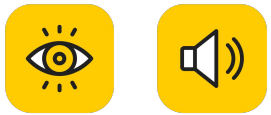
Microservice Architecture

Выпускной проект

otus.ru

Меня хорошо видно & слышно?



Защита проекта
Тема: Интернет-магазин

Каперусов Сергей

Инженер-Программист
ООО “СИСТЕМА”

План защиты

- Описание проекта
- Используемые технологии и паттерны
- Что получилось: схемы/архитектура
- Демонстрация проекта
- Выводы

Общее описание проекта

Проект представляет собой учебное приложение в формате интернет-магазина, реализованное на основе микросервисной архитектуры.

В рамках разработки был успешно внедрен комплекс современных архитектурных паттернов, включая Orchestration Saga для управления распределенными транзакциями, обеспечение идемпотентности критических операций и построение многоуровневой системы наблюдаемости.

Каждый сервис был спроектирован как независимый модуль с четко определенными контрактами, что обеспечило высокую степень слабой связанности и устойчивости системы к отказам.

Участники процесса

- **Клиент** - покупатель, инициирующий заказ
- **Keycloak** - сервер аутентификации/авторизации
- **API Gateway** - WSGI сервис (прокси + аутентификация)
- **OrderService** - оркестратор процесса создания заказа
- **BillingService** - управление платежами и счетами
- **WarehouseService** - управление складскими запасами
- **DeliveryService** - организация доставки
- **NotificationService** - отправка уведомлений

Используемые технологии и паттерны

1. Общий стек технологий: Java, Python, H2, JPA Hibernate
2. Паттерны: API Gateway, REST API, orchestration Saga, Transaction, Idempotency, Observability
3. Перечень используемых инструментов: Minikube, Helm, Prometheus, Grafana, Keyclock, Postman, newman
4.

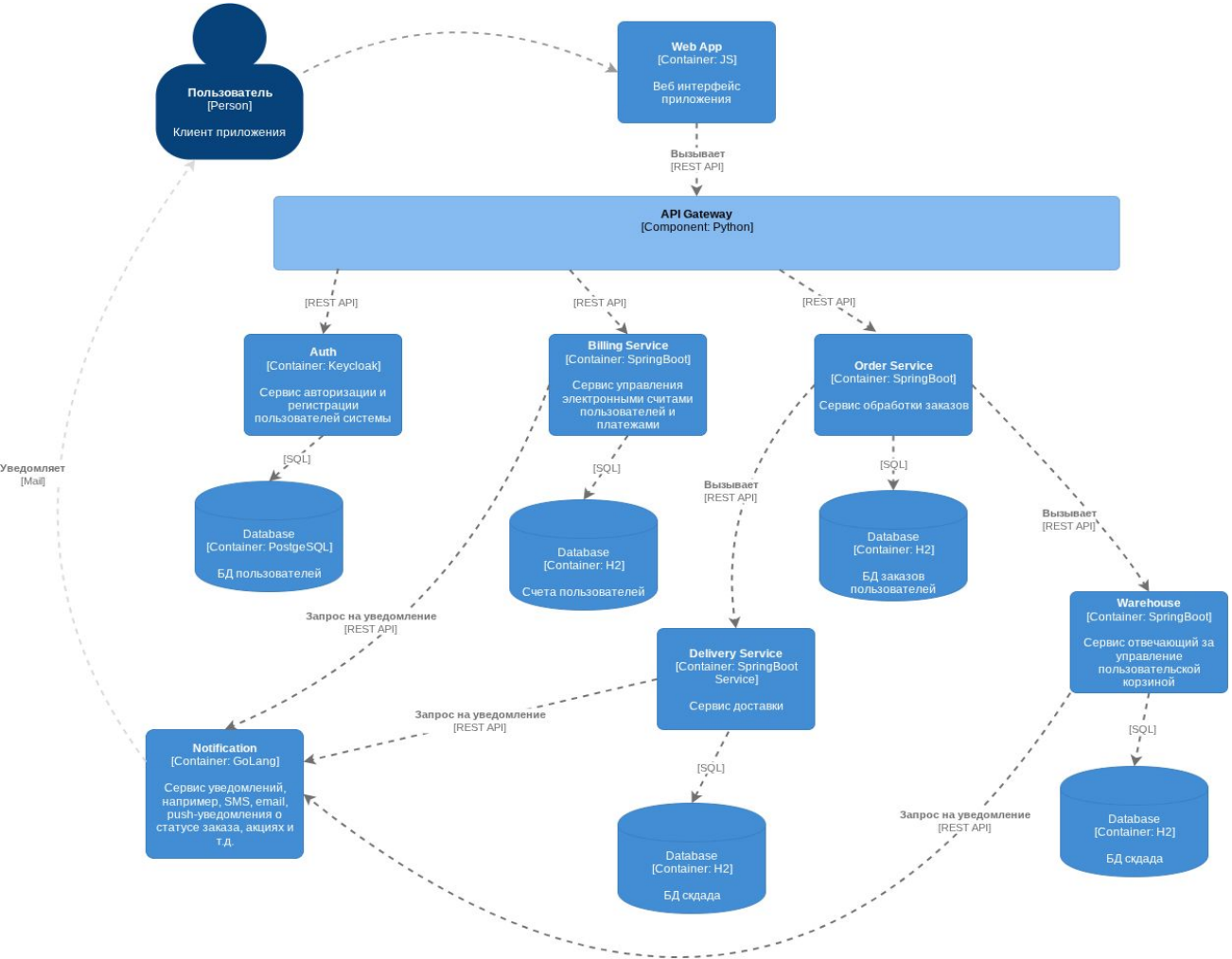


Диаграмма сервисов

1. API Gateway
2. Auth (keycloak)
3. Billing
4. Order
5. Warehouse
6. Delivery
7. Notification

\+ Prometheus
\+ Graphana

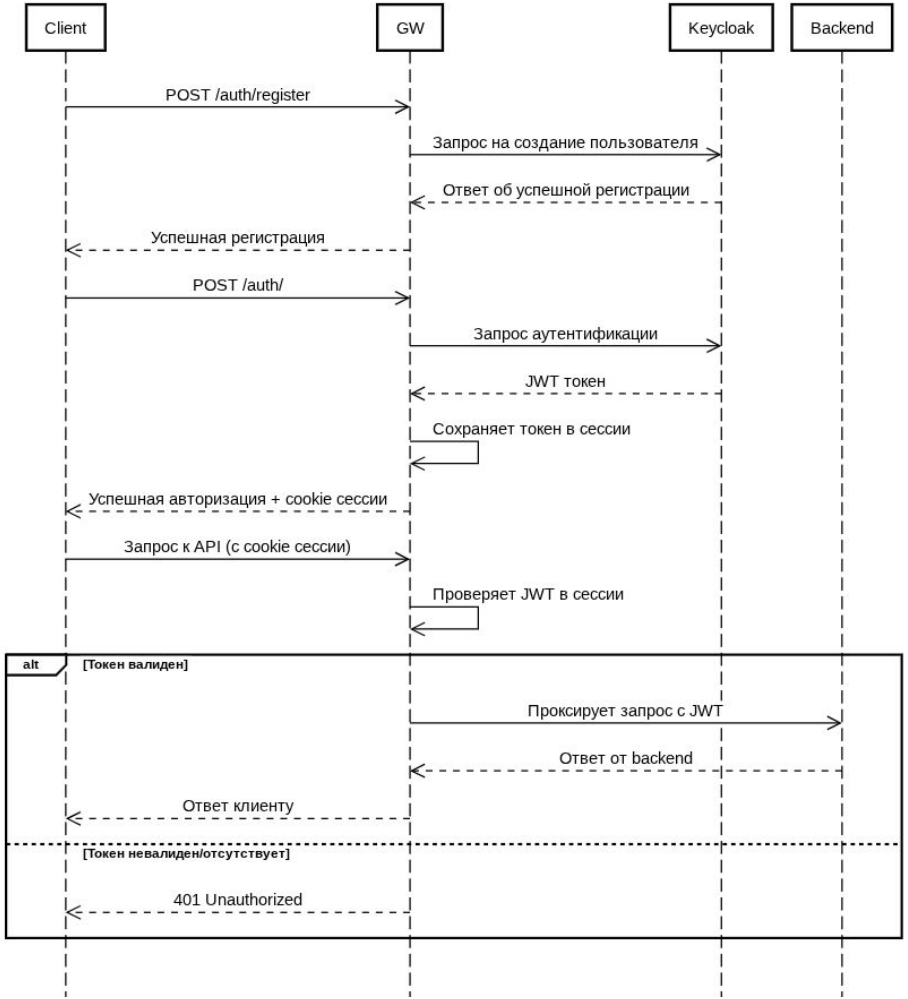


Схема аутентификации

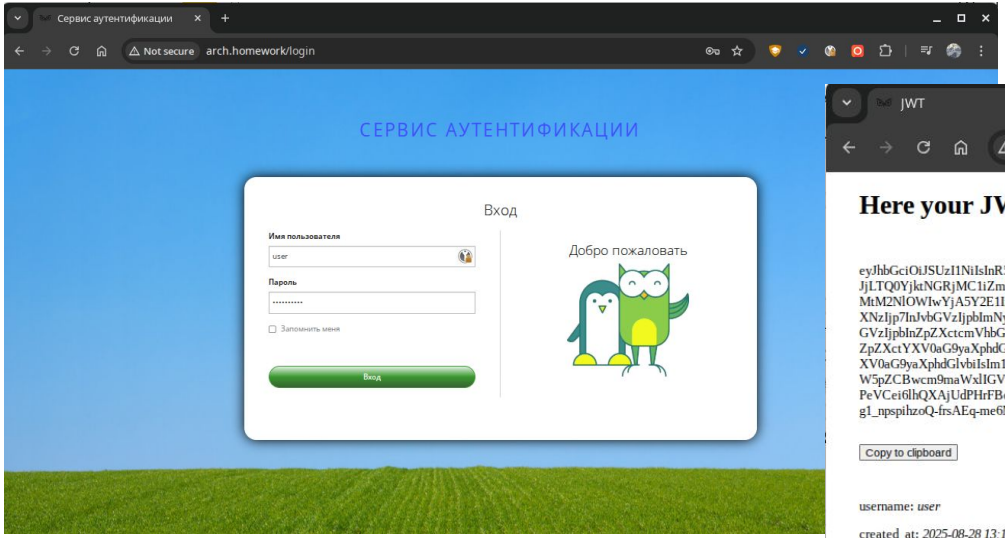
Client - веб-браузер или любой другое приложение

API Gateway - WSGI сервис (прокси + аутентификация)

Keycloak - сервер аутентификации/авторизации

Backend сервисы - защищенные микросервисы

Web интерфейс WSGI сервиса



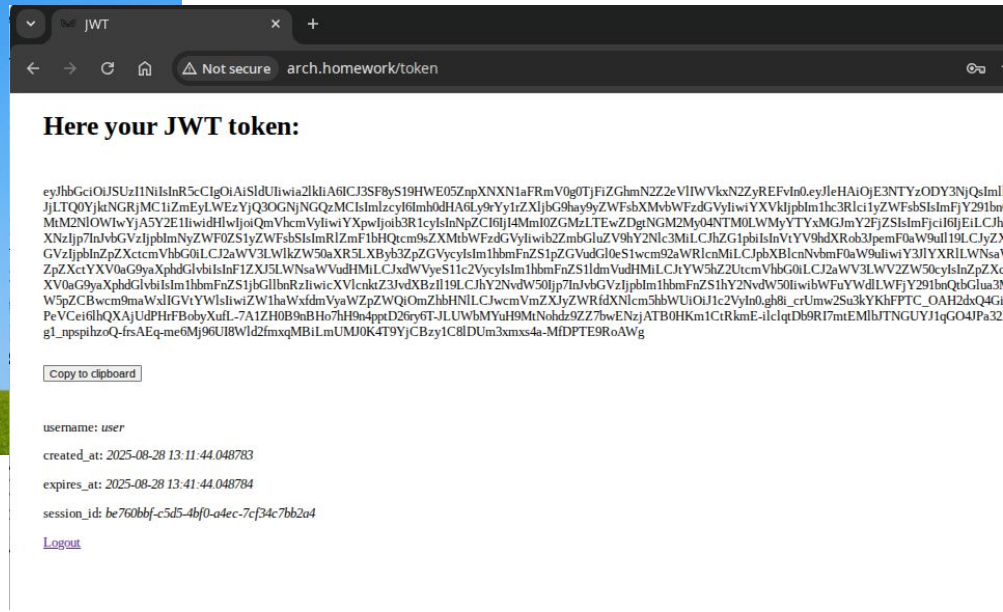
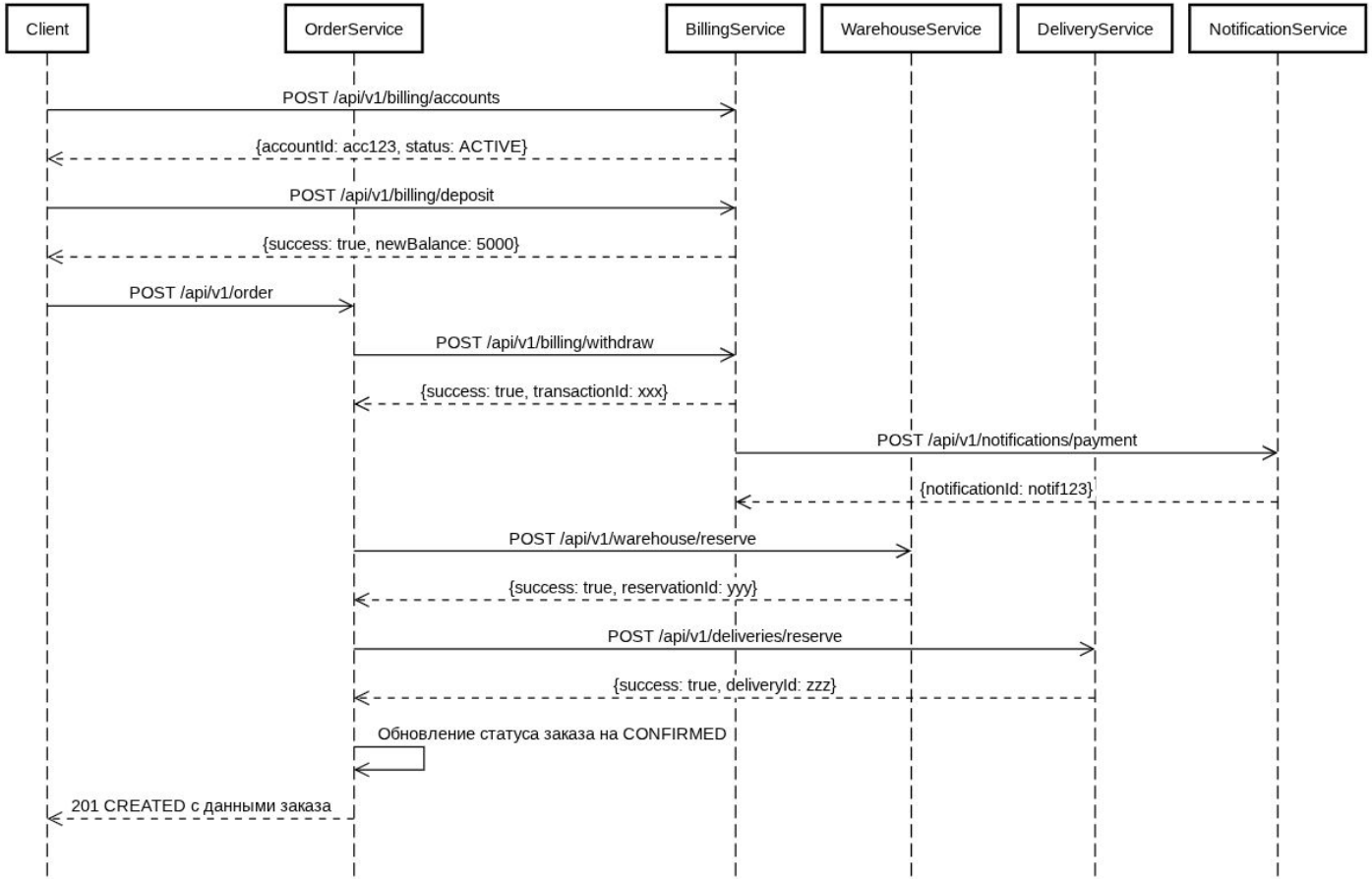


Диаграмма последовательности для ключевого сценария



Сценарий отката

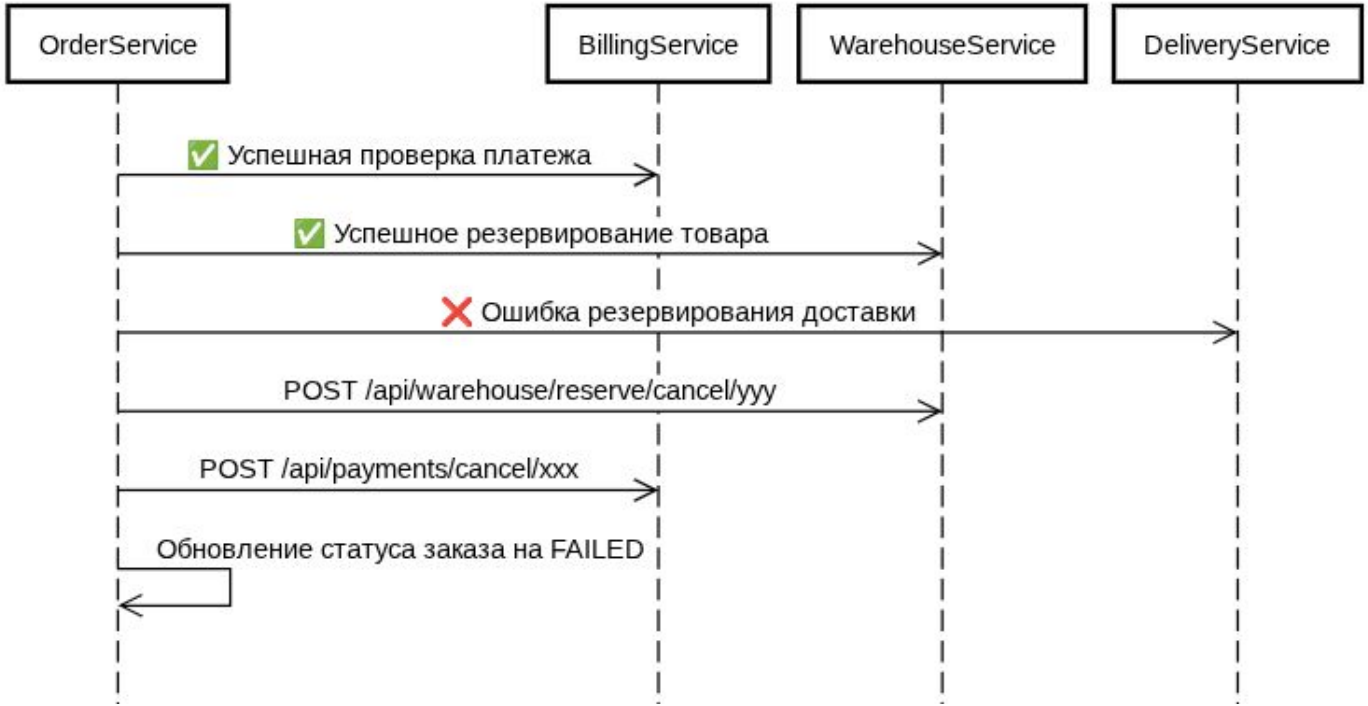
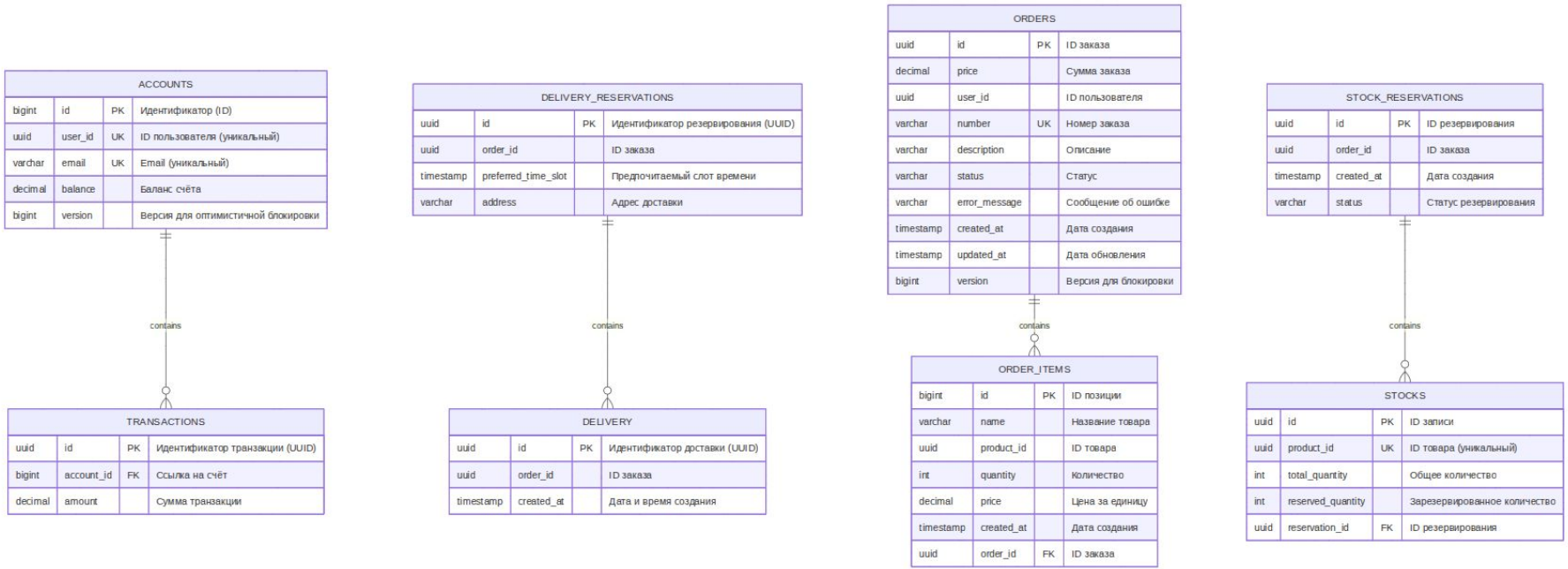


Схема хранения данных



Saga. Ключевые аспекты реализации

Order Service (Orchestrator)

- Управляет workflow всей саги
- Выполняет последовательные вызовы к другим сервисам
- Обрабатывает ошибки и запускает компенсирующие транзакции
- Обновляет статус заказа на каждом этапе

Сервис-участники (Participants)

- Billing Service: Сервис проведения платежных операций
- Warehouse Service: Резервирование товаров на складе
- Delivery Service: Бронирование временного слота доставки
- Notification Service: Сервис уведомления пользователя о состоянии заказа

Механизм компенсации

Каждый сервис предоставляет:

- Основную операцию (например, /reserve)
- Компенсирующую операцию (например, /reserve/cancel/{id})

DTO для межсервисного взаимодействия

```java
@Data
public class SagaStepResult {
    private boolean success;
    private String message;
    private UUID transactionId; // ID для компенсации
    private Object data;
}
```

Жизненный цикл заказа

```java
public enum OrderStatus {
    NEW,              // Новый заказ (начало саги)
    PROCESSING,       // В обработке
    PAID,             // Оплачен
    ITEMS_RESERVED,   // Товары зарезервированы
    DELIVERY_BOOKED,  // Доставка забронирована
    CONFIRMED,        // Заказ обработан
    FAILED            // Ошибка при обработке
}
```

Механизм отката

```
SagaStepResult warehouseResult = reserveItems(orderId, request.getItems());
log.debug( "2. Warehouse reservation result: {}", warehouseResult);
if (warehouseResult.isSuccess()) {
    updateOrderStatus(orderId, OrderStatus.ITEMS_RESERVED, null);
} else {
    cancelPayment(orderId, paymentResult.getTransactionId());
    throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR,
            "Warehouse reservation failed: " + warehouseResult.getMessage());
}

SagaStepResult deliveryResult = reserveDelivery(orderId, request.getDeliveryInfo());
log.debug( "3. Delivery reservation result: {}", deliveryResult);
if (deliveryResult.isSuccess()) {
    updateOrderStatus(orderId, OrderStatus.DELIVERY_BOOKED, null);
} else {
    cancelPayment(orderId, paymentResult.getTransactionId());
    cancelReservation(orderId, warehouseResult.getTransactionId());
    throw new ResponseStatusException(HttpStatus.INTERNAL_SERVER_ERROR,
            "Delivery reservation failed: " + deliveryResult.getMessage());
}
```

Идемпотентность

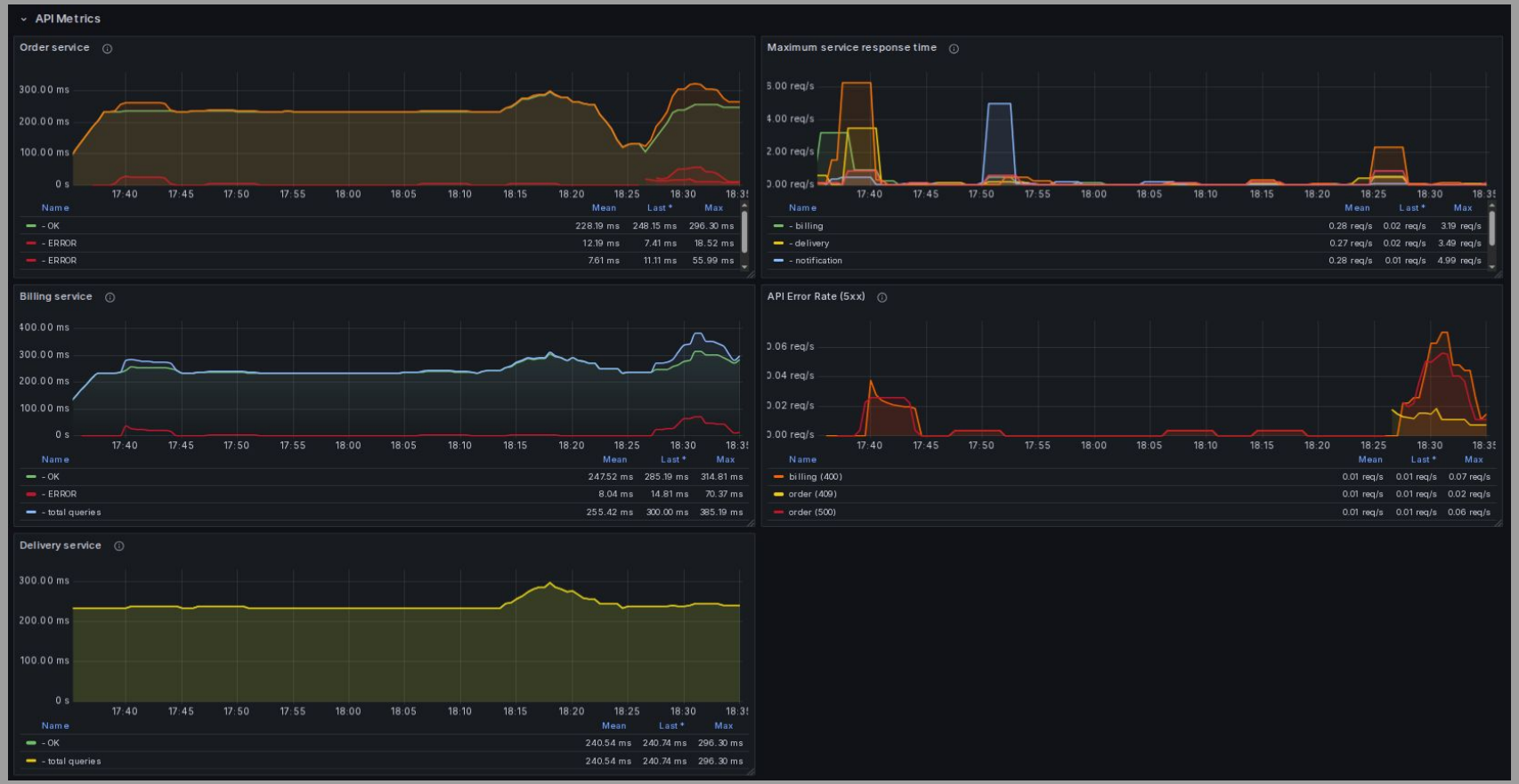
Реализован механизм гарантированной идемпотентности на основе уникального ключа, который клиент передает при каждом запросе создания заказа.

Сервер кэширует результаты обработки запросов и при повторных запросах с тем же ключом возвращает результат из кэша без повторного выполнения бизнес-логики.

Дополнительно проверяется наличие существующего заказа в БД по бизнес-правилам: тот же пользователь + похожие товары + пятиминутный интервал времени, в котором похожий заказ мог бы быть сделан.

1. В случае, если заказ успешно созадётся в первый раз, сервер возвращает код **201 (CREATED)**.
2. При нахождении запроса на заказ в кеше по ключу, сервер вернёт код **200 (OK)**. То есть это показывает, что на сервере не был создан новый ресурс (в виде записи в БД).
3. А в случае нахождения похожего заказа в БД, сервер ответит ошибкой **409 (CONFLICTED)**, но всё равно вернёт найденный заказ в теле ответа.

Мониторинг


API Metrics
Order service
300.00 ms
200.00 ms
100.00 ms
0 s
17:40 17:45 17:50 17:55 18:00 18:05 18:10 18:15 18:20 18:25 18:30
Name Mean Last * Max
- OK 228.19 ms 248.15 ms 296.30 ms
- ERROR 12.19 ms 7.41 ms 18.52 ms
- ERROR 7.61 ms 11.11 ms 55.99 ms
Maximum service response time
5.00 req/s
4.00 req/s
2.00 req/s
0.00 req/s
17:40 17:45 17:50 17:55 18:00 18:05 18:10 18:15 18:20 18:25 18:30
Name Mean Last * Max
- billing 0.28 req/s 0.02 req/s 3.19 req/s
- delivery 0.27 req/s 0.02 req/s 3.49 req/s
- notification 0.28 req/s 0.01 req/s 4.99 req/s
Billing service
400.00 ms
300.00 ms
200.00 ms
100.00 ms
0 s
17:40 17:45 17:50 17:55 18:00 18:05 18:10 18:15 18:20 18:25 18:30
Name Mean Last * Max
- OK 247.52 ms 285.19 ms 314.81 ms
- ERROR 8.04 ms 14.81 ms 70.37 ms
- total queries 255.42 ms 300.00 ms 385.19 ms
API Error Rate (5xx)
0.06 req/s
0.04 req/s
0.02 req/s
0.00 req/s
17:40 17:45 17:50 17:55 18:00 18:05 18:10 18:15 18:20 18:25 18:30
Name Mean Last * Max
billing (400) 0.01 req/s 0.01 req/s 0.07 req/s
order (409) 0.01 req/s 0.01 req/s 0.02 req/s
order (500) 0.01 req/s 0.01 req/s 0.06 req/s
Delivery service
300.00 ms
200.00 ms
100.00 ms
0 s
17:40 17:45 17:50 17:55 18:00 18:05 18:10 18:15 18:20 18:25 18:30
Name Mean Last * Max
- OK 240.54 ms 240.74 ms 296.30 ms
- total queries 240.54 ms 240.74 ms 296.30 ms

С какими сложностями столкнулись

Главную сложность представила собой настройка мониторинга и observability-инфраструктура. Интеграция Prometheus и Grafana потребовала глубокого понимания принципов работы метрик в Spring Boot приложениях, правильной конфигурации Actuator endpoints и организации сбора данных в Kubernetes-окружении.

Я столкнулся с проблемами обнаружения сервисов Prometheus через ServiceMonitor и настройки корректных label selectors и обеспечения необходимых RBAC прав для доступа к метрикам. Требовалась тонкая настройка аннотаций pod'ов и service'ов для автоматического обнаружения targets.

Вся работа над проектом и домашними заданиями принесла мне значительный профессиональный опыт. Мне кажется, я довольно глубоко вник в принципы построения распределенных систем.

Выводы и планы по развитию

1. Приобретен ценный практический опыт построения полноценной микросервисной архитектуры с использованием современных паттернов (Saga, идемпотентность, Circuit Breaker)

2. Освоены ключевые принципы обеспечения наблюдаемости распределенных систем и важность комплексного мониторинга для поддержания SLA

3. Хотелось бы ещё освоить внедрение распределенного трейсинга через OpenTelemetry для получения полной картины взаимодействия между сервисами и реализовать автоматический alerting на основе метрик бизнес-логики

4. Ну и планирую получить больше опыта в разработке event-driven систем. В рамках данного проекта у меня не хватило времени на это…

Спасибо за внимание!