ПРАВООБЛАДАТЕЛЬ:

ОБЩЕСТВО С ОГРАНИЧЕННОЙ ОТВЕТСТВЕННОСТЬЮ «ИНТЕНСА»

ИНН 7107540675

ОПИСАНИЕ ТЕХНИЧЕСКОЙ АРХИТЕКТУРЫ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ «INTENSA E-COMMERCE PLATFORM» (альтернативное название «INTENSA SHOP»)

Электронный документ

Листов 54

г.Тула

2025

1. СПИСОК СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

API – Application Programming Interface (интерфейс программирования приложений)

REST – Representational State Transfer (архитектурный стиль взаимодействия)

HTTP – HyperText Transfer Protocol (протокол передачи гипертекста)

HTTPS – HyperText Transfer Protocol Secure (безопасный протокол передачи гипертекста)

JSON – JavaScript Object Notation (текстовый формат обмена данными)

СУБД – Система управления базами данных

БД – База данных

ПО – Программное обеспечение

DDD – Domain-Driven Design (предметно-ориентированное проектирование)

SOLID – принципы объектно-ориентированного программирования

ORM – Object-Relational Mapping (объектно-реляционное отображение)

CSRF – Cross-Site Request Forgery (межсайтовая подделка запроса)

XSS – Cross-Site Scripting (межсайтовый скриптинг)

SQL – Structured Query Language (язык структурированных запросов)

SEO – Search Engine Optimization (поисковая оптимизация)

UUID – Universally Unique Identifier (универсальный уникальный идентификатор)

CDN – Content Delivery Network (сеть доставки контента)

CDP – Customer Data Platform (платформа клиентских данных)

SSR – Server-Side Rendering (рендеринг на стороне сервера)

SPA – Single Page Application (одностраничное приложение)

---

2. ОБЩИЕ СВЕДЕНИЯ

Документ содержит описание технической архитектуры программного обеспечения (ПО) «Intensa Shop».

ПО «Intensa Shop» представляет собой серверную часть (backend) интернет-магазина, разработанную с использованием фреймворка Laravel. Проект реализован в виде headless API, что позволяет использовать его с различными клиентскими приложениями, в том числе с основным frontend-приложением на Nuxt.

Основное назначение проекта — обеспечение полного функционала интернет-магазина, включая:

- Каталог товаров с фильтрацией и поиском
- Корзину покупок
- Оформление заказов
- Управление пользователями
- Интеграцию с внешними системами (1С, платежные системы, CDP Mindbox)
- Административную панель для управления контентом

Основные цели проекта:

- Создание масштабируемого и гибкого API для интернет-магазина
- Обеспечение высокой производительности и отказоустойчивости системы
- Реализация всех необходимых бизнес-процессов интернет-магазина
- Обеспечение безопасности данных и транзакций
- Интеграция с внешними системами учета и управления

Технологический стек:

- PHP 8.4 – основной язык программирования
- Laravel 11.x – фреймворк для разработки веб-приложений
- PostgreSQL 17 – система управления базами данных
- Redis 7.x – система кэширования и управления очередями
- Nuxt 3.x – фреймворк для frontend-приложения
- Laravel Sail – среда разработки на основе Docker

Архитектурные принципы:

- Domain-Driven Design (DDD) – подход к проектированию, ориентированный на предметную область
- Clean Architecture – архитектурный подход, обеспечивающий разделение ответственности
- SOLID – набор принципов объектно-ориентированного программирования

Информация о документе и версионировании

Версия документа: 1.0
Версия ПО: 1.0.0
Дата выпуска архитектуры: 16 июня 2025 года

Статус версии: Финальная (Final Release)

Истории версий архитектуры:
- v1.0 (16.06.2025) -- Первая версия архитектуры для ПО v1.0.0

Совместимость:
- Данная архитектура описывает ПО «Intensa Shop» версии 1.0.0
- Версионирование следует семантическому версионированию (Semantic Versioning)
- При выпуске новых версий архитектура должна обновляться

Авторы документа:
- Шишкин Иван Андреевич – Технический директор
- Козлов Дмитрий Андреевич – Руководитель отдела разработки

Дата согласования документа: 16 июня 2025 года
Статус согласования: Утвержден

---

3. ПРОЕКТИРОВАНИЕ И ОПИСАНИЕ СЦЕНАРИЕВ ИСПОЛЬЗОВАНИЯ СИСТЕМЫ

3.1. Сценарии по ролям в представлении вариантов использования

Общие сценарии, доступные всем пользователям (Гость):

1. **Просмотр каталога товаров**
    - Просмотр категорий товаров
    - Просмотр списка товаров с пагинацией
    - Просмотр детальной информации о товаре
    - Фильтрация товаров по различным параметрам
    - Поиск товаров
2. **Работа с корзиной**
    - Добавление товаров в корзину
    - Изменение количества товаров в корзине
    - Удаление товаров из корзины
    - Применение промокодов
    - Просмотр общей стоимости заказа
3. **Оформление заказа**
    - Выбор способа доставки
    - Выбор способа оплаты
    - Указание контактной информации
    - Создание заказа
4. **Просмотр контента**
    - Просмотр публикаций и новостей
    - Просмотр информационных страниц

Сценарии пользователя с ролью «Пользователь» (Клиент):

1. **Аутентификация и профиль**

- Регистрация в системе
- Вход в систему по номеру телефона с SMS-кодом
- Выход из системы
- Редактирование личных данных
- Управление подписками на рассылки

2. **Управление профилями покупателей**
   - Просмотр сохраненных профилей покупателей
   - Создание нового профиля покупателя
   - Редактирование профиля покупателя
   - Использование профиля при оформлении заказа
3. **Работа с заказами**
   - Просмотр истории заказов
   - Просмотр детальной информации о заказе
   - Отмена заказа (в определенных статусах)
4. **Работа с избранным**
   - Добавление товаров в избранное
   - Удаление товаров из избранного
   - Просмотр списка избранных товаров
5. **Программа лояльности**
   - Просмотр баланса бонусов
   - Просмотр истории начислений и списаний

Сценарии пользователя с ролью «Администратор»:

1. **Вход в систему**
   - Вход в административную панель
   - Редактирование своего профиля
2. **Управление товарами**
   - Просмотр списка товаров
   - Создание нового товара
   - Редактирование товара
   - Деактивация/активация товара
   - Управление предложениями товара
   - Управление ценами товара
   - Управление остатками на складах
3. **Управление категориями**
   - Просмотр списка категорий
   - Создание новой категории
   - Редактирование категории
   - Деактивация/активация категории
   - Управление иерархией категорий
4. **Управление атрибутами и фильтрами**
   - Создание и редактирование атрибутов
   - Управление списками значений атрибутов
   - Создание и настройка фильтров для каталога
5. **Управление заказами**
   - Просмотр списка заказов
   - Просмотр детальной информации о заказе

    - Изменение статуса заказа
    - Редактирование состава заказа
    - Управление оплатой заказа
6. **Управление пользователями**
    - Просмотр списка пользователей
    - Редактирование данных пользователей
    - Управление ролями пользователей
7. **Управление контентом**
    - Создание и редактирование публикаций
    - Управление разделами сайта
    - Управление категориями публикаций
8. **Управление складами**
    - Просмотр и редактирование складов
    - Управление остатками товаров
9. **Управление доставкой и оплатой**
    - Настройка способов доставки
    - Настройка способов оплаты
10. **Обмен данными с 1С**
    - Просмотр истории обменов
    - Инициация обмена каталогом
    - Просмотр логов обмена

Сценарии внешних систем:

1. **Система 1С**
    - Выгрузка каталога товаров
    - Выгрузка цен и остатков
    - Получение новых заказов
    - Обновление статусов заказов
2. **Платежная система Paykeeper**
    - Создание платежа
    - Уведомление о статусе оплаты
    - Подтверждение платежа
3. **CDP система Mindbox**
    - Синхронизация данных клиентов
    - Расчет промокодов и акций
    - Управление программой лояльности
    - Персонализация предложений

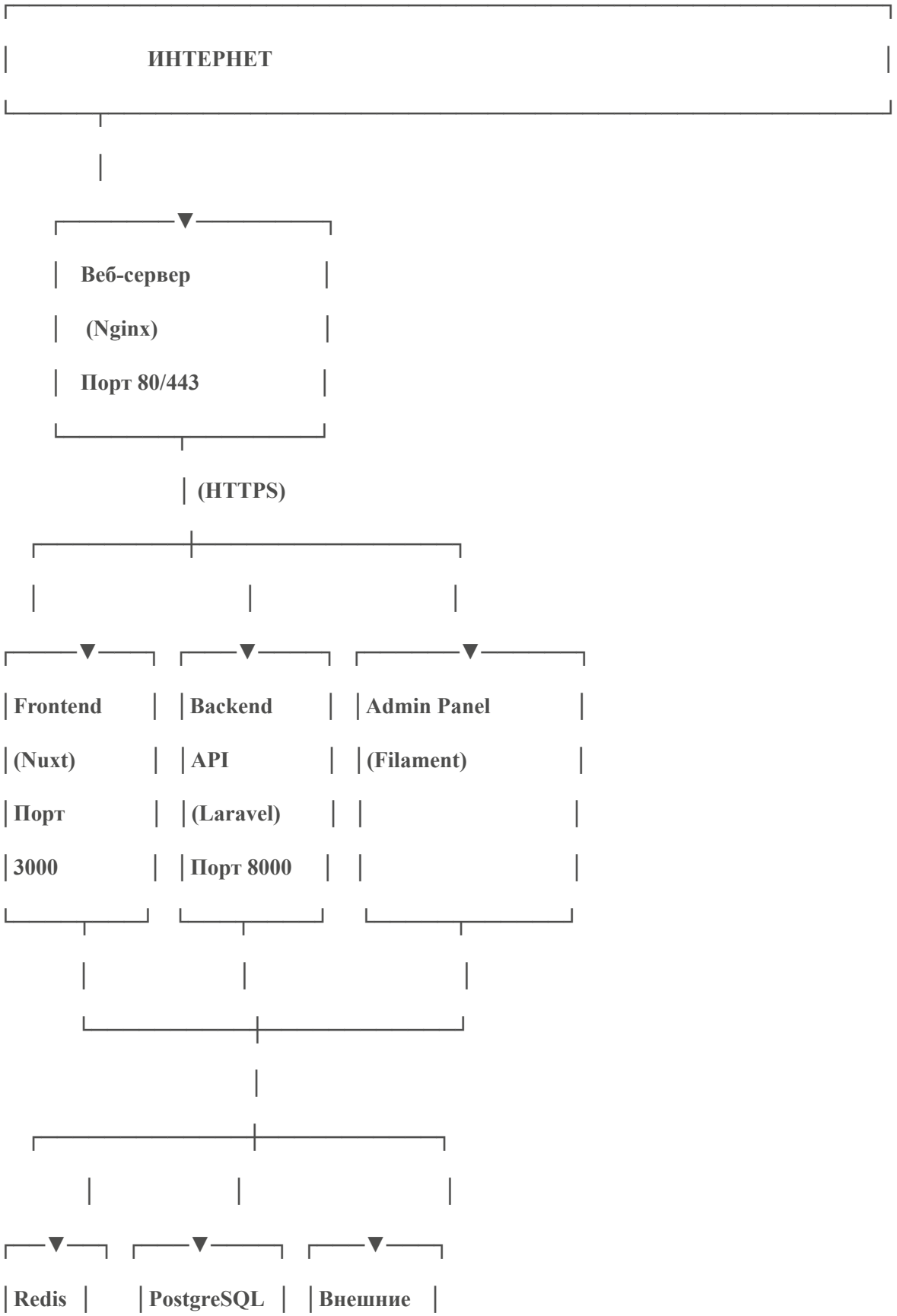
---

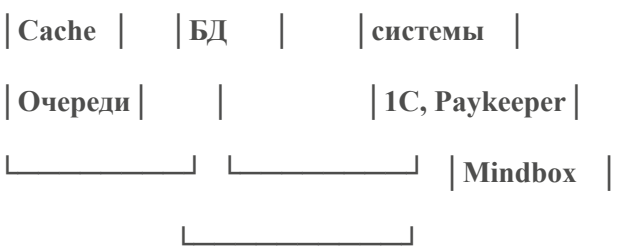
4. СТРУКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

4.1. Общая схема технологического стека

На рисунке 4.1 показана общая схема технологического стека ПО «Intensa Shop».

Рисунок 4.1 – Общая схема технологического стека ПО «Intensa Shop»





Описание компонентов:

Frontend (Nuxt.js на порту 3000):

- Клиентское приложение на Vue 3 + Nuxt 3

- Server-Side Rendering (SSR) для SEO

- Управление состоянием через Pinia

- HTTPS защита

Backend API (Laravel на порту 8000):

- RESTful API версии 1.0

- Обработка бизнес-логики

- Маршрутизация запросов

- Аутентификация и авторизация

- Версионирование API (v1)

- Middleware: CORS, аутентификация, валидация, rate limiting

Admin Panel (Filament):

- Управление товарами и категориями

- Управление заказами и пользователями

- Управление контентом и интеграциями

Redis 7.x:

- In-memory кэш для часто используемых данных

- Управление очередями асинхронных задач

- Хранение сессий пользователей

- Rate limiting и throttling

- Session cache (время жизни: 2 часа)

PostgreSQL 17 (БД):

- Главное хранилище данных

- Поддержка транзакций ACID

- Индексирование для оптимизации запросов

- JSON поля для гибкого хранения

Внешние системы:

- 1С: обмен каталогом, ценами, остатками, заказами

- Paykeeper: обработка платежей

- Mindbox: CDP, программа лояльности, персонализация

- SMS-сервис: отправка кодов подтверждения

Протоколы и стандарты:

- HTTPS/TLS 1.2+ для всех соединений

- REST для API

- JSON для обмена данными

- XML для интеграции с 1С

4.2. Взаимодействие компонентов

ПО «Intensa Shop» состоит из следующих компонентов:

«Backend API» – выполняет функции:

- Авторизации и аутентификации пользователей
- Обработки бизнес-логики приложения
- Взаимодействия с базой данных
- Интеграции с внешними системами
- Предоставления RESTful API для клиентских приложений

«Frontend-приложение (Nuxt)» – предоставляет функционал:

- Отображения пользовательского интерфейса
- Взаимодействия с Backend API
- Управления состоянием приложения на клиенте
- Роутинга и навигации
- Server-Side Rendering для SEO

«Административная панель (Filament)» – реализует:

- Управление каталогом товаров
- Управление заказами
- Управление пользователями
- Управление контентом
- Настройку системы

«Система кэширования (Redis)» – выполняет функции:

- Кэширования данных для повышения производительности
- Управления очередями задач
- Хранения сессий пользователей
- Rate limiting для API

«База данных (PostgreSQL)» – выполняет функции по хранению:

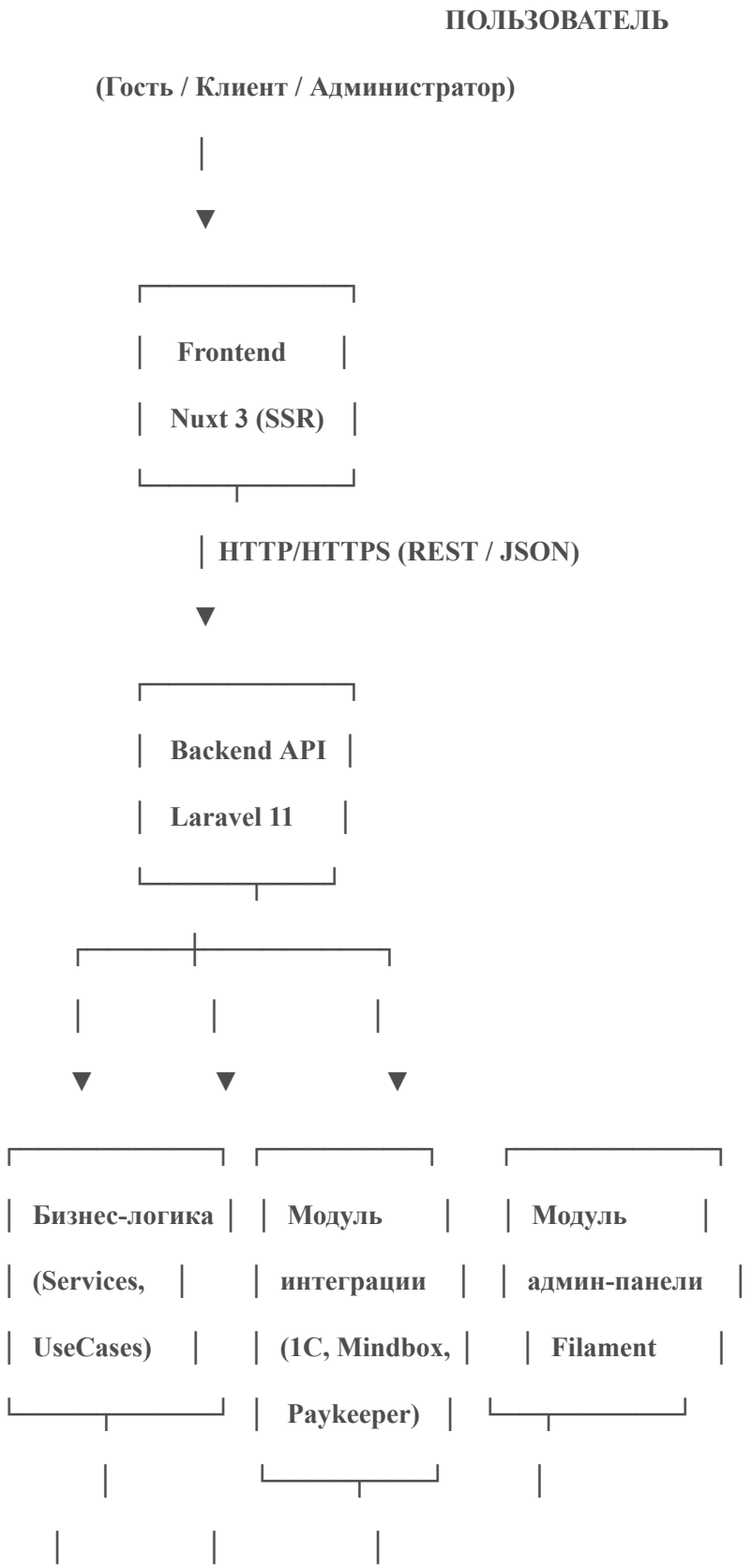
- Данных о товарах и категориях
- Информации о заказах
- Данных пользователей
- Контента сайта
- Логов и истории операций

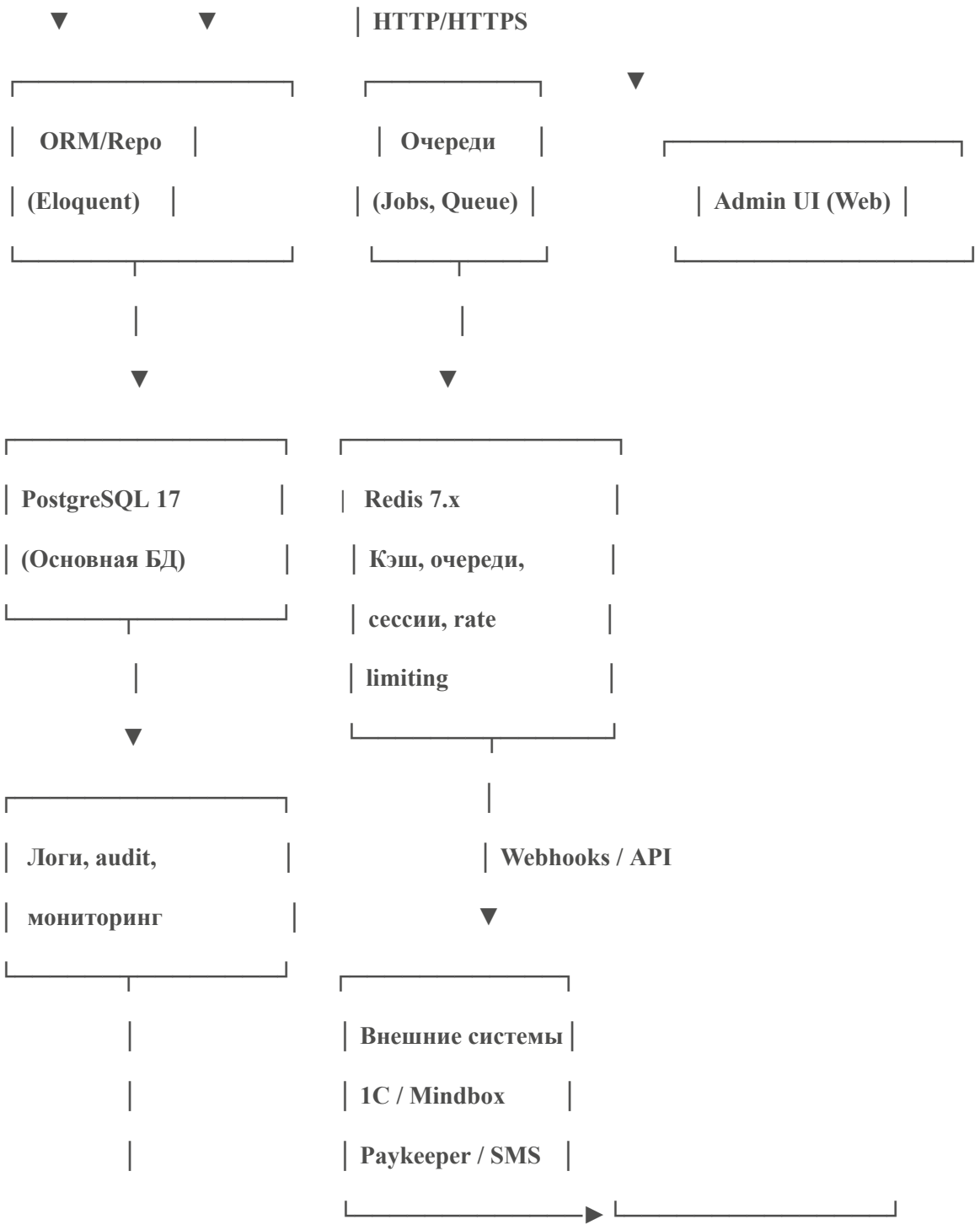
«Библиотека интеграции» – реализует:

- Подключение к внешним системам (1С, Mindbox, Paykeeper)
- Синхронизацию данных
- Обработку вебхуков
- Отправку уведомлений

На рисунке 4.2 показана детальная схема взаимодействия компонентов.

Рисунок 4.2 – Детальная схема взаимодействия компонентов ПО «Intensa Shop»

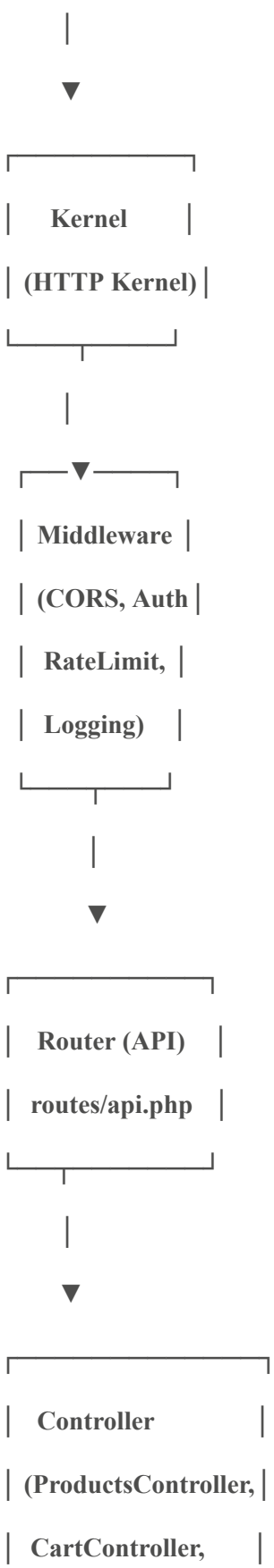


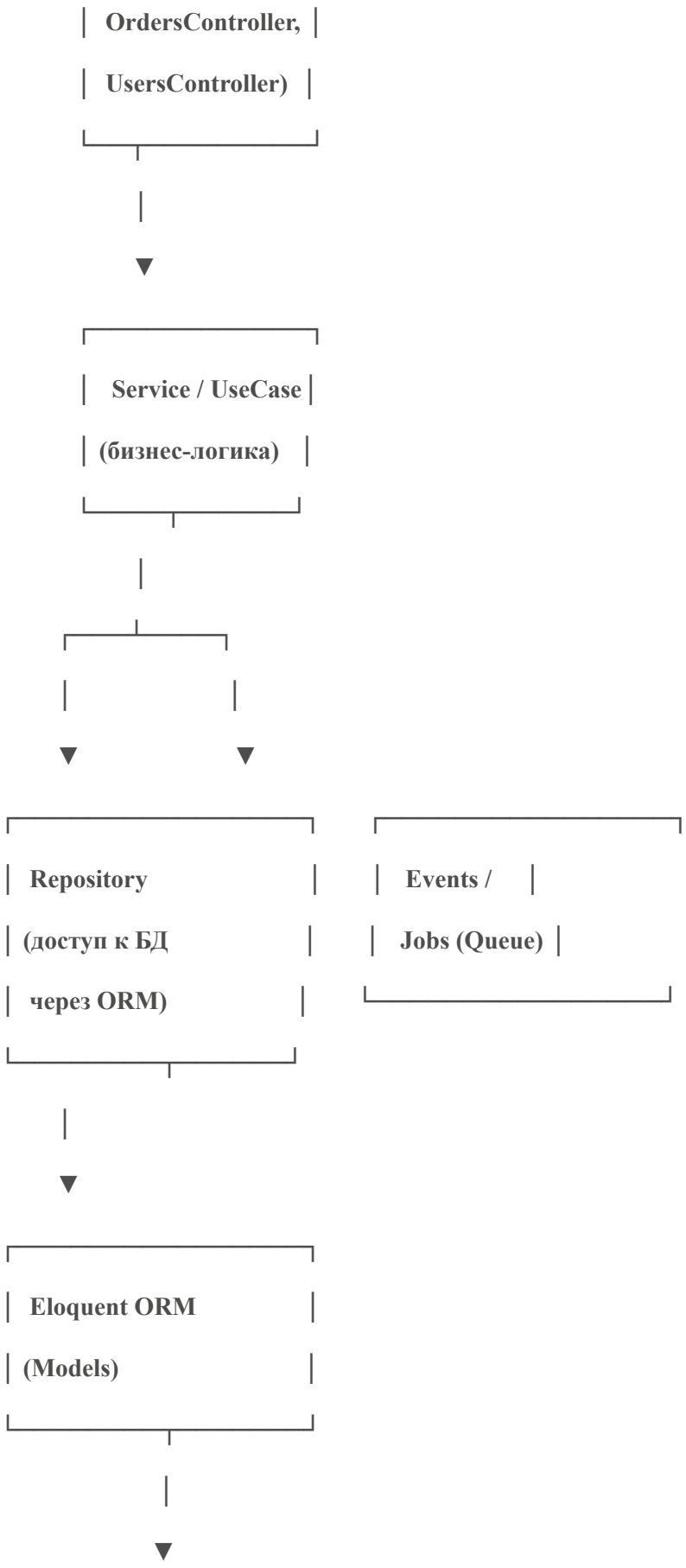


4.3. Схема взаимодействия модулей Laravel

На рисунке 4.3 показана схема взаимодействия основных модулей в рамках Laravel-приложения.

Рисунок 4.3 – Схема взаимодействия модулей Laravel

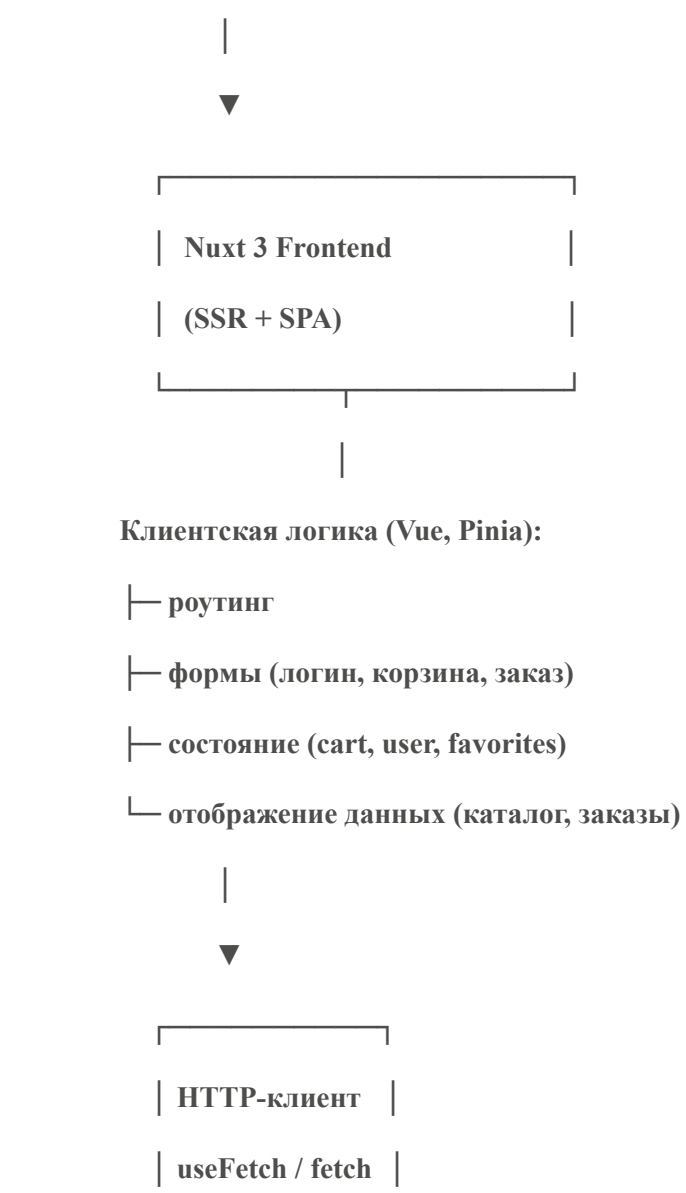

HTTP ЗАПРОС
Kernel
(HTTP Kernel)
Middleware
(CORS, Auth
RateLimit,
Logging)
Router (API)
routes/api.php
Controller
(ProductsController,
CartController,

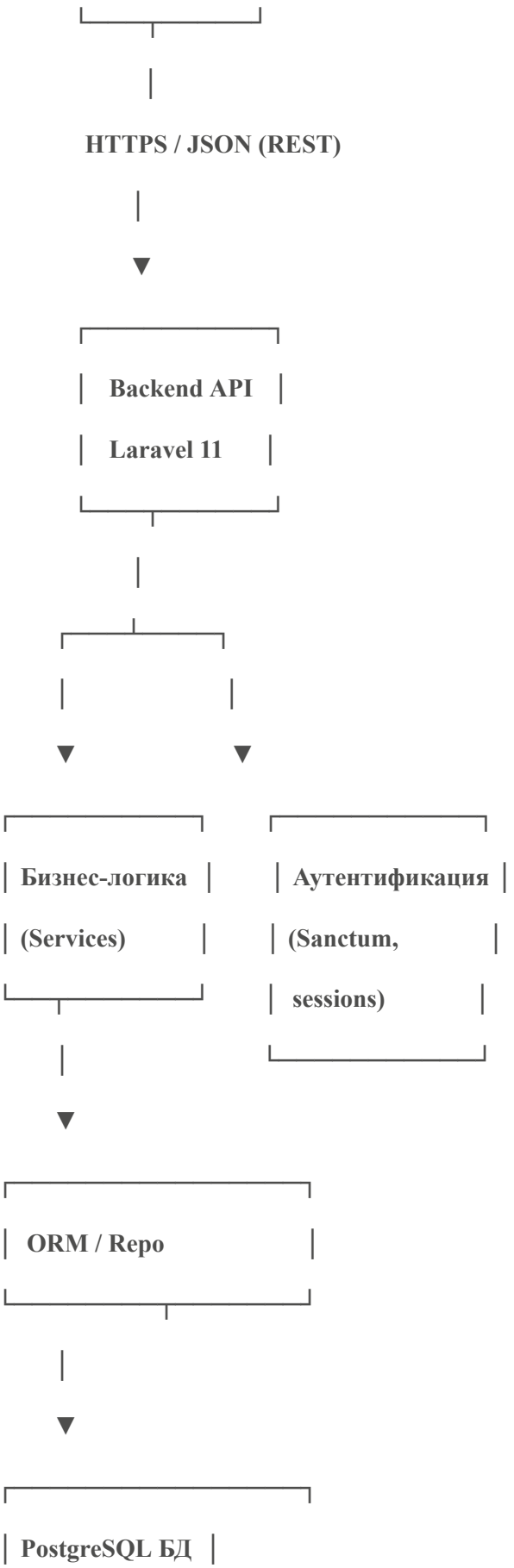

OrdersController,
UsersController)
Service / UseCase
(бизнес-логика)
Repository
(доступ к БД
через ORM)
Events /
Jobs (Queue)
Eloquent ORM
(Models)

4.4. Схема взаимодействия Frontend (Nuxt) и Backend (Laravel)

На рисунке 4.4 показана детальная схема взаимодействия между Frontend и Backend приложениями.

Рисунок 4.4 – Схема взаимодействия Frontend (Nuxt) и Backend (Laravel)




HTTPS / JSON (REST)
Backend API
Laravel 11
Бизнес-логика
(Services)
Аутентификация
(Sanctum,
sessions)
ORM / Repo
PostgreSQL БД

└──────────────────────┘

Ответ Backend API:

- HTTP статус (200, 201, 400, 401, 422, 500)

- JSON-структура (data, meta, errors)

Nuxt:

- обновляет состояние Pinia

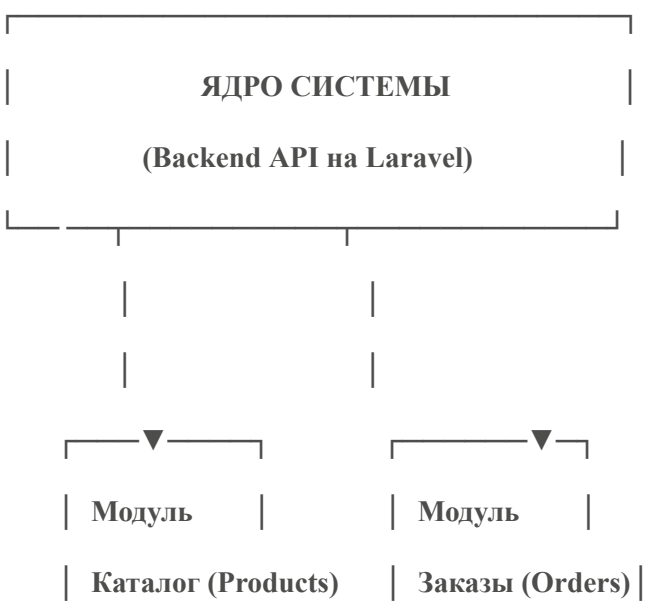
- перерисовывает компоненты

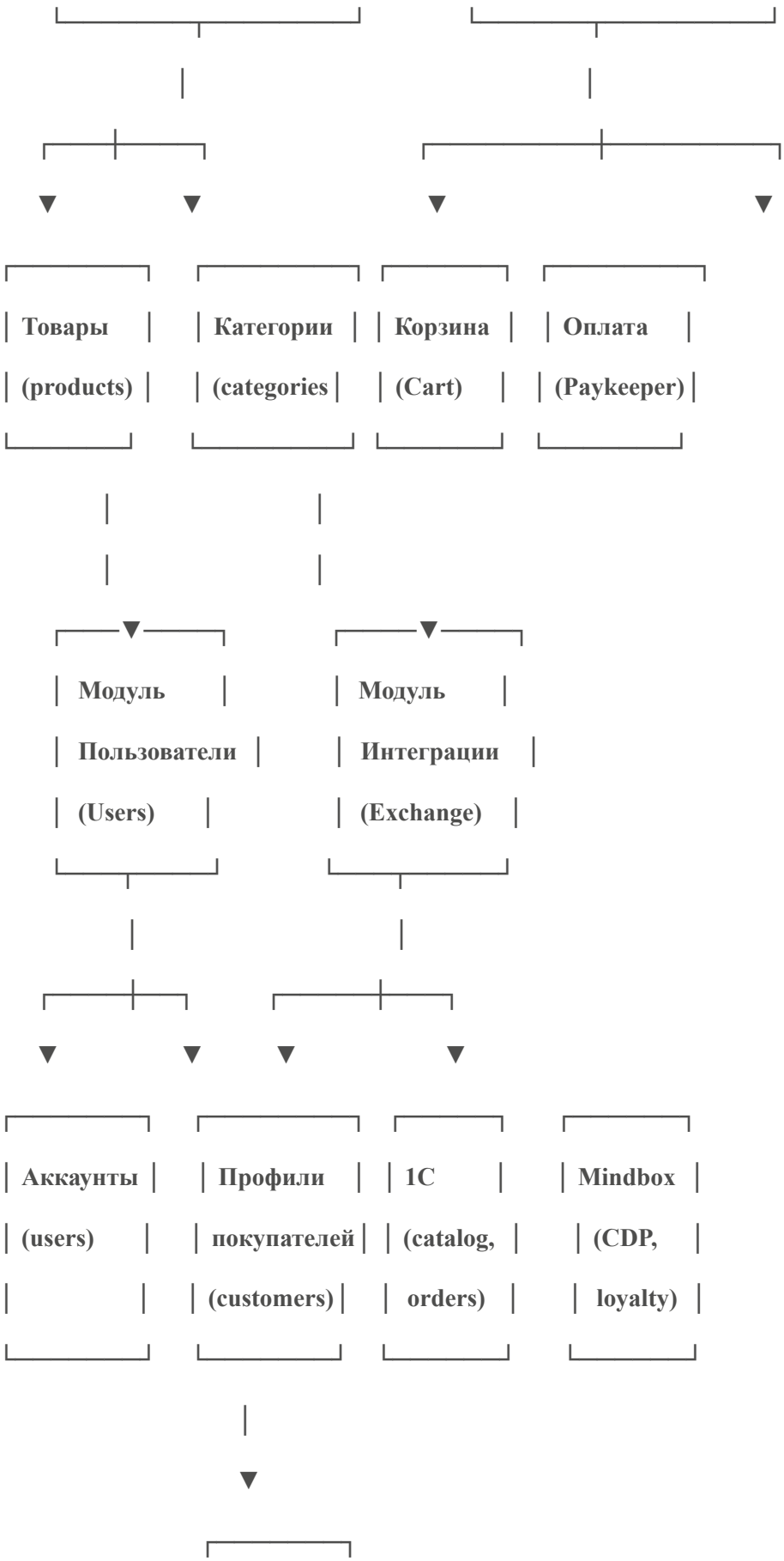
- при SSR часть страниц рендерится на сервере Nuxt

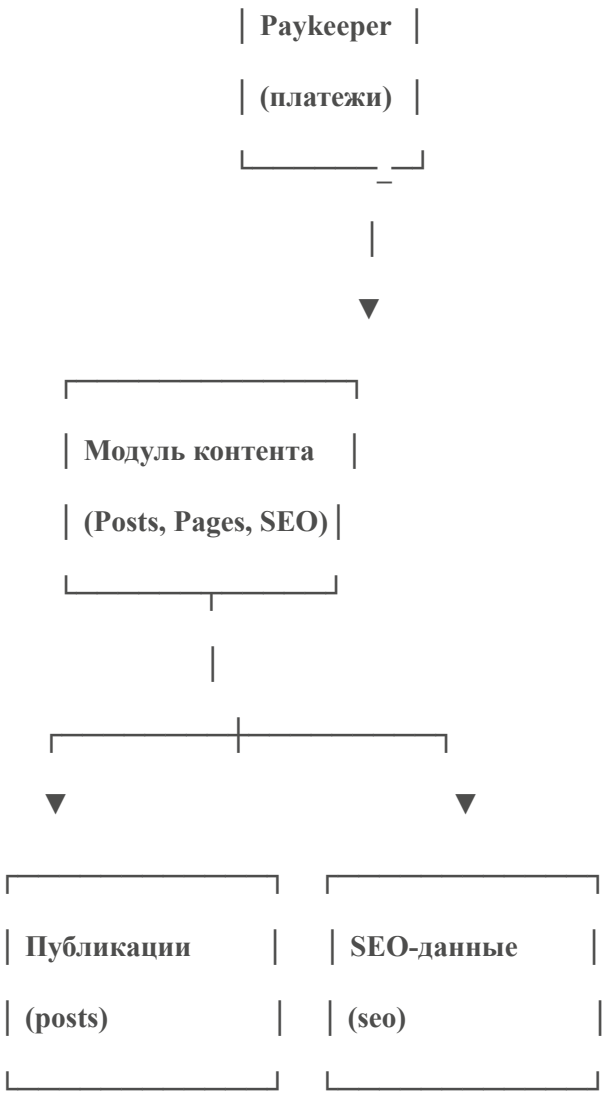
4.5. Функциональная структура

На рисунке 4.5 показана функциональная структура ПО «Intensa Shop».

Рисунок 4.5 – Функциональная структура ПО «Intensa Shop»




Товары (products)
Категории (categories)
Корзина (Cart)
Оплата (Paykeeper)
Модуль Пользователи (Users)
Модуль Интеграции (Exchange)
Аккаунты (users)
Профили покупателей (customers)
1С (catalog, orders)
Mindbox (CDP, loyalty)



Над всеми модулями:

- Модуль аутентификации и авторизации (roles, permissions)

- Модуль логирования и аудита

- Модуль очередей и фоновых задач

- Модуль админ-панели Filament (управление всеми сущностями)

4.6. Описание основных модулей

4.6.1. Ядро системы

Ядро системы представляет собой веб-сервис на базе фреймворка Laravel, обеспечивающий:

- Обработку HTTP-запросов по протоколу HTTPS

- Маршрутизацию запросов к соответствующим контроллерам
- Применение middleware для обработки CORS, аутентификации, валидации
- Версионирование API (текущая версия v1)

Основные API контракты:

Контроллер Products (Товары):

- GET /api/v1/products/paginate – пагинация товаров
- GET /api/v1/products/search – поиск товаров
- GET /api/v1/products/{id} – получение товара
- GET /api/v1/products/{id}/recommend – рекомендуемые товары
- GET /api/v1/products/{id}/stocks – наличие товара
- GET /api/v1/products/{id}/seo – SEO-данные товара

Контроллер Cart (Корзина):

- GET /api/v1/cart – получение корзины
- POST /api/v1/cart/add – добавление товара
- POST /api/v1/cart/remove – удаление товара
- POST /api/v1/cart/setQuantity – изменение количества
- POST /api/v1/cart/setPromoCode – применение промокода

Контроллер Orders (Заказы):

- GET /api/v1/orders – список заказов
- POST /api/v1/orders – создание заказа
- POST /api/v1/orders/init – инициализация заказа
- GET /api/v1/orders/{id} – получение заказа
- POST /api/v1/orders/{id}/cancel – отмена заказа

Контроллер Users (Пользователи):

- POST /api/v1/users/register – регистрация
- POST /api/v1/users/login/phone – вход по телефону
- POST /api/v1/users/logout – выход
- GET /api/v1/users/current – текущий пользователь
- PATCH /api/v1/users/{id} – обновление данных

GET /api/v1/products/paginate

Полный пример API контракта с JSON схемой:

Request Parameters:

├─ page (integer, required): номер страницы (default: 1)

├─ per_page (integer): товаров на странице (default: 20, max: 100)

├─ sort (string): поле сортировки (name, price, created_at)

└─ category_id (integer): фильтр по категории

Response (200 OK):


```
{
  "data": [
    {
      "id": 1,
      "name": "Product Name",
      "price": 1999.99,
      "description": "...",
      "image_url": "...",
      "available": true,
      "stock": 50
    }
  ],
  "meta": {
    "current_page": 1,
    "total": 1000,
    "per_page": 20
  }
}
```


Errors:

├─ 400 Bad Request: неверные параметры

├─ 401 Unauthorized: требуется аутентификация

├─ 429 Too Many Requests: rate limit превышен

└─ 500 Internal Server Error: ошибка сервера

Rate limit headers:

├─ X-RateLimit-Limit: 1000

├─ X-RateLimit-Remaining: 999

└─ X-RateLimit-Reset: timestamp

4.6.2. Модуль веб-интерфейса (Frontend Nuxt)

Формирует пользовательские формы и компоненты, обеспечивающие:

- Отображение каталога товаров с фильтрацией
- Работу с корзиной покупок
- Процесс оформления заказа
- Личный кабинет пользователя
- Управление избранным
- Просмотр истории заказов
- Отображение контента (новости, статьи)

4.6.3. Модуль аутентификации и авторизации

Реализует следующий функционал:

Stateful аутентификация:

- Аутентификация по номеру телефона с SMS-кодом
- Создание и управление сессиями
- Хранение сессий в Redis
- Автоматическая привязка посетителя к пользователю

Bearer токены для внешних систем:

- Генерация токенов для API-интеграций
- Проверка прав доступа токенов
- Управление сроком действия токенов

Система ролей и прав:

- Гость – ограниченный доступ
- Пользователь – доступ к личному кабинету
- Администратор – полный доступ к системе

4.6.4. Модуль обработки платежей

Обеспечивает:

- Создание платежей через Paykeeper
- Обработку уведомлений от платежной системы
- Подтверждение оплаты заказов
- Поддержку оплаты при получении
- Возврат средств

4.6.5. Модуль расчета доставки

Реализует:

- Курьерскую доставку с расчетом стоимости
- Самовывоз из пунктов выдачи
- Расчет даты доставки
- Интеграцию со складами для определения наличия

4.6.6. Модуль интеграции с внешними системами

Интеграция с 1С:

- Импорт каталога товаров из XML
- Обновление цен и остатков
- Экспорт новых заказов
- Обновление статусов заказов

Интеграция с Mindbox:

- Синхронизация данных клиентов
- Расчет промокодов и скидок
- Управление программой лояльности
- Персонализация предложений

Интеграция с Paykeeper:

- Создание платежных ссылок
- Обработка вебхуков
- Подтверждение платежей

4.6.7. Модуль административной панели (Filament)

Предоставляет интерфейс для управления:

- Каталогом товаров и категориями
- Заказами и их статусами
- Пользователями и ролями
- Контентом сайта
- Складами и остатками

- Настройками доставки и оплаты
- Обменом данными с 1С

4.6.8. Модуль SEO

Обеспечивает:

- Генерацию мета-тегов для страниц
- Создание SEO-дружественных URL
- Генерацию sitemap.xml
- Управление мета-данными через админ-панель

4.6.9. Модуль хранения результатов в БД

Сохраняет в базе данных:

- Данные о товарах и категориях
- Информацию о заказах и платежах
- Данные пользователей и их профили
- Историю просмотров и избранное
- Логи обмена с внешними системами
- Контент сайта (публикации, страницы)

При первом запуске ПО модуль выполняет миграции, создавая необходимую схему данных в PostgreSQL.

---

5. СТРУКТУРА БАЗЫ ДАННЫХ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

5.1. Поддерживаемые СУБД

ПО поддерживает работу со следующими системами управления базами данных (СУБД):

- **PostgreSQL 17** (основная СУБД)

5.2. Основные таблицы базы данных

5.2.1. Таблицы модуля Products

products – хранит информацию о товарах

- id – уникальный идентификатор
- name – название товара
- description – описание
- article – артикул
- brand_id – связь с брендом
- is_active – статус активности

- created_at, updated_at – метки времени

categories – хранит категории товаров

- id – уникальный идентификатор
- name – название категории
- parent_id – родительская категория (иерархия)
- slug – URL-адрес
- sort – порядок сортировки
- is_active – статус активности

offers – хранит предложения товаров

- id – уникальный идентификатор
- product_id – связь с товаром
- article – артикул предложения
- is_active – статус активности

attributes – хранит атрибуты товаров

- id – уникальный идентификатор
- code – символьный код
- name – название атрибута
- type – тип атрибута (строка, список, число)

prices – хранит цены товаров и предложений

- id – уникальный идентификатор
- priceable_type – тип сущности (товар/предложение)
- priceable_id – идентификатор сущности
- type_id – тип цены
- value – значение цены
- currency – валюта

stocks – хранит остатки товаров на складах

- id – уникальный идентификатор
- offer_id – связь с предложением
- warehouse_id – связь со складом
- quantity – количество

5.2.2. Таблицы модуля Cart

carts – хранит корзины

- id – уникальный идентификатор
- visitor_id – связь с посетителем
- promo_code – примененный промокод

cart_lines – хранит позиции корзины

- id – уникальный идентификатор
- cart_id – связь с корзиной
- offer_id – связь с предложением
- quantity – количество
- price – цена на момент добавления

5.2.3. Таблицы модуля Orders

orders – хранит заказы

- id – уникальный идентификатор
- user_id – связь с пользователем
- status – статус заказа
- total_amount – общая сумма
- delivery_type – тип доставки
- payment_type – тип оплаты
- created_at, updated_at – метки времени

order_lines – хранит позиции заказа

- id – уникальный идентификатор
- order_id – связь с заказом
- offer_id – связь с предложением
- quantity – количество
- price – цена на момент заказа
- discount – скидка

invoices – хранит счета на оплату

- id – уникальный идентификатор
- order_id – связь с заказом
- amount – сумма
- status – статус оплаты
- payment_url – ссылка на оплату
- paid_at – дата оплаты

5.2.4. Таблицы модуля Users

users – хранит пользователей

- id – уникальный идентификатор
- phone – номер телефона
- email – электронная почта
- name – имя
- is_active – статус активности

customers – хранит профили покупателей

- id – уникальный идентификатор

- user_id – связь с пользователем
- name – имя получателя
- phone – телефон получателя
- address – адрес доставки

roles – хранит роли

- id – уникальный идентификатор
- name – название роли
- permissions – права доступа

5.2.5. Таблицы модуля Visitors

visitors – хранит посетителей

- id – уникальный идентификатор (UUID)
- user_id – связь с пользователем (после авторизации)
- created_at – дата создания

favorites – хранит избранные товары

- id – уникальный идентификатор
- visitor_id – связь с посетителем
- product_id – связь с товаром

views – хранит просмотры товаров

- id – уникальный идентификатор
- visitor_id – связь с посетителем
- product_id – связь с товаром
- viewed_at – дата просмотра

5.2.6. Таблицы модуля Exchange

exchanges – хранит задачи обмена

- id – уникальный идентификатор
- type – тип обмена (catalog/orders)
- driver – драйвер обмена (1c)
- status – статус выполнения
- started_at, finished_at – метки времени

entries – хранит записи обмена

- id – уникальный идентификатор
- exchange_id – связь с обменом
- type – тип данных
- data – данные для обработки
- priority – приоритет обработки
- status – статус обработки

exchange_logs – хранит логи обменов

- id – уникальный идентификатор
- type – тип обмена
- driver – драйвер обмена
- executed_at – дата выполнения

5.2.7. Таблицы модуля Posts

posts – хранит публикации

- id – уникальный идентификатор
- section_id – связь с разделом
- name – название
- description – описание
- path – путь в URL
- picture – изображение
- published_at – дата публикации
- activated_at – дата активации
- sort – порядок сортировки

sections – хранит разделы

- id – уникальный идентификатор
- name – название
- path – путь в URL
- activated_at – дата активации

5.2.8. Таблицы модуля Forms

form_entries – хранит записи форм

- id – уникальный идентификатор
- type – тип формы
- data – данные формы
- status – статус обработки
- reason – причина (если есть ошибка)

5.2.9. Вспомогательные таблицы

warehouses – хранит склады

- id – уникальный идентификатор
- name – название склада
- address – адрес
- can_take – возможность самовывоза
- is_active – статус активности

seo – хранит SEO-данные

- id – уникальный идентификатор
- seoable_type – тип сущности
- seoable_id – идентификатор сущности
- title – заголовок
- h1 – основной заголовок
- description – описание
- keywords – ключевые слова

sessions – хранит сессии пользователей

- id – идентификатор сессии
- user_id – связь с пользователем
- payload – данные сессии
- last_activity – время последней активности

5.3. Индексы базы данных

Для оптимизации производительности созданы следующие индексы:

Индексы для таблицы products:

- idx_products_is_active
- idx_products_brand_id
- idx_products_article

Индексы для таблицы offers:

- idx_offers_product_id
- idx_offers_article
- idx_offers_is_active

Индексы для таблицы prices:

- idx_prices_priceable
- idx_prices_type_id

Индексы для таблицы stocks:

- idx_stocks_offer_id
- idx_stocks_warehouse_id

Индексы для таблицы orders:

- idx_orders_user_id
- idx_orders_status
- idx_orders_created_at

Индексы для таблицы categories:

- idx_categories_parent_id

- idx_categories_slug
- idx_categories_is_active

---

6. ПРИМЕНЯЕМЫЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

6.1. Серверная часть (Backend)

Основной сервис ПО реализован как веб-приложение на фреймворке Laravel 11.x и развернут на серверах с использованием следующих технологий:

Язык программирования:

- PHP 8.4

Фреймворк и библиотеки:

- Laravel 11.x – основной фреймворк приложения
- Laravel Sanctum – аутентификация API и управление токенами
- Laravel Telescope – инструмент отладки и мониторинга

Архитектурные паттерны:

- Domain-Driven Design (DDD)
- Clean Architecture
- Repository Pattern
- Command Query Responsibility Segregation (CQRS)

ORM и работа с БД:

- Eloquent ORM – объектно-реляционное отображение
- Query Builder – построитель запросов
- Миграции – версионирование схемы БД

6.2. Клиентская часть (Frontend)

Логика клиентского приложения реализована с использованием:

Фреймворк:

- Nuxt 3.x – фреймворк для Vue.js

Язык программирования:

- TypeScript – типизированный JavaScript

Управление состоянием:

- Pinia – хранилище состояния для Vue

HTTP-клиент:

- Встроенный fetch API / useFetch из Nuxt

Сборка:

- Vite – инструмент сборки

6.3. База данных

Логика хранения данных приложения реализована с использованием:

СУБД:

- PostgreSQL 17 – реляционная СУБД

Особенности использования:

- Транзакции для обеспечения целостности данных
- Индексы для оптимизации запросов
- Полнотекстовый поиск
- JSON-поля для гибкого хранения данных

6.4. Кэширование и очереди

Система кэширования и управления очередями:

Redis 7.x используется для:

- Кэширования данных
- Хранения сессий пользователей
- Управления очередями фоновых задач
- Хранения временных данных

6.5. Веб-сервер

Логика обработки HTTP-запросов реализована с использованием:

Веб-сервер:

- Nginx – высокопроизводительный HTTP-сервер

Обработка PHP:

- PHP-FPM 8.4 – FastCGI Process Manager

SSL/TLS:

- Let's Encrypt – бесплатные SSL-сертификаты

6.6. Административная панель

Логика управления системой реализована с использованием:

Фреймворк:

- Filament 3.x – фреймворк для создания административных панелей в Laravel

Пакеты:

- intensa/filament – базовый пакет для админ-панели
- intensa/product – управление товарами
- intensa/post – управление контентом
- intensa/order – управление заказами
- awcodes/curator – управление медиа-файлами

6.7. Интеграции с внешними системами

Обмен данными с 1С:

- XML – формат обмена данными
- HTTP/HTTPS – протокол передачи данных
- Очереди Laravel – асинхронная обработка данных

Интеграция с Mindbox:

- REST API – взаимодействие с CDP
- JSON – формат обмена данными
- Вебхуки – уведомления о событиях

Интеграция с Paykeeper:

- REST API – создание платежей
- Вебхуки – уведомления о статусе оплаты
- Подписи запросов – проверка подлинности

6.8. Контейнеризация (для разработки)

Среда разработки:

- Docker – контейнеризация приложения
- Laravel Sail – обертка над Docker для Laravel
- Docker Compose – оркестрация контейнеров

Контейнеры:

- PHP 8.4

- PostgreSQL 17
- Redis 7.x
- Nginx
- Node.js 20.x (для Nuxt)

6.9. Инструменты разработки и тестирования

Тестирование:

- PHPUnit – фреймворк для unit и feature тестов
- Pest – альтернативный тестовый фреймворк

Отладка:

- Laravel Telescope – мониторинг запросов, очередей, логов
- Xdebug – отладчик PHP

Документация API:

- Swagger/OpenAPI – документирование API

Контроль качества кода:

- PHP CS Fixer – автоматическое форматирование кода
- PHPStan – статический анализ кода

6.10. Система управления зависимостями

PHP пакеты:

- Composer – менеджер зависимостей для PHP

JavaScript пакеты:

- npm – менеджер пакетов для Node.js

6.11. Система контроля версий

Git – распределенная система контроля версий

6.12. Требования к серверному окружению

Минимальные требования:

- ОС: Linux (Ubuntu 22.04 LTS или выше)
- PHP 8.4 с расширениями: PDO, pgsql, redis, mbstring, xml, curl, bcmath, intl
- PostgreSQL 17
- Redis 7.x
- Nginx

- 4 ГБ оперативной памяти
- 20 ГБ свободного дискового пространства (SSD)
- 2 ядра процессора

Рекомендуемые требования:

- 8 ГБ оперативной памяти
- 4 ядра процессора
- 50 ГБ свободного дискового пространства (SSD)

---

7. БЕЗОПАСНОСТЬ АРХИТЕКТУРЫ

7.1. ОБЩИЕ ПРИНЦИПЫ БЕЗОПАСНОСТИ

ПО построено в соответствии со следующими принципами безопасности:

- Principle of Least Privilege (принцип наименьших привилегий)

- Defense in Depth (многоуровневая защита)

- Secure by Default (безопасность по умолчанию)

- Never Trust User Input (не доверять пользовательским данным)

7.2. УРОВНИ ЗАЩИТЫ

Уровень 1: Сетевой уровень

├─ HTTPS/TLS 1.2+ для всех соединений

├─ Брандмауэр (firewall) на уровне сервера

├─ DDoS защита (на уровне хостера)

├─ Rate limiting: 1000 запросов в минуту на пользователя

└─ Ограничение попыток входа: 3 неудачных попытки = блокировка на 15 минут

Уровень 2: Уровень приложения

├─ Аутентификация:

│ ├─ SMS-коды для веб-пользователей (6 цифр, TTL=5 минут)

│ ├─ Bearer токены для API (JWT, TTL=24 часа)

│ └─ Сессии: зашифрованы в Redis (HttpOnly, Secure флаги)

│

├─ Авторизация (RBAC):

│ ├─ Гость (guest): просмотр каталога

│ ├─ Пользователь (user): личный кабинет, заказы

│ └─ Администратор (admin): полный доступ

│

├─ Валидация входных данных:

│ ├─ Client-side: предварительная проверка в браузере

│ ├─ Server-side: обязательная проверка всех входных данных

│ ├─ Санитизация: очистка от XSS-кода

│ └─ Типизация: использование TypeScript на фронте, типа Laravel на бэке

│

├─ Защита от типовых атак:

│ ├─ SQL-инъекции: параметризованные запросы (ORM Eloquent)

│ ├─ XSS-атаки: экранирование выводимых данных, CSP заголовки

│ ├─ CSRF-атаки: CSRF-токены в формах, SameSite cookies

│ ├─ XXSS: отключение XML внешних сущностей

│ └─ Brute Force: ограничение попыток входа

│

└─ Content Security Policy:

  ├─ Только HTTPS для всех ресурсов

├─ Запрет на inline скрипты

├─ Запрет на eval()

└─ Запрет на доступ к локальному хранилищу (localStorage) для критичных данных

Уровень 3: Уровень базы данных

├─ Шифрование:

│ ├─ Пароли: bcrypt с salt factor 10

│ ├─ Чувствительные данные: AES-256 (опционально)

│ └─ Данные в покое: зависит от конфигурации хостера

│

├─ Права доступа:

│ ├─ Раздельные пользователи БД для приложения и администратора

│ ├─ Минимизация прав (только необходимые для работы)

│ └─ Изоляция по ролям

│

├─ Резервное копирование:

│ ├─ Ежедневное резервное копирование всех данных

│ ├─ Хранение копий: минимум 30 дней

│ └─ Тестирование восстановления: 1 раз в месяц

│

└─ Логирование и аудит:

├─ Все операции с заказами логируются

├─ Логирование попыток несанкционированного доступа

├─ Хранение логов: минимум 6 месяцев

└─ Инструмент: файловая система + логирование в Redis

7.3. УПРАВЛЕНИЕ КЛЮЧАМИ И СЕРТИФИКАТАМИ

SSL/TLS сертификаты:

├─ Провайдер: Let's Encrypt

├─ Тип: Domain Validation (DV)

├─ Период действия: 90 дней

├─ Автоматическое обновление: ДА (через Certbot)

└─ Версия TLS: 1.2+

API ключи и токены:

├─ Хранение: переменные окружения (.env файл)

├─ Шифрование в покое: через провайдера хостинга

├─ Ротация: каждые 6 месяцев

└─ Процедура: автоматическое создание новых, постепенное удаление старых

7.4. ЗАЩИТА ПЕРСОНАЛЬНЫХ ДАННЫХ

В соответствии с ФЗ-152 "О персональных данных":

├─ Согласие пользователя: получается при регистрации (чекбокс)

├─ Мета-данные о ПД:

│ ├─ Оператор ПД: ООО «Интенса»

│ ├─ Обработчик: (если используется облачный хостинг)

│ └─ Субъекты ПД: пользователи системы

│

├─ Категории ПД:

│ ├─ ФИО, номер телефона, адрес (обязательные)

│ ├─ Email (опциональная)

│ ├─ История покупок, IP-адрес (технические данные)

│ └─ Данные платёжных средств (обрабатывает Paykeeper, не хранится у нас)

│

├─ Права пользователей:

│ ├─ Право доступа: /api/v1/users/current (полные данные)

│ ├─ Право на удаление: запрос на удаление аккаунта

│ └─ Право на передачу: экспорт данных (формат JSON/CSV)

│

└─ Сроки хранения:

├─ Активный пользователь: бессрочно (пока использует систему)

├─ Неактивный пользователь: 1 год (затем анонимизация)

└─ Служебные логи: 6 месяцев

7.5. ИНТЕГРАЦИОННАЯ БЕЗОПАСНОСТЬ

Безопасность при взаимодействии с внешними системами:

1С Интеграция:

├─ Протокол: HTTPS (TLS 1.2+)

├─ Аутентификация: API ключ (в заголовке X-API-Key)

├─ Валидация: проверка подписи запроса (HMAC-SHA256)

├─ Rate limiting: 100 запросов в час

└─ Логирование: все запросы и ответы

Paykeeper (платежи):

├─ Протокол: REST API с подписями

├─ Аутентификация: API ключ + API пароль

├─ Валидация платежей: проверка подписи вебхука

├─ Хранение карт: НЕ хранятся, используется токенизация

├─ PCI-DSS: Соответствие (через Paykeeper)

└─ Rate limiting: в соответствии с документацией Paykeeper

Mindbox (CDP/лояльность):

├─ Протокол: REST API с OAuth 2.0

├─ Аутентификация: Bearer токен (TTL определяется Mindbox)

├─ Синхронизация: шифрованный HTTPS

├─ Данные: клиентские данные, не ПД третьих лиц

└─ Режим обновления: real-time через очереди

SMS-провайдер:

├─ Протокол: REST API

├─ Аутентификация: API ключ

├─ Шифрование: HTTPS

├─ Логирование: все отправки логируются (без текста, только статус)

└─ TTL кодов подтверждения: 5 минут

7.6. МОНИТОРИНГ БЕЗОПАСНОСТИ И ИНЦИДЕНТНОСТЬ

Мониторинг:

├─ Real-time мониторинг ошибок (через Sentry или аналог)

├─ Контроль доступа: логирование всех административных операций

├─ Проверка целостности данных: ежедневно

└─ Проверка на скомпрометированные зависимости: еженедельно (Snyk)

Процедура при инциденте безопасности:

├─ Обнаружение: автоматическое или ручное

├─ Изоляция: остановка скомпрометированного компонента

├─ Анализ: определение причины и масштаба

├─ Уведомление: пользователей (в течение 24 часов) и компетентных органов

├─ Исправление: патч и развертывание

└─ Восстановление: проверка целостности данных

7.7. ТЕСТИРОВАНИЕ БЕЗОПАСНОСТИ

├─ OWASP ZAP сканирование: ежемесячно

├─ Penetration testing: ежеквартально (внешний подрядчик)

├─ Проверка зависимостей: еженедельно (Snyk, Dependabot)

├─ Статический анализ кода: при каждом push (PHPStan, ESLint)

└─ Проверка паролей: HIBP (Have I Been Pwned) интеграция

7.8. СООТВЕТСТВИЕ СТАНДАРТАМ

└─ ГОСТ Р 34.201-2020 (Сертификация ПО)

└─ ГОСТ Р ИСО/МЭК 27001-2021 (Информационная безопасность)

└─ OWASP Top 10 (2021) - все критичные уязвимости исправлены

└─ RFC 6749 (OAuth 2.0)

└─ RFC 7231 (HTTP/1.1)

---

8. МАСШТАБИРУЕМОСТЬ И ПРОИЗВОДИТЕЛЬНОСТЬ АРХИТЕКТУРЫ

8.1. ОБЩИЕ ПРИНЦИПЫ МАСШТАБИРУЕМОСТИ

ПО построено с учетом следующих принципов:
├─ Stateless приложение: каждый экземпляр API независим
├─ Горизонтальная масштабируемость: добавление серверов
├─ Кэширование: многоуровневое кэширование данных
├─ Асинхронность: очереди для длительных операций
└─ CDN: доставка статического контента через сеть доставки

8.2. ГОРИЗОНТАЛЬНАЯ МАСШТАБИРУЕМОСТЬ

Архитектура поддерживает добавление новых серверов без изменения кода:

Frontend масштабирование (Nuxt):
├─ Развертывание: несколько instances за Load Balancer
├─ Состояние: хранится в cookie (отправляется с каждым запросом)
├─ Сессии: не требуются (stateless)
└─ Масштаб: поддерживает N инстансов

Backend API масштабирование (Laravel):
├─ Развертывание: несколько instances за Load Balancer (Nginx, HAProxy)
├─ Состояние: в Redis (shared session store)
├─ Сессии: Redis на отдельном сервере (master-slave репликация)
├─ База данных: read replicas для SELECT запросов
├─ Асинхронные задачи: распределённые очереди (Redis Queues)
└─ Масштаб: поддерживает N инстансов

Load Balancer:
├─ Способ распределения: Round Robin (циклическое)
├─ Health checks: каждые 10 секунд
├─ Sticky sessions: НЕ требуются (благодаря Redis сессиям)
├─ Failover: автоматический переход на резервный сервер
└─ Тип: Nginx с конфигурацией upstream

8.3. ВЕРТИКАЛЬНАЯ МАСШТАБИРУЕМОСТЬ

Увеличение ресурсов одного сервера:

Текущие требования (на 500 одновременных пользователей):
├─ CPU: 4 ядра
├─ RAM: 8 ГБ
├─ Disk: 40 ГБ (SSD)

└─ Сетевой канал: 100 Mbps

Расширенные требования (на 5000+ пользователей):
├─ CPU: 16+ ядер
├─ RAM: 64+ ГБ
├─ Disk: 500+ ГБ (SSD)
└─ Сетевой канал: 1 Gbps

8.4. КЭШИРОВАНИЕ И ОПТИМИЗАЦИЯ

Многоуровневое кэширование:

Уровень 1: Browser Cache (клиент)
├─ Время жизни: зависит от типа ресурса
├─ HTML страницы: max-age=3600 (1 час)
├─ CSS/JS: max-age=31536000 (1 год, с версионированием)
├─ Изображения: max-age=2592000 (30 дней)
└─ API ответы: Cache-Control: no-cache (требуется валидация)

Уровень 2: CDN Cache (глобальная сеть)
├─ Провайдер: CloudFlare, Akamai или аналог
├─ Типы данных: статические файлы (CSS, JS, изображения)
├─ TTL: 1 день для медленно меняющегося контента
└─ Инвалидация: purge по требованию при обновлении

Уровень 3: Server-side Cache (Redis)
├─ Кэширование данных:
│ ├─ Каталог товаров: key="products:all" TTL=1 час
│ ├─ Категории: key="categories:{id}" TTL=2 часа
│ ├─ Цены: key="prices:offer:{id}" TTL=30 минут
│ └─ Профиль пользователя: key="user:{id}:profile" TTL=1 час
│
├─ Кэширование запросов:
│ ├─ SELECT запросы к БД: кэшируются на 5-30 минут
│ ├─ Инвалидация: при UPDATE/DELETE/INSERT
│ └─ Key strategy: query hash (детерминированное преобразование)
│
├─ Сессии:
│ ├─ Хранилище: Redis (session store)
│ ├─ TTL: 2 часа неактивности
│ └─ Ротация: новый session ID при усиленной аутентификации
│
└─ Rate limiting:
   ├─ Per IP: 1000 requests/min
   ├─ Per user: 5000 requests/min
   └─ Per API endpoint: специфичные лимиты (например, cart: 100 req/min)

8.5. АСИНХРОННАЯ ОБРАБОТКА

Длительные операции выполняются в фоне через очереди:

Queue-based задачи (Laravel Queue + Redis):

1. Email рассылки
   ├─ Сценарий: уведомление о новом заказе
   ├─ Время выполнения: может быть до 5 сек (SMTP задержка)
   ├─ Queue: mail
   ├─ Workers: 2-4 процесса
   └─ Retry: 3 попытки при ошибке

2. Синхронизация с 1С
   ├─ Сценарий: импорт каталога (может быть 10k+ товаров)
   ├─ Время выполнения: 5-30 минут
   ├─ Queue: exchange
   ├─ Workers: 1-2 процесса
   ├─ Priority: 2 (высокий приоритет)
   └─ Batching: обработка порциями по 500 товаров

3. Генерация отчётов
   ├─ Сценарий: экспорт заказов в PDF/Excel
   ├─ Время выполнения: 10-60 сек
   ├─ Queue: reports
   ├─ Workers: 1 процесс
   └─ Уведомление: email с ссылкой на скачивание

4. Вебхуки и уведомления
   ├─ Сценарий: уведомление Mindbox о покупке
   ├─ Время выполнения: 1-3 сек
   ├─ Queue: notifications
   ├─ Workers: 4-8 процессов
   ├─ Retry: 5 попыток с экспоненциальной задержкой
   └─ Timeout: 30 сек на операцию

8.6. БАЗА ДАННЫХ МАСШТАБИРОВАНИЕ

PostgreSQL архитектура:

Master-Slave репликация:
├─ Master: основной сервер (write operations)
├─ Slave replicas: 1-3 реплики (read operations)
├─ Latency: <100ms (синхронная репликация)
├─ Failover: автоматический переход на новый master
└─ Инструмент: patroni или аналог

Read replicas:
├─ Распределение READ запросов:

│ ├─ SELECT товаров: на read replica (non-critical data)
│ ├─ SELECT пользователей: на master (must-have actual data)
│ └─ SELECT заказов: на master (critical data)
│
├─ Load balancing: вручную в коде ORM (Eloquent replica())
└─ Fallback: автоматический fallback на master если реплика недоступна

Индексирование:
├─ Индексы по всем JOIN колонкам
├─ Full-text индексы для поиска товаров
├─ Partial индексы для фильтров (is_active = true)
├─ Регулярная реанализация: ANALYZE (автоматически)
└─ Мониторинг slow queries: слог запросов дольше 1 сек

Шардирование (будущее):
├─ По user_id для данных пользователей (если масштаб >1M юзеров)
├─ По product_id для каталога (если масштаб >100k товаров)
└─ Миграция: подготовлена архитектура, но не требуется сейчас

8.7. МЕТРИКИ ПРОИЗВОДИТЕЛЬНОСТИ (ЦЕЛЕВЫЕ ЗНАЧЕНИЯ)

При нормальной нагрузке (500 одновременных пользователей):

API Response Time:
├─ p50 (медиана): 100-150 мс
├─ p95: <500 мс
├─ p99: <1 сек
└─ max: <3 сек

Пропускная способность:
├─ Requests per second (RPS): 100+ RPS на одном инстансе
├─ Bandwidth: <100 Mbps при среднем запросе 10 KB
└─ Concurrent connections: 5000+

Database Performance:
├─ Query time (p95): <50 мс
├─ Connection pool: 20 connections от приложения
├─ Slow query threshold: > 1 сек логируется
└─ Lock wait time: <100 мс

Memory usage:
├─ Laravel instance: 100-150 MB per worker
├─ Redis: <2 GB (зависит от объёма кэша)
├─ PostgreSQL: <4 GB buffer pool

Disk I/O:
├─ Read latency: <5 мс (SSD)
├─ Write latency: <10 мс (SSD)

└─ IOPS: >3000 IOPS (случайный доступ)

8.8. SCALING ROADMAP

Текущая стадия (0-6 месяцев):
├─ Нагрузка: 500 одновременных пользователей
├─ 1 instan Laravel + 1 instan Nuxt + 1 инстан БД
└─ Кэширование через Redis

Средняя стадия (6-12 месяцев):
├─ Нагрузка: 5000+ одновременных пользователей
├─ 4-8 instances Laravel за load balancer
├─ 2 instances Nuxt за CDN
├─ Master-slave PostgreSQL с read replicas
├─ Distributed Redis cluster
└─ Асинхронные очереди для длительных операций

Большая стадия (12+ месяцев):
├─ Нагрузка: 50000+ одновременных пользователей
├─ Микросервисная архитектура
├─ Шардирование БД (по user_id, product_id)
├─ Kubernetes для оркестрации
└─ Мультирегиональное размещение

9. ОТКАЗОУСТОЙЧИВОСТЬ И ВОССТАНОВЛЕНИЕ

9.1. ЦЕЛЕВЫЕ ПОКАЗАТЕЛИ НАДЕЖНОСТИ

RTO (Recovery Time Objective):
├─ Критичные сервисы (заказы, платежи): 15 минут
├─ Стандартные сервисы (каталог, поиск): 1 час
└─ Некритичные сервисы (админ-панель, отчёты): 4 часа

RPO (Recovery Point Objective):
├─ Финансовые данные: 1 час
├─ Клиентские данные: 1 час
└─ Товары и каталог: 4 часа

SLA (Service Level Agreement):
├─ Uptime: 99.5% (5 часов простоя в месяц)
├─ Плановое обслуживание: макимум 4 часа в месяц (в нерабочее время)
└─ Неплановые простои: минимизация через redundancy

9.2. АРХИТЕКТУРА ОТКАЗОУСТОЙЧИВОСТИ

Redundancy по компонентам:

Frontend (Nuxt):
├─ Статус: Single Point of Failure (должно быть 2+ инстанса)
├─ Решение: за Load Balancer (Nginx, HAProxy)
├─ Health check: HTTP GET /health каждые 10 сек
├─ Failover: автоматический (<10 сек обнаружение)
└─ Data loss: нет (stateless)

Backend API (Laravel):
├─ Статус: Single Point of Failure (должно быть 4+ инстанса)
├─ Решение: за Load Balancer
├─ Health check: GET /api/v1/health каждые 5 сек
├─ Failover: автоматический (<5 сек)
└─ Data loss: нет (stateless, данные в Redis/БД)

Database (PostgreSQL):
├─ Статус: Single Point of Failure (master-slave репликация)
├─ Решение: Master-Slave с автоматическим failover
├─ Failover инструмент: Patroni, etcd
├─ Failover время: 10-30 сек (automatic)
├─ Data loss: нет (синхронная репликация)
└─ Monitoring: Prometheus + Alertmanager

Cache (Redis):
├─ Статус: Может быть Single Point of Failure
├─ Решение: Redis Sentinel или Redis Cluster
├─ Failover: автоматический (10-30 сек)
├─ Data loss: допустимо (только кэш, не критичные данные)
└─ Recovery: автоматическое заполнение кэша

9.3. СЦЕНАРИИ ОТКАЗА И ВОССТАНОВЛЕНИЕ

Сценарий 1: Отказ одного инстанса Backend
├─ Обнаружение: health check не отвечает (5 сек)
├─ Действие: Load Balancer убирает instan из пула
├─ Impact: остальные инстансы берут нагрузку (+20-30%)
├─ Восстановление: автоматический перезапуск (systemd/Docker)
├─ RTO: <1 минуты
└─ Data loss: нет

Сценарий 2: Отказ главной БД (Master)
├─ Обнаружение: потеря соединения (1-3 сек)
├─ Action: Patroni инициирует failover на Slave
├─ Steps:
│ 1. Выбор новой master (на основе LSN)
│ 2. Реконфигурация остальных slave
│ 3. Перенаправление приложения на новую master
├─ RTO: 10-30 сек
├─ Data loss: нет (синхронная репликация)

└─ Manual intervention: нет (автоматический failover)

Сценарий 3: Отказ Redis (Cache)
├─ Обнаружение: connection timeout (1 сек)
├─ Impact: кэш недоступен, повышение нагрузки на БД
├─ Fallback: прямые запросы в PostgreSQL (медленнее на 50%)
├─ Производительность: деградирует, но работает
├─ Восстановление: перезапуск Redis, перестроение кэша
├─ RTO: 1-2 минуты (кэш загружается постепенно)
└─ Data loss: нет (кэш не содержит критичные данные)

Сценарий 4: Потеря сетевого соединения с 1С
├─ Обнаружение: connection timeout (10 сек)
├─ Action: очередь задач сохраняет попытку в Redis
├─ Retry: экспоненциальная задержка (1, 2, 4, 8, 16 минут)
├─ MaxRetries: 5 (итого ~31 минуты попыток)
├─ Уведомление: alert администратору при 3-й попытке
├─ RTO: данные восстановятся при восстановлении соединения
└─ Data loss: нет (в очереди сохранены данные)

Сценарий 5: Отказ платежной системы Paykeeper
├─ Обнаружение: при попытке создать платеж
├─ Action: пользователю предлагается попробовать позже
├─ Fallback: предложение "оплаты при получении"
├─ Data: заказ сохраняется как "ожидание оплаты"
├─ Retry: автоматический retry каждые 5 минут (фон)
├─ Notification: email уведомление об ошибке оплаты
└─ RTO: платеж пройдёт при восстановлении

9.4. РЕЗЕРВНОЕ КОПИРОВАНИЕ

Стратегия резервного копирования:

PostgreSQL:
├─ Частота: ежедневное (в 03:00 UTC)
├─ Тип: полная копия + WAL архивирование
├─ Хранилище: S3 облако + локальный NAS
├─ Retention: 30 дней полных копий, 7 дней WAL архивов
├─ Тестирование: восстановление раз в неделю на отдельном сервере
├─ Recovery time: <1 часа (с использованием WAL)
└─ Point-in-time recovery: доступна (любой момент в прошлые 7 дней)

Redis:
├─ Частота: RDB снимки каждый час + AOF логирование
├─ Тип: снимок состояния (RDB) + логирование операций (AOF)
├─ Хранилище: локальное + S3 для архива
├─ Retention: последние 7 дней
├─ Recovery: полное восстановление из RDB + replay AOF

└─ Time: <5 минут

Файлы приложения (исходный код, конфиги):
├─ Хранилище: Git (GitLab) с remote backups
├─ Частота: при каждом commit (автоматический)
├─ Retention: вся история (неограниченно)
└─ Recovery: git clone любой версии

Медиа-файлы (изображения товаров):
├─ Хранилище: S3 облако с версионированием
├─ Backup: replicate в secondary S3 регион
├─ Retention: вся история (неограниченно)
├─ Recovery: restore из secondary региона
└─ Monitoring: CloudWatch alerts

9.5. МОНИТОРИНГ И ОПОВЕЩЕНИЕ

Инструменты мониторинга:
├─ Prometheus: сбор метрик (CPU, memory, disk, network)
├─ Grafana: визуализация метрик и создание дашбордов
├─ AlertManager: управление оповещениями
├─ ELK Stack: логирование и анализ логов
└─ Sentry: мониторинг ошибок приложения

Ключевые метрики для мониторинга:

Система:
├─ CPU utilization: alert если >80% на 5 минут
├─ Memory usage: alert если >85% на 5 минут
├─ Disk space: alert если <10% свободного места
├─ Network latency: alert если >100 мс
└─ Network errors: alert если rate >1/min

Приложение:
├─ Request rate: baseline и anomaly detection
├─ Error rate: alert если >1% 4xx/5xx ошибок
├─ Response time p95: alert если >500 мс
├─ Exception rate: alert при любом Exception
└─ Active connections: alert если >500 соединений

База данных:
├─ Query time p95: alert если >100 мс
├─ Replication lag: alert если >1 сек (master-slave)
├─ Connection pool: alert если >15 из 20 connections
├─ Slow queries: логирование запросов дольше 1 сек
└─ Disk usage: alert если >80%

Оповещение каналы:

├─ Severity P1 (Critical): SMS + Slack + Phone call
├─ Severity P2 (Warning): Slack + Email
├─ Severity P3 (Info): Email + Slack
└─ Escalation: если не ответили за 15 минут -> call CTO

9.6. DISASTER RECOVERY PLAN (DRP)

План восстановления при техногенной катастрофе:

Уровень 1 (отказ одного компонента):
├─ Действие: автоматический failover (см раздел 9.3)
├─ RTO: <5 минут
└─ Документация: процесс автоматизирован

Уровень 2 (отказ всей инфраструктуры датацентра):
├─ Действие: переключение на secondary datacenter
├─ RTO: 30 минут (потребуется ручное переключение DNS)
├─ Требования: горячий резервный datacenter (в другом регионе)
├─ Infrastructure as Code: всё в Terraform для быстрого развёртывания
└─ Тестирование: DR drill квартально

Уровень 3 (полная потеря всех данных):
├─ Восстановление: из S3 резервных копий (может занять часы)
├─ RTO: 4-6 часов
├─ Data loss: максимум 1 час (последняя резервная копия)
└─ Manual process: требуется вмешательство DBA

9.7. ДОКУМЕНТИРОВАНИЕ ИНЦИДЕНТОВ

После каждого инцидента:
├─ Заполняется incident report (шаблон)
├─ Root cause analysis: определяется причина
├─ Timeline: фиксируется время обнаружения и восстановления
├─ Action items: создаются задачи на предотвращение
├─ Post-mortem meeting: обсуждение в команде
└─ Update runbook: обновление документации процедур

Хранение инцидентов:
├─ Система: Jira / GitLab Issues
├─ Отчётность: ежемесячный отчёт stakeholders
└─ Trend analysis: поиск паттернов и систематических проблем

10. РАЗВЁРТЫВАНИЕ И DEVOPS ПРОЦЕССЫ

10.1. CI/CD PIPELINE

├─ Trigger: push в main branch

├─ Steps:
│ 1. Unit tests (PHPUnit, Jest) - 5 минут
│ 2. Integration tests - 10 минут
│ 3. Static analysis (PHPStan, ESLint) - 3 минуты
│ 4. Security scanning (OWASP ZAP) - 5 минут
│ 5. Build artifacts (Docker image) - 5 минут
│ 6. Push to registry (GitLab Container Registry) - 2 минуты
│ 7. Deploy to staging - 5 минут
│ 8. Smoke tests (проверка критичных функций) - 2 минуты
│
└─ Total time: ~37 минут

Deploy в production:
├─ Trigger: manual approval (при условии успеха staging)
├─ Strategy: Blue-Green deployment (0 downtime)
├─ Steps:
│ 1. Spin up new instances (зелёные) - 5 минут
│ 2. Run migrations - 2-10 минут (зависит от размера БД)
│ 3. Health checks - 1 минута
│ 4. Switch traffic (синие -> зелёные) - 10 сек
│ 5. Старые инстансы (синие) выключаются - 1 минута
│
└─ Rollback: простое переключение обратно (10 сек)

10.2. ВЕРСИОНИРОВАНИЕ И РЕЛИЗЫ

Semantic Versioning: MAJOR.MINOR.PATCH
├─ MAJOR: несовместимые изменения API
├─ MINOR: добавление новой функциональности
└─ PATCH: исправление багов

Release process:
├─ Git tag: v1.0.0
├─ Changelog: обновляется автоматически (commitizen)
├─ Release notes: публикуются в GitLab Releases
└─ Artifact: Docker image tagged v1.0.0

10.3. МОНИТОРИНГ РАЗВЁРТЫВАНИЯ

Post-deploy checks:
├─ API health check: GET /api/v1/health
├─ Database connectivity: ping PostgreSQL
├─ Redis connectivity: ping Redis
├─ External services: проверка интеграций (1C, Mindbox, Paykeeper)
├─ Error rate: должна быть <0.1%
└─ Response time: p95 <500 мс

Automated rollback triggers:

├─ Error rate >5% в течение 5 минут
├─ Response time p95 >2 sec в течение 5 минут
├─ Database replication lag >10 sec
└─ Memory usage >90%

11. СТРАТЕГИЯ ТЕСТИРОВАНИЯ АРХИТЕКТУРЫ

11.1. УРОВНИ ТЕСТИРОВАНИЯ

Unit Tests (PHPUnit, Jest):
├─ Покрытие: 100% критичного кода
├─ Примеры:
│ ├─ Test для ProductRepository::getById()
│ ├─ Test для CartService::addToCart()
│ └─ Test для PaymentProcessor::processPayment()
├─ Время выполнения: <5 минут для 2000+ тестов
└─ Инструменты: PHPUnit, Jest, Mockito

Integration Tests (Pest, E2E):
├─ Покрытие: взаимодействие модулей
├─ Примеры:
│ ├─ API -> БД (добавление товара)
│ ├─ API -> внешние системы (интеграция с 1С)
│ └─ Frontend -> Backend (оформление заказа)
├─ Время выполнения: <15 минут
└─ Database: в-памяти (SQLite для тестов)

System/End-to-End Tests (Cypress, Selenium):
├─ Покрытие: полные пользовательские сценарии
├─ Примеры:
│ ├─ Пользователь регистрируется -> просматривает каталог -> оформляет заказ
│ ├─ Администратор добавляет товар через админ-панель
│ └─ Синхронизация данных с 1С
├─ Время выполнения: <30 минут
└─ Инструменты: Cypress, Selenium

Performance Tests (JMeter, K6):
├─ Сценарии:
│ ├─ Нагрузка: 500 одновременных пользователей
│ ├─ Длительность: 10 минут
│ └─ Ramp-up: 100 пользователей в минуту
├─ Целевые метрики:
│ ├─ p95 response time: <500 мс
│ ├─ Throughput: >100 RPS
│ └─ Error rate: <0.1%
└─ Частота: еженедельно перед производством

Security Tests (OWASP ZAP, Burp):
├─ Сканирование:
│ ├─ SQL-инъекции
│ ├─ XSS-уязвимости
│ ├─ CSRF-защита
│ └─ Неверная конфигурация безопасности
├─ Частота: ежемесячно
└─ Результаты: обязательное исправление критичных

Load Tests (K6, Gatling):
├─ Сценарии:
│ ├─ Normal load: 500 юзеров
│ ├─ Peak load: 2000 юзеров (spike тест)
│ └─ Soak test: 100 юзеров в течение 24 часов
├─ Мониторинг:
│ ├─ Resource usage (CPU, memory, disk)
│ ├─ Database performance (query time)
│ └─ Network bandwidth
└─ Результаты: документируются, тренды отслеживаются

11.2. CONTINUOUS TESTING

├─ Unit tests: при каждом commit (pre-commit hook)
├─ Integration tests: при каждом push (CI pipeline)
├─ Static analysis: при каждом push (CI pipeline)
├─ Security scan: при каждом merge request
├─ Performance baseline: еженедельно
└─ Full regression suite: перед каждым production deployment

11.3. ТЕСТИРОВАНИЕ ОТКАЗОУСТОЙЧИВОСТИ

Chaos Engineering тесты:
├─ Отключение PostgreSQL: система должна быть недоступна gracefully
├─ Отключение Redis: кэш теряется, но приложение работает
├─ Отключение одного API инстанса: другие берут нагрузку
├─ Сеть latency: +100ms задержка
└─ Packet loss: 1% потерь пакетов

Проверка восстановления:
├─ Перезагрузка сервера: приложение должно перезапуститься автоматически
├─ Перезагрузка БД: данные должны быть целостными
├─ Восстановление из резервной копии: RTO <1 часа
└─ Failover master-slave: <30 сек, без потерь данных

12. ОБЕСПЕЧЕНИЕ КАЧЕСТВА КОДА И АРХИТЕКТУРЫ

12.1. CODE QUALITY METRICS

Инструменты анализа:
├─ PHPStan (level 9): статический анализ PHP
├─ Psalm: type checker для PHP
├─ SonarQube: комплексный анализ качества
├─ Codecov: покрытие unit tests
└─ Dependabot: обновление зависимостей

Целевые метрики:
├─ Code coverage: >80% для critical path
├─ Cyclomatic complexity: <10 для методов
├─ Duplicate code: <3%
├─ Technical debt: <5%
└─ CRAP index: <30

12.2. АРХИТЕКТУРНЫЕ ПРОВЕРКИ

Запреты (enforced by code):
├─ Circular dependencies: запрещены
├─ Direct database access вне Repository: запрещено
├─ Бизнес-логика в контроллерах: запрещена
├─ Global состояние: запрещено (кроме DI контейнера)
└─ Hardcoded конфигурация: запрещена

Проверка слоёв:
├─ Controllers -> Services -> Repositories
├─ Services не должны зависеть от Controllers
├─ Models не должны содержать бизнес-логику
└─ Database queries только в Repositories

Инструменты проверки:
├─ deptrac: анализ зависимостей между слоями
├─ php-architecture-tester: проверка архитектурных правил
└─ Code review: обязательный для каждого MR

12.3. CODE REVIEW ПРОЦЕСС

Требования для merge:
├─ ✅ Все тесты passed
├─ ✅ Code coverage не уменьшилось
├─ ✅ Static analysis passed (PHPStan level 9)
├─ ✅ Security scan passed (no critical/high)
├─ ✅ Одобрение от минимум 2 разработчиков
├─ ✅ Lead разработчик одобрил архитектурные изменения
└─ ✅ Changelog обновлён

Review criteria:
├─ Функциональность: соответствует requirement

├─ Тестовое покрытие: новый код протестирован
├─ Производительность: нет регрессии
├─ Безопасность: нет новых уязвимостей
├─ Читаемость: код понятен без комментариев
└─ Архитектура: соответствует принципам