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

- выделить самостоятельные модули в вашем приложении;
- провести разделение своего API на микросервисы (минимум, их должно быть 3);
- настроить сетевое взаимодействие между микросервисами.

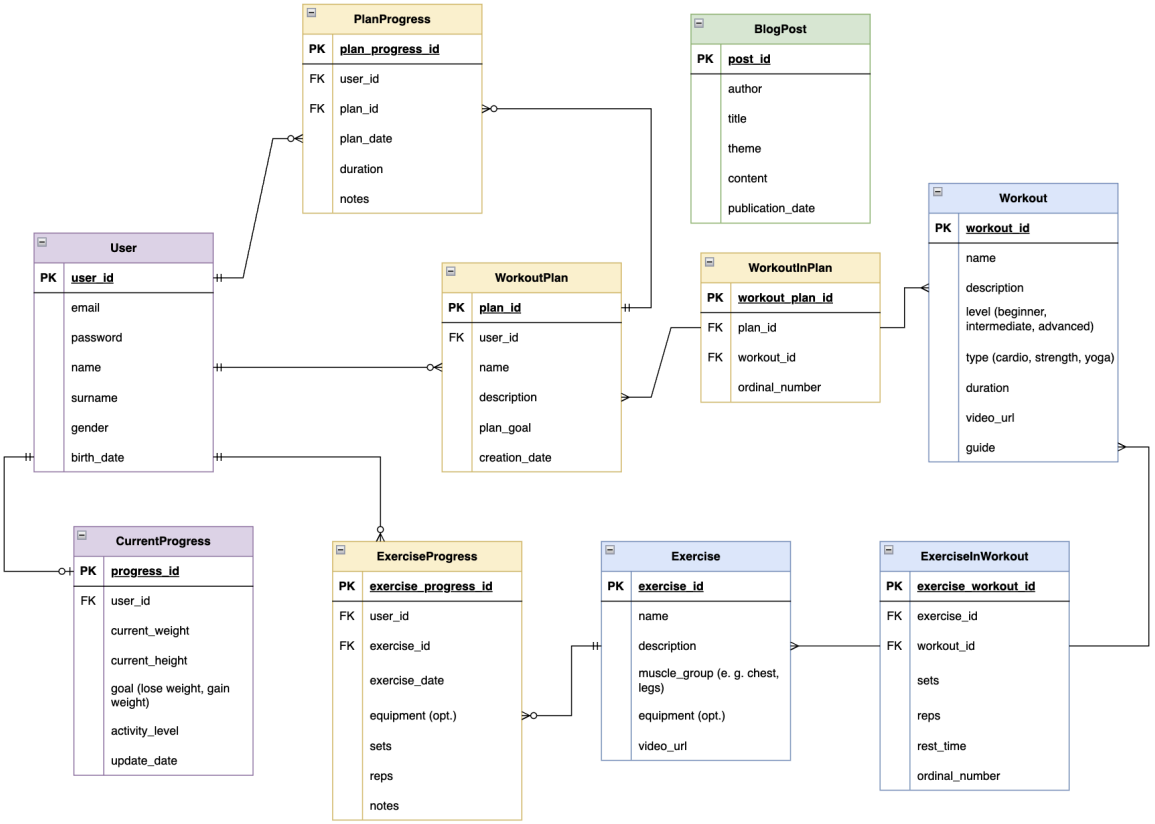
Ход работы

Работа началась с анализа существующего монолитного приложения, целью которого было выявление логически связанных и независимых по данным частей. В результате этого анализа были выделены следующие ключевые доменные области, каждая из которых стала основой для отдельного микросервиса:

1. Сервис пользователей (User Service): этот сервис отвечает за управление данными пользователей. Он оперирует такими моделями, как User (пользователи) и CurrentProgress (текущий прогресс пользователя) и обрабатывает все операции, связанные с аутентификацией, регистрацией, профилями пользователей и их текущими прогрессами.
2. Сервис тренировок и упражнений (Workout-Exercise Service): данный сервис сфокусирован на управлении информацией о тренировках и упражнениях. Он включает в себя модели Workout (тренировки), Exercise (упражнения) и ExerciseWorkout (связь между тренировками и упражнениями). Этот сервис предоставляет функциональность по созданию, редактированию и получению данных о тренировочных программах.
3. Сервис прогресса (Plan-Progress Service): этот сервис предназначен для отслеживания прогресса пользователей в выполнении планов тренировок и отдельных упражнений. Он работает с моделями PlanProgress (прогресс по плану тренировок), ExerciseProgress (прогресс по упражнениям), WorkoutPlan (планы тренировок), WorkoutInPlan (связь между тренировкой и планом). Важной особенностью этого сервиса является его зависимость от данных из других сервисов для корректной работы.
4. Сервис блога (Blog Service): данный сервис отвечает за логику работы блога в приложении. Он включает модель BlogPost (посты

блога), обрабатывает все CRUD-методы связанные с нею. Данный сервис никак не связан с другими.

Наглядно можно увидеть на схеме базы данных, где модели, вошедшие в разные микросервсисы, имеют разные цвета:



После выделения доменных областей, каждый сервис был разработан как независимое Node.js приложение с собственной базой данных и отдельным API.

Для организации взаимодействия между этими сервисами и обеспечения единой точки входа для клиентских приложений был реализован API Gateway. Этот подход позволил централизовать маршрутизацию запросов, а также внедрить сквозную функциональность, такую как аутентификация и логирование. В файле src/index.ts API Gateway используется библиотека http-proxy-middleware для маршрутизации входящих HTTP-запросов к соответствующим внутренним микросервисам. Конфигурация прокси осуществляется динамически, позволяя легко добавлять новые сервисы и их эндпоинты. Каждый сервис имеет свой базовый URL (например, USER_SERVICE_URL, WORKOUT_EXERCISE_SERVICE_URL,

PLAN_PROGRESS_SERVICE_URL), которые хранятся в файле src/config/index.ts и загружаются из переменных окружения.

Особое внимание было уделено настройке аутентификации. В API Gateway был разработан auth.middleware.ts, который перехватывает все входящие запросы. Этот middleware отвечает за валидацию JWT-токенов, прикрепленных к запросам. В случае успешной валидации, userId из токена извлекается и передается в заголовке x-user-id дальше в микросервисы. Это позволяет бэкенд-сервисам определять пользователя, инициировавшего запрос, без необходимости повторной валидации токена. Для регистрации и логина пользователей, auth.middleware специально пропускает аутентификацию, позволяя этим запросам дойти до User Service напрямую.

Внутри сервиса прогресса (Plan-Progress Service) сетевое взаимодействие с другими микросервисами реализовано через файлы клиентов. Так, user.service.client.ts и workout.exercise.service.client.ts используют библиотеку axios для выполнения HTTP-запросов к User Service и Workout-Exercise Service соответственно. Эти клиенты предоставляют методы, такие как doesUserExist или doesExerciseExist, которые позволяют сервису проверять существование связанных сущностей в других доменах, обеспечивая целостность данных перед выполнением операций создания или обновления.

Для улучшения обработки ошибок и обеспечения консистентности ответов в микросервисах были введены универсальные утилиты: CustomError и asyncHandler.

- CustomError (src/utils/custom-error.util.ts) позволяет выбрасывать типизированные ошибки с определенным HTTP-статусом, что упрощает их обработку на глобальном уровне.
- asyncHandler (src/middlewares/async.handler.ts) является оберткой для асинхронных функций контроллеров, которая автоматически перехватывает любые промисы, отклоненные (rejected) в этих функциях, и передает их в глобальный обработчик ошибок.
- Глобальный обработчик ошибок (логика в src/index.ts) перехватывает все выбрасываемые CustomError и отправляет клиенту соответствующий HTTP-статус и сообщение, а для неожиданных ошибок возвращает общий статус 500.

Вывод

В ходе данной лабораторной работы была осуществлена декомпозиция монолитного приложения в микросервисную архитектуру, состоящую из четырех специализированных сервисов (User Service, Workout-Exercise Service, Plan-Progress Service, Blog Service), взаимодействующих через централизованный API Gateway. Настройка аутентификации на уровне шлюза и использование межсервисных клиентов обеспечили безопасность и согласованность данных.