

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ  
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Дисциплина:** Бэк-энд разработка

Отчет

Домашнее задание №2

Выполнил:

Корчагин Вадим

Группа  
К3341

Проверил:

Добряков Д. И.

Санкт-Петербург

2025 г.

## Тема

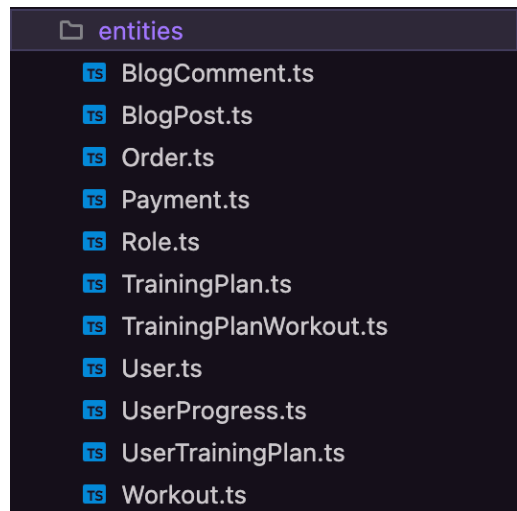
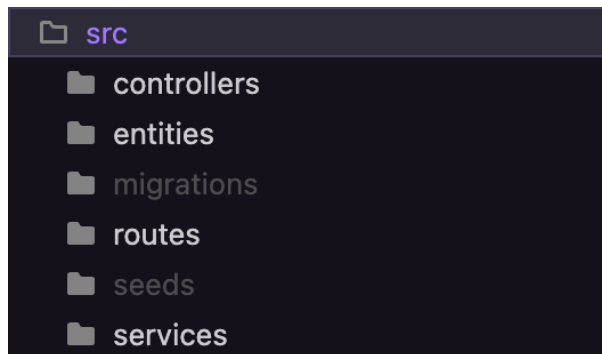
Платформа для фитнес-тренировок и здоровья.

## Задача

- Реализовать все модели данных, спроектированные в рамках ДЗ1
- Реализовать набор из CRUD-методов для работы с моделями данных средствами Express + TypeScript
- Реализовать API-эндпоинт для получения пользователя по id/email

## Ход работы

Были реализованы все Entity, которые были созданы на ERD-диаграмме в ДЗ1. А также создана файловая структура в соответствии с рекомендациями преподавателя.

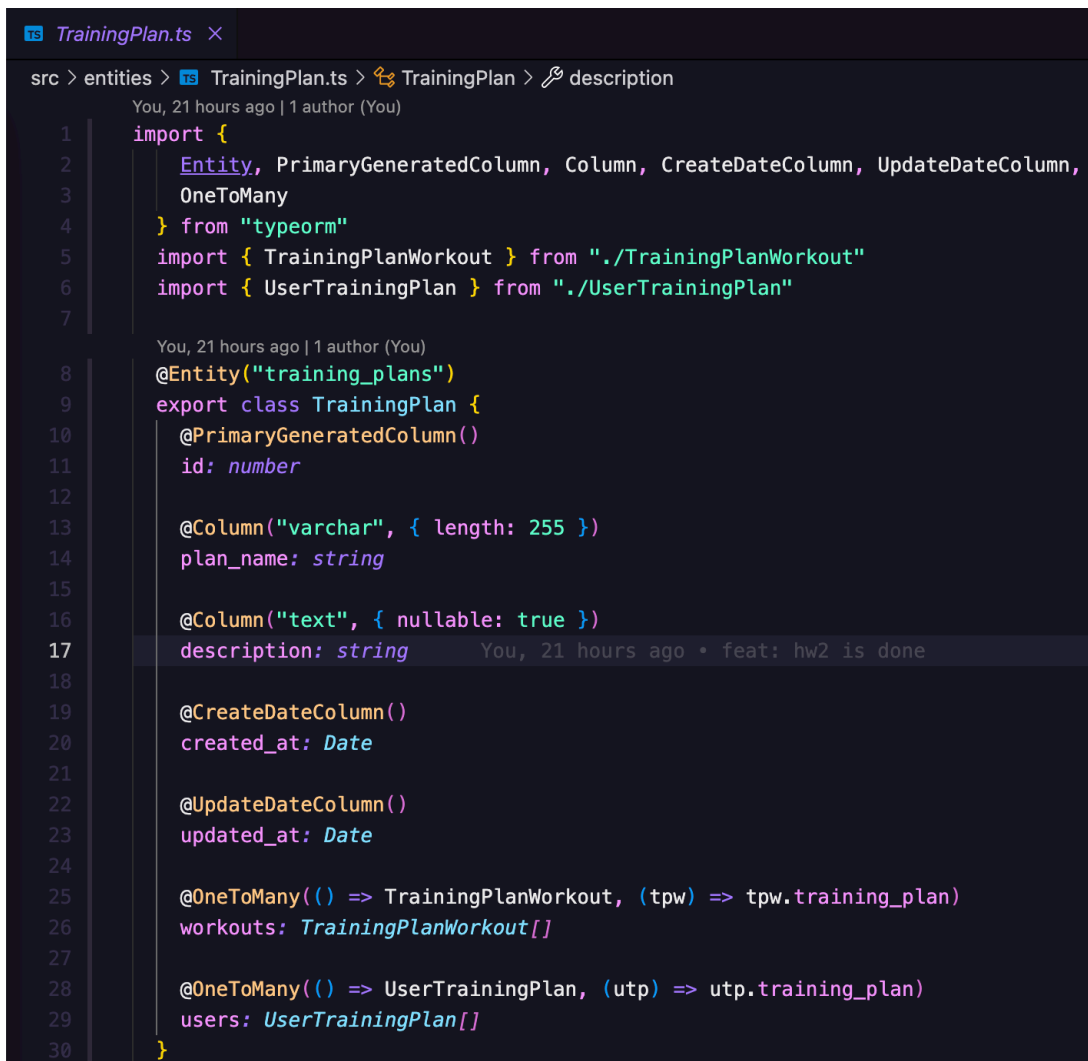


## Entities

Пример созданных Entity можно увидеть на Рисунке 1, на котором отображен TrainingPlan.

- `@Entity("training_plans")` — указывает на таблицу `training_plans` в базе данных.
- `@PrimaryGeneratedColumn()` — автоинкрементируемый первичный ключ.

- `@Column(...)` — поля `plan_name` и `description` с типами `varchar` и `text`.
- `@CreateDateColumn`, `@UpdateDateColumn` — автоматически задаются при создании и обновлении.
- `@OneToMany(...)` — связи, возможность автоматически загружать связанные сущности (relations), автоматическая поддержка внешних ключей и каскадных операций:
  - с `TrainingPlanWorkout` — список тренировок, входящих в план



```

1  import {
2      Entity, PrimaryGeneratedColumn, Column, CreateDateColumn, UpdateDateColumn,
3      OneToMany
4  } from "typeorm"
5  import { TrainingPlanWorkout } from "../TrainingPlanWorkout"
6  import { UserTrainingPlan } from "../UserTrainingPlan"
7
8  @Entity("training_plans")
9  export class TrainingPlan {
10     @PrimaryGeneratedColumn()
11     id: number
12
13     @Column("varchar", { length: 255 })
14     plan_name: string
15
16     @Column("text", { nullable: true })
17     description: string
18
19     @CreateDateColumn()
20     created_at: Date
21
22     @UpdateDateColumn()
23     updated_at: Date
24
25     @OneToMany(() => TrainingPlanWorkout, (tpw) => tpw.training_plan)
26     workouts: TrainingPlanWorkout[]
27
28     @OneToMany(() => UserTrainingPlan, (utp) => utp.training_plan)
29     users: UserTrainingPlan[]
30 }

```

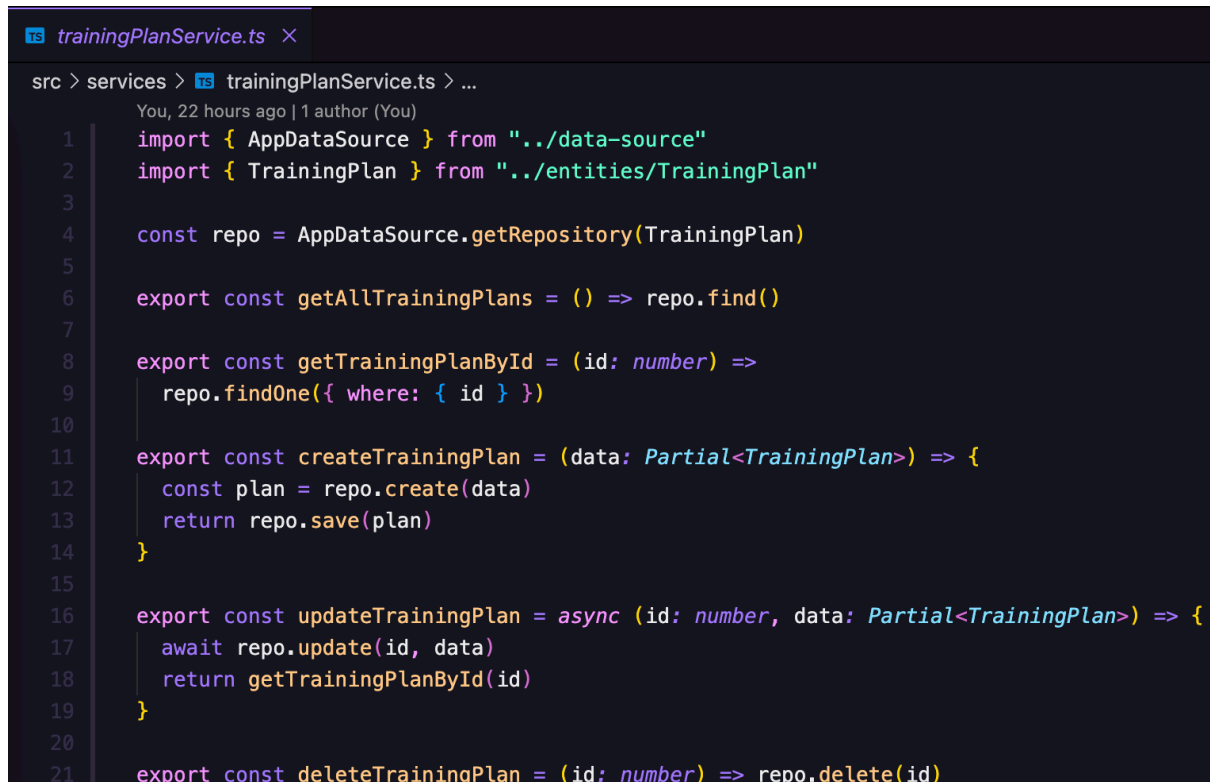
Рисунок 1 - TrainingPlan.ts

- с `UserTrainingPlan` — список пользователей, у которых назначен этот план

Все остальные сущности реализованы в соответствии с указанным примером.

## Services

В рамках задания также был реализован полный набор CRUD-методов (Create, Read, Update, Delete) для работы с моделями, как в примере с TrainingPlan на Рисунке 2.



```
1 import { AppDataSource } from "../data-source"
2 import { TrainingPlan } from "../entities/TrainingPlan"
3
4 const repo = AppDataSource.getRepository(TrainingPlan)
5
6 export const getAllTrainingPlans = () => repo.find()
7
8 export const getTrainingPlanById = (id: number) =>
9   repo.findOne({ where: { id } })
10
11 export const createTrainingPlan = (data: Partial<TrainingPlan>) => {
12   const plan = repo.create(data)
13   return repo.save(plan)
14 }
15
16 export const updateTrainingPlan = async (id: number, data: Partial<TrainingPlan>) => {
17   await repo.update(id, data)
18   return getTrainingPlanById(id)
19 }
20
21 export const deleteTrainingPlan = (id: number) => repo.delete(id)
```

Рисунок 2 - trainingPlanService.ts

Получен репозиторий TrainingPlan через AppDataSource.getRepository(...) — это обеспечивает доступ к сущности в базе данных.

Реализованы функции:

- getAllTrainingPlans() — возвращает все планы тренировок из БД через find().
- getTrainingPlanById(id) — возвращает конкретный план по ID через findOne().
- createTrainingPlan(data) — создаёт новый план:

- используется `repo.create(...)` для создания экземпляра
- затем `repo.save(...)` для сохранения в БД
- `updateTrainingPlan(id, data)` — обновляет план:
  - `repo.update(...)` модифицирует нужную запись
  - затем возвращается обновлённый объект через `getTrainingPlanById(...)`
- `deleteTrainingPlan(id)` — удаляет план по идентификатору через `delete()`.

Методы используют Promise-ориентированный синтаксис TypeORM, позволяют обрабатывать запросы асинхронно.

## Controllers and Routes

А также соответственно была сделана обработка HTTP-запросов (CRUD) средствами Express. Функциональность разбита по слоям: контроллер и роуты на Рисунке 3 и Рисунке 4.

- `getAll` - Получает все планы тренировок из БД и возвращает в виде JSON
- `getById` - Ищет план по ID, возвращает его или 404 при отсутствии
- `create` - Получает данные из `req.body`, создаёт и сохраняет новый план, возвращает 201 Created
- `update` - Обновляет существующий план по ID, возвращает обновлённую версию
- `remove` - Удаляет план по ID, возвращает статус 204 No Content

Создан файл маршрутов, подключающий функции контроллера к конкретным HTTP-методам и URL. Это позволяет клиенту взаимодействовать с сущностью `TrainingPlan` через REST API:

- `/training-plans/` - GET Получить все планы
- `/training-plans/:id` - GET Получить конкретный план по ID
- `/training-plans/` - POST Создать новый план

- /training-plans/:id - PUT Обновить план по ID
- /training-plans/:id - DELETE Удалить план по ID

```
src > controllers > trainingPlanController.ts > ...
You, 22 hours ago | 1 author (You)
1 import { Request, Response } from "express"
2 import * as service from "../services/trainingPlanService"
3
4 export const getAll = async (_, Request, res: Response) => {
5   const plans = await service.getAllTrainingPlans()
6   res.json(plans)
7 }
8
9 export const getById = async (req: Request, res: Response) => {
10   const plan = await service.getTrainingPlanById(Number(req.params.id))
11   plan ? res.json(plan) : res.status(404).json({ error: "Plan not found" })
12 }
13
14 export const create = async (req: Request, res: Response) => {
15   const plan = await service.createTrainingPlan(req.body)
16   res.status(201).json(plan)
17 }
18 | You, 22 hours ago • feat: hw2 is done ...
19 export const update = async (req: Request, res: Response) => {
20   const updated = await service.updateTrainingPlan(Number(req.params.id), req.body)
21   res.json(updated)
22 }
23
24 export const remove = async (req: Request, res: Response) => {
25   await service.deleteTrainingPlan(Number(req.params.id))
26   res.status(204).send()
27 }
```

Рисунок 3 - trainingPlanController.ts

```
src > routes > trainingPlanRoutes.ts > ...
You, 22 hours ago | 1 author (You)
1 import { Router } from "express"
2 import * as controller from "../controllers/trainingPlanController"
3
4 const router = Router()
5
6 router.get("/", controller.getAll)
7 router.get("/:id", controller.getById)
8 router.post("/", controller.create)
9 router.put("/:id", controller.update)
10 router.delete("/:id", controller.remove)
11
12 export default router
```

Рисунок 4 - trainingPlanRoutes.ts

## Swagger

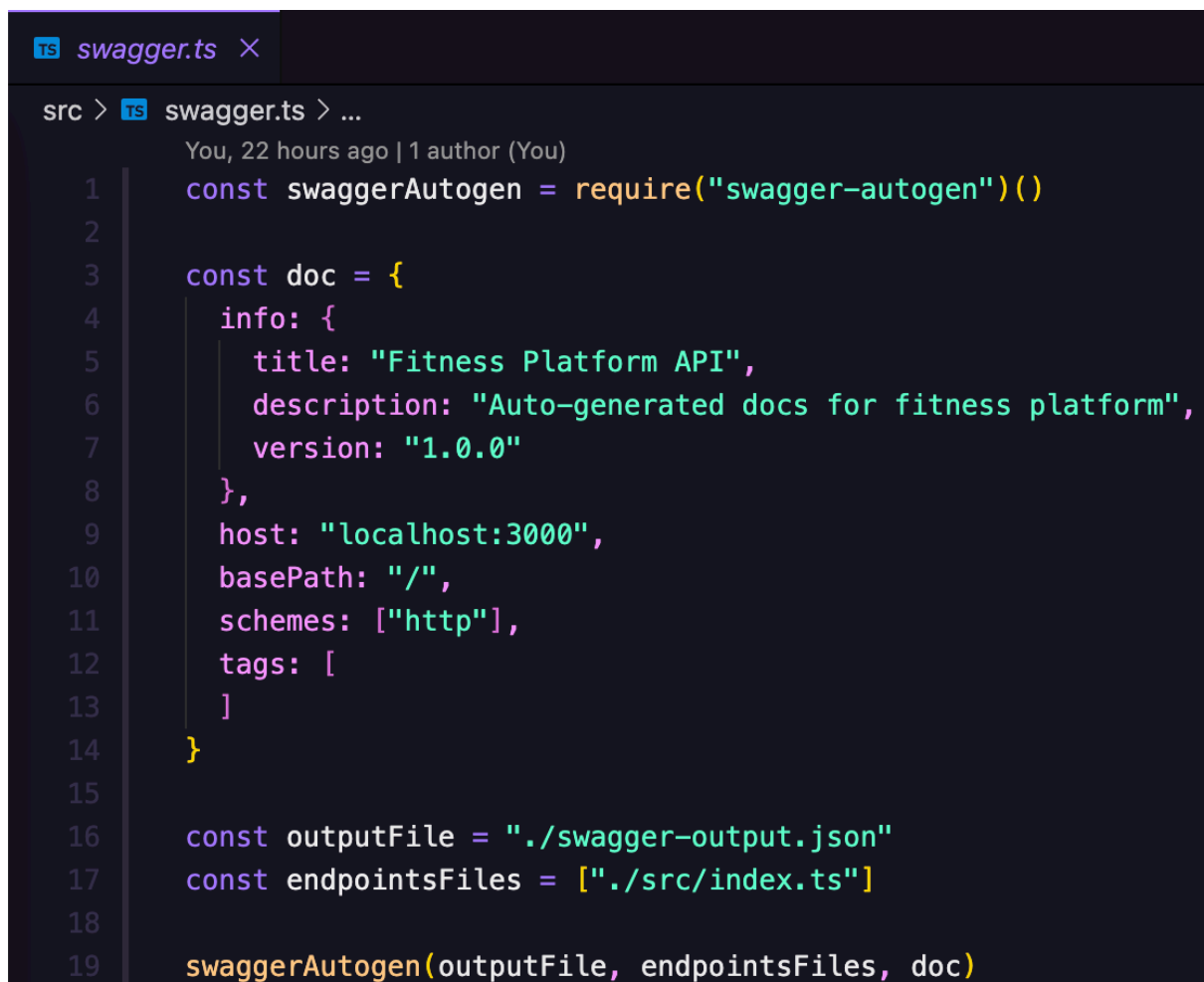
В рамках проекта реализована автоматическая генерация документации REST API с использованием Swagger UI и swagger-autogen.

- swagger-ui-express — отображение Swagger UI в браузере
- swagger-autogen — автоматическая генерация спецификации на основе кода

Создан конфигурационный файл `src/swagger.ts` на Рисунке 5, где определены:

- название и описание API
- основные теги для группировки маршрутов
- список файлов с маршрутами, по которым строится документация

Аннотации на данный момент не были сделаны, так как не было данного требования.



```
src > TS swagger.ts > ...
You, 22 hours ago | 1 author (You)
1  const swaggerAutogen = require("swagger-autogen")()
2
3  const doc = {
4    info: {
5      title: "Fitness Platform API",
6      description: "Auto-generated docs for fitness platform",
7      version: "1.0.0"
8    },
9    host: "localhost:3000",
10   basePath: "/",
11   schemes: ["http"],
12   tags: [
13   ]
14 }
15
16 const outputFile = "./swagger-output.json"
17 const endpointsFiles = ["./src/index.ts"]
18
19 swaggerAutogen(outputFile, endpointsFiles, doc)
```

Рисунок 5 - swagger.ts

# User

```
src > entities > User.ts > User > updated_at
You, 22 hours ago | 1 author (You)

1  import {
2      Entity, PrimaryGeneratedColumn, Column,
3      ManyToOne, OneToMany, JoinColumn, CreateDateColumn, UpdateDateColumn
4  } from "typeorm"
5  import { Role } from "../Role"
6  import { BlogPost } from "../BlogPost"
7  import { BlogComment } from "../BlogComment"
8  import { Order } from "../Order"
9  import { UserProgress } from "../UserProgress"
10 import { UserTrainingPlan } from "../UserTrainingPlan"
11 +
You, 22 hours ago | 1 author (You)
12 @Entity("users")
13 export class User {
14     @PrimaryGeneratedColumn()
15     id: number
16
17     @Column("int")
18     role_id: number
19
20     @ManyToOne(() => Role, (role) => role.users)
21     @JoinColumn({ name: "role_id" })
22     role: Role
23
24     @Column("varchar", { length: 100 })
25     name: string
26
27     @Column("varchar", { length: 100 })
28     email: string
29
30     @Column("varchar", { length: 255 })
31     password_hash: string
32
33     @Column("date", { nullable: true })
34     date_of_birth: Date
35
36     @Column("varchar", { length: 10, nullable: true })
37     gender: string
38
39     @CreateDateColumn()
40     created_at: Date
41
42     @UpdateDateColumn()
43     updated_at: Date
44
45     @OneToMany(() => BlogPost, (post) => post.author)
46     blogPosts: BlogPost[]
47
48     @OneToMany(() => BlogComment, (comment) => comment.user)
49     comments: BlogComment[]
50
51     @OneToMany(() => Order, (order) => order.user)
52     orders: Order[]
53
54     @OneToMany(() => UserProgress, (progress) => progress.user)
55     progress: UserProgress[]
56
57     @OneToMany(() => UserTrainingPlan, (plan) => plan.user)
58     trainingPlans: UserTrainingPlan[]
59 }
```



TS userController.ts X

src > controllers > TS userController.ts > ...

You, 22 hours ago | 1 author (You)

```
1 import { Request, Response } from "express"
2 import * as userService from "../services/userService" You, 22 hours ago •
3
4 export const getAll = async (_, Request, res: Response) => {
5   const users = await userService.getAllUsers()
6   res.json(users)
7 }
8
9 export const getById = async (req: Request, res: Response) => {
10   const user = await userService.getUserById(Number(req.params.id))
11   user ? res.json(user) : res.status(404).json({ error: "User not found" })
12 }
13
14 export const getEmail = async (req: Request, res: Response) => {
15   const user = await userService.getUserByEmail(req.params.email)
16   user ? res.json(user) : res.status(404).json({ error: "User not found" })
17 }
18 +
19 export const create = async (req: Request, res: Response) => {
20   const user = await userService.createUser(req.body)
21   res.status(201).json(user)
22 }
23
24 export const update = async (req: Request, res: Response) => {
25   const updated = await userService.updateUser(Number(req.params.id), req.body)
26   res.json(updated)
27 }
28
29 export const remove = async (req: Request, res: Response) => {
30   await userService.deleteUser(Number(req.params.id))
31   res.status(204).send()
32 }
```

TS userRoutes.ts X

src > routes > TS userRoutes.ts > ...

You, 22 hours ago | 1 author (You)

```
1 import { Router } from "express"
2 import * as userController from "../controllers/userController"
3
4 const router = Router()
5
6 router.get("/", userController.getAll)
7 router.get("/id/:id", userController.getById)
8 router.get("/email/:email", userController.getEmail)
9 router.post("/", userController.create)
10 router.put("/:id", userController.update)
11 router.delete("/:id", userController.remove)
12
13 export default router
```

## **Выводы**

В ходе работы был разработан полноценный backend-сервер на Express + TypeScript с использованием TypeORM. Проект структурирован по слоям, реализованы все сущности базы данных, настроены связи, миграции, CRUD-операции, маршруты и контроллеры. Также подключена Swagger-документация.