Проектирование архитектуры программных систем

Д7

Проект: Платформа для управления персональными финансами "FinTrack"

Выполнили:

Сидоров Артемий БПИ225

Шмараев Артём БПИ225

(БПИ228)

ПИ ФКН НИУ ВШЭ · 2024/2025 учебный год

Содержание

1. ЗАР по применяемым стекам технологий в проектируемом ПО "FinTrack"

1.1 Frontend (SPA)

Компонент: Веб-интерфейс пользователя

Назначение: Отображение пользовательского интерфейса, взаимодействие с API, реализация клиентской логики

Выбранная технология: React

Альтернативы: Vue.js

Обоснование выбора: React обладает зрелой экосистемой, активным сообществом и поддержкой TypeScript. Позволяет строить масштабируемые и производительные одностраничные приложения (SPA). В проекте уже применяется и интегрирован в CI/CD.

Риски и меры: Усложнение архитектуры — предотвращается использованием линтинга, модульного подхода и unit-тестирования.

1.2 Backend (API-сервер)

Компонент: Серверная логика, REST API

Назначение: Реализация бизнес-логики, валидация данных, маршрутизация запросов, взаимодействие с СУБД и кэшем

Выбранная технология: FastAPI

Альтернативы: Django

Обоснование выбора: FastAPI обеспечивает высокую производительность благодаря асинхронности, автоматически генерирует OpenAPI-документацию и быстро разворачивается. Подходит для микросервисной архитектуры и событийной обработки.

Риски и меры: Молодая экосистема — компенсируется выбором LTS-библиотек и широким покрытием тестами.

1.3 Auth-сервис

Компонент: Микросервис аутентификации

Назначение: Регистрация, вход, авторизация, генерация и проверка JWT-токенов

Выбранная технология: FastAPI + OAuth2 (JWT)

Альтернативы: Keycloak

Обоснование выбора: Простота интеграции с остальной системой, высокая безопасность за счет стандарта OAuth2 и JWT. Масштабируется горизонтально, не требует внешнего IDM-сервиса.

Риски и меры: Неправильная реализация авторизации — смягчается использованием проверенных библиотек (oauthlib, pyjwt) и тестированием.

1.4 СУБД

Компонент: Хранилище данных

Назначение: Персистентное хранение данных о пользователях, транзакциях, целях и уведомлениях

Выбранная технология: PostgreSQL

Альтернативы: MySQL

Обоснование выбора: PostgreSQL поддерживает ACID-транзакции, JSONB, оконные функции и масштабирование. Обладает высокой надежностью и гибкой системой индексов, что критично для финансовых операций.

Риски и меры: Высокая нагрузка на память — решается настройкой конфигурации (shared_buffers, work_mem).

1.5 Кэш / Брокер

Компонент: In-memory кэш и очередь задач

Назначение: Ускорение доступа к данным, очередь задач для worker'ов

Выбранная технология: Redis

Альтернативы: RabbitMQ

Обоснование выбора: Redis одновременно может выполнять роль кэша и брокера задач. Обеспечивает высокую скорость обработки, прост в установке и широко поддерживается. Поддерживает Celery.

Риски и меры: Потеря данных при сбое — используется снапшоты (RDB/AOF) и бэкапы в S3.

1.6 Очереди сообщений

Компонент: Механизм обмена событиями

Назначение: Асинхронное взаимодействие микросервисов, обработка событий от клиентов

Выбранная технология: Apache Kafka

Альтернативы: Redis Streams

Обоснование выбора: Kafka устойчив к сбоям, обеспечивает сохранение событий, масштабируем, подходит для high-throughput задач. Используется для событийной модели FinTrack.

Риски и меры: Сложность поддержки — минимизируется использованием Helm-чартов и managed Kafka в облаке.

1.7 Worker-процессы

Компонент: Воркеры фоновых задач

Назначение: Обработка долгих операций: уведомления, генерация отчетов, интеграции

Выбранная технология: Celery

Альтернативы: FastStream

Обоснование выбора: Celery — зрелое решение с поддержкой повторных попыток, планировщиков и мониторинга. Широко используется, интегрируется с Redis.

Риски и меры: Проблемы с производительностью — решаются автоскейлингом, выделением очередей.

1.8 Object Storage

Компонент: Объектное хранилище

Назначение: Хранение чеков, экспортов, пользовательских отчетов

Выбранная технология: Yandex Object Storage (S3)

Альтернативы: MinIO

Обоснование выбора: Облачное хранилище не требует администрирования, масштабируется автоматически, имеет встроенное резервное копирование и жизненные циклы объектов.

Риски и меры: Затраты — решаются контролем объемов и использованием политики хранения.

1.9 CI/CD

Компонент: Автоматизация сборки и развертывания

Назначение: Сборка Docker-образов, деплой, тестирование, линтинг

Выбранная технология: GitHub Actions

Альтернативы: GitLab CI

Обоснование выбора: GitHub Actions нативно интегрирован в репозиторий, легко настраивается и поддерживает секреты, Docker, pytest, flake8 и GitOps-деплой.

Риски и меры: Ограничение по runtime — решается выделением runner'ов и кэшированием слоев.

1.10 Инфраструктура и оркестрация

Компонент: Оркестратор контейнеров

Назначение: Автоскейлинг, отказоустойчивость, zero-downtime деплой, управление микросервисами

Выбранная технология: Kubernetes + Helm

Альтернативы: Docker Compose

Обоснование выбора: Kubernetes — индустриальный стандарт, поддерживает rolling update, autoscaling, liveness/readiness probes. Helm-чарты позволяют управлять конфигурациями и окружениями.

Риски и меры: Сложность конфигурации — компенсируется использованием Yandex Managed Kubernetes и готовыми Helm-чартами.

2. Дополненная UML диаграмма развертывания проектируемого ПО, с результатом выбранных технологий