D1%83%D0%BB%D1%8F%D1%80%D0%BD%D0%BE%D1%81%D1%82%D1%8C%2F%D1%86%D0%B5%D0%BD%D0%B0%2F%D0%BD%D0%BE%D0%B2%D0%B8%D0%B7%D0%BD%D0%B0%29). Эти настройки будут заданы в конфигурации индексатора (Meilisearch автоматически может управлять этим через параметры typoTolerance). Так пользователь найдёт товар даже если ошибся в раскладке или написании названия.

**Синонимы и словарь:** На первом этапе (MVP) поиск будет работать по прямым соответствиям и опечаткам. В **этапе 2 (рост)** планируется подключить словарь синонимов (двуязычных RU/EN)[[20]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%94%D0%BB%D0%B8%D0%BD%D0%BD%D1%8B%D0%B9%20%D1%85%D0%B2%D0%BE%D1%81%D1%82%20%D1%84%D0%B0%D1%81%D0%B5%D1%82%D0%BE%D0%B2%3A%20%D1%81%D0%B2%D0%BE%D1%80%D0%B0%D1%87%D0%B8%D0%B2%D0%B0%D0%B5%D0%BC%20%D0%B4%D0%BE,search%20%D0%BF%D1%80%D0%B8%20%3E10%20%D0%B7%D0%BD%D0%B0%D1%87%D0%B5%D0%BD%D0%B8%D0%B9). Это значит, что мы подготовим список синонимов и вариантных названий товаров. Например, "флешка" ≈ "USB-накопитель", или английские названия основных категорий. Эти синонимы можно будет внести в Meilisearch (он поддерживает синонимы на уровне индекса) или даже связать с админкой Strapi (например, отдельный content type "SearchSynonym" для управления словарём, а при обновлении его – триггерить обновление настроек Meilisearch). В результате, пользователь найдёт товары даже по альтернативным названиям.

**UX поиска:** На фронтенде поиск будет представлен как поле ввода с автодополнением. Возможна реализация подсказок: при вводе запроса в поле, клиентский скрипт обращается к Meilisearch (с определённой задержкой) за *наилучшими совпадениями* и отображает выпадающий список подсказок (например, товары или категории). Это улучшит пользовательский опыт. Если пользователь просто жмёт "Enter", он попадает на страницу результатов поиска, где отображается список товаров, релевантных запросу, и боковая панель с фильтрами (похожие на страницу категории). Страница поиска, однако, не индексируется поисковиками (мы поставим noindex для неё), поэтому можно её генерировать на клиенте. Но для консистентности и возможности шарить ссылку на поиск – мы реализуем её как динамическую страницу Next.js: при первоначальном рендере можно серверно выполнить запрос к Meilisearch для заданного q (если передан query param) и отдать первые результаты, а дальнейшая фильтрация/пагинация уже будет происходить AJAX-запросами.

Наконец, поиск должен быть **быстрым** – Meilisearch легко укладывается в миллисекунды на тысячи товаров, но важно мониторить производительность. Будут настроены индексы, отключены лишние поля из выдачи, чтобы ответы были минимально необходимыми. Это поможет удержать время отклика поиска даже под нагрузкой в требуемом диапазоне (p95 API < 300 мс)[[21]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%9F%D1%80%D0%BE%D0%B8%D0%B7%D0%B2%D0%BE%D0%B4%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE%D1%81%D1%82%D1%8C%3A%20API%20p95%20,Web%20Vitals%20%E2%80%94%20%C2%AB%D0%B7%D0%B5%D0%BB%D1%91%D0%BD%D0%B0%D1%8F%C2%BB%20%D0%B7%D0%BE%D0%BD%D0%B0).

## Фронтенд (Next.js) и стратегия рендеринга страниц

Фронтенд реализован на **Next.js**, что позволяет использовать React-компоненты и одновременно получать преимущество серверного рендеринга. Основные моменты реализации фронтенда Marcus:

* **Структура приложения:** Будет настроен единый Layout для всех страниц, включающий шапку (header) и подвал (footer), а также компоненты, повторяющиеся на страницах (например, виджет чата, модальное окно уведомлений о куки). Next.js позволит определить Layout на уровне \_app или использовать новую систему App Router (Next 13+) с шаблонами. Метатеги (title, description, OpenGraph) настраиваются на уровне страниц, используя данные из Strapi (SEO-поля в товарах, категориях, а для статических страниц – ручной ввод). Также добавим глобальные компоненты: например, компонент для <Head> с общими тегами, включающими фавиконки, подключение шрифтов, скриптов аналитики и т.п. (эти общие части входят в задание 5.1 фронтенда[[22]](file://file-1yjcu1K12osyUYjxJV5SHb#:~:text=%D0%AD%D1%82%D0%B0%D0%BF%205%20%E2%80%94%20Frontend%20%28Next,7%20%D0%9F%D0%BE%D1%80%D1%82%D1%84%D0%BE%D0%BB%D0%B8%D0%BE)).
* **SSR/ISR реализация:** Как описано ранее, Next.js будет генерировать страницы товаров и категорий статически (с ISR). Конкретно:
* **Страницы товаров (PDP)** – будут реализованы как динамические маршруты с getStaticPaths + getStaticProps. На этапе сборки (или при первом обращении с fallback) Next.js получит список популярных товаров для предварительной генерации, а остальные – через fallback. После создания/обновления товара Strapi через вебхук вызовет /api/revalidate?secret=...&path=/catalog/product-slug на фронтенде, чтобы тот обновил кеш страницы[[5]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=lifecycle%2FIA). Таким образом, информация о товаре (цена, наличие) всегда актуальна максимум с задержкой в несколько минут.
* **Страницы категорий (PLP)** – аналогично, генерируются статически. Можно предварительно сгенерировать основные категории, а вложенные – тоже (если немного) или через fallback при первом запросе. Категории меняются редко, поэтому ISR с большим интервалом подходит.
* **Главная страница** – генерируется статически при каждом деплое. Если содержимое главной (баннеры, тексты) управляется из Strapi, при изменении можно также дергать revalidate для /.
* **Статические страницы** («О компании», «Помощь/Как купить», «Контакты», «Политика конфиденциальности» и др.) – можно реализовать как SSG (они редактируются через Strapi, поэтому, либо использовать ISR с большим таймаутом, либо дергать ондеманд).
* **Страница поиска** – особый случай: она не требуется для SEO (не индексируется), и её содержимое зависит от запроса пользователя. Мы можем рендерить её на клиенте (React состояние) либо сделать SSR при наличии query. План такой: по URL /search?q=... Next.js может SSR-отрендерить начальный результат (чтобы при открытии ссылки напрямую человек увидел результаты). Но основные фильтрации будут работать через JS на клиенте (то есть Next.js, отрендерив страницу, подключит скрипт, который при взаимодействии будет обращаться к Meilisearch).
* **Страницы корзины и оформления заказа** – эти страницы не нужны поисковикам, но для лучшего UX мы сделаем SSR/SSG. Например, корзину можно отрендерить на сервере пустой (если у пользователя нет cookie с товарами) или с уже добавленными товарами (если сохранять корзину в cookie/localStorage, можно передать её на сервер через cookie). Оформление заказа – SSR, но там в основном форма, особой выгоды SSR нет кроме быстрого отображения каркаса.
* **Личный кабинет** (этап 2) – доступен только авторизованным пользователям; можно рендерить на сервере после проверки сессии (SSR с проверкой токена).
* **Обработка данных на фронте:** Для взаимодействия с Strapi будет использоваться либо REST API (через fetch/Axios) либо GraphQL (если включим плагин). В MVP, вероятно, достаточно REST. Next.js серверные методы (getStaticProps, getServerSideProps) будут обращаться к Strapi API по HTTP(S) – для этого храним API URL и токен доступа (для публикаций Strapi может открыть публичный доступ к нужным типам контента в режиме чтения, или использовать API Token). При отрисовке страниц Next.js получит JSON от Strapi и передаст данные React-компонентам.
* **Интерактивность и состояние:** Корзина будет храниться на стороне клиента (в localStorage или context) для мгновенного отклика при добавлении товара. Однако, чтобы не потерять данные, при оформлении заказа эти данные будут отправлены на бэкенд (Strapi) для создания сущности Order. Некоторый функционал (например, выбор города доставки, виджет ПВЗ) будет подключаться как сторонние скрипты на странице оформления заказа (в этапе 2, с интеграцией доставок).
* **Оптимизация производительности:** Next.js предоставляет оптимизированный компонент для изображений <Image>, который будет использоваться для всех товарных картинок – это даст ресайзинг и оптимальный формат (WebP) на лету, плюс lazy-loading для картинок вне экрана[[15]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=Tiered,%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B1%D0%BB%D0%BE%D0%BA%20%D0%BD%D0%B0%20MVP). Сами оригиналы изображений хранятся на S3, а через CDN будут раздаваться быстро по географии. Кроме того, применим стандартные практики: код сплиттинг (разделение бандла, чтобы, например, страницы админки/личного кабинета не тянулись на главную), prefetch критичных данных (Next может предзагрузить JSON для соседних страниц, например, на странице категории предзагрузить данные популярных товаров). Библиотеки, не используемые сразу, будут загружаться динамически (import('module')).
* **Core Web Vitals:** Цель – показатели в зелёной зоне (LCP, FID, CLS). Для этого: SSR/ISR обеспечит быстрый LCP (первый контент уже на сервере), обдуманный загрузчик изображений улучшит CLS (избежим скачков, задавая размеры img). FID – обеспечим за счёт минимизации тяжелых скриптов, использование CDN. Также подключим мониторинг Perfomance API – можно отправлять показатели на аналитический сервер или использовать Google Analytics 4 для отслеживания Core Web Vitals.

В целом, фронтенд будет реализован модульно и c соблюдением SEO требований: на всех страницах пропишем человеко-понятные URL и мета-теги (с динамическим подстановкой из CMS), добавим **микроразметку Schema.org** для товаров (Product, Offer) и хлебных крошек (BreadcrumbList) прямо в HTML[[23]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%98%D0%BD%D0%B4%D0%B5%D0%BA%D1%81%D0%B0%D1%86%D0%B8%D1%8F%3A%20%D0%BA%D0%B0%D1%82%D0%B5%D0%B3%D0%BE%D1%80%D0%B8%D0%B8%20%D0%B8%20%D1%82%D0%BE%D0%B2%D0%B0%D1%80%D1%8B%3B%20%D0%BA%D0%BE%D0%BC%D0%B1%D0%B8%D0%BD%D0%B0%D1%86%D0%B8%D0%B8,%D0%BD%D0%B0%20%D0%B1%D0%B0%D0%B7%D0%BE%D0%B2%D1%83%D1%8E%20%D0%BA%D0%B0%D1%82%D0%B5%D0%B3%D0%BE%D1%80%D0%B8%D1%8E), что повысит представление в поисковых системах.

## Инфраструктура на VPS и DevOps

Развёртывание приложения будет происходить на **виртуальном выделенном сервере (VPS)** под управлением Ubuntu 24.04. В рамках MVP предполагается единый сервер, который будет выполнять роли веб-сервера, сервера приложений и базы данных (для упрощения), хотя компоненты изолируются в Docker-контейнерах. Ниже описаны ключевые элементы инфраструктуры и DevOps:

### Сервер, сеть и безопасность

* **Сервер и ОС:** Ubuntu LTS 24.04 обеспечивает актуальную и стабильную платформу. Будут настроены пользователи для разработчиков/деплоя, SSH-ключи для входа без пароля, а также базовая безопасность: отключение root-входа, настройка брандмауэра (UFW) для разрешения только нужных портов (80, 443 для Nginx, 22 для SSH; порты приложений – закрыты извне).
* **Reverse Proxy (Nginx):** На сервере устанавливается Nginx, который будет выполнять несколько задач: терминация SSL (приём HTTPS-трафика), редиректы HTTP→HTTPS, сжатие (Gzip/Brotli) и кеш статических файлов, а также проксирование запросов к backend-сервисам. Nginx настроим с поддержкой HTTP/2, HSTS (Strict-Transport-Security) для улучшенной безопасности[[3]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=Ubuntu%20LTS%2C%20users%2FSSH%2F%D1%84%D0%B0%D0%B5%D1%80%D0%B2%D0%BE%D0%BB%3B%20Nginx%20,js%20%2B%20PM2%20cluster%2C%20logrotate). Он будет слушать 80/443 порты; сертификаты выпускаются через **Let's Encrypt** (Certbot) и авто-продлеваются каждые ~60 дней[[24]](file://file-1yjcu1K12osyUYjxJV5SHb#:~:text=%D0%A0%D0%B0%D0%B7%D0%B2%D1%91%D1%80%D0%BD%D1%83%D1%82%20VDS%2C%20%D0%BD%D0%B0%D1%81%D1%82%D1%80%D0%BE%D0%B5%D0%BD%D1%8B%20Docker%2FDocker%20Compose%2C,proxy%20%28nginx%29%2C%20%D0%BB%D0%BE%D0%B3%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)[[25]](file://file-1yjcu1K12osyUYjxJV5SHb#:~:text=%D0%92%D1%8B%D0%BF%D1%83%D1%89%D0%B5%D0%BD%D1%8B%20%D0%B8%20%D0%B0%D0%B2%D1%82%D0%BE%D0%BF%D1%80%D0%BE%D0%B4%D0%BB%D0%B5%D0%B2%D0%B0%D1%8E%D1%82%D1%81%D1%8F%20HTTPS,Encrypt%2FCertbot).
* **Маршрутизация запросов:** Все внешние запросы идут через Nginx. Запросы к пользовательскому фронтенду (Next.js) будут проксированы на порт, на котором работает Next (например, :3000). Запросы к Strapi CMS (админка и API) – на порт Strapi (например, :1337). Nginx по location определит, что, скажем, https://marcus.su/admin или /api/ – это Strapi, а остальные – Next.js. Также Nginx может раздавать напрямую некоторые статические ресурсы, если настроить путь (например, если Next генерирует файл sitemap.xml или robots.txt, Nginx может их отдавать с диска, хотя проще проксировать к Next).
* **Docker и окружение:** На сервере устанавливается Docker Engine и docker-compose. Планируется упаковать компоненты (Next.js фронт, Strapi бэкенд, Meilisearch, а также Redis, PostgreSQL) в контейнеры. Таким образом, инфраструктура будет описана в **docker-compose.yml**, где каждый сервис – отдельный контейнер, что облегчает управление и обновление. Например, образы: node:18 для Next и Strapi (или специальные образы, если сделаем multi-stage build, об этом ниже), postgres:15 для БД, redis:7, getmeili/meilisearch для поиска.
* **Сеть Docker:** Все контейнеры будут в одной приватной сети, Nginx на хосте будет связан с контейнерами через их экспонированные порты. (Либо альтернативно Nginx тоже можно завернуть в контейнер и использовать Docker-сеть, тогда 80/443 маппить на Nginx-контейнер. Но часто удобнее Nginx на хосте оставить для simplicity).
* **Безопасность Strapi:** Административная панель Strapi будет доступна извне (например, по URL /admin), но мы примем меры защиты. Во-первых, настроим **basic auth или IP restriction** в Nginx для /admin пути – например, разрешим доступ только с определённого IP офиса или через VPN. Если это неудобно, как альтернатива – включим двухфакторную аутентификацию (2FA) для учётных записей админки. Strapi из коробки не имеет 2FA, но существуют плагины или можно через обратный прокси требовать клиентский сертификат. Минимально реализуем ограничение по IP на уровне firewall или Nginx, чтобы снизить поверхность атаки[[26]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%91%D0%B5%D0%B7%D0%BE%D0%BF%D0%B0%D1%81%D0%BD%D0%BE%D1%81%D1%82%D1%8C%3A%20HTTPS%2C%20HSTS%2C%20%D1%81%D0%B5%D0%BA%D1%80%D0%B5%D1%82%D1%8B%20%D0%B2,%D0%B4%D0%BE%D1%81%D1%82%D1%83%D0%BF%20%D0%BA%20%D0%B0%D0%B4%D0%BC%D0%B8%D0%BD%D0%BA%D0%B5%20%E2%80%94%20IP%2F2FA). Кроме того, все учетные записи админки получат сложные пароли, и отключим создание новых админов через публичный API (оставим только вручную).
* **Хранение секретов:** На сервере все чувствительные данные (ключи API, токены, пароли БД, секреты JWT Strapi, секреты Next ISR) хранятся **в виде переменных окружения**, а не в репозитории кода. В CI/CD конвейере эти переменные задаются через Secrets. Можно использовать отдельный Vault или .env файл, но важно, чтобы .env не попадал в Гит. Переменные (особенно JWT\_SECRET Strapi, API Token) будут сгенерированы и хранятся безопасно. Это соответствует требованию хранения секретов в окружении[[26]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%91%D0%B5%D0%B7%D0%BE%D0%BF%D0%B0%D1%81%D0%BD%D0%BE%D1%81%D1%82%D1%8C%3A%20HTTPS%2C%20HSTS%2C%20%D1%81%D0%B5%D0%BA%D1%80%D0%B5%D1%82%D1%8B%20%D0%B2,%D0%B4%D0%BE%D1%81%D1%82%D1%83%D0%BF%20%D0%BA%20%D0%B0%D0%B4%D0%BC%D0%B8%D0%BD%D0%BA%D0%B5%20%E2%80%94%20IP%2F2FA).
* **Логи и мониторинг:** Включим логирование доступа и ошибок Nginx в файлы, с ротацией (logrotate). Логи Node-приложений (Next, Strapi) можно писать в stdout (Docker будет собирать) или настроить Winston logger на файл. Будет настроено наблюдение за сервером: минимум, подключим внешний сервис мониторинга доступности (например, UptimeRobot или собственный скрипт cron с ping). Желательно также метрики: например, Netdata или Prometheus Node exporter для ресурсов сервера. Алерты: при падении сайта (код 5xx или недоступность) — уведомление разработчиков по почте/мессенджеру. Также настроим специфичные алерты: если ночной импорт завершился с ошибками или не обновил данные – оповещение; если индекс поиска не успел обновиться – сигнал; если в логах Strapi появились ошибки 500 – сигнал разработке[[27]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%9D%D0%B0%D0%B1%D0%BB%D1%8E%D0%B4%D0%B0%D0%B5%D0%BC%D0%BE%D1%81%D1%82%D1%8C%3A%20%D0%BB%D0%BE%D0%B3%D0%B8%20%D0%B7%D0%B0%D0%BF%D1%80%D0%BE%D1%81%D0%BE%D0%B2%2F%D0%BE%D1%88%D0%B8%D0%B1%D0%BE%D0%BA%2C%20%D0%BC%D0%BE%D0%BD%D0%B8%D1%82%D0%BE%D1%80%D0%B8%D0%BD%D0%B3%20%D0%B0%D0%BF%D1%82%D0%B0%D0%B9%D0%BC%D0%B0%2C,%D0%B8%D0%BC%D0%BF%D0%BE%D1%80%D1%82%2C%20%D1%80%D0%B5%D0%B8%D0%BD%D0%B4%D0%B5%D0%BA%D1%81%2C%205xx). Это поможет быстро реагировать на проблемы.
* **Резервное копирование:** Организуем ежедневные бэкапы ключевых данных. PostgreSQL – автоматический дамп базы каждую ночь (pg\_dump), хранящий N последних копий на удалённом хранилище (можно на том же S3 или другом сервере)[[28]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%91%D1%8D%D0%BA%D0%B0%D0%BF%D1%8B%3A%20PostgreSQL%20,Meilisearch%20snapshot%2C%20S3%20lifecycle%2FIA). Meilisearch – поддерживает снапшоты, их тоже можно сохранять раз в сутки, но поскольку индекс можно перестроить, критично бэкапить скорее контент. Медиафайлы на S3 – настроим Lifecycle-политику (например, копирование на холодное хранение или версионирование, если поддерживается)[[28]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%91%D1%8D%D0%BA%D0%B0%D0%BF%D1%8B%3A%20PostgreSQL%20,Meilisearch%20snapshot%2C%20S3%20lifecycle%2FIA). Также перед крупными обновлениями (деплой) можно делать snapshot БД. Восстановление (RTO) нацелено на 4 часа, а максимальная потеря данных (RPO) – 24 часа[[29]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%9D%D0%B0%D0%B4%D1%91%D0%B6%D0%BD%D0%BE%D1%81%D1%82%D1%8C%3A%2099.9,%D0%91%D0%94%2F%D0%BC%D0%B5%D0%B4%D0%B8%D0%B0), что соответствует ночному бэкапу.

### CI/CD: автоматизация сборки и деплоя

Разработка ведётся с GitHub, поэтому для CI/CD выбраны **GitHub Actions**. Будут настроены отдельные пайплайны для фронтенда и бэкенда (или объединённый монорепо-пайплайн, если код фронта и бэка хранятся вместе).

Основные шаги CI/CD:

1. **Сборка (Build):** При push в ветку dev (для стейджинга) или main (для продакшна) запускается workflow. Он использует базовый образ сборки Ubuntu 24.04 (Codex Universal image) – это среда, где уже установлены основные утилиты. В pre-install скрипте Action мы установим необходимые зависимости, например: обновления, npm ci для установки пакетов. Убедимся, что образ содержит Node.js нужной версии (например, Node 18 LTS). Если нет – явно установим Node через actions/setup-node. Также убедимся, что для сборки Next.js требуется sharp – на Ubuntu 24.04 возможно понадобится libvips (если sharp не предустановлен). Такие вещи можно прописать в шаг preinstall: apt-get install -y libvips (псевдокод).

2. **Тестирование и линтинг:** (При наличии тестов) запуск юнит-тестов, линтер (ESLint) и сборка TypeScript. В MVP, возможно, auto-tests минимальны, основная проверка – что проект компилируется.

3. **Сборка Docker-образов:** Поскольку используем Docker, pipeline может собирать образы: один для фронтенда, другой для бэкенда. Например, для Next.js: используем multi-stage Dockerfile – на первом этапе собираем production build (next build), на втором – берем легковесный Node образ и копируем туда .next и node\_modules (production only) и запускаем next start. Для Strapi: похожий подход – первый этап устанавливает пакеты, запускает strapi build (сборка админки), второй – копирует всё нужное (build, API, plugins) и запускает strapi start. После сборки образы публикуются в контейнерный регистр (например, GitHub Container Registry или частный). Версионирование образов – по git SHA или тегу.

4. **Миграции БД:** Прежде чем запускать новую версию бэкенда, выполняются миграции. Strapi имеет систему миграций контента (конфигурационные изменения). Если мы добавляли новые content type, Strapi применит их автоматически при стартe, но на проде лучше выполнять strapi develop --no-run или миграционный CLI. Можно также иметь SQL миграции для ручных изменений. В CI/CD обеспечим шаг: «выполнить миграции». Например, можно временно поднять контейнер Strapi с командой для миграции, либо использовать утилиту. В планировании указано «build + миграции БД + zero-downtime deploy»[[3]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=Ubuntu%20LTS%2C%20users%2FSSH%2F%D1%84%D0%B0%D0%B5%D1%80%D0%B2%D0%BE%D0%BB%3B%20Nginx%20,js%20%2B%20PM2%20cluster%2C%20logrotate), значит мы заранее продумываем обновление схем без простоя.

5. **Развёртывание (Deploy):** После успешной сборки (и тестов) пайплайн автоматически деплоит на целевой сервер. Так как у нас VDS, выберем способ: либо SSH-подключение к серверу и выполнение docker-compose pull/up, либо настроим GitHub Actions runner прямо на сервере (self-hosted runner) для деплоя. Рассмотрим вариант с SSH: в Actions хранится приватный ключ сервера, шаг деплоя делает ssh user@server 'cd /opt/marcus && docker-compose pull && docker-compose up -d'. Compose подтянет новые образы и перезапустит сервисы. **Zero-downtime:** Docker Compose перезапускает контейнеры — чтобы исключить простой, можно перезагружать по одному: сначала обновить бэкенд, затем фронт. При обновлении фронтенда на Next.js можно реализовать плавный рестарт PM2 (если без docker). Но с Docker, вероятно, будет несколько секунд перерыва. Для минимизации, поднимем второй экземпляр контейнера Next на другом порту, переключим Nginx (blue-green deployment). Однако, учитывая 99.9% uptime требование[[29]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%9D%D0%B0%D0%B4%D1%91%D0%B6%D0%BD%D0%BE%D1%81%D1%82%D1%8C%3A%2099.9,%D0%91%D0%94%2F%D0%BC%D0%B5%D0%B4%D0%B8%D0%B0), постараемся, чтобы развёртывание занимало секунды ночью и не влияло на пользователей.

6. **Переменные окружения:** В workflows для stage и prod будут использоваться разные наборы env (наборы секретов). Например, для stage – своя БД, своя базовая URL, test API ключи; для prod – боевые ключи. GitHub Actions позволяет создавать **Environments** (например, "Staging" и "Production") с соответствующими секретами (DB\_PASSWORD, JWT\_SECRET, S3\_KEYS, и т.д.). Перед деплоем на сервер эти секреты загружаются и записываются либо в .env файл контейнера, либо в docker-compose.yml (как environment).

7. **Изолированные контуры:** dev, stage, prod – это три независимых окружения. Dev – локальная разработка (с локальными Docker или SQLite, etc). Stage – сервер, где собирается такая же инфраструктура, но с тестовыми данными. Прод – боевой. CI/CD настроен так, что деплой в stage происходит например при пуше в develop или по ручному запуску workflow, а в prod – при пуше тэга релиза или слиянии в main с повышением версии. Это предотвращает случайный выкат сырого кода на боевой. Также, на staging можно прогонять UAT (user acceptance testing) по чек-листам, и только после подтверждения – продакшн. Участие AI-инструмента Codex здесь: он может проверить конфигурацию, сгенерировать типовой Dockerfile, или подсказать оптимизации в yaml – это используется разработчиками, но сам pipeline выполнит развёртывание.

8. **Контроль качества на этапе деплоя:** После деплоя, workflow может отправить уведомление (например, в Telegram или email) о успешном релизе. Кроме того, сразу после выката мы выполняем smoke-тесты: пингуем главную страницу, делаем тестовый запрос к API статической страницы, чтобы убедиться что контейнеры подняты и отвечают. Если что-то не так – pipeline может автоматически откатить изменения (например, вернёт старую версию образов). На практике, можно просто уведомить ответственных, а откат выполнить вручную через docker-compose rollback (если храним предыдущий compose).

Особенность интеграции с **Codex** (AI): помимо помощи в написании кода, Codex может быть использован для генерации частей CI/CD конфигурации. Например, можно поручить Codex написать GitHub Actions YAML, включающий кеширование npm, или написать Terraform скрипт для поднятия инфраструктуры. Эти артефакты потом проверяются разработчиками. Codex может ускорить такие задачи, но финальное решение и ответственность – за DevOps-инженером.

### Инфраструктура приложений и сервисов

* **Node.js приложения (Next, Strapi):** будут запускаться либо внутри Docker с одной репликой каждого (в начальной конфигурации), либо с использованием PM2 cluster mode (если не контейнеризировать, PM2 может поднять, например, 2-4 экземпляра Next.js для распараллеливания). В Docker-режиме масштабирование вертикальное (увеличение CPU/RAM) или ручное горизонтальное (запустить несколько контейнеров и балансировать). Пока трафик невелик, хватит 1 инстанса Next (Node.js) и 1 инстанса Strapi. PM2 тем не менее можно использовать внутри контейнера Next.js, чтобы использовать все ядра – тогда контейнер стартует PM2, который запускает несколько воркеров Next.js. Однако это усложняет graceful shutdown. Возможно, лучше не использовать PM2 внутри Docker, а если потребуется масштаб – масштабировать контейнеры и добавить load-balancing в Nginx (Nginx upstream с несколькими targets).
* **База данных (PostgreSQL):** Для простоты, PostgreSQL развернём на том же сервере, в контейнере. Данные БД хранятся в volume на диске сервера (для сохранности при пересоздании контейнера). Настроим оптимальный конфиг для работы с Strapi (соединений немного, основные запросы – чтение). В продакшне можно вынести Postgres на отдельный управляемый сервис, но MVP допускает локальный. Не забудем про регулярные бэкапы как отмечалось. Мониторинг БД: хотя бы следить за объемом данных, длинными запросами (можно включить slow query log).
* **Кеш и очереди (Redis):** Redis тоже поднимается в контейнере. Он используется Strapi-Backend и, возможно, фронтом. В Strapi мы можем использовать Redis как кеш для часто запрашиваемых API (например, страница категории – кешировать JSON на несколько минут чтобы не гонять SQL каждый раз). Так как основной поиск вынесен, нагрузка на БД мала, но Redis все равно поможет сессии хранить, если потребуются авторизации пользователей. Также Redis послужит для очередей: например, отправка email подтверждения заказа – можно помещать задачу в очередь, которую обрабатывает воркер (можно запустить маленький Node-скрипт, подписанный на Redis, или использовать встроенные возможности Strapi Hook/Lifecycle). Это разгрузит основной поток запросов.
* **Хранилище медиа (S3):** Медиафайлы (изображения товаров, может быть файлы сертификатов, если будут) хранятся в S3-совместимом облаке. Возможные варианты: AWS S3, либо российские аналоги – Яндекс Облако Object Storage, Selectel, либо self-hosted MinIO на том же сервере (но последний вариант менее отказоустойчив). Мы выберем, например, сервис Object Storage, чтобы не утяжелять VPS. Strapi настраивается через Upload Provider: укажем ключи доступа, bucket и регион. При загрузке изображения через Strapi, файл пойдёт прямо в облако, а Strapi сохранит URL. Для ускорения доставки подключим CDN: либо облачный (если используем, например, Cloudflare перед S3, либо у самого провайдера Object Storage может быть встроенный CDN). Это даст быстрые загрузки картинок пользователям по всему РФ.
* **Дополнительные сервисы:** Meilisearch, как сказано, работает в контейнере. Ему нужен persistent volume для хранения индексов. Размер индекса не очень большой (зависит от числа товаров; даже на 100k товаров – порядка сотен МБ). Meili будет настроен без аутентификации на localhost (он не торчит наружу, только фронт обращается с сервера, или через Next API proxy). При желании, можно задавать ключ API для безопасности.

### Правовые и регуляторные аспекты

Проект Marcus должен соответствовать не только функциональным требованиям, но и правовым: - Будут реализованы страницы **«Политика конфиденциальности»** и **«Пользовательское соглашение (Оферта)»** – эти документы доступны на сайте, хранятся в Strapi для редактирования[[30]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=GA4%20%2B%20%D0%B1%D0%B0%D0%B7%D0%BE%D0%B2%D1%8B%D0%B5%20e,%D0%BF%D0%BB%D0%B0%D1%82%D0%B5%D0%B6%D0%B5%D0%B9%20%D0%B2%20MVP). При оформлении заказа пользователь должен подтвердить согласие с офертой (чекбокс). - На сайт добавим баннер **Cookie Consent** – уведомление о использовании cookies, с возможностью согласиться (для соответствия законодательству и требованиям аналитики)[[30]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=GA4%20%2B%20%D0%B1%D0%B0%D0%B7%D0%BE%D0%B2%D1%8B%D0%B5%20e,%D0%BF%D0%BB%D0%B0%D1%82%D0%B5%D0%B6%D0%B5%D0%B9%20%D0%B2%20MVP). - **Аналитика (GA4):** Google Analytics 4 будет подключена через Google Tag Manager либо напрямую. Будут настроены базовые e-commerce события: просмотр товара (view\_item), добавление в корзину (add\_to\_cart), начало оформления (begin\_checkout) и т.д., вплоть до покупки (purchase)[[31]](file://file-1yjcu1K12osyUYjxJV5SHb#:~:text=%D0%B7%D0%B0%D0%BA%D0%B0%D0%B7%D0%B0%20%D0%B2%20%D0%B1%D0%B5%D0%BA%D0%B5%D0%BD%D0%B4%D0%B5%2C%20%D1%81%D0%B5%D1%80%D0%B2%D0%B8%D1%81%D0%BD%D1%8B%D0%B5%20%D0%BF%D0%B8%D1%81%D1%8C%D0%BC%D0%B0)[[32]](file://file-1yjcu1K12osyUYjxJV5SHb#:~:text=GA4%2FTag%20Manager%20%D0%BD%D0%B0%D1%81%D1%82%D1%80%D0%BE%D0%B5%D0%BD%D1%8B%2C%20%D1%81%D0%BE%D0%B1%D1%8B%D1%82%D0%B8%D1%8F%20e,%D0%BC%D0%B8%D0%BD%D0%B8%D0%BC%D1%83%D0%BC%3A%20%D0%BF%D1%80%D0%BE%D1%81%D0%BC%D0%BE%D1%82%D1%80%D1%8B%2C%20%D0%B4%D0%BE%D0%B1%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F%2C%20%D1%87%D0%B5%D0%BA%D0%B0%D1%83%D1%82). В MVP, так как онлайн оплаты нет, событие покупки фиксируется при оформлении заказа (как "purchase" с пометкой offline). Эти данные помогут маркетингу отслеживать конверсию. - **Обработка платежей:** в MVP оплаты онлайн нет, клиент платит по счёту. Но на этапе добавления онлайн-оплаты (ЮKassa) важно соблюдать PCI DSS. Мы выберем интеграцию по минимальному скоупу (редирект или виджет ЮKassa), чтобы наш сайт не обрабатывал банковские данные, а лишь получал токены платежа. Соответственно, **в базе мы не храним данные карт**, а только ID транзакций и статус оплаты, полученные от ЮKassa[[33]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%A1%D1%82%D1%80%D0%B0%D0%BD%D0%B8%D1%86%D1%8B%3A%20%D0%9F%D0%BE%D0%BB%D0%B8%D1%82%D0%B8%D0%BA%D0%B0%2C%20%D0%9E%D1%84%D0%B5%D1%80%D1%82%D0%B0%3B%20%D0%BA%D1%83%D0%BA%D0%B8). Это позволяет соответствовать уровню SAQ A (когда сайт не имеет доступа к чувствительным данным карт). - **Конфиденциальность данных клиентов:** данные заказов (ФИО, адреса, телефоны) хранятся в БД, и доступ к ним ограничен админами. Будут предприняты меры защиты БД и резервных копий (шифрование диска, ограничение IP). При работе с персональными данными соблюдаем Закон о персональных данных (152-ФЗ в РФ): разместим Политику, обеспечим защиту данных, получим согласие (например, при оформлении заказа фразой "Нажимая оформить, вы даёте согласие на обработку персональных данных").

## UI/UX и компоненты интерфейса

Дизайн-макет сайта Marcus (PDF прототип) показывает современный, минималистичный интерфейс в фирменном стиле. При разработке фронтенда важно точно воспроизвести макет на всех трёх целевых разрешениях (desktop, tablet, mobile). Для этого будет выбран подходящий UI-фреймворк или библиотека компонентов.

**Выбор библиотеки компонентов:** После анализа дизайн-макета можно сделать вывод, что интерфейс довольно кастомный: фирменные цвета, уникальный стиль элементов (например, карточек товаров, фильтров). Использование готовой библиотеки (например, Chakra UI) может ускорить работу, но её стили из коробки отличаются от макета, и кастомизация может занять время. Альтернативный подход – использовать утилитарный CSS-фреймворк **Tailwind CSS** для быстрой верстки по макету, а сложные компоненты (модальные окна, выпадающие списки, карусели) реализовать с помощью библиотеки низкого уровня. Хорошим вариантом является **shadcn/ui** – набор готовых React-компонентов на базе Radix UI и Tailwind, который обеспечивает доступность и легко стилизуется под дизайн. Шаблоны shadcn/ui покрывают типовые элементы (форма, диалог, поповер, табы и т.д.) и их внешний вид можно настроить через CSS переменные и слои Tailwind. Ещё вариант – использовать коммерческие наборы готовых блоков **Tailwind UI** как основу: там есть примеры навигации, карточек товаров, списков, оформленных в современном стиле, близком к макету Marcus. Можно взять их за основу и адаптировать.

**Решение:** Будем использовать **Tailwind CSS** для стилизации и компонентный подход от **shadcn/ui** для сложных интерактивных элементов. Этот выбор обоснован тем, что: - Tailwind даст быструю адаптацию стилей под макет, контролируя каждую деталь (отступы, цвета) без необходимости переопределять стили готовой библиотеки. - Компоненты shadcn (Radix) обеспечат доступность (ARIA) и функциональность сложных элементов – выпадающих фильтров, модальных окон (например, для изображения товара или уведомления о добавлении в корзину). - Chakra UI, хотя и ускоряет разработку форм и базовых компонентов, имеет свой дизайн-систему, и подгонка под индивидуальный макет могла бы занять время. - Используя Tailwind, мы можем и самостоятельно верстать компоненты, а там, где нужно, подключать Headless UI или Radix (например, для выпадающего меню категорий, которое по макету явно присутствует).

**Адаптивность:** Макет предусматривает три основных разрешения (desktop, tablet, mobile). Мы определим брейкпоинты, например: mobile < 768px, tablet ~ 768-1440px, desktop > 1440px. Верстка будет относительно **mobile-first** – т.е. начинаем с простого линейного расположения элементов для телефона, и по мере увеличения экрана раскладываем блоки в две/три колонки, увеличиваем отступы, показываем дополнительные элементы (например, на desktop-версии в меню видны сразу пункты, а на мобильной – меню скрыто за "бургер"-иконкой).

Проверим, чтобы все интерактивные элементы были удобны на тач-устройствах: достаточно большие кнопки, выпадающие списки на мобиле занимают большую часть экрана (как модальные). Некоторые компоненты могут иметь специальные реализации: например, таблицу сравнения (если бы была) на мобиле показывать скроллом, но в нашем случае, в основном, список товаров и колонки превращаются в карточки 2-в ряд на таблет, 1-в-ряд на телефоне.

Ниже приведена **декомпозиция страниц и UI-компонентов** согласно макету:

### Главная страница (Home)

* **Шапка сайта (Header):** включает логотип Marcus, основное меню навигации и контакты. По макету меню содержит пункты "Компания", "Нанесение", "Помощь", "Портфолио", "Контакты" и отдельный раздел "Каталог" с выпадающим списком категорий[[34]](file://file-8njwtbzoHADixaYdrroLxW#:~:text=%D0%9A%D0%BE%D0%BC%D0%BF%D0%B0%D0%BD%D0%B8%D1%8F%20%D0%9F%D0%BE%D1%80%D1%82%D1%84%D0%BE%D0%BB%D0%B8%D0%BE%20%D0%9A%D0%BE%D0%BD%D1%82%D0%B0%D0%BA%D1%82%D1%8B%20%D0%9A%D0%B0%D1%82%D0%B0%D0%BB%D0%BE%D0%B3%20%D0%9E%D0%B4%D0%B5%D0%B6%D0%B4%D0%B0,%D0%9F%D0%BE%D1%81%D1%83%D0%B4%D0%B0%20%D0%95%D0%B6%D0%B5%D0%B4%D0%BD%D0%B5%D0%B2%D0%BD%D0%B8%D0%BA%D0%B8%20%D0%B8%20%D0%B1%D0%BB%D0%BE%D0%BA%D0%BD%D0%BE%D1%82%D1%8B%20%D0%9D%D0%B0%D0%BD%D0%B5%D1%81%D0%B5%D0%BD%D0%B8%D0%B5)[[35]](file://file-8njwtbzoHADixaYdrroLxW#:~:text=%D0%9F%D0%BE%D1%80%D1%82%D1%84%D0%BE%D0%BB%D0%B8%D0%BE%20%D0%9A%D0%BE%D0%BD%D1%82%D0%B0%D0%BA%D1%82%D1%8B%20%D0%9A%D0%B0%D1%82%D0%B0%D0%BB%D0%BE%D0%B3%20%D0%9E%D0%B4%D0%B5%D0%B6%D0%B4%D0%B0%20%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%BD%D1%8B%D0%B5,%D0%95%D0%B6%D0%B5%D0%B4%D0%BD%D0%B5%D0%B2%D0%BD%D0%B8%D0%BA%D0%B8%20%D0%B8%20%D0%B1%D0%BB%D0%BE%D0%BA%D0%BD%D0%BE%D1%82%D1%8B%20%D0%9D%D0%B0%D0%BD%D0%B5%D1%81%D0%B5%D0%BD%D0%B8%D0%B5%20%D0%A2%D0%B0%D0%BC%D0%BF%D0%BE%D0%BF%D0%B5%D1%87%D0%B0%D1%82%D1%8C). Реализуем header как фиксированный верхний блок, с адаптацией: на мобильном – иконка меню (бургер) для раскрытия бокового меню навигации, на десктопе – горизонтальное меню. Контактный телефон и email тоже вынесены в шапку, на мобильном они могут быть в меню или под логотипом. "Ответим в чате" – возможно кнопка для вызова онлайн-чата (может интеграция с внешним чат-сервисом).
* *UI:* будем использовать компонент **Navigation Menu** (в Radix UI) или собственную реализацию на основе Headless UI Menu для выпадающего списка категорий. Элемент меню "Каталог" при наведении/клике показывает мегаменю с колонками категорий (как в макете перечислены "Одежда", "Электронные товары", ...[[36]](file://file-8njwtbzoHADixaYdrroLxW#:~:text=%D0%9A%D0%B0%D1%82%D0%B0%D0%BB%D0%BE%D0%B3%20%D0%9E%D0%B4%D0%B5%D0%B6%D0%B4%D0%B0%20%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%BD%D1%8B%D0%B5%20%D1%82%D0%BE%D0%B2%D0%B0%D1%80%D1%8B%20%D0%9F%D0%BE%D1%81%D1%83%D0%B4%D0%B0,%D0%A2%D0%B0%D0%BC%D0%BF%D0%BE%D0%BF%D0%B5%D1%87%D0%B0%D1%82%D1%8C%20%D0%9B%D0%B0%D0%B7%D0%B5%D1%80%D0%BD%D0%B0%D1%8F%20%D0%B3%D1%80%D0%B0%D0%B2%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0%20%D0%A3%D0%A4%20%D0%BF%D0%B5%D1%87%D0%B0%D1%82%D1%8C)). Это сделаем через Radix <NavigationMenu> или <Popover> + Tailwind стили.
* *Search bar:* Текстовое поле поиска в шапку со значком лупы.
* **Основной баннер / промо-блок:** Главная страница скорее всего имеет крупный баннер с ключевым посланием. Например, слоган или акция. В PDF мы видим, например, текст "Кому подойдут наши сувениры? Бизнес-клиентам ..."[[37]](file://file-8njwtbzoHADixaYdrroLxW#:~:text=marcus%40marcus,%D0%BA%D0%BB%D0%B8%D0%B5%D0%BD%D1%82%D0%B0%D0%BC) – возможно блок с текстом и фоном. Реализуем как компонент **Hero Section** с фоновым изображением или цветом, заголовком, подписью и кнопкой (CTA).
* **Блок "Каталог" или категории:** Вероятно, на главной будут представлены основные разделы каталога (например, "Одежда с логотипом", "Электроника", "Посуда", ...). Это может быть сетка из карточек категорий с картинкой. Сделаем компонент **CategoryCard** и разместим 3-4 карточки в ряд (на desktop) с изображением категории и названием. На мобиле это будет слайдер или две в ряд.
* **Блоки преимуществ / информации:** Может быть секция "Почему Marcus" или "Наши преимущества". (В макете упоминается "Во что вы одеваете свой бренд?..."[[38]](file://file-8njwtbzoHADixaYdrroLxW#:~:text=%2B7%20%28800%29%2070,%D0%BD%D0%B0%20%D0%BD%D0%B5%D0%BC%20%D1%84%D1%83%D1%82%D0%B1%D0%BE%D0%BB%D0%BA%D0%B8%2C%20%D0%BF%D0%BE%D0%BB%D0%BE%2C%20%D1%82%D0%BE%D0%BB%D1%81%D1%82%D0%BE%D0%B2%D0%BA%D0%B8) – возможно это текстовый блок о значимости сувениров). Это оформим либо как текст с иконками, либо простой параграф с заголовком.
* **Портфолио превью:** Если есть раздел портфолио, на главной можно показать несколько примеров работ (например, 3-6 фотографий реализованных сувениров с логотипом). Сделаем слайдер или галерею.
* **Подвал (Footer):** содержит дублированное меню (см. PDF – там перечислены те же ссылки, контакты)[[39]](file://file-8njwtbzoHADixaYdrroLxW#:~:text=marcus%40marcus,%D0%9A%D0%BE%D0%BD%D1%82%D0%B0%D0%BA%D1%82%D1%8B%20%D0%9A%D0%B0%D1%82%D0%B0%D0%BB%D0%BE%D0%B3%20%D0%9E%D0%B4%D0%B5%D0%B6%D0%B4%D0%B0%20%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%BD%D1%8B%D0%B5%20%D1%82%D0%BE%D0%B2%D0%B0%D1%80%D1%8B)[[40]](file://file-8njwtbzoHADixaYdrroLxW#:~:text=match%20at%20L875%20%D0%9A%D0%B0%D1%82%D0%B0%D0%BB%D0%BE%D0%B3%20%2B7,%D0%9D%D0%B0%D0%BD%D0%B5%D1%81%D0%B5%D0%BD%D0%B8%D0%B5%20%D0%9F%D0%BE%D0%BC%D0%BE%D1%89%D1%8C%20%D0%9F%D0%BE%D1%80%D1%82%D1%84%D0%BE%D0%BB%D0%B8%D0%BE%20%D0%9A%D0%BE%D0%BD%D1%82%D0%B0%D0%BA%D1%82%D1%8B%20%D0%9A%D0%BE%D0%BC%D0%BF%D0%B0%D0%BD%D0%B8%D1%8F). Реализуем футер с колонками: колонка компании (о компании, портфолио, контакты), колонка "Каталог" (список основных категорий), колонка "Нанесение" (список методов печати), колонка "Помощь" (список страниц FAQ, доставка, гарантия), плюс контакты и копирайт. На мобильном футер складывается в аккордеон или просто в одну колонку.

### Страница каталога (список товаров в категории)

Это страница отображения списка товаров выбранной категории (PLP – Product Listing Page). Компоненты: - **Хлебные крошки (Breadcrumbs):** Путь навигации, например "Каталог / Одежда / Футболки". Позволяет вернуться вверх. Реализуем компонент Breadcrumbs (можно semantically <nav aria-label="breadcrumb">). Структура хлебных крошек строится по иерархии категорий. - **Заголовок категории и описание:** В макете видим пример "Одежда с логотипом" и ниже текст "Во что вы одеваете свой бренд?..."[[38]](file://file-8njwtbzoHADixaYdrroLxW#:~:text=%2B7%20%28800%29%2070,%D0%BD%D0%B0%20%D0%BD%D0%B5%D0%BC%20%D1%84%D1%83%D1%82%D0%B1%D0%BE%D0%BB%D0%BA%D0%B8%2C%20%D0%BF%D0%BE%D0%BB%D0%BE%2C%20%D1%82%D0%BE%D0%BB%D1%81%D1%82%D0%BE%D0%B2%D0%BA%D0%B8). Это значит у категории есть описание – его выводим под заголовком. Возможно, также показываем количество товаров в категории. - **Блок фильтров и сортировки:** На странице категории, слева – панель фильтров (на desktop). Фильтры включают атрибуты (цвет, объем и т.д.), наличие (в макете строка "Наличие") и, возможно, диапазон цен. Также сверху или над списком – выпадающий список сортировки (например: по возрастанию цены, по убыванию, по популярности)[[41]](file://file-8njwtbzoHADixaYdrroLxW#:~:text=marcus%40marcus,67%2068%20%D0%A1%D0%BB%D0%B5%D0%B4%D1%83%D1%8E%D1%89%D0%B0%D1%8F). - Реализация: на широком экране фильтры постоянно видны слева. На планшете/мобиле – кнопка "Фильтры", открывающая модальное окно с чекбоксами. Сортировка – выпадающий <select> или кастомный menu. - Каждая категория атрибутов (Цвет, Объём) будет компонентом **Accordion** (чтобы сворачивать список фильтров). Внутри – список чекбоксов с названием значения и количеством результатов (Meilisearch даст counts). Например: "Цвет: [ ] Красный (24) [ ] Синий (18) ..." – макет указывает "Цвет корпуса, Объём, мл"[[42]](file://file-8njwtbzoHADixaYdrroLxW#:~:text=%2B7%20%28800%29%2070,%D0%BA%D0%BE%D0%BD%D1%84%D0%B8%D0%B4%D0%B5%D0%BD%D1%86%D0%B8%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D1%81%D1%82%D0%B8%20%D0%A6%D0%B2%D0%B5%D1%82%20%D0%BA%D0%BE%D1%80%D0%BF%D1%83%D1%81%D0%B0%20%D0%9E%D0%B1%D1%8A%D1%91%D0%BC%2C%20%D0%BC%D0%BB) и где-то "Сбросить фильтры"[[41]](file://file-8njwtbzoHADixaYdrroLxW#:~:text=marcus%40marcus,67%2068%20%D0%A1%D0%BB%D0%B5%D0%B4%D1%83%D1%8E%D1%89%D0%B0%D1%8F). Последняя кнопка сбрасывает все фильтры. - UI libraries: можно использовать компонент Accordion из shadcn/ui (Radix Collapsible) для фильтров и Checkbox from Radix for each option, styled via Tailwind. - **Список товаров (превью карточки):** Отображаются карточки товаров (название, изображение, возможно цена). Если индексируются модели, каждая карточка представляет товарную модель, и, возможно, нужно указать, что “от N рублей” если есть варианты с разной ценой. Макет, вероятно, показывает сетку карточек 3 или 4 в ряд на desktop, 2 на tablet, 1 на mobile. Карточка включает картинку товара, название, возможно небольшой список характеристик или отметку "есть нанесение логотипа". Реализуем компонент **ProductCard**. - UI: можно воспользоваться готовым примером из Tailwind UI (раздел e-commerce) и адаптировать. Добавим быстрые действия: при наведении – кнопка “В корзину” или "Подробнее". В MVP, клик на карточку ведёт на страницу товара; возможно, по клику на "В корзину" сразу добавляет (но лучше внутри страницы товара после выбора варианта). - **Пагинация:** Если товаров много, снизу страницация: “Предыдущая 1 2 3 … 67 68 Следующая”[[41]](file://file-8njwtbzoHADixaYdrroLxW#:~:text=marcus%40marcus,67%2068%20%D0%A1%D0%BB%D0%B5%D0%B4%D1%83%D1%8E%D1%89%D0%B0%D1%8F). Реализуем пагинацию на фронте (Next.js может сгенерировать пагинацию как разные статические страницы или сделать одну страницу категории, где на клиенте подгружает следующие страницы). Лучше SSR разделить категории по страницам для SEO (каждая страница пагинации тоже индексируется, но можно указать rel="prev/next"). Сделаем компонент **Pagination**. UI: список ссылок, на мобильном – кнопки вперед/назад без номеров или с прокруткой номеров.

### Страница товара (PDP – Product Detail Page)

Страница детальной информации о товаре или конкретной вариации: - **Галерея изображений товара:** Большое основное изображение и превью других ракурсов. Возможно, зум при наведении. На мобильном – свайпер. Реализуем компонент **ImageGallery**. Можно использовать библиотеку (например, KeenSlider для галереи) или легковесно через <div overflow-x scroll> на мобиле. Shadcn/ui пока не имеет готового слайдера, поэтому скорее свой код или tiny-slider lib. - **Основная информация:** Название товара, артикул (SKU), цена. Если есть выбор варианта – выпадающие списки или переключатели (например, цвет – кружочки цветов, размер – кнопки). Реализуем компоненты **VariantSelector**. Можно использовать Radix Select или Toggle Group для вариаций. - **Характеристики:** Список атрибутов товара (например, "Материал: хлопок, 100%", "Производитель: ..."). Будет блок "Характеристики" как таб или просто секция под описанием[[43]](file://file-8njwtbzoHADixaYdrroLxW#:~:text=%2B7%20%28800%29%2070,%D0%9F%D0%BE%D0%BB%D0%B8%D1%82%D0%B8%D0%BA%D0%B0%20%D0%BA%D0%BE%D0%BD%D1%84%D0%B8%D0%B4%D0%B5%D0%BD%D1%86%D0%B8%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D1%81%D1%82%D0%B8%20%D0%A5%D0%B0%D1%80%D0%B0%D0%BA%D1%82%D0%B5%D1%80%D0%B8%D1%81%D1%82%D0%B8%D0%BA%D0%B8%20%D0%A6%D0%B2%D0%B5%D1%82%20%D0%BA%D0%BE%D1%80%D0%BF%D1%83%D1%81%D0%B0). - **Описание товара:** Текстовое описание, возможно параграфы о товаре (например, преимущества, как наносится логотип). - **Блок нанесения логотипа:** Если hasLogoPrinting=true, показываем инфоблок: какие методы доступны (напр. "На этот товар можно нанести логотип методом тампопечати или УФ-печати"). Может быть ссылка "Подробнее о нанесении". Этот блок важен для Marcus (сувенирка с логотипом). Мы выведем список logoMethods (связанные PrintingMethod) – в виде иконок или текста. - **Выбор количества и кнопка в корзину:** Поле ввода количества (с кнопками +/-) и кнопка **"Добавить в корзину"**. При нажатии товар (выбранная вариация) добавляется в состояние корзины. Пользователю может показываться всплывающее окно или выпадашка "Товар добавлен в корзину [Перейти в корзину]". - Реализация: компонент **QuantityInput** и кнопка, нажатие которой вызывает, например, Radix Dialog с подтверждением. Либо просто надпись появляется. - **Рекомендуемые / Похожие товары:** Внизу страницы можно показать блок "Похожие товары" или "С этим покупают". Это 3-4 карточки (тот же компонент ProductCard). Можно запросить их через Strapi (например, связанные той же категории). - **SEO элементы:** На странице настроим <title> как "Купить {Название товара} с логотипом – Marcus" и meta description. Добавим микроразметку JSON-LD Product + Offer с ценой и наличием[[44]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=Sitemap%20,%D0%B8%D0%BC%D0%BF%D0%BE%D1%80%D1%82%D0%B5).

### Корзина

Страница корзины показывает все добавленные товары перед оформлением: - **Список позиций:** Таблица или список товаров: каждое позиция – название (с ссылкой на товар), выбранные атрибуты (цвет, размер), цена за единицу, количество (с возможностью изменить), итог по позиции. Иконка удаления позиции. - **Итоговая стоимость:** Под списком – суммирование, указание общей стоимости, НДС (если нужен) или "без НДС". - **Кнопки:** "Продолжить покупки" (ссылка назад в каталог) и основной CTA "Оформить заказ". - UI: Можно взять пример корзины из Tailwind UI. Сделаем так, чтобы на мобильном элементы были вертикально, на desktop – таблица. Из библиотек можем применить Radix Dialog если корзина будет всплывающей (но здесь отдельная страница, так что не нужно).

### Оформление заказа (Checkout)

В MVP оформление заказа – это форма, где пользователь вводит свои данные и информацию для доставки, а затем подтверждает заказ (без онлайн оплаты): - **Форма контактных данных:** Поля: имя, компания (если B2B), телефон, email. Возможно, сразу адрес доставки (город, улица, индекс) если нужна доставка курьером. - **Выбор способа получения:** Так как в MVP нет интеграции, можно просто радиокнопки: самовывоз или доставка транспортной компанией. Если доставка – выпадает поле адреса/города. В будущем (этап 2) здесь появится виджет выбора пункта выдачи (для СДЭК и др.), но на MVP просто текст "мы свяжемся для уточнения доставки". - **Оплата:** В MVP онлайн-оплата не используется, поэтому отображаем текст "Менеджер свяжется и вышлет счет на оплату". Можно все же предложить выбрать "Безналичная оплата" или "Оплата при получении" для информации. На этапе 2 вместо этого – интеграция с ЮKassa: либо сразу iframe оплаты, либо кнопка "Оплатить" перекидывающая на платёжную страницу. - **Комментарий к заказу:** Текстовое поле, опционально. - **Подтверждение:** Кнопка "Отправить заказ". Перед ней – чекбокс согласия с офертой. Нажатие создаёт Order через API (POST запрос на Strapi endpoint или через Strapi SDK). - **После отправки:** Показывается страница/сообщение "Спасибо за заказ! Ваш номер заказа #XYZ. Менеджер скоро свяжется." В бэкенде Order со статусом "Новый". - **Email уведомление:** Trigger: Strapi (или Node) отправляет письмо на email клиента с подтверждением заказа, перечислением позиций и инструкциями (например, реквизиты для оплаты)[[45]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%9E%D1%84%D0%BE%D1%80%D0%BC%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5%20%D0%B1%D0%B5%D0%B7%20%D0%BE%D0%BD%D0%BB%D0%B0%D0%B9%D0%BD,%D0%9E%D0%B6%D0%B8%D0%B4%D0%B0%D0%B5%D1%82%20%D0%BF%D0%BE%D0%B4%D1%82%D0%B2%D0%B5%D1%80%D0%B6%D0%B4%D0%B5%D0%BD%D0%B8%D1%8F%20%2F%20%D0%9A%20%D0%BE%D1%82%D0%B3%D1%80%D1%83%D0%B7%D0%BA%D0%B5%C2%BB). Это письмо настроим через SMTP (например, Mailgun API или локальный почтовик с SPF/DKIM настройками для домена)[[46]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%BF%D0%BE%D0%B4%D1%82%D0%B2%D0%B5%D1%80%D0%B6%D0%B4%D0%B5%D0%BD%D0%B8%D1%8F%20%2F%20%D0%9A%20%D0%BE%D1%82%D0%B3%D1%80%D1%83%D0%B7%D0%BA%D0%B5%C2%BB).

В плане UI, форма оформляется просто и понятно, можно использовать готовые компоненты форм (из shadcn/ui есть Form, Input, Radio, etc.). Валидация – на клиенте (простая, через HTML required, + маска телефона) и на сервере дубль.

### Личный кабинет (этап 2)

При расширении функционала будет добавлен личный кабинет для постоянных клиентов: - **Регистрация/Авторизация:** Скорее всего B2B-клиенты могут иметь аккаунт для просмотра истории заказов. Реализуем на Strapi плагине Users & Permissions: клиенты регистрируются (или менеджеры создают им логин). Вход по email+паролю. Можно также OAuth, но лишнее. - **Страница профиля:** изменение контактной информации, пароля. - **История заказов:** список прошлых заказов пользователя (из таблицы Order, фильтр по user). Отображается номер, дата, сумма, статус. Можно кликнуть увидеть детали заказа. - **Повтор заказа:** Nice-to-have – кнопка "повторить заказ" создаёт корзину из этого заказа. - **Сохранённые адреса:** если нужно, или платёжные реквизиты (для компаний). - *UI:* Кабинет будет доступен после входа, для неавторизованных – редирект на страницу входа. Верстка кабинета обычно табами или меню сбоку. Можно применить Radix Tabs для переключения "Профиль | Мои заказы | ..." или отдельные страницы.

### Статические информационные страницы

Раздел "Компания": страница "О компании" (история, миссия), "Контакты" (адрес, карта, формы обратной связи), раздел "Помощь" включает несколько страниц: - **Как купить?** – инструкция по заказу. - **Доставка и самовывоз** – условия доставки. - **Оплата и гарантии** – способы оплаты, гарантийные обязательства. - **Поддержка** – возможно контакты техподдержки или менеджера. - **Частые вопросы (FAQ)** – список вопросов-ответов.

Каждая такая страница редактируется через Strapi. Структурно они могут иметь заголовок и контент (текст, списки, изображения). Верстаем их как простой шаблон: заголовок страницы + контент, выровненный по центру, шириной, например, 800px для удобства чтения. Контент в Strapi можно хранить как Markdown или Rich Text; для отображения возможно потребуется парсер (если Markdown) либо dangerouslySetHTML (если готовый HTML). В любом случае, применим стили для контентной области (например, ограничим размеры заголовков, таблиц, списков внутри).

FAQ можно сделать чуть интерактивным: список вопросов как аккордеон (в макете пункт FAQ есть, возможно там аккордеон). Но для редактируемости, проще как статический текст с вопросами и ответами.

### Портфолио

Раздел "Портфолио" предназначен для демонстрации кейсов – выполненных корпоративных сувениров для клиентов. Здесь, вероятно, галерея изображений с описаниями: - **Список кейсов:** например, плитка из карточек, каждая – фото готовой продукции с логотипом клиента, название проекта или клиента, краткое описание. В макете упоминается "Портфолио" в меню, а также, возможно, "Кому подойдут наши сувениры? Бизнес-клиентам" – возможно, имеется в виду, что портфолио организовано по типам клиентов. - Если детальной страницы кейса нет, то портфолио может быть просто галерея. Но можно сделать и страницы проектов: тогда при клике – страница с несколькими фото, описанием, отзывом клиента. - **UI:** Сделаем компонент **Gallery**. Можно использовать CSS grid или Masonry. Если нужна фильтрация по отрасли или типу сувенира, добавим фильтр. Но MVP может иметь статичный список.

Страницу портфолио тоже можно редактировать через Strapi: либо content type "Case" (поля: название, описание, изображения, категория клиента) и вывести все. Либо захардкодить. Предпочтем сделать content type "Portfolio Item" для гибкости.

Все перечисленные страницы будут связаны общей стилистикой и компонентами. Мы составили библиотеку повторно используемых **UI-компонентов**: - Buttons (кнопки разных видов: основные CTA, вторичные, иконки-кнопки). - Forms (input, checkbox, radio, select, styled по макету). - Modal/Dialog (например, для корзины на мобильном или для подтверждений). - Dropdown Menu (например, юзерпик с меню выхода в ЛК, либо меню сортировки). - Tooltip (подсказки, если нужны). - Card (базовый компонент карточки контента, расширяем для товара, для категории, для портфолио). - Badge/Label (напр. бейдж "New" на товаре). - Loader/Spinner (при ожидании). - Pagination controls (как упомянуто). - Accordion (для FAQ и фильтров). - Tab (если в ЛК или характеристиках/отзывах товара нужно табы). - Toast/Notification (вдруг нужен, например всплывашки "товар добавлен").

Отдельно учтем **совместимость и тестирование UI**: будем проверять в современных браузерах Chrome, Firefox, Safari, а также на мобильных (Chrome Android, Safari iOS). Верстка должна быть кроссбраузерной и доступной: проверим с клавиатуры (focus states), для экранных читалок (aria-labels на иконках, alt на изображениях товаров). Компоненты от Radix UI в этом помогут, т.к. уже имеют aria-атрибуты.

## План реализации (Roadmap проекта)

Проект разбит на этапы, охватывающие как создание MVP, так и развитие на втором этапе (рост). Ниже представлен поэтапный план с указанием задач, ключевых артефактов и критериев приёмки для каждого этапа, а также как в процессе будет применяться Codex (AI-помощник) для ускорения разработки:

**Этап 0 – Предпроектная подготовка и финализация ТЗ**  
*Задачи:* Анализ требований заказчика, уточнение деталей. Финализация технического задания: фиксируются все фасетные атрибуты, структура URL (человекочитаемые пути для категорий, товаров), макеты страниц, схемы импорта и обновления данных, критерии приёмки по каждой фиче[[47]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%AD%D1%82%D0%B0%D0%BF%200%20%E2%80%94%20%D0%9F%D1%80%D0%B5%D0%B4%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%20%D0%B8,%D1%84%D0%B8%D0%BD%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F%20%D0%A2%D0%97).  
*Артефакты:* Итоговое ТЗ (документ) утверждено обеими сторонами; mindmap или схема архитектуры; план испытаний (черновик чек-листов).  
*Приёмка:* Заказчик подтверждает, что ТЗ полно описывает функционал MVP и этапа роста. Можно переходить к реализации.  
*Роль Codex:* На этом этапе Codex может генерировать части документации или помочь составить mindmap функциональности на основе требований, проверяя, не упущены ли какие-то детали.

**Этап 1 – Архитектура и DevOps (VDS)**  
*Описание:* Развёртывание и базовая настройка инфраструктуры. Провиженинг VPS: установка ОС, Docker, настройка Nginx, TLS-сертификатов, базового firewall. Подготовка репозиториев и CI/CD. Докеризация приложения.  
*Работы:* Настроить Docker Compose с сервисами (Postgres, Strapi, Next, Meili, Redis). Настроить Nginx конфиг для прокси и SSL. Настроить GitHub Actions для автоматической сборки и деплоя на сервер (стейджинг и прод)[[48]](file://file-1yjcu1K12osyUYjxJV5SHb#:~:text=CI%2FCD%20%D0%B4%D0%BB%D1%8F%20%D1%84%D1%80%D0%BE%D0%BD%D1%82%D0%B0%20%D0%B8%20%D0%B1%D1%8D%D0%BA%D0%B0,Terraform%2FIaC%20%E2%80%94%20%D0%BA%D0%B0%D0%BA%20%D0%B1%D0%B5%D0%BD%D1%87%D0%BC%D0%B0%D1%80%D0%BA%20%D0%BF%D1%80%D0%B0%D0%BA%D1%82%D0%B8%D0%BA%D0%B8). Проверить автопродление сертификатов Let's Encrypt[[25]](file://file-1yjcu1K12osyUYjxJV5SHb#:~:text=%D0%92%D1%8B%D0%BF%D1%83%D1%89%D0%B5%D0%BD%D1%8B%20%D0%B8%20%D0%B0%D0%B2%D1%82%D0%BE%D0%BF%D1%80%D0%BE%D0%B4%D0%BB%D0%B5%D0%B2%D0%B0%D1%8E%D1%82%D1%81%D1%8F%20HTTPS,Encrypt%2FCertbot).  
*Артефакты:* VDS с поднятым стеком (пока может быть заглушечный Hello World сервис). Файлы: docker-compose.yml, Dockerfile(и), nginx.conf, CI/CD workflow файлы. Документация: описание инфраструктуры (что где расположено), **playbook деплоя** – шаги как выкатывать обновление и как бэкапить/восстанавливать при сбое[[49]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%98%D0%BD%D1%84%D1%80%D0%B0%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0%20%D0%B8%20DevOps).  
*Приёмка:* Доступность тестового сайта по HTTPS (например, dev.domain) с корректным сертификатом. Проверка, что при push происходит автоматический деплой на staging. Nginx отдает статическую страницу без ошибок.  
*Роль Codex:* Codex AI используется для генерации шаблонов конфигов – например, сгенерировать типовой docker-compose или github actions yaml, которые затем дорабатываются вручную. Это экономит время на написании boilerplate.

**Этап 2 – Backend (Strapi CMS, моделирование схем)**  
*Описание:* Установка и настройка Strapi, моделирование схем данных (Content Types) согласно разработанной модели, настройка прав доступа и API.  
*Работы:* Развернуть Strapi (в Docker). Создать Content Types: Category, Product, ProductVariant, Attribute, PrintingMethod, Order, Page, PortfolioItem и т.д. настроить поля, отношения. Подключить плагин загрузки файлов к S3. Настроить роли и права: анонимному доступу дать права GET на необходимые коллекции (товары, категории) для фронтенда[[50]](file://file-1yjcu1K12osyUYjxJV5SHb#:~:text=%D0%A1%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D1%8B%20%D1%82%D0%B8%D0%BF%D1%8B%20%D0%BA%D0%BE%D0%BD%D1%82%D0%B5%D0%BD%D1%82%D0%B0%20%28%D0%BA%D0%B0%D1%82%D0%B5%D0%B3%D0%BE%D1%80%D0%B8%D0%B8%2F%D1%82%D0%BE%D0%B2%D0%B0%D1%80%D1%8B%2F%D0%B0%D1%82%D1%80%D0%B8%D0%B1%D1%83%D1%82%D1%8B%2F%D0%BC%D0%B5%D0%B4%D0%B8%D0%B0%29). Создать админ-аккаунты. Настроить CORS, если нужно, чтобы фронт мог обращаться. Если используем GraphQL плагин – установить и настроить схемы.  
*Артефакты:* Strapi CMS доступен на /admin. Документация по API (можно включить Swagger или сгенерировать Postman коллекцию)[[51]](file://file-1yjcu1K12osyUYjxJV5SHb#:~:text=%D0%A7%D0%B5%D1%80%D0%BD%D0%BE%D0%B2%D0%B0%D1%8F%20%D0%B4%D0%BE%D0%BA%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D0%B0%D1%86%D0%B8%D1%8F%20%D0%BF%D0%BE%20%D1%81%D1%85%D0%B5%D0%BC%D0%B0%D0%BC%20%D0%B8,%D1%80%D0%B5%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B5%20%D0%BE%D1%82%D0%B2%D0%B5%D1%82%D1%8B%20API%20%D0%B8%D0%B7%20staging). Экспорт или описание схем контента. Первоначальные справочники (пустые или заполненные тестовыми данными).  
*Приёмка:* Через REST API можно получить список категорий, список товаров (пока пусто или с парой тестовых записей) – демонстрируется в Postman или Swagger UI[[52]](file://file-1yjcu1K12osyUYjxJV5SHb#:~:text=%D0%94%D0%B5%D0%BC%D0%BE%D0%BD%D1%81%D1%82%D1%80%D0%B0%D1%86%D0%B8%D1%8F%2F%D0%BF%D1%80%D0%B8%D1%91%D0%BC%D0%BA%D0%B0%3A%20Postman,%D1%80%D0%B5%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B5%20%D0%BE%D1%82%D0%B2%D0%B5%D1%82%D1%8B%20API%20%D0%B8%D0%B7%20staging). Права работают (неавторизованный пользователь не может POST/PUT). Админка открывается (с ограничением доступа) и позволяет редактировать записи.  
*Роль Codex:* Помогает сгенерировать модели контента (например, на языке схем Strapi – JSON или Strapi code). Может быстро написать скрипт для создания большого количества фейковых данных, что полезно для теста производительности.

**Этап 3 – Импорт данных Oasis + Медиа + ISR/реиндексация**  
*Описание:* Разработка скрипта импорта товаров из внешнего API, загрузка медиа, интеграция с ISR и поиском.  
*Работы:* Написать модуль интеграции с Oasis API: получить категории и товары батчами, создать/обновить их в Strapi. Обработать изображения – скачать, загрузить в S3 (через Strapi endpoints или напрямую). Обработать ошибки и ретраи. Настроить cron (например, Strapi Cron или системный cron job, или PM2 cron) на запуск каждую ночь. Реализовать триггеры: по окончании импорта дергать Next.js revalidate API и обновлять Meilisearch индекс. Если Strapi поддерживает lifecycle hooks, можно привязаться к созданию/обновлению product – но лучше явно вызывать после массового импорта.  
*Артефакты:* Скрипт import\_oasis.js (или услуга Strapi) со всеми функциями. Логи импорта (в консоли или файл) с информацией об успешных операциях и ошибках[[53]](file://file-1yjcu1K12osyUYjxJV5SHb#:~:text=%D0%A1%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D1%8B%20%D0%B8%D0%BC%D0%BF%D0%BE%D1%80%D1%82%D0%B0%20%D0%BA%D0%B0%D1%82%D0%B0%D0%BB%D0%BE%D0%B3%D0%B0%2F%D0%BC%D0%B5%D0%B4%D0%B8%D0%B0%2C%20%D0%BB%D0%BE%D0%B3%20%D0%B8%D0%BC%D0%BF%D0%BE%D1%80%D1%82%D0%B0,%D1%81%20%D0%BE%D1%82%D1%87%D1%91%D1%82%D0%BE%D0%BC%20%D0%BE%D0%B1%20%D0%BE%D1%88%D0%B8%D0%B1%D0%BA%D0%B0%D1%85). Конфигурация cron. Документ со схемой процесса импорта (например, flowchart).  
*Приёмка:* Демонстрация: запускаем импорт на staging – новые товары появляются в Strapi и сразу становятся доступны на фронте. Проверяется, что после импорта обновился индекс поиска (можно сразу найти новый товар через поиск) и при открытии страницы нового товара Next.js сгенерировал страницу (либо при первом запросе)[[54]](file://file-1yjcu1K12osyUYjxJV5SHb#:~:text=%D0%9D%D0%B0%D1%81%D1%82%D1%80%D0%BE%D0%B5%D0%BD%D0%B0%20%D1%80%D0%B5%D0%B3%D0%B5%D0%BD%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D1%8F%20%D1%81%D1%82%D1%80%D0%B0%D0%BD%D0%B8%D1%86%20%D0%B1%D0%B5%D0%B7%20%D0%BF%D0%BE%D0%BB%D0%BD%D0%BE%D0%B3%D0%BE,%D0%BE%D0%B1%D0%BD%D0%BE%D0%B2%D0%B8%D0%BB%D0%B0%D1%81%D1%8C%20%D0%B1%D0%B5%D0%B7%20%D0%BF%D0%BE%D0%BB%D0%BD%D0%BE%D0%B3%D0%BE%20%C2%ABnext%20build%C2%BB). Изменение существующего товара (например, цена) и вызов revalidate – обновляет страницу товара без полного деплоя.  
*Роль Codex:* Используется для ускорения написания парсера: может помочь сгенерировать код обращения к REST API Oasis, разбора JSON, и даже с написанием SQL запросов, если бы они понадобились. Codex также может предложить структуру логирования или готовую реализацию ретраев.

**Этап 4 – Поиск (Meilisearch) интеграция**  
*Описание:* Развёртывание и настройка Meilisearch, индексирование товаров, настройка фасетного поиска.  
*Работы:* Поднять Meilisearch (контейнер). Написать скрипт, который берет товары из Strapi и загружает их в Meili-индекс (initial indexing). Настроить поля для поиска, задать фасетные атрибуты (как обсуждалось). Установить параметры typoTolerance (1 и 2)[[19]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%A4%D0%B0%D1%81%D0%B5%D1%82%D1%8B%20%D0%BF%D0%BE%20%D0%B0%D1%82%D1%80%D0%B8%D0%B1%D1%83%D1%82%D0%B0%D0%BC%3B%20%D1%81%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B8%20%28%D0%BF%D0%BE%D0%BF%D1%83%D0%BB%D1%8F%D1%80%D0%BD%D0%BE%D1%81%D1%82%D1%8C%2F%D1%86%D0%B5%D0%BD%D0%B0%2F%D0%BD%D0%BE%D0%B2%D0%B8%D0%B7%D0%BD%D0%B0%29), включить поддержу синонимов (пока список пустой). Реализовать обновление индекса при изменении отдельных товаров (например, Strapi webhook on update -> call Meili to update doc). На фронтенде: создать API-методы или использовать Meili JS-клиент для запросов поиска. *Артефакты:* Конфигурационный файл или команды Meilisearch (если нужны для relevancy). Сниппеты кода для поиска на фронте. Документ с примерами запросов (например, пример REST запросов к Meili и ответов).  
*Приёмка:* Демо: выполняются примерные поисковые запросы. Например, ищем с опечаткой – результат находится (показать что "тасовка" находит "тусовка" если бы был такой товар). Фасетные фильтры: сделать запрос через API с фильтром цвет=красный – убедиться, что в ответе только товары с красным, и Meili вернул counts по другим фильтрам[[55]](file://file-1yjcu1K12osyUYjxJV5SHb#:~:text=%D0%92%D0%BA%D0%BB%D1%8E%D1%87%D0%B5%D0%BD%D1%8B%20facet,search%3B%20%D0%B1%D0%B5%D0%BD%D1%87%D0%BC%D0%B0%D1%80%D0%BA%20%D0%BD%D0%B5%D1%81%D0%BA%D0%BE%D0%BB%D1%8C%D0%BA%D0%B8%D1%85%20%D0%B7%D0%B0%D0%BF%D1%80%D0%BE%D1%81%D0%BE%D0%B2). Проверить сортировку: например, получить результаты по возрастанию цены и убедиться в порядке. Все это демонстрируется через небольшую утилиту (Postman или встроенный UI).  
*Роль Codex:* AI может помочь с формированием запросов Meilisearch, написанием функций сортировки, а также сгенерировать тестовые сценарии: "попробуй такие-то комбинации фильтров". Это ускорит отладку релевантности.

**Этап 5 – Frontend (Next.js) разработка**  
*(Примечание: этап крупный, разбит на подпункты согласно макету и компонентам)*[*[22]*](file://file-1yjcu1K12osyUYjxJV5SHb#:~:text=%D0%AD%D1%82%D0%B0%D0%BF%205%20%E2%80%94%20Frontend%20%28Next,7%20%D0%9F%D0%BE%D1%80%D1%82%D1%84%D0%BE%D0%BB%D0%B8%D0%BE)*.*

* **5.1 Каркас и SEO:** Настроить routing в Next.js, общий Layout. Подключить Tailwind CSS. Настроить <Head> компоненты: теги <meta>, OpenGraph, favicon. Реализовать общие части: Header, Footer, чтобы они отображались на всех страницах. Подключить шрифты, если в дизайне используются кастомные. Настроить переходы (например, Link из next/link для SPA-навигации). *Приёмка:* На черновом фронте есть работающие ссылки между страницами (пустые страницы заглушки), корректно отображается header/footer на всех, метатеги уникальны (можно заглушить "Страница - Marcus").
* **5.2 Главная страница:** Сверстать главную по макету. Реализовать компоненты: баннер/hero, секции категорий, текста, портфолио превью. Данные пока можно захардкодить или получить из Strapi (например, подтянуть 4 категории с флагом "showOnMain"). *Приёмка:* Главная выглядит как в дизайне на всех разрешениях, проверена адаптивность. Все ссылки/кнопки ведут куда надо (пока если страниц нет – на заглушки).
* **5.3 Страница категории (PLP) с фильтрацией:** Реализовать получение данных категории (SSR getStaticProps: получаем категорию по slug, список товаров из Meilisearch по категории). Сверстать фильтры: слева панель, сверху сортировка. Подключить реальный поиск: при взаимодействии с фильтрами – либо через навигацию (query params) перезагружать страницу, либо AJAX. Вероятно, сделаем на React: при выборе фильтра вызываем API /api/search (Next.js API route) который обращается к Meilisearch и возвращает JSON с товарами и фасетами, обновляем состояние. Т.е. страница будет гидратирована SSR результатом, далее работает как SPA для фильтров. Сверстать список карточек товаров (с реальными данными). *Приёмка:* На staging можно зайти в категорию, увидеть товары, применить фильтр (например, цвет) – список обновляется без перезагрузки всей страницы, выводятся правильные товары. Проверить пагинацию: переход на стр.2 работает. SEO: на странице категории <title> и h1 совпадают с названием категории, присутствует канонический URL (без параметров фильтра)[[23]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%98%D0%BD%D0%B4%D0%B5%D0%BA%D1%81%D0%B0%D1%86%D0%B8%D1%8F%3A%20%D0%BA%D0%B0%D1%82%D0%B5%D0%B3%D0%BE%D1%80%D0%B8%D0%B8%20%D0%B8%20%D1%82%D0%BE%D0%B2%D0%B0%D1%80%D1%8B%3B%20%D0%BA%D0%BE%D0%BC%D0%B1%D0%B8%D0%BD%D0%B0%D1%86%D0%B8%D0%B8,%D0%BD%D0%B0%20%D0%B1%D0%B0%D0%B7%D0%BE%D0%B2%D1%83%D1%8E%20%D0%BA%D0%B0%D1%82%D0%B5%D0%B3%D0%BE%D1%80%D0%B8%D1%8E).
* **5.4 Страница товара (PDP):** Сверстать страницу товара, выводить данные реального продукта (название, описание из Strapi). Сделать галерею изображений, переключение вариаций (если вариации имеют отдельные страницы, можно и переключать через ссылки, но удобнее на одной странице – смена варианта меняет URL или просто данные цены). Добавить кнопку "в корзину" (пока без функционала, либо складывать в context). *Приёмка:* Открывается страница товара через каталог, отображает всю информацию, изображения листаются, характеристики видны. Если товар с логотипом – виден блок методов печати.
* **5.5 Поиск, подсказки, 404:** Реализовать страницу поиска /search – можно на базе компонента категории, но с выводом результатов по запросу. Сделать чтобы при пустом результате отображалось сообщение "Ничего не найдено" (empty state). Страницу 404 оформить по макету (что-то дружелюбное). Автодополнение поиска: скрипт, вызывающий /api/search\_suggest?q= для подсказок – вернём несколько топ-результатов, отобразим выпадающим списком под строкой поиска. *Приёмка:* Введя в поиске несколько букв, появляются подсказки, при нажатии переходит на нужный товар. Страница 404 показывается при вводе несуществующего URL, выглядит стильно.
* **5.6 Статические страницы (Оферта, Политика, Контакты):** Получить контент из Strapi (можно SSR на each request или SSG). Сверстать по простому шаблону. Контакты: форма обратной связи (можно пока без бекенда, либо через Strapi email отправку). *Приёмка:* Страницы доступны через меню "Помощь/Контакты", отображают контент, при правке текста в Strapi – через ISR обновляются. Форма контактов отправляет письмо на почту админа (можно протестировать, если настроено SMTP).
* **5.7 Портфолио:** Вывести список портфолио-элементов (можно красивой сеткой). Если предусмотрены фильтры (например, по отрасли) – реализовать. *Приёмка:* Страница портфолио открывается, картинки загружаются быстро (оптимизированы), при клике на элемент – либо лайтбокс, либо переход (в зависимости от реализации).
* **5.8 Оптимизация изображений и производительности:** Проверить, что везде стоит <Image> с нужными sizes атрибутами. Настроить заглавные изображения через <link rel="preload"> для важного контента. Убедиться, что лишние бандлы не грузятся. *Приёмка:* Линтинг Lighthouse: показатели performance близки к зелёным, изображения в next-devtools не ругаются на отсутствие width/height.

*Роль Codex:* В ходе этапа фронтенда AI-помощник используется активно: генерация типового кода компонентов, помощь с Tailwind классами, отладка сложных взаимодействий. Например, Codex может сгенерировать алгоритм пагинации, предложить решение для автокомплита, помочь написать тесты для React-компонентов. Это ускоряет работу на каждом подпункте.

**Этап 6 – Корзина и оформление заказа (MVP, офлайн оплата)**  
*Описание:* Добавление функционала электронной торговли: корзина, оформление заказа, уведомления, интеграция с аналитикой.  
*Работы:* Реализовать состояние корзины на фронте: при нажатии "в корзину" товар (id вариации, кол-во) сохраняется (в React state или локальном хранилище). Страница корзины отображает это состояние, позволяет редактировать. Оформление заказа: создать форму, валидировать, отправить данные через вызов Strapi API – для этого может быть создан кастомный endpoint (например, REST POST /orders) или напрямую писать в PostgreSQL (лучше через Strapi). Статусы заказа: при создании присваивается "Новый". Настроить отправку email: используя либо Strapi Email plugin (SMTP) – шаблон письма с данными заказа и инструкцией по оплате[[45]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%9E%D1%84%D0%BE%D1%80%D0%BC%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5%20%D0%B1%D0%B5%D0%B7%20%D0%BE%D0%BD%D0%BB%D0%B0%D0%B9%D0%BD,%D0%9E%D0%B6%D0%B8%D0%B4%D0%B0%D0%B5%D1%82%20%D0%BF%D0%BE%D0%B4%D1%82%D0%B2%D0%B5%D1%80%D0%B6%D0%B4%D0%B5%D0%BD%D0%B8%D1%8F%20%2F%20%D0%9A%20%D0%BE%D1%82%D0%B3%D1%80%D1%83%D0%B7%D0%BA%D0%B5%C2%BB). Email менеджеру о новом заказе тоже можно отправлять. Добавить на фронте Google Analytics события (GA4): отправлять add\_to\_cart, begin\_checkout, purchase (покупку регистрируем сразу после оформления, т.к. оплаты нет, но это "offline purchase stub")[[56]](file://file-1yjcu1K12osyUYjxJV5SHb#:~:text=%D0%B7%D0%B0%D0%BA%D0%B0%D0%B7%D0%B0%20%D0%B2%20%D0%B1%D0%B5%D0%BA%D0%B5%D0%BD%D0%B4%D0%B5%2C%20%D1%81%D0%B5%D1%80%D0%B2%D0%B8%D1%81%D0%BD%D1%8B%D0%B5%20%D0%BF%D0%B8%D1%81%D1%8C%D0%BC%D0%B0)[[32]](file://file-1yjcu1K12osyUYjxJV5SHb#:~:text=GA4%2FTag%20Manager%20%D0%BD%D0%B0%D1%81%D1%82%D1%80%D0%BE%D0%B5%D0%BD%D1%8B%2C%20%D1%81%D0%BE%D0%B1%D1%8B%D1%82%D0%B8%D1%8F%20e,%D0%BC%D0%B8%D0%BD%D0%B8%D0%BC%D1%83%D0%BC%3A%20%D0%BF%D1%80%D0%BE%D1%81%D0%BC%D0%BE%D1%82%D1%80%D1%8B%2C%20%D0%B4%D0%BE%D0%B1%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F%2C%20%D1%87%D0%B5%D0%BA%D0%B0%D1%83%D1%82).  
*Артефакты:* Код фронта для корзины/чекаута, Strapi endpoint (например, создаём Order через Strapi controllers). Шаблон письма (в Strapi можно через Email templates, с переменными заказа). Документация для менеджеров – как обрабатывать заказ (например, в админке Strapi перейти в Orders, изменить статус по мере обработки).  
*Приёмка:* Тестовый сценарий: пользователь добавляет товары в корзину, переходит к оформлению, заполняет форму и жмёт "Оформить". В админке Strapi появляется новый Order с правильными данными[[57]](file://file-1yjcu1K12osyUYjxJV5SHb#:~:text=%D0%9F%D1%80%D0%BE%D0%B1%D1%80%D0%BE%D1%81%20%D0%BE%D1%81%D0%BD%D0%BE%D0%B2%D0%BD%D1%8B%D1%85%20e,%D0%B2%20GA4%20DebugView%20%D0%B2%D0%B8%D0%B4%D0%BD%D0%BE%20%D1%81%D0%BE%D0%B1%D1%8B%D1%82%D0%B8%D1%8F). Пользователю на почту приходит письмо с подтверждением. В Google Analytics (режим DebugView) видны события add\_to\_cart, begin\_checkout, purchase с соответствующими данными[[58]](file://file-1yjcu1K12osyUYjxJV5SHb#:~:text=%D0%9F%D1%80%D0%BE%D0%B1%D1%80%D0%BE%D1%81%20%D0%BE%D1%81%D0%BD%D0%BE%D0%B2%D0%BD%D1%8B%D1%85%20e,%D0%BF%D0%BE%D0%BA%D0%B0%D0%B7%D1%8B%D0%B2%D0%B0%D0%B5%D0%BC%20%D0%B7%D0%B0%D0%BF%D0%B8%D1%81%D1%8C%20%D0%B2%20%D0%B0%D0%B4%D0%BC%D0%B8%D0%BD%D0%BA%D0%B5%2F%D0%91%D0%94%20%D0%B8)[[59]](file://file-1yjcu1K12osyUYjxJV5SHb#:~:text=%D0%AD%D1%82%D0%B0%D0%BF%207%20%E2%80%94%20%D0%90%D0%BD%D0%B0%D0%BB%D0%B8%D1%82%D0%B8%D0%BA%D0%B0%2F%D0%BF%D1%80%D0%B0%D0%B2%D0%BE%D0%B2%D1%8B%D0%B5%20%D0%A0%D0%B5%D0%B7%D1%83%D0%BB%D1%8C%D1%82%D0%B0%D1%82%3A). После этого менеджер меняет статус заказа в Strapi, и (по желанию) клиент получает уведомление (можно реализовать отправку письма при смене статуса на "Отгружен" например).  
*Роль Codex:* AI помогает с написанием серверного кода для обработки заказа, с шаблоном письма (можно попросить его сгенерировать HTML-письмо с таблицей товаров), а также с интеграцией аналитики (правильный формат данных для GA4 события).

**Этап 7 – Аналитика и юридические разделы**  
*Описание:* Финализация SEO-настроек, подключение аналитики, реализация всех обязательных статических страниц и баннеров.  
*Работы:* Настроить Google Tag Manager и GA4 на сайте – добавить скрипты в <Head>. Протестировать e-commerce трекинг (с этапа 6). Реализовать микроразметку JSON-LD Product для страницы товара, проверить через Google Rich Results Test[[60]](file://file-1yjcu1K12osyUYjxJV5SHb#:~:text=SEO,%D0%BA%D0%B0%D0%BD%D0%BE%D0%BD%D0%B8%D0%BA%D0%B0%D0%BB%2F%D0%BC%D0%B5%D1%82%D0%B0%2C%20%D0%B1%D0%B0%D0%B7%D0%BE%D0%B2%D0%B0%D1%8F%20%D0%BA%D0%B0%D1%80%D1%82%D0%B0%20%D1%81%D0%B0%D0%B9%D1%82%D0%B0). Сгенерировать sitemap.xml (можно динамически с API route, возвращающим объединённый sitemap категорий, товаров, статичных страниц, который обновляется при импорте)[[23]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%98%D0%BD%D0%B4%D0%B5%D0%BA%D1%81%D0%B0%D1%86%D0%B8%D1%8F%3A%20%D0%BA%D0%B0%D1%82%D0%B5%D0%B3%D0%BE%D1%80%D0%B8%D0%B8%20%D0%B8%20%D1%82%D0%BE%D0%B2%D0%B0%D1%80%D1%8B%3B%20%D0%BA%D0%BE%D0%BC%D0%B1%D0%B8%D0%BD%D0%B0%D1%86%D0%B8%D0%B8,%D0%BD%D0%B0%20%D0%B1%D0%B0%D0%B7%D0%BE%D0%B2%D1%83%D1%8E%20%D0%BA%D0%B0%D1%82%D0%B5%D0%B3%D0%BE%D1%80%D0%B8%D1%8E). Заполнить контент стат. страниц: Политика конф., Оферта – согласовать текст с юристами и внести в Strapi. Вывести предупреждение о cookies (можно библиотеку или самому сделать простой див).  
*Артефакты:* Файл sitemap.xml доступный по /sitemap.xml, robots.txt разрешающий индексировать все кроме возможно /admin и /search. GA4 счетчик виден в исходном коде. Документ с выводом тестов: скриншот из Structured Data Testing Tool (успешно), проверка в GA DebugView событий.  
*Приёмка:* Проверка SEO: страница товара имеет теги <h1> = названию, теги meta. В коде страницы товара виден скрипт application/ld+json с корректными полями (name, image, sku, price, availability, etc.). Сайтмап открывается в браузере, содержит URL всех товаров и категорий. Robots.txt имеется, содержит ссылку на sitemap. Cookie-banner появляется при первом заходе и исчезает после согласия. Все юридические страницы доступны и соответствуют требованиям.  
*Роль Codex:* Помог при генерации JSON-LD шаблона (можно было скормить ему пример товара, он бы составил JSON-LD), с проверкой robots.txt best practices. Возможно даже написал черновик текста политики конфиденциальности, хотя юридический текст лучше предоставить заказчик.

**Этап 8 – Тестирование и загрузочное тестирование**  
*Описание:* Проведение интеграционного и пользовательского тестирования, поиск и исправление багов, подготовка к релизу.  
*Работы:* Составить **UAT чек-листы** по основным сценариям: регистрация заказа, поиск товара, фильтрация, адаптивность макета, и т.д. Провести серию испытаний: разработчики и/или тестировщики проходят по сценарию, отмечают Pass/Fail. Все баги собираются в трекере, производится их исправление и регресс-тест. Также провести нагрузочное тестирование (например, JMeter или k6) на поиск и открытие страниц, чтобы убедиться, что p95 TTFB укладывается в <800 мс при, скажем, 50 одновременных пользователях. Оптимизировать, если что-то обнаружено (например, добавить кеширование Redis на какой-то запрос).  
*Артефакты:* Документ с чек-листами и результатами (где указано, что пройдено, что найдено и исправлено)[[61]](file://file-1yjcu1K12osyUYjxJV5SHb#:~:text=%D0%A0%D0%B5%D0%B7%D1%83%D0%BB%D1%8C%D1%82%D0%B0%D1%82%3A). Отчёты об устранении багов (лог изменений).  
*Приёмка:* Все тест-кейсы успешно выполняются (100% Pass)[[62]](file://file-1yjcu1K12osyUYjxJV5SHb#:~:text=%D0%A7%D0%B5%D0%BA,%D0%BF%D1%80%D0%BE%D0%B7%D1%80%D0%B0%D1%87%D0%BD%D1%8B%D0%B5%20pass%2Ffail). Заказчик лично проводит приёмочное тестирование ключевых функций (демо-сценарии) и остётся удовлетворённым. Производительность и безопасность подтверждены (например, сканером типа OWASP ZAP критичных уязвимостей не найдено).  
*Роль Codex:* AI может сгенерировать ряд тестовых сценариев на основе требований (что еще проверить), а также помогать в отладке найденных проблем, предлагая решения по ошибкам. Мог даже участвовать в создании авто-тестов (например, Puppeteer скриптов) по сценарию.

**Этап 9 – Запуск в продакшен и стабилизация**  
*Описание:* Финальный релиз MVP на боевой сервер, контроль работы и исправление возможных пост-реблизных проблем.  
*Работы:* Развернуть актуальную версию на production окружении (если до этого работали на staging). Настроить доменное имя (marcus.su) на IP сервера, проверить DNS. Провести ещё раз все миграции на прод-базе. Сделать smoke-test после запуска: ключевые страницы открываются, заказ можно оформить. Настроить мониторинг на бою: подпись на алерты uptime. В течение первых 1-2 недель – оперативно реагировать на сбои или ошибки. Собрать отзывы первых пользователей, исправить UX-мелочи. *Артефакты:* Чек-лист деплоя (что сделать перед/во время запуска). Доклад об инцидентах за первую неделю (если были) и как решены[[63]](file://file-1yjcu1K12osyUYjxJV5SHb#:~:text=%D0%9F%D1%80%D0%BE%D0%B4,%D0%BC%D0%BE%D0%BD%D0%B8%D1%82%D0%BE%D1%80%D0%B8%D0%BD%D0%B3). *Приёмка:* Сайт успешно работает под нагрузкой боевых пользователей, заказчик подтверждает, что релиз состоялся. Контрольные метрики: uptime за первый месяц ~99.9%, ср. время ответа соответствует ожиданиям. Если были инциденты (падение, утечка) – их разобрали и устранили причины. Проект переходит в фазу поддержки или разработки новых функций.  
*Роль Codex:* В этой фазе Codex может помогать быстро анализировать логи ошибок, рекомендовать фиксы по stack trace. Также может предложить улучшения, которые заметны постфактум (рефакторинг, например).

**Фаза 2 – Рост и новые функции:** После успешного MVP запуска, планируется расширение функциональности. Ключевые направления развития: - **Онлайн-оплата (ЮKassa):** Добавить интеграцию с платёжным шлюзом. Будет разработан модуль, который при оформлении заказа перенаправляет на платёжную страницу ЮKassa или открывает платежный виджет на сайте. После успешной оплаты – обновляет статус заказа на "Оплачен". Также, возможно, реализовать pre-auth (блокировка средств) и списание после подтверждения менеджером. *Критерий:* Оплата проходит с тестовой и боевой карт, статусы корректно меняются, данные карты нигде у нас не хранятся (только id платежа, соответствуя PCI SAQ A)[[33]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%A1%D1%82%D1%80%D0%B0%D0%BD%D0%B8%D1%86%D1%8B%3A%20%D0%9F%D0%BE%D0%BB%D0%B8%D1%82%D0%B8%D0%BA%D0%B0%2C%20%D0%9E%D1%84%D0%B5%D1%80%D1%82%D0%B0%3B%20%D0%BA%D1%83%D0%BA%D0%B8). - **Интеграция служб доставки:** Подключить API СДЭК, Деловых Линий, Яндекс.Доставки. Это включает виджет выбора пункта выдачи (например, у СДЭК есть готовый JS-виджет), расчет стоимости доставки по API при оформлении заказа. После интеграции пользователь сможет на этапе checkout выбирать удобный способ, видеть стоимость доставки, а в админке менеджер будет получать эти данные и, возможно, может сразу создавать отправление в системе доставки. *Критерий:* В форме заказа появился выбор пункта выдачи на карте, стоимость рассчитывается по индексу/городу, данные о выборе сохраняются в Order. Тест: выбрать разные города – видеть разные цены, убедиться что запросы к API идут и ответы обрабатываются. - **Личный кабинет и программа лояльности:** Реализовать функционал аккаунтов: регистрация, вход, сохранение истории заказов. Возможно, бонусная программа (начисление баллов) – если актуально для Marcus. *Критерий:* Пользователь может зарегистрироваться, посмотреть историю, повторить заказ. Без багов безопасности (пароль хэшируется, JWT токен хранится httpOnly cookie). - **Умный поиск и SEO-улучшения:** Добавить словарь синонимов (например, через интерфейс админки пополняемый) и мультиязычный поиск (возможно, поиск по английским терминам, если каталог станет двуязычным – хотя мультиязычность не планируется на старте). Также можно реализовать **SEO-лендинги** – отдельные страницы под ключевые запросы (например, "Новогодние корпоративные подарки" – подборка товаров, оптимизированный контент). Такие лендинги можно генерировать через Strapi (тип контента "SEO Landing" с привязкой к поисковому запросу или категории)[[64]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%93%D0%BB%D0%B0%D0%B2%D0%BD%D0%B0%D1%8F%3B%20%D0%BA%D0%B0%D1%82%D0%B5%D0%B3%D0%BE%D1%80%D0%B8%D0%B8%2F%D1%84%D0%B8%D0%BB%D1%8C%D1%82%D1%80%D1%8B%2F%D0%BF%D0%B0%D0%B3%D0%B8%D0%BD%D0%B0%D1%86%D0%B8%D1%8F%3B%20PDP%20%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C%2F%D0%B2%D0%B0%D1%80%D0%B8%D0%B0%D1%86%D0%B8%D1%8F%3B%20%D0%BF%D0%BE%D0%B8%D1%81%D0%BA%2F%D0%BF%D0%BE%D0%B4%D1%81%D0%BA%D0%B0%D0%B7%D0%BA%D0%B8%3B,%D1%81%D1%82%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B5%3B%20%D0%BF%D0%BE%D1%80%D1%82%D1%84%D0%BE%D0%BB%D0%B8%D0%BE%3B%20%D0%BE%D0%BF%D1%82%D0%B8%D0%BC%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F%20%D0%B8%D0%B7%D0%BE%D0%B1%D1%80%D0%B0%D0%B6%D0%B5%D0%BD%D0%B8%D0%B9). *Критерий:* По слову-синониму (например, "USB drive" для "флешка") поиск выдает результаты. SEO-лендинг доступен по красивому URL, содержит заданный контент и список товаров, индексируется в Google (можно проверить через site:). - **Интеграция с «Моё дело» (бухгалтерия):** Вероятно, автоматизация бухгалтерии – экспорт заказов в систему учета "Моё дело" через их API. После каждого подтвержденного заказа дергать API для создания счета/акта. *Критерий:* После смены статуса заказа на "Готов к отгрузке" в системе "Моё дело" появляется соответствующий документ. - **Улучшение UX и конверсии:** Например, чат-бот на сайте для консультаций, онлайн-статус заказа (трекинг номера отправления через API доставки), отзывы клиентов на товары, сравнение товаров, и мультивитринность (если Marcus решит сделать версии сайта для разных регионов или направлений). Эти улучшения будут планироваться по мере роста.

Каждое из этих направлений будет оформлено отдельным мини-проектом/этапом с ТЗ, разработкой и тестированием. В них также возможно задействовать Codex для ускорения (например, генерация кода интеграции API по документации).

**Заключение:** Данный план охватывает все аспекты проекта Marcus – от технической архитектуры до поэтапной реализации и роста. При его выполнении особое внимание будет уделено качеству (тестирование, код-ревью), безопасности (защита данных, доступов) и соблюдению сроков этапов. Использование современных инструментов (Next.js, Strapi, Meilisearch) и автоматизации (CI/CD, Codex AI) позволит эффективно разработать и запустить продукт, который удовлетворит бизнес-требования и обеспечит основу для дальнейшего развития интернет-магазина Marcus.

**Источники:** Основой для архитектуры и плана послужило согласованное техническое задание Marcus[[65]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=Frontend%3A%20Next.js%20%28SSR%2FSSG%2FISR%20%2B%20on,revalidate%2C%20next%2Fimage)[[66]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=Product%20,SEO%2C%20publishedAt), детализированное поэтапное планирование разработки[[67]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%AD%D1%82%D0%B0%D0%BF%203%20%E2%80%94%20%D0%98%D0%BC%D0%BF%D0%BE%D1%80%D1%82%20Oasis,%D0%BC%D0%B5%D0%B4%D0%B8%D0%B0%20%2B%20ISR%2F%D1%80%D0%B5%D0%B8%D0%BD%D0%B4%D0%B5%D0%BA%D1%81)[[68]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%AD%D1%82%D0%B0%D0%BF%205%20%E2%80%94%20Frontend%20%28Next), а также дизайн-макеты интерфейса (PDF) и лучшие практики разработки e-commerce приложений на выбранном стеке.

[[1]](file://file-2pJTK1XjeeH2gT5WqUDEpg" \l ":~:text=Frontend%3A%20Next.js%20%28SSR%2FSSG%2FISR%20%2B%20on,revalidate%2C%20next%2Fimage) [[2]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=DB%3A%20PostgreSQL%3B%20%D0%BA%D0%B5%D1%88%2F%D0%BE%D1%87%D0%B5%D1%80%D0%B5%D0%B4%D0%B8%3A%20Redis%3B%20%D0%BC%D0%B5%D0%B4%D0%B8%D0%B0%3A,%D1%81%D0%BE%D0%B2%D0%BC%D0%B5%D1%81%D1%82%D0%B8%D0%BC%D0%BE%D0%B5%20%D1%85%D1%80%D0%B0%D0%BD%D0%B8%D0%BB%D0%B8%D1%89%D0%B5%20%2B%20CDN) [[3]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=Ubuntu%20LTS%2C%20users%2FSSH%2F%D1%84%D0%B0%D0%B5%D1%80%D0%B2%D0%BE%D0%BB%3B%20Nginx%20,js%20%2B%20PM2%20cluster%2C%20logrotate) [[4]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%98%D1%81%D1%82%D0%BE%D1%87%D0%BD%D0%B8%D0%BA%3A%20Oasis%20Catalog%20API%20,%D1%81%20%D1%86%D0%B5%D0%BD%D0%B0%D0%BC%D0%B8%2C%20%D0%BE%D1%81%D1%82%D0%B0%D1%82%D0%BA%D0%B0%D0%BC%D0%B8%20%D0%B8%20%D0%BC%D0%B5%D0%B4%D0%B8%D0%B0) [[5]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=lifecycle%2FIA) [[6]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%94%D0%BE%D1%81%D1%82%D0%B0%D0%B2%D0%BA%D0%B0) [[7]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%9F%D1%80%D0%BE%D0%B8%D0%B7%D0%B2%D0%BE%D0%B4%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE%D1%81%D1%82%D1%8C%3A%20API%20p95%20,Web%20Vitals%20%E2%80%94%20%C2%AB%D0%B7%D0%B5%D0%BB%D1%91%D0%BD%D0%B0%D1%8F%C2%BB%20%D0%B7%D0%BE%D0%BD%D0%B0) [[8]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=SEO%20%D0%B8%20%D0%BA%D0%BE%D0%BD%D1%82%D0%B5%D0%BD%D1%82) [[9]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%9A%D0%B0%D1%82%D0%B0%D0%BB%D0%BE%D0%B3%2C%20%D0%BA%D0%B0%D1%80%D1%82%D0%BE%D1%87%D0%BA%D0%B0%2C%20%D0%BC%D0%B5%D0%B4%D0%B8%D0%B0) [[10]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=Product%20,SEO%2C%20publishedAt) [[11]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=ProductVariant%20,isDefault) [[12]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%9F%D0%BE%D0%B8%D1%81%D0%BA%20%D0%B8%20%D1%84%D0%B8%D0%BB%D1%8C%D1%82%D1%80%D1%8B%20) [[13]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%9A%D0%BE%D1%80%D0%B7%D0%B8%D0%BD%D0%B0%20%D0%B8%20%D0%BE%D1%84%D0%BE%D1%80%D0%BC%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5%20%28%D0%B1%D0%B5%D0%B7%20%D0%BE%D0%BD%D0%BB%D0%B0%D0%B9%D0%BD) [[14]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%A7%D0%B0%D1%81%D1%82%D0%BE%D1%82%D0%B0%3A%20%D0%B5%D0%B6%D0%B5%D0%B4%D0%BD%D0%B5%D0%B2%D0%BD%D0%BE%20%D0%BD%D0%BE%D1%87%D1%8C%D1%8E%20%28cron%29,demand%20ISR) [[15]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=Tiered,%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B1%D0%BB%D0%BE%D0%BA%20%D0%BD%D0%B0%20MVP) [[16]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%9F%D0%B0%D0%B3%D0%B8%D0%BD%D0%B0%D1%86%D0%B8%D1%8F%20%E2%80%94%20rel%3D) [[17]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%9E%D0%BF%D0%B5%D1%87%D0%B0%D1%82%D0%BA%D0%B8%3A%201%20%D0%B4%D0%BB%D1%8F%20%D0%BA%D0%BE%D1%80%D0%BE%D1%82%D0%BA%D0%B8%D1%85%20,%D0%9D%D0%B0%D1%81%D1%82%D1%80%D0%B0%D0%B8%D0%B2%D0%B0%D0%B5%D1%82%D1%81%D1%8F%20%D0%BF%D0%B0%D1%80%D0%B0%D0%BC%D0%B5%D1%82%D1%80%D0%B0%D0%BC%D0%B8%20Meilisearch) [[18]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%9F%D0%BE%D0%B8%D1%81%D0%BA%20%D0%B8%20%D1%84%D0%B8%D0%BB%D1%8C%D1%82%D1%80%D1%8B%20) [[19]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%A4%D0%B0%D1%81%D0%B5%D1%82%D1%8B%20%D0%BF%D0%BE%20%D0%B0%D1%82%D1%80%D0%B8%D0%B1%D1%83%D1%82%D0%B0%D0%BC%3B%20%D1%81%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B8%20%28%D0%BF%D0%BE%D0%BF%D1%83%D0%BB%D1%8F%D1%80%D0%BD%D0%BE%D1%81%D1%82%D1%8C%2F%D1%86%D0%B5%D0%BD%D0%B0%2F%D0%BD%D0%BE%D0%B2%D0%B8%D0%B7%D0%BD%D0%B0%29) [[20]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%94%D0%BB%D0%B8%D0%BD%D0%BD%D1%8B%D0%B9%20%D1%85%D0%B2%D0%BE%D1%81%D1%82%20%D1%84%D0%B0%D1%81%D0%B5%D1%82%D0%BE%D0%B2%3A%20%D1%81%D0%B2%D0%BE%D1%80%D0%B0%D1%87%D0%B8%D0%B2%D0%B0%D0%B5%D0%BC%20%D0%B4%D0%BE,search%20%D0%BF%D1%80%D0%B8%20%3E10%20%D0%B7%D0%BD%D0%B0%D1%87%D0%B5%D0%BD%D0%B8%D0%B9) [[21]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%9F%D1%80%D0%BE%D0%B8%D0%B7%D0%B2%D0%BE%D0%B4%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%BE%D1%81%D1%82%D1%8C%3A%20API%20p95%20,Web%20Vitals%20%E2%80%94%20%C2%AB%D0%B7%D0%B5%D0%BB%D1%91%D0%BD%D0%B0%D1%8F%C2%BB%20%D0%B7%D0%BE%D0%BD%D0%B0) [[23]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%98%D0%BD%D0%B4%D0%B5%D0%BA%D1%81%D0%B0%D1%86%D0%B8%D1%8F%3A%20%D0%BA%D0%B0%D1%82%D0%B5%D0%B3%D0%BE%D1%80%D0%B8%D0%B8%20%D0%B8%20%D1%82%D0%BE%D0%B2%D0%B0%D1%80%D1%8B%3B%20%D0%BA%D0%BE%D0%BC%D0%B1%D0%B8%D0%BD%D0%B0%D1%86%D0%B8%D0%B8,%D0%BD%D0%B0%20%D0%B1%D0%B0%D0%B7%D0%BE%D0%B2%D1%83%D1%8E%20%D0%BA%D0%B0%D1%82%D0%B5%D0%B3%D0%BE%D1%80%D0%B8%D1%8E) [[26]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%91%D0%B5%D0%B7%D0%BE%D0%BF%D0%B0%D1%81%D0%BD%D0%BE%D1%81%D1%82%D1%8C%3A%20HTTPS%2C%20HSTS%2C%20%D1%81%D0%B5%D0%BA%D1%80%D0%B5%D1%82%D1%8B%20%D0%B2,%D0%B4%D0%BE%D1%81%D1%82%D1%83%D0%BF%20%D0%BA%20%D0%B0%D0%B4%D0%BC%D0%B8%D0%BD%D0%BA%D0%B5%20%E2%80%94%20IP%2F2FA) [[27]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%9D%D0%B0%D0%B1%D0%BB%D1%8E%D0%B4%D0%B0%D0%B5%D0%BC%D0%BE%D1%81%D1%82%D1%8C%3A%20%D0%BB%D0%BE%D0%B3%D0%B8%20%D0%B7%D0%B0%D0%BF%D1%80%D0%BE%D1%81%D0%BE%D0%B2%2F%D0%BE%D1%88%D0%B8%D0%B1%D0%BE%D0%BA%2C%20%D0%BC%D0%BE%D0%BD%D0%B8%D1%82%D0%BE%D1%80%D0%B8%D0%BD%D0%B3%20%D0%B0%D0%BF%D1%82%D0%B0%D0%B9%D0%BC%D0%B0%2C,%D0%B8%D0%BC%D0%BF%D0%BE%D1%80%D1%82%2C%20%D1%80%D0%B5%D0%B8%D0%BD%D0%B4%D0%B5%D0%BA%D1%81%2C%205xx) [[28]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%91%D1%8D%D0%BA%D0%B0%D0%BF%D1%8B%3A%20PostgreSQL%20,Meilisearch%20snapshot%2C%20S3%20lifecycle%2FIA) [[29]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%9D%D0%B0%D0%B4%D1%91%D0%B6%D0%BD%D0%BE%D1%81%D1%82%D1%8C%3A%2099.9,%D0%91%D0%94%2F%D0%BC%D0%B5%D0%B4%D0%B8%D0%B0) [[30]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=GA4%20%2B%20%D0%B1%D0%B0%D0%B7%D0%BE%D0%B2%D1%8B%D0%B5%20e,%D0%BF%D0%BB%D0%B0%D1%82%D0%B5%D0%B6%D0%B5%D0%B9%20%D0%B2%20MVP) [[33]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%A1%D1%82%D1%80%D0%B0%D0%BD%D0%B8%D1%86%D1%8B%3A%20%D0%9F%D0%BE%D0%BB%D0%B8%D1%82%D0%B8%D0%BA%D0%B0%2C%20%D0%9E%D1%84%D0%B5%D1%80%D1%82%D0%B0%3B%20%D0%BA%D1%83%D0%BA%D0%B8) [[44]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=Sitemap%20,%D0%B8%D0%BC%D0%BF%D0%BE%D1%80%D1%82%D0%B5) [[45]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%9E%D1%84%D0%BE%D1%80%D0%BC%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5%20%D0%B1%D0%B5%D0%B7%20%D0%BE%D0%BD%D0%BB%D0%B0%D0%B9%D0%BD,%D0%9E%D0%B6%D0%B8%D0%B4%D0%B0%D0%B5%D1%82%20%D0%BF%D0%BE%D0%B4%D1%82%D0%B2%D0%B5%D1%80%D0%B6%D0%B4%D0%B5%D0%BD%D0%B8%D1%8F%20%2F%20%D0%9A%20%D0%BE%D1%82%D0%B3%D1%80%D1%83%D0%B7%D0%BA%D0%B5%C2%BB) [[46]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%BF%D0%BE%D0%B4%D1%82%D0%B2%D0%B5%D1%80%D0%B6%D0%B4%D0%B5%D0%BD%D0%B8%D1%8F%20%2F%20%D0%9A%20%D0%BE%D1%82%D0%B3%D1%80%D1%83%D0%B7%D0%BA%D0%B5%C2%BB) [[47]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%AD%D1%82%D0%B0%D0%BF%200%20%E2%80%94%20%D0%9F%D1%80%D0%B5%D0%B4%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%20%D0%B8,%D1%84%D0%B8%D0%BD%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F%20%D0%A2%D0%97) [[49]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%98%D0%BD%D1%84%D1%80%D0%B0%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0%20%D0%B8%20DevOps) [[64]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%93%D0%BB%D0%B0%D0%B2%D0%BD%D0%B0%D1%8F%3B%20%D0%BA%D0%B0%D1%82%D0%B5%D0%B3%D0%BE%D1%80%D0%B8%D0%B8%2F%D1%84%D0%B8%D0%BB%D1%8C%D1%82%D1%80%D1%8B%2F%D0%BF%D0%B0%D0%B3%D0%B8%D0%BD%D0%B0%D1%86%D0%B8%D1%8F%3B%20PDP%20%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C%2F%D0%B2%D0%B0%D1%80%D0%B8%D0%B0%D1%86%D0%B8%D1%8F%3B%20%D0%BF%D0%BE%D0%B8%D1%81%D0%BA%2F%D0%BF%D0%BE%D0%B4%D1%81%D0%BA%D0%B0%D0%B7%D0%BA%D0%B8%3B,%D1%81%D1%82%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B5%3B%20%D0%BF%D0%BE%D1%80%D1%82%D1%84%D0%BE%D0%BB%D0%B8%D0%BE%3B%20%D0%BE%D0%BF%D1%82%D0%B8%D0%BC%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F%20%D0%B8%D0%B7%D0%BE%D0%B1%D1%80%D0%B0%D0%B6%D0%B5%D0%BD%D0%B8%D0%B9) [[65]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=Frontend%3A%20Next.js%20%28SSR%2FSSG%2FISR%20%2B%20on,revalidate%2C%20next%2Fimage) [[66]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=Product%20,SEO%2C%20publishedAt) [[67]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%AD%D1%82%D0%B0%D0%BF%203%20%E2%80%94%20%D0%98%D0%BC%D0%BF%D0%BE%D1%80%D1%82%20Oasis,%D0%BC%D0%B5%D0%B4%D0%B8%D0%B0%20%2B%20ISR%2F%D1%80%D0%B5%D0%B8%D0%BD%D0%B4%D0%B5%D0%BA%D1%81) [[68]](file://file-2pJTK1XjeeH2gT5WqUDEpg#:~:text=%D0%AD%D1%82%D0%B0%D0%BF%205%20%E2%80%94%20Frontend%20%28Next) Marcus\_TZ\_Final\_2025-08-28 (3).docx

<file://file-2pJTK1XjeeH2gT5WqUDEpg>

[[22]](file://file-1yjcu1K12osyUYjxJV5SHb#:~:text=%D0%AD%D1%82%D0%B0%D0%BF%205%20%E2%80%94%20Frontend%20%28Next,7%20%D0%9F%D0%BE%D1%80%D1%82%D1%84%D0%BE%D0%BB%D0%B8%D0%BE) [[24]](file://file-1yjcu1K12osyUYjxJV5SHb#:~:text=%D0%A0%D0%B0%D0%B7%D0%B2%D1%91%D1%80%D0%BD%D1%83%D1%82%20VDS%2C%20%D0%BD%D0%B0%D1%81%D1%82%D1%80%D0%BE%D0%B5%D0%BD%D1%8B%20Docker%2FDocker%20Compose%2C,proxy%20%28nginx%29%2C%20%D0%BB%D0%BE%D0%B3%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) [[25]](file://file-1yjcu1K12osyUYjxJV5SHb#:~:text=%D0%92%D1%8B%D0%BF%D1%83%D1%89%D0%B5%D0%BD%D1%8B%20%D0%B8%20%D0%B0%D0%B2%D1%82%D0%BE%D0%BF%D1%80%D0%BE%D0%B4%D0%BB%D0%B5%D0%B2%D0%B0%D1%8E%D1%82%D1%81%D1%8F%20HTTPS,Encrypt%2FCertbot) [[31]](file://file-1yjcu1K12osyUYjxJV5SHb#:~:text=%D0%B7%D0%B0%D0%BA%D0%B0%D0%B7%D0%B0%20%D0%B2%20%D0%B1%D0%B5%D0%BA%D0%B5%D0%BD%D0%B4%D0%B5%2C%20%D1%81%D0%B5%D1%80%D0%B2%D0%B8%D1%81%D0%BD%D1%8B%D0%B5%20%D0%BF%D0%B8%D1%81%D1%8C%D0%BC%D0%B0) [[32]](file://file-1yjcu1K12osyUYjxJV5SHb#:~:text=GA4%2FTag%20Manager%20%D0%BD%D0%B0%D1%81%D1%82%D1%80%D0%BE%D0%B5%D0%BD%D1%8B%2C%20%D1%81%D0%BE%D0%B1%D1%8B%D1%82%D0%B8%D1%8F%20e,%D0%BC%D0%B8%D0%BD%D0%B8%D0%BC%D1%83%D0%BC%3A%20%D0%BF%D1%80%D0%BE%D1%81%D0%BC%D0%BE%D1%82%D1%80%D1%8B%2C%20%D0%B4%D0%BE%D0%B1%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F%2C%20%D1%87%D0%B5%D0%BA%D0%B0%D1%83%D1%82) [[48]](file://file-1yjcu1K12osyUYjxJV5SHb#:~:text=CI%2FCD%20%D0%B4%D0%BB%D1%8F%20%D1%84%D1%80%D0%BE%D0%BD%D1%82%D0%B0%20%D0%B8%20%D0%B1%D1%8D%D0%BA%D0%B0,Terraform%2FIaC%20%E2%80%94%20%D0%BA%D0%B0%D0%BA%20%D0%B1%D0%B5%D0%BD%D1%87%D0%BC%D0%B0%D1%80%D0%BA%20%D0%BF%D1%80%D0%B0%D0%BA%D1%82%D0%B8%D0%BA%D0%B8) [[50]](file://file-1yjcu1K12osyUYjxJV5SHb#:~:text=%D0%A1%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D1%8B%20%D1%82%D0%B8%D0%BF%D1%8B%20%D0%BA%D0%BE%D0%BD%D1%82%D0%B5%D0%BD%D1%82%D0%B0%20%28%D0%BA%D0%B0%D1%82%D0%B5%D0%B3%D0%BE%D1%80%D0%B8%D0%B8%2F%D1%82%D0%BE%D0%B2%D0%B0%D1%80%D1%8B%2F%D0%B0%D1%82%D1%80%D0%B8%D0%B1%D1%83%D1%82%D1%8B%2F%D0%BC%D0%B5%D0%B4%D0%B8%D0%B0%29) [[51]](file://file-1yjcu1K12osyUYjxJV5SHb#:~:text=%D0%A7%D0%B5%D1%80%D0%BD%D0%BE%D0%B2%D0%B0%D1%8F%20%D0%B4%D0%BE%D0%BA%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D0%B0%D1%86%D0%B8%D1%8F%20%D0%BF%D0%BE%20%D1%81%D1%85%D0%B5%D0%BC%D0%B0%D0%BC%20%D0%B8,%D1%80%D0%B5%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B5%20%D0%BE%D1%82%D0%B2%D0%B5%D1%82%D1%8B%20API%20%D0%B8%D0%B7%20staging) [[52]](file://file-1yjcu1K12osyUYjxJV5SHb#:~:text=%D0%94%D0%B5%D0%BC%D0%BE%D0%BD%D1%81%D1%82%D1%80%D0%B0%D1%86%D0%B8%D1%8F%2F%D0%BF%D1%80%D0%B8%D1%91%D0%BC%D0%BA%D0%B0%3A%20Postman,%D1%80%D0%B5%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B5%20%D0%BE%D1%82%D0%B2%D0%B5%D1%82%D1%8B%20API%20%D0%B8%D0%B7%20staging) [[53]](file://file-1yjcu1K12osyUYjxJV5SHb#:~:text=%D0%A1%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D1%8B%20%D0%B8%D0%BC%D0%BF%D0%BE%D1%80%D1%82%D0%B0%20%D0%BA%D0%B0%D1%82%D0%B0%D0%BB%D0%BE%D0%B3%D0%B0%2F%D0%BC%D0%B5%D0%B4%D0%B8%D0%B0%2C%20%D0%BB%D0%BE%D0%B3%20%D0%B8%D0%BC%D0%BF%D0%BE%D1%80%D1%82%D0%B0,%D1%81%20%D0%BE%D1%82%D1%87%D1%91%D1%82%D0%BE%D0%BC%20%D0%BE%D0%B1%20%D0%BE%D1%88%D0%B8%D0%B1%D0%BA%D0%B0%D1%85) [[54]](file://file-1yjcu1K12osyUYjxJV5SHb#:~:text=%D0%9D%D0%B0%D1%81%D1%82%D1%80%D0%BE%D0%B5%D0%BD%D0%B0%20%D1%80%D0%B5%D0%B3%D0%B5%D0%BD%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D1%8F%20%D1%81%D1%82%D1%80%D0%B0%D0%BD%D0%B8%D1%86%20%D0%B1%D0%B5%D0%B7%20%D0%BF%D0%BE%D0%BB%D0%BD%D0%BE%D0%B3%D0%BE,%D0%BE%D0%B1%D0%BD%D0%BE%D0%B2%D0%B8%D0%BB%D0%B0%D1%81%D1%8C%20%D0%B1%D0%B5%D0%B7%20%D0%BF%D0%BE%D0%BB%D0%BD%D0%BE%D0%B3%D0%BE%20%C2%ABnext%20build%C2%BB) [[55]](file://file-1yjcu1K12osyUYjxJV5SHb#:~:text=%D0%92%D0%BA%D0%BB%D1%8E%D1%87%D0%B5%D0%BD%D1%8B%20facet,search%3B%20%D0%B1%D0%B5%D0%BD%D1%87%D0%BC%D0%B0%D1%80%D0%BA%20%D0%BD%D0%B5%D1%81%D0%BA%D0%BE%D0%BB%D1%8C%D0%BA%D0%B8%D1%85%20%D0%B7%D0%B0%D0%BF%D1%80%D0%BE%D1%81%D0%BE%D0%B2) [[56]](file://file-1yjcu1K12osyUYjxJV5SHb#:~:text=%D0%B7%D0%B0%D0%BA%D0%B0%D0%B7%D0%B0%20%D0%B2%20%D0%B1%D0%B5%D0%BA%D0%B5%D0%BD%D0%B4%D0%B5%2C%20%D1%81%D0%B5%D1%80%D0%B2%D0%B8%D1%81%D0%BD%D1%8B%D0%B5%20%D0%BF%D0%B8%D1%81%D1%8C%D0%BC%D0%B0) [[57]](file://file-1yjcu1K12osyUYjxJV5SHb#:~:text=%D0%9F%D1%80%D0%BE%D0%B1%D1%80%D0%BE%D1%81%20%D0%BE%D1%81%D0%BD%D0%BE%D0%B2%D0%BD%D1%8B%D1%85%20e,%D0%B2%20GA4%20DebugView%20%D0%B2%D0%B8%D0%B4%D0%BD%D0%BE%20%D1%81%D0%BE%D0%B1%D1%8B%D1%82%D0%B8%D1%8F) [[58]](file://file-1yjcu1K12osyUYjxJV5SHb#:~:text=%D0%9F%D1%80%D0%BE%D0%B1%D1%80%D0%BE%D1%81%20%D0%BE%D1%81%D0%BD%D0%BE%D0%B2%D0%BD%D1%8B%D1%85%20e,%D0%BF%D0%BE%D0%BA%D0%B0%D0%B7%D1%8B%D0%B2%D0%B0%D0%B5%D0%BC%20%D0%B7%D0%B0%D0%BF%D0%B8%D1%81%D1%8C%20%D0%B2%20%D0%B0%D0%B4%D0%BC%D0%B8%D0%BD%D0%BA%D0%B5%2F%D0%91%D0%94%20%D0%B8) [[59]](file://file-1yjcu1K12osyUYjxJV5SHb#:~:text=%D0%AD%D1%82%D0%B0%D0%BF%207%20%E2%80%94%20%D0%90%D0%BD%D0%B0%D0%BB%D0%B8%D1%82%D0%B8%D0%BA%D0%B0%2F%D0%BF%D1%80%D0%B0%D0%B2%D0%BE%D0%B2%D1%8B%D0%B5%20%D0%A0%D0%B5%D0%B7%D1%83%D0%BB%D1%8C%D1%82%D0%B0%D1%82%3A) [[60]](file://file-1yjcu1K12osyUYjxJV5SHb#:~:text=SEO,%D0%BA%D0%B0%D0%BD%D0%BE%D0%BD%D0%B8%D0%BA%D0%B0%D0%BB%2F%D0%BC%D0%B5%D1%82%D0%B0%2C%20%D0%B1%D0%B0%D0%B7%D0%BE%D0%B2%D0%B0%D1%8F%20%D0%BA%D0%B0%D1%80%D1%82%D0%B0%20%D1%81%D0%B0%D0%B9%D1%82%D0%B0) [[61]](file://file-1yjcu1K12osyUYjxJV5SHb#:~:text=%D0%A0%D0%B5%D0%B7%D1%83%D0%BB%D1%8C%D1%82%D0%B0%D1%82%3A) [[62]](file://file-1yjcu1K12osyUYjxJV5SHb#:~:text=%D0%A7%D0%B5%D0%BA,%D0%BF%D1%80%D0%BE%D0%B7%D1%80%D0%B0%D1%87%D0%BD%D1%8B%D0%B5%20pass%2Ffail) [[63]](file://file-1yjcu1K12osyUYjxJV5SHb#:~:text=%D0%9F%D1%80%D0%BE%D0%B4,%D0%BC%D0%BE%D0%BD%D0%B8%D1%82%D0%BE%D1%80%D0%B8%D0%BD%D0%B3) Результаты\_и\_демонстрация\_по\_этапам.docx

<file://file-1yjcu1K12osyUYjxJV5SHb>

[[34]](file://file-8njwtbzoHADixaYdrroLxW#:~:text=%D0%9A%D0%BE%D0%BC%D0%BF%D0%B0%D0%BD%D0%B8%D1%8F%20%D0%9F%D0%BE%D1%80%D1%82%D1%84%D0%BE%D0%BB%D0%B8%D0%BE%20%D0%9A%D0%BE%D0%BD%D1%82%D0%B0%D0%BA%D1%82%D1%8B%20%D0%9A%D0%B0%D1%82%D0%B0%D0%BB%D0%BE%D0%B3%20%D0%9E%D0%B4%D0%B5%D0%B6%D0%B4%D0%B0,%D0%9F%D0%BE%D1%81%D1%83%D0%B4%D0%B0%20%D0%95%D0%B6%D0%B5%D0%B4%D0%BD%D0%B5%D0%B2%D0%BD%D0%B8%D0%BA%D0%B8%20%D0%B8%20%D0%B1%D0%BB%D0%BE%D0%BA%D0%BD%D0%BE%D1%82%D1%8B%20%D0%9D%D0%B0%D0%BD%D0%B5%D1%81%D0%B5%D0%BD%D0%B8%D0%B5) [[35]](file://file-8njwtbzoHADixaYdrroLxW#:~:text=%D0%9F%D0%BE%D1%80%D1%82%D1%84%D0%BE%D0%BB%D0%B8%D0%BE%20%D0%9A%D0%BE%D0%BD%D1%82%D0%B0%D0%BA%D1%82%D1%8B%20%D0%9A%D0%B0%D1%82%D0%B0%D0%BB%D0%BE%D0%B3%20%D0%9E%D0%B4%D0%B5%D0%B6%D0%B4%D0%B0%20%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%BD%D1%8B%D0%B5,%D0%95%D0%B6%D0%B5%D0%B4%D0%BD%D0%B5%D0%B2%D0%BD%D0%B8%D0%BA%D0%B8%20%D0%B8%20%D0%B1%D0%BB%D0%BE%D0%BA%D0%BD%D0%BE%D1%82%D1%8B%20%D0%9D%D0%B0%D0%BD%D0%B5%D1%81%D0%B5%D0%BD%D0%B8%D0%B5%20%D0%A2%D0%B0%D0%BC%D0%BF%D0%BE%D0%BF%D0%B5%D1%87%D0%B0%D1%82%D1%8C) [[36]](file://file-8njwtbzoHADixaYdrroLxW#:~:text=%D0%9A%D0%B0%D1%82%D0%B0%D0%BB%D0%BE%D0%B3%20%D0%9E%D0%B4%D0%B5%D0%B6%D0%B4%D0%B0%20%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%BD%D1%8B%D0%B5%20%D1%82%D0%BE%D0%B2%D0%B0%D1%80%D1%8B%20%D0%9F%D0%BE%D1%81%D1%83%D0%B4%D0%B0,%D0%A2%D0%B0%D0%BC%D0%BF%D0%BE%D0%BF%D0%B5%D1%87%D0%B0%D1%82%D1%8C%20%D0%9B%D0%B0%D0%B7%D0%B5%D1%80%D0%BD%D0%B0%D1%8F%20%D0%B3%D1%80%D0%B0%D0%B2%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0%20%D0%A3%D0%A4%20%D0%BF%D0%B5%D1%87%D0%B0%D1%82%D1%8C) [[37]](file://file-8njwtbzoHADixaYdrroLxW#:~:text=marcus%40marcus,%D0%BA%D0%BB%D0%B8%D0%B5%D0%BD%D1%82%D0%B0%D0%BC) [[38]](file://file-8njwtbzoHADixaYdrroLxW#:~:text=%2B7%20%28800%29%2070,%D0%BD%D0%B0%20%D0%BD%D0%B5%D0%BC%20%D1%84%D1%83%D1%82%D0%B1%D0%BE%D0%BB%D0%BA%D0%B8%2C%20%D0%BF%D0%BE%D0%BB%D0%BE%2C%20%D1%82%D0%BE%D0%BB%D1%81%D1%82%D0%BE%D0%B2%D0%BA%D0%B8) [[39]](file://file-8njwtbzoHADixaYdrroLxW#:~:text=marcus%40marcus,%D0%9A%D0%BE%D0%BD%D1%82%D0%B0%D0%BA%D1%82%D1%8B%20%D0%9A%D0%B0%D1%82%D0%B0%D0%BB%D0%BE%D0%B3%20%D0%9E%D0%B4%D0%B5%D0%B6%D0%B4%D0%B0%20%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%BD%D1%8B%D0%B5%20%D1%82%D0%BE%D0%B2%D0%B0%D1%80%D1%8B) [[40]](file://file-8njwtbzoHADixaYdrroLxW#:~:text=match%20at%20L875%20%D0%9A%D0%B0%D1%82%D0%B0%D0%BB%D0%BE%D0%B3%20%2B7,%D0%9D%D0%B0%D0%BD%D0%B5%D1%81%D0%B5%D0%BD%D0%B8%D0%B5%20%D0%9F%D0%BE%D0%BC%D0%BE%D1%89%D1%8C%20%D0%9F%D0%BE%D1%80%D1%82%D1%84%D0%BE%D0%BB%D0%B8%D0%BE%20%D0%9A%D0%BE%D0%BD%D1%82%D0%B0%D0%BA%D1%82%D1%8B%20%D0%9A%D0%BE%D0%BC%D0%BF%D0%B0%D0%BD%D0%B8%D1%8F) [[41]](file://file-8njwtbzoHADixaYdrroLxW#:~:text=marcus%40marcus,67%2068%20%D0%A1%D0%BB%D0%B5%D0%B4%D1%83%D1%8E%D1%89%D0%B0%D1%8F) [[42]](file://file-8njwtbzoHADixaYdrroLxW#:~:text=%2B7%20%28800%29%2070,%D0%BA%D0%BE%D0%BD%D1%84%D0%B8%D0%B4%D0%B5%D0%BD%D1%86%D0%B8%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D1%81%D1%82%D0%B8%20%D0%A6%D0%B2%D0%B5%D1%82%20%D0%BA%D0%BE%D1%80%D0%BF%D1%83%D1%81%D0%B0%20%D0%9E%D0%B1%D1%8A%D1%91%D0%BC%2C%20%D0%BC%D0%BB) [[43]](file://file-8njwtbzoHADixaYdrroLxW#:~:text=%2B7%20%28800%29%2070,%D0%9F%D0%BE%D0%BB%D0%B8%D1%82%D0%B8%D0%BA%D0%B0%20%D0%BA%D0%BE%D0%BD%D1%84%D0%B8%D0%B4%D0%B5%D0%BD%D1%86%D0%B8%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D1%81%D1%82%D0%B8%20%D0%A5%D0%B0%D1%80%D0%B0%D0%BA%D1%82%D0%B5%D1%80%D0%B8%D1%81%D1%82%D0%B8%D0%BA%D0%B8%20%D0%A6%D0%B2%D0%B5%D1%82%20%D0%BA%D0%BE%D1%80%D0%BF%D1%83%D1%81%D0%B0) Сайт Marcus (1).pdf

<file://file-8njwtbzoHADixaYdrroLxW>