САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа

Выполнила:

Гусейнова Марьям

БР 1.2

Проверил:
Добряков Д. И.

Санкт-Петербург

2025 г.

Задача

- реализовать Dockerfile для каждого сервиса;
- написать общий docker-compose.yml;
- настроить сетевое взаимодействие между сервисами.

Ход работы

Для достижения поставленных задач были выполнены следующие шаги по контейнеризации и оркестрации микросервисного приложения "Fitness App":

1. Разработка Dockerfile для каждого микросервиса. Для каждого микросервиса (API Gateway, User Service, Workout & Exercise Service, Plan & Progress Service, Blog Service) был разработан индивидуальный Dockerfile. Структура этих файлов идентична, за исключением портов, которые каждый сервис использует. Пример Dockerfile для api-gateway выглядит следующим образом:

```
FROM node:20-alpine

WORKDIR /app

COPY package*.json ./

RUN npm install

COPY . .

RUN npm run build

EXPOSE 3000

CMD ["node", "dist/index.js"]
```

Здесь мы определяем базовый образ (в нашем случае легковесный образ Node.js версии 20 на базе Alpine Linux), устанавливаем рабочую директорию внутри контейнера, копируем в нее файлы зависимостей, устанавливаем все необходимые зависимости, копируем весь исходный код в контейнер, запускаем команду сборки, сообщаем, на каком порту будет слушать контейнер и определяем команду, которая будет выполнена при запуске контейнера.

2. Разработка общего docker-compose.yml. Файл docker-compose.yml был создан для определения и запуска многоконтейнерного

приложения. Он описывает все сервисы, их образы (или инструкции по сборке), переменные окружения, порты, тома и зависимости между ними.

- Сервис базы данных (db): использует образ postgres:16.9-alpine для легковесной и стабильной работы. Он настроен на restart: always для автоматического перезапуска. Учетные данные для PostgreSQL (POSTGRES_USER, POSTGRES_PASSWORD) передаются через переменные окружения. Порт базы данных 5432 мапируется на внешний порт 5433 для доступа извне контейнерной сети.
- Персистентность данных: для сохранения данных базы данных используется именованный том db_data (volumes: - db_data:/var/lib/postgresql/data). Это гарантирует, что данные не будут потеряны при удалении или пересоздании контейнера БД.
- Инициализация базы данных: при первом запуске контейнера db выполняется SQL-скрипт из директории ./db-init. Файл db-init/init.sql содержит команды CREATE DATABASE для создания отдельных баз данных для каждого микросервиса (fitness_users_db, fitness_workouts_db, fitness_plans_progress_db, fitness_blog_db). Такой подход обеспечивает изоляцию данных между сервисами.
- Проверка состояния базы данных (healthcheck): для сервиса db был добавлен healthcheck, который периодически проверяет готовность PostgreSQL к приему соединений с помощью команды pg_isready. Это нужно для обеспечения корректного порядка запуска сервисов.
- Микросервисы (User Service, Workout & Exercise Service, Plan & Progress Service, Blog Service): каждый микросервис собирается из своего Dockerfile. Они настроены на restart: always. Порты каждого сервиса мапируются для обеспечения доступа и отладки. Каждый микросервис использует свою базу данных внутри одного Postgres.
- API Gateway: выступает в качестве единой точки входа для клиентских запросов. Он также собирается из своего Dockerfile и мапирует свой порт.
- Зависимости сервисов (depends_on):
    - Все микросервисы (кроме api-gateway) зависят от сервиса db с условием condition: service_healthy. Это означает, что контейнеры микросервисов будут запускаться только после того, как база данных полностью инициализируется и будет

готова принимать соединения, предотвращая ошибки подключения на старте.
    - Сервис plan-progress-service также зависит от user-service и workout-exercise-service с условием condition: service_started, поскольку ему могут потребоваться данные или функции от этих сервисов.
    - api-gateway зависит от всех остальных микросервисов с условием condition: service_started, так как он не должен запускаться, пока его бэкенд-сервисы не будут доступны.

3. Разработка общего .env файла и передача переменных окружения. Был создан общий файл .env в корневой директории проекта. Этот файл содержит все необходимые переменные окружения, такие как учетные данные для базы данных (DB_USER, DB_PASSWORD), порты для всех сервисов (API_GATEWAY_PORT, USER_SERVICE_INTERNAL_PORT и т.д.) и секретный ключ для JWT (JWT_SECRET). Переменные из .env автоматически подгружаются Docker Compose и передаются в секцию environment каждого сервиса в docker-compose.yml. Внутри каждого микросервиса в файлах config/index.ts, эти переменные окружения читаются с помощью библиотеки dotenv через process.env.

Вывод

В рамках данной лабораторной работы было реализовано и развернуто микросервисное приложение "Fitness App" с использованием Docker и Docker Compose. Каждый микросервис был контейнеризован с помощью индивидуального Dockerfile, что обеспечило их изоляцию и переносимость. Общий файл docker-compose.yml позволил эффективно оркестрировать все компоненты приложения, включая базу данных PostgreSQL, и настроить их взаимодействие. Особое внимание было уделено персистентности данных для БД с помощью томов и автоматической инициализации баз данных. Использование healthcheck и depends_on гарантирует правильный порядок запуска сервисов, минимизируя проблемы, связанные с зависимостями.