САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа №5

Выполнил:

Акулов Даниил

К3439

Проверил:
Добряков Д. И.

Санкт-Петербург

2026 г.

Задача

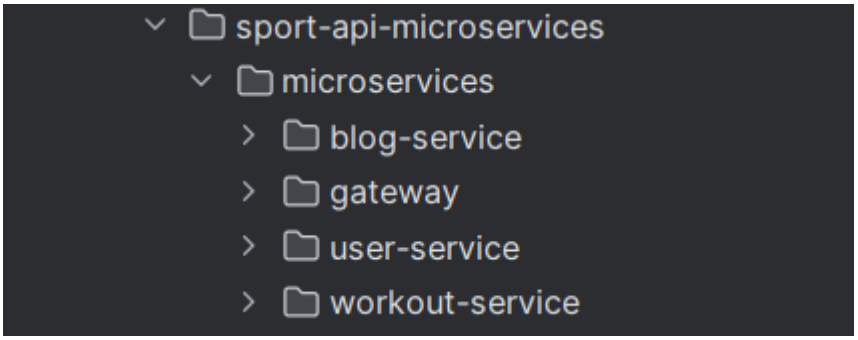
• выделить самостоятельные модули в вашем приложении;

• провести разделение своего API на микросервисы (минимум, их должно быть 3);

• настроить сетевое взаимодействие между микросервисами.

Ход работы

В процессе разработки приложения была проведена декомпозиция системы на независимые модули. Каждый модуль отвечает за свой домен, а все зависимости между ними минимизированы и оформлены через REST API. Ниже приведён список модулей:

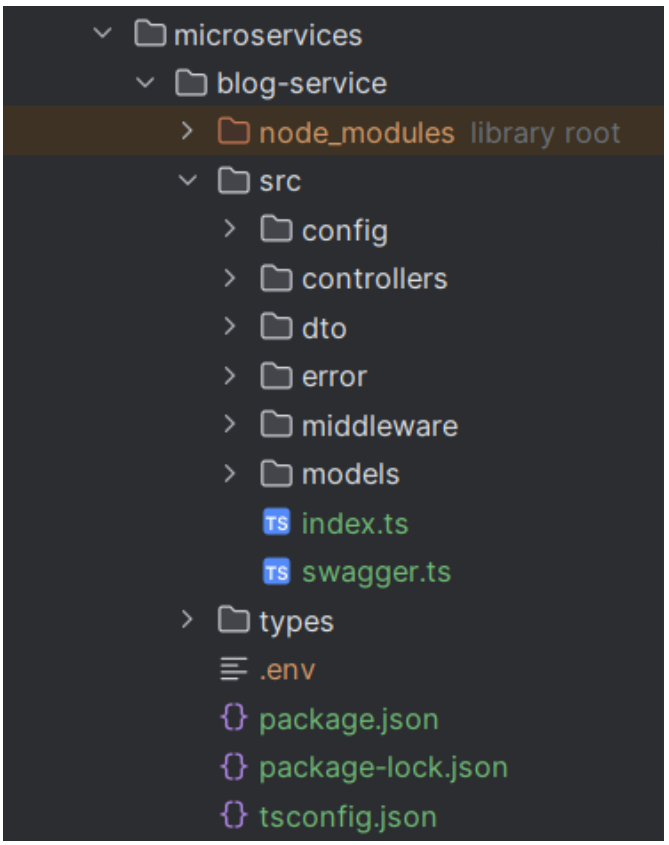


User-service - регистрация, авторизация, профили пользователей

Workout-sevice - планы тренировок, логи тренировки, упражнения

Blog-service - блог по тренировкам и питанию

Gateway - общая точка входа



Пример структуры одного сервиса.

Сетевое взаимодействие осуществляется с помощью axios. Пример запроса к user-service-api для получения профиля пользователя:

```
let user = null
    try{
        const response = await axios.get(
`http://localhost:${SETTINGS.API_USER_PORT}/api/user/get-one/${post.authorId}`)
        user = response.data.user;
    } catch(error) {
        console.log("Axios error: ", error)
    }
```

Реализация endpoint getWorkoutPlans для получения планов тренировки определенного пользователя из user-service:

```
@OpenAPI({})
  @Get('/get-workout-plans/:userId')
  async getWorkoutPlans(
      @Req() req: Request,
      @Res() res: Response,
  ) {
      const userId = parseInt(req.params.userId);
      let workoutPlans = []
      try{
          const response = await axios.get(
`http://localhost:${SETTINGS.API_WORKOUT_PORT}/api/workout-plan/get-workout-plans/${userId}`)
          workoutPlans = response.data.workoutPlans;
      } catch(error) {
          console.log("Axios error: ", error)
      }
      return res.json({workoutPlans});
  }
```

Реализация index.ts файла gateway сервиса:

```
import express from "express";
import { createProxyMiddleware } from "http-proxy-middleware";
import {SETTINGS} from "./config/settings";
const app = express();
app.use("/user", createProxyMiddleware({target: `http://localhost:${SETTINGS.API_USER_PORT}`, changeOrigin: true,}));
app.use("/blog", createProxyMiddleware({target: `http://localhost:${SETTINGS.API_BLOG_PORT}`, changeOrigin: true,}));
```

```
app.use("/workout", createProxyMiddleware({target: `http://localhost:${SETTINGS.API_WORKOUT_PORT}`, changeOrigin: true,}));
app.listen(SETTINGS.API_PORT, () => {
    console.log(`API Gateway running on port ${SETTINGS.API_PORT}`);
});
```

Вывод

Приложение разбито на независимые сервисы с чёткой ответственностью, что повысило модульность, упростило масштабирование и развертывание. Единый API-Gateway обеспечивает централизованную маршрутизацию и авторизацию, а межсервисные вызовы гарантируют согласованность данных. Такая архитектура делает систему более гибкой, надежной и удобной.