Техническое задание

# 1. Введение

Учебный проект предназначен для отработки компетенций разработчика уровня Java Middle при работе с современным стеком корпоративных технологий:  
  
Java 17+, Stream API, Spring Boot 3, Spring Security 6, Apache Kafka, PostgreSQL, MongoDB, Redis, Docker, Kubernetes  
  
Важно: фронт-енд (UI/SPA) не входит в объём работ; все функции предоставляются через REST/gRPC‑API.  
  
Архитектура построена в виде набора микросервисов, взаимодействующих посредством событийной шины и REST/gRPC‑API.

# 2. Цели проекта

* Продемонстрировать способность проектировать и реализовывать распределённые системы на базе Spring‑стека.
* Отработать приёмы реактивного и событийного программирования с использованием Stream API и Kafka.
* Освоить безопасную аутентификацию и авторизацию (JWT, OAuth 2.1) средствами Spring Security.
* Научиться проектировать схему данных под реляционное (PostgreSQL) и документо‑ориентированное (MongoDB) хранилища.
* Использовать Redis для кэширования и распределённых блокировок.
* Освоить CI/CD, контейнеризацию (Docker) и оркестрацию (Kubernetes, Helm).

# 3. Предметная область

Платформа обработки заказов (Event‑Driven Order Processing Platform) для малого e‑commerce‑магазина:

* каталог товаров;
* управление пользователями и ролями;
* оформление заказов;
* учёт запасов;
* отправка уведомлений.

# 4. Общее описание архитектуры

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Сервис | Назначение | БД | Протоколы | Зависимости |
| api-gateway | Единая точка входа, rate‑limiting | — | REST, gRPC | auth-service |
| auth-service | JWT/OAuth 2.1, refresh‑tokens | PostgreSQL | REST | kafka-broker |
| user-service | CRUD пользователей, RBAC | PostgreSQL | REST, Kafka | kafka-broker |
| product-service | Каталог товаров | MongoDB | REST, Kafka | kafka-broker |
| inventory-service | Запасы и резервы | PostgreSQL | gRPC, Kafka-streams | product-service |
| order-service | Создание и жизненный цикл заказа | PostgreSQL | REST, gRPC, Kafka | inventory-service |
| notification-service | Email/SMS/push | MongoDB | Kafka | — |

Использование gRPC: внутренние синхронные вызовы с низкой задержкой выполняются через gRPC. Примеры: api‑gateway ↔ product‑service и order‑service ↔ inventory‑service при проверке наличия товара.

Все сервисы собираются отдельными контейнерами (Docker Buildx) и разворачиваются в Kubernetes (Helm‑чарты). Секреты — в Kubernetes Secrets (sealed‑secrets).

# 5. Функциональные требования

## 5.1 Пользователи

* Регистрация по e‑mail/паролю с подтверждением.
* OAuth 2 Login (Google, GitHub).
* Управление ролями (ROLE\_USER, ROLE\_MANAGER, ROLE\_ADMIN).

## 5.2 Товары

* CRUD позиций каталога (администраторы).
* Публикация/скрытие товаров.
* Поиск + фильтры (price range, category, text).

## 5.3 Заказы

* Оформление заказа (ROLE\_USER).
* Проверка наличия товара в inventory-service через Kafka request/response или gRPC.
* Статусы: NEW → RESERVED → PAID → SHIPPED → COMPLETED / CANCELLED.
* Саговая компенсация при недостатке товара.

## 5.4 Уведомления

* Отправка письма о статусе заказа.
* Rate‑limiting по пользователю (Redis leaky bucket).

# 6. Нефункциональные требования

|  |  |
| --- | --- |
| Категория | Требование |
| Производительность | 95-й персентиль latency REST‑запросов ≤ 150 мс при 500 RPS. |
| Доступность | 99.5 % в месяц (SLA). |
| Масштабируемость | Горизонтальное масштабирование каждого микросервиса до ×4 реплик без даунтайма. |
| Безопасность | OWASP Top‑10, TLS 1.3, секреты — Vault/Secrets. |
| Логирование | Структурированные логи (JSON) → Loki; трассировка → OpenTelemetry + Jaeger. |
| Мониторинг | Prometheus + Grafana, alert‑rules для критичных метрик. |

# 7. Интеграции и события Kafka

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Topic | Key | Схема | Producer | Consumer(s) |
| user.created | userId | Avro | auth-service | user-service |
| order.created | orderId | Avro | order-service | inventory-service, notification-service |
| inventory.reserved | orderId | Avro | inventory-service | order-service |
| order.status-changed | orderId | Avro | order-service | notification-service |

Схемы хранятся в Confluent Schema Registry.

# 8. Хранение данных

* PostgreSQL 15 — OLTP‑операции (пользователи, заказы, запасы).
* MongoDB 7 — документо‑ориентированные сущности (каталог, шаблоны уведомлений, логи).
* Redis 7 — кэш горячих данных, распределённые блокировки (Redlock).

# 9. CI/CD и локальное развёртывание

## 9.0 Pipeline

* PR → GitHub Actions: checkstyle, unit‑tests, code‑coverage ≥ 80 %.
* Docker build → push в registry.
* Helm‑release в dev‑namespace.
* Интеграционные API‑тесты (Testcontainers + Spring WebTestClient).
* Manual approval → prod‑namespace.

## 9.1 Локальное развёртывание

Требования: Windows 10/11 с Docker Desktop (WSL 2 backend), 2 CPU, 8 GB RAM, GNU Make (choco install make) или PowerShell 5+.  
  
Запуск стека локально:  
WSL2/Git Bash — make dev-up / make dev-down.  
PowerShell — ./dev-up.ps1 / ./dev-down.ps1 (скрипты входят в репозиторий).  
  
docker-compose.yaml поднимает Kafka (Redpanda), PostgreSQL, MongoDB, Redis, Confluent Schema Registry и все микросервисы.  
  
Скрипт make k3d-up разворачивает кластер k3d и Helm‑чарты для локального k8s‑тестирования.  
Секреты dev‑окружения лежат в .secrets/dev; make inject-secrets загружает их в k3d.

# 10. План‑график (4‑недельный спринт)

|  |  |  |
| --- | --- | --- |
| Неделя | Основные работы | Итог |
| 1 | Анализ требований, проектирование, настройка репо, CI‑пайплайн, базовые Dockerfile/Helm, запуск «пустых» сервисов | Каркас платформы |
| 2 | Реализация auth-service (JWT), api-gateway, CRUD product‑ и inventory-service, Kafka-topics + schema registry | Аутентификация и каталог |
| 3 | order-service с сагой резервирования (gRPC + Kafka), notification-service (email), интеграционные тесты | Полный цикл заказа |
| 4 | Observability (Prometheus, Grafana, Loki, Jaeger), security‑тестирование, оптимизации, финальный деплой в prod, подготовка документации и демо | Готовый MVP |

Stretch goals (если останется время): OAuth‑логины соцсетей, Redis‑rate‑limiter, gRPC‑стриминг‑API.

# 11. Критерии приёмки

* Все сервисы проходят unit‑ и интеграционные тесты; pipeline «зелёный».
* K8s‑кластер развёрнут в k3d (или WSL‑based kind) локально.
* Заказ проходит весь жизненный цикл без ручных вмешательств.
* Swagger‑UI (/swagger‑ui.html) отражает актуальное OpenAPI, все эндпоинты работают.
* Security‑сканирование (OWASP Dependency‑Check + Trivy) показывает 0 критичных уязвимостей.
* Полный стек разворачивается локально одной командой make dev-up (или ./dev-up.ps1) без дополнительных действий.
* Для проверки API используется встроенный Swagger‑UI и авто‑генерированный OpenAPI‑yaml.