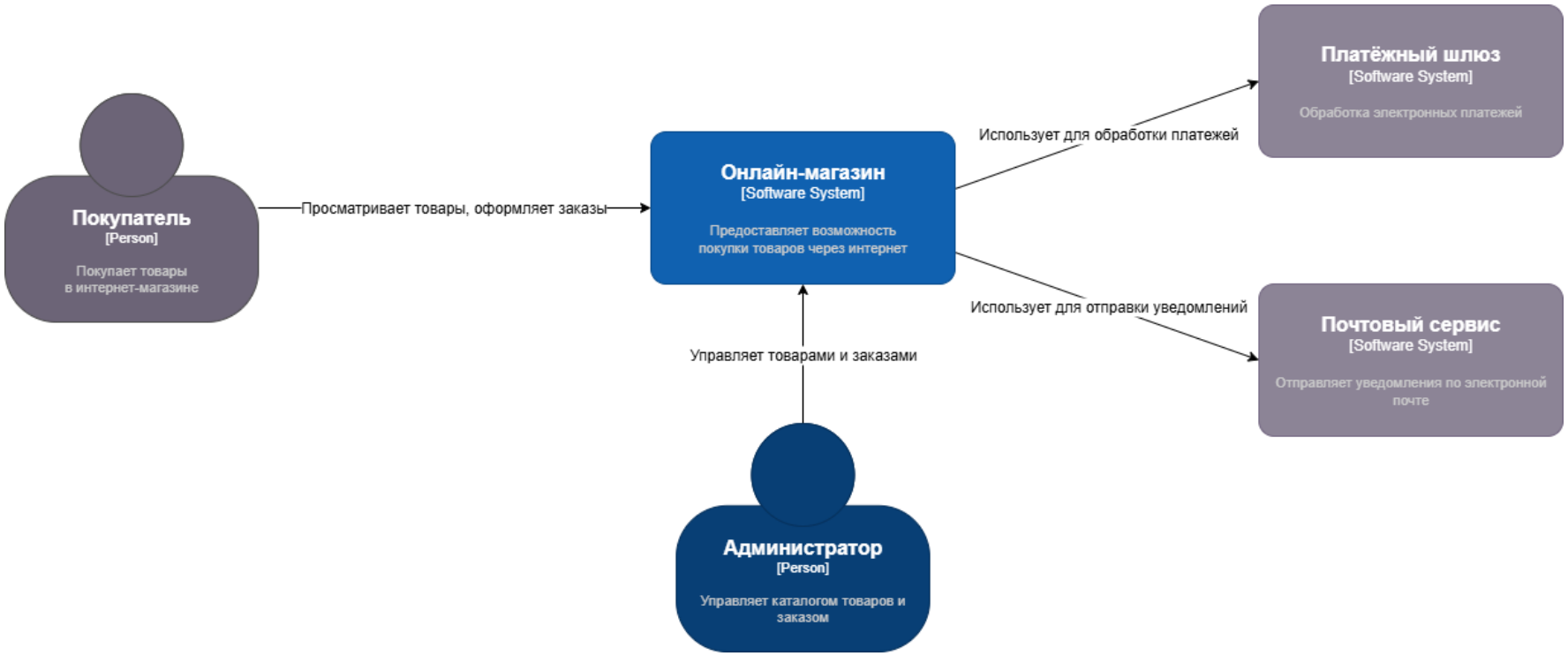
Архитектура REST API онлайн магазина

Автор: *Горохов Семён*

C4. Системный контекст

Цели проекта: высокая производительность и отзывчивость, горизонтальная масштабируемость, чистая архитектура, полная RESTful спецификация.

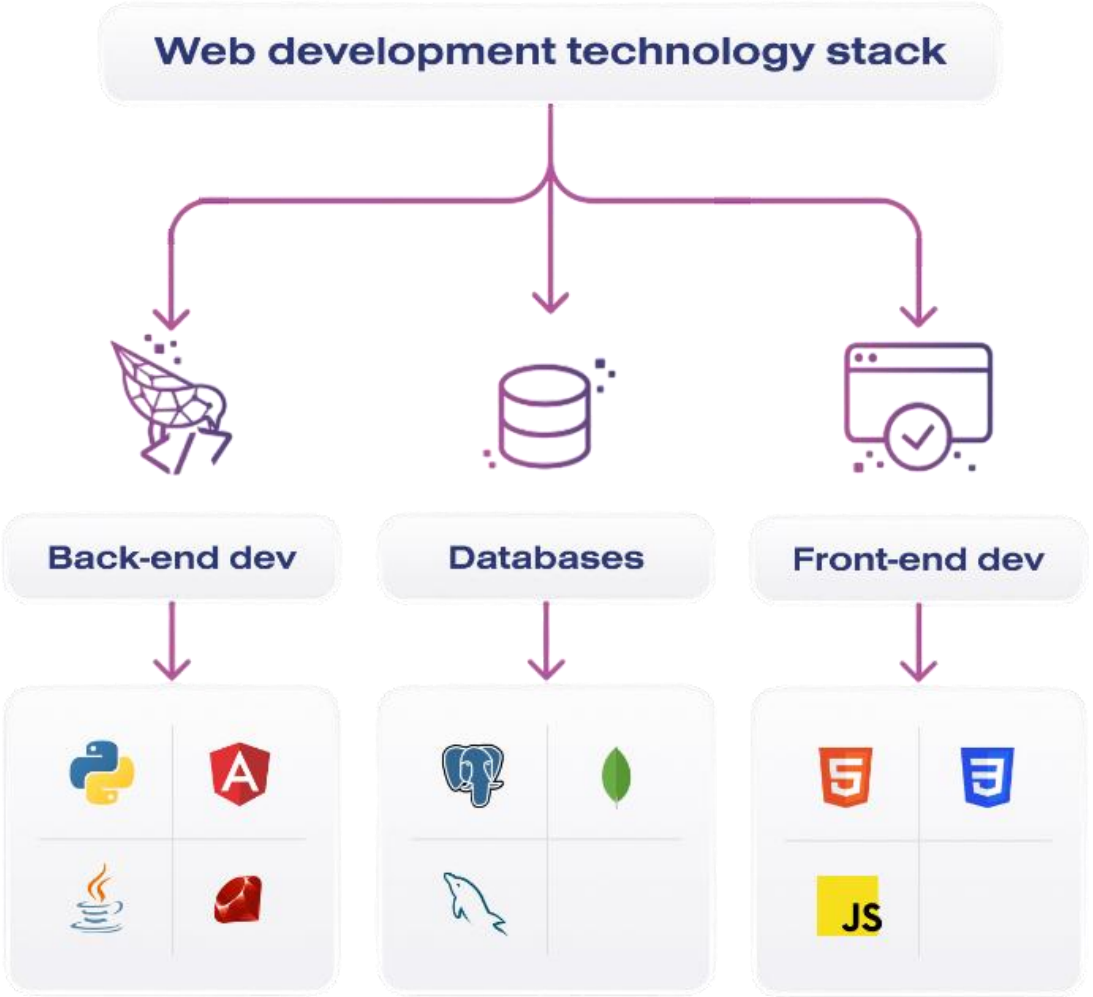
Архитектура: Многослойная (Layered Architecture).
Уровни детализации: C4 Model.



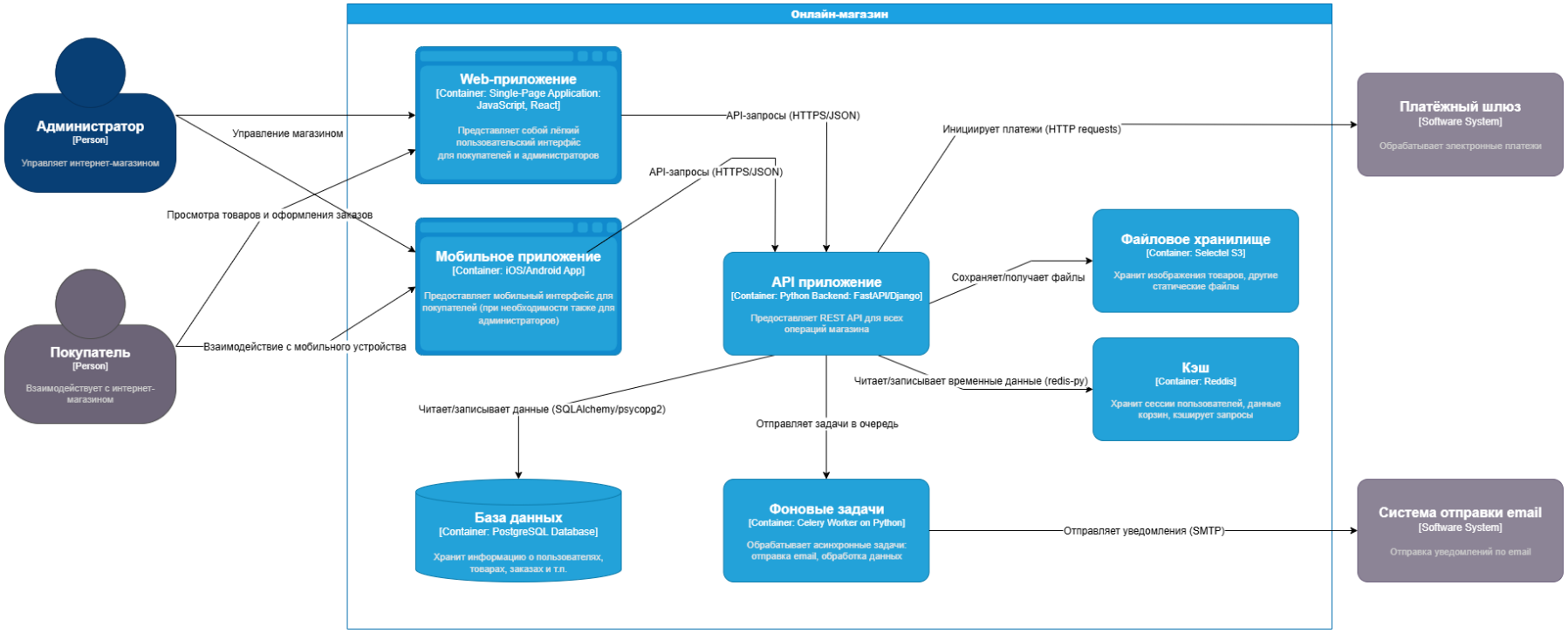
Технологический стек

Backend: Python 3. 11+ FastAPI.
Database: PostgreSQL + SQLAlchemy ORM.
Cache: Redis (сессии, корзины, кэш).
Docs: OpenAPI 3.0 (Swagger).
Auth: JWT tokens.

Развертывание решения осуществляется через контейнеры Docker с помощью Kubernetes для оркестрации.



C4. Уровень контейнеров



Покупатели (основные пользователи) — действия: просмотр товаров, управление корзиной, оформление заказов.

Администраторы (управление системой) — действия: CRUD товаров, управление заказами, аналитика.

Внешние системы: платёжный шлюз (critical dependency), email сервис (async communication).

Требования к REST API онлайн-магазина

Функциональные требования

Бизнес-требования:
- Полный CRUD для товаров, заказов, пользователей
- Автоматизация бизнес-процессов через API
- Валидация данных и контроль доступа (JWT)
- Разделение прав: пользователи → заказы, админы → товары

Клиентское приложение:
- Просмотр товаров с пагинацией и фильтрацией
- Управление корзиной покупок
- Создание и отслеживание заказов
- Регистрация и аутентификация пользователей

Административная система:
- Полное управление товарным каталогом (CRUD)
- Управление статусами заказов и пользователями
- Аналитика продаж
- Модерация пользователей

Ключевые эндпоинты:
- Товары: GET/POST/PUT/DELETE /api/products
- Заказы: GET/POST/PUT/DELETE /api/orders

Нефункциональные требования

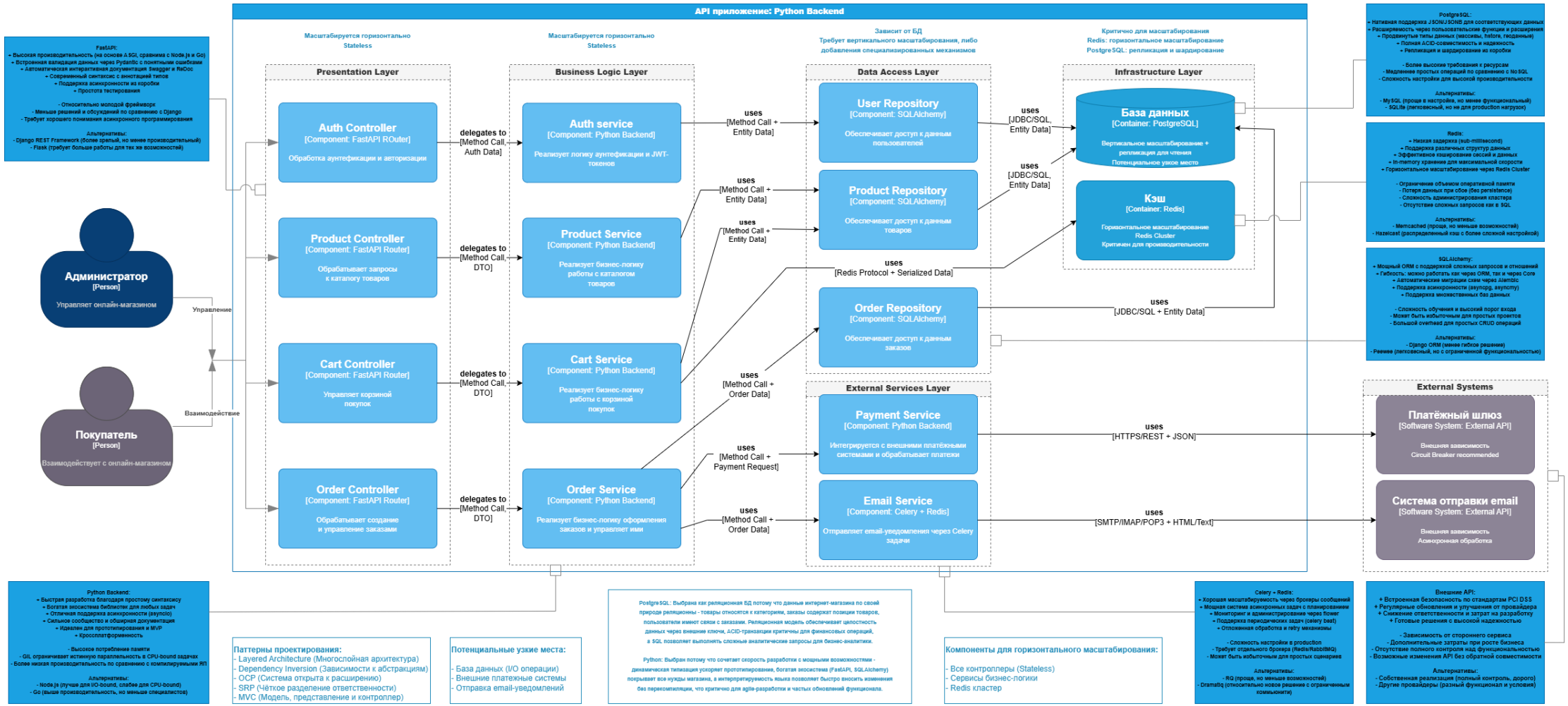
Производительность:
Время ответа:
- Операции чтения: ≤ 50 мс
- Создание/обновление: ≤ 100 мс
- Транзакции (заказы): ≤ 200 мс

Нагрузка:
- Эндпоинты чтения: 1000 RPS
- Эндпоинты записи: 100 RPS

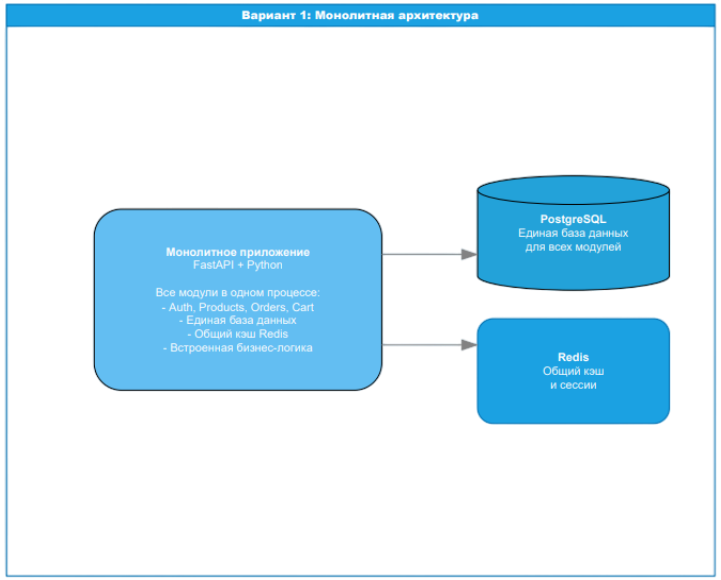
Надежность:
- Атомарность транзакций (PostgreSQL)
- Целостность данных (ссылочная целостность на уровне API)
- Откат при ошибках в сложных операциях

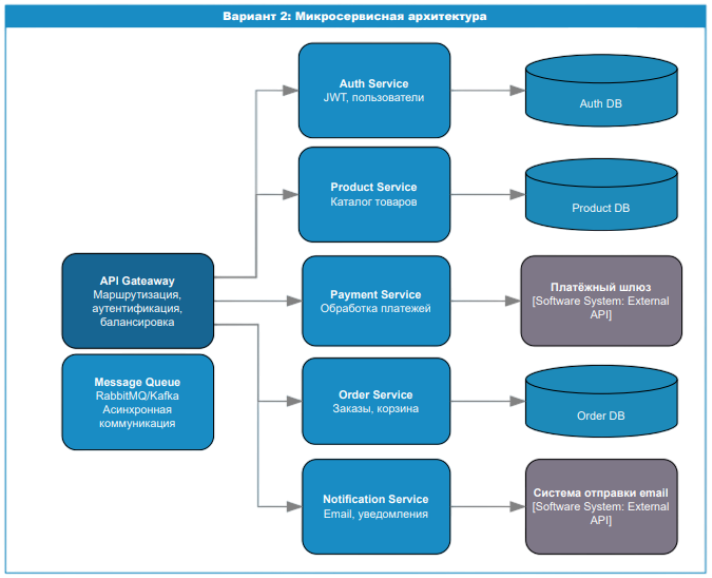
Архитектура:
- RESTful стандарты (HTTP методы, коды состояния)
- Валидация входных данных для всех операций
- Интеграция с клиентскими приложениями

C4. Уровень компонентов



Варианты развёртывания





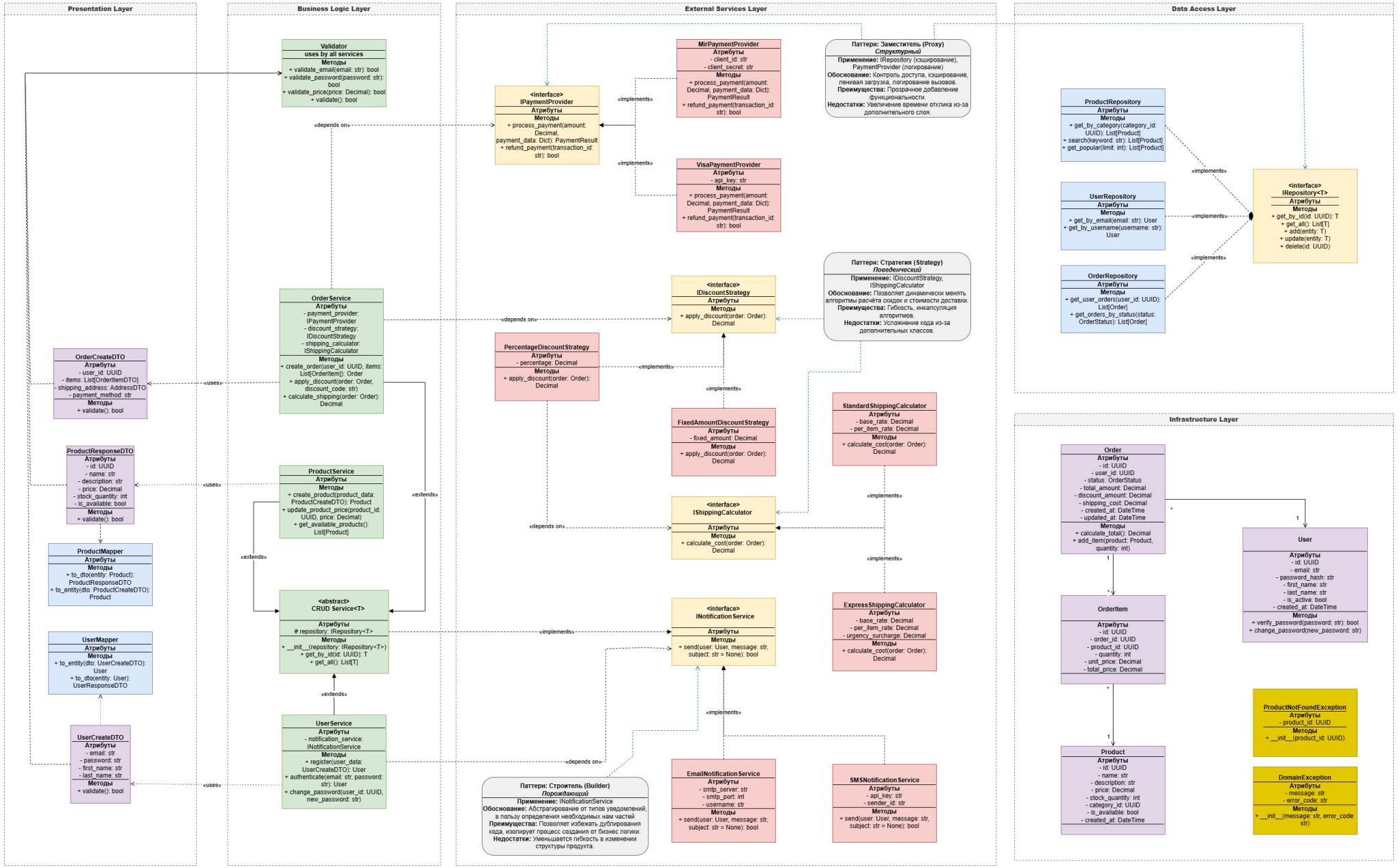
- ✅ Простая отладка
- ✅ Единая кодовая база
- ✅ Простота разработки и деплоя
- ✅ Минимальные накладные расходы

- ⛔ Зависимость технологического стека
- ⛔ Сложность внесения изменений
- ⛔ Сложность масштабирования
- ⛔ Единая точка отказа

- ✅ Разные технологии
- ✅ Гибкость разработки
- ✅ Устойчивость к отказам
- ✅ Независимое масштабирование

- ⛔ Проблема согласованности данных
- ⛔ Накладные расходы сети
- ⛔ Сложность оркестрации
- ⛔ Сложность отладки

Масштаб	CPU	RAM	Storage	Монолит	Микросервисы
10^3 пользователей	2-4 ядра	4-8 GB	50-100 GB	✓ Экономия	✗ Избыточно
10^5 пользователей	8-16 ядер	16-32 GB	500 GB - 1 TB	~ Окей	✓ Оптимально
10^6 пользователей	32-64+ ядер	64-128+ GB	2-5+TB	✗ Сложно	✓ Идеально



C4. Уровень классов

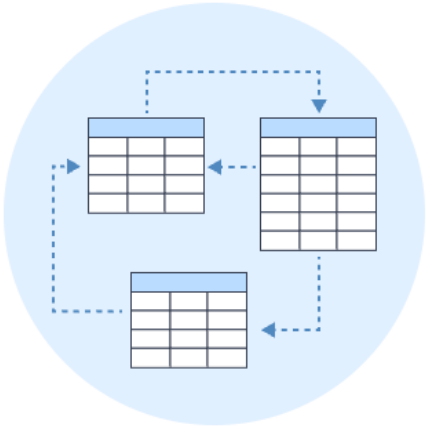
Модели данных и репозитории

SQLAlchemy models:

- *User*: представляет пользователя системы.
  - Поля: id, email, хэшированный пароль, роль, дата создания и обновления.
- *Product*: представляет товар в магазине.
  - Поля: id, название, описание, цена, количество на складе, категория, активен ли товар, дата создания.
- *Order*: представляет заказ, сделанный пользователем.
  - Поля: id, пользователь (связь с User), статус заказа, общая сумма, адрес доставки, статус оплаты, дата создания.
- *OrderItem*: представляет позицию в заказе.
  - Поля: id, заказ (связь с Order), товар (связь с Product), количество, цена за единицу, общая цена за позицию.

Репозитории:

- *ProductRepository*: отвечает за операции с товарами (поиск, фильтрация, обновление количества на складе)
- *OrderRepository*: отвечает за операции с заказами (создание заказа, получение заказов пользователя, обновление статуса)
- *CartRepository*: отвечает за операции с корзиной (добавление товаров, удаление, получение корзины)



Безопасность и аутентификация

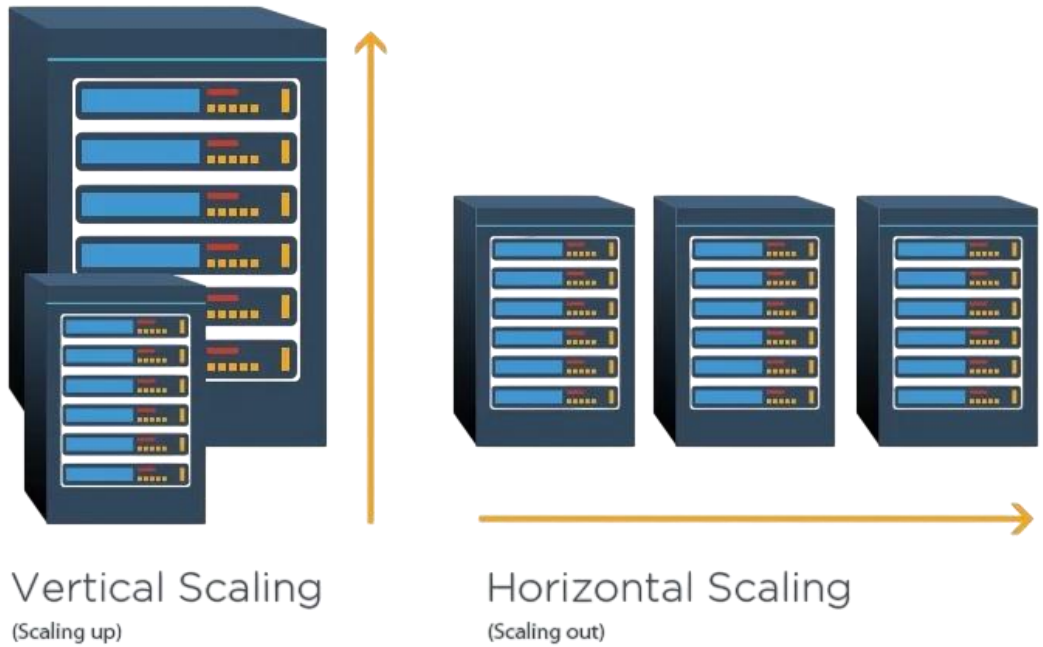
JWT аутентификация: структура токенов, security features.

Защита endpoints: Rate Limiting, Data Validation, Input Sanitization.

Авторизация: ролевая модель, permission system.

Внедрение надлежащих механизмов обработки ошибок имеет решающее значение для поддержания безопасности и предоставления значимой обратной связи пользователям. Мониторинг и регистрация подозрительных действий помогают выявлять и предотвращать потенциальные нарушения безопасности.

Масштабирование



Горизонтальное масштабирование:

- Stateless компоненты: API серверы, Celery workers
- Балансировка нагрузки

Вертикальное масштабирование:

- Stateful компоненты: PostgreSQL (RAM, CPU, SSD), Redis (RAM)
- Критические сервисы: где горизонтальное масштабирование невозможно или неэффективно
    - Монолит: преимущественно вертикальное масштабирование
    - Микросервисы: комбинированный подход

Масштабирование базы данных

- PostgreSQL оптимизации: read replicas, индексы, шардирование
- Connection pooling: PgBouncer, управление соединениями
- Гибридный подход: вертикальное для записи + горизонтальное для чтения

Спасибо за внимание!