САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Домашнее задание 2: Работа с TypeORM

Выполнил:

Лазебный Всеволод

Группа К3344

Проверил:
Добряков Д. И.

Санкт-Петербург

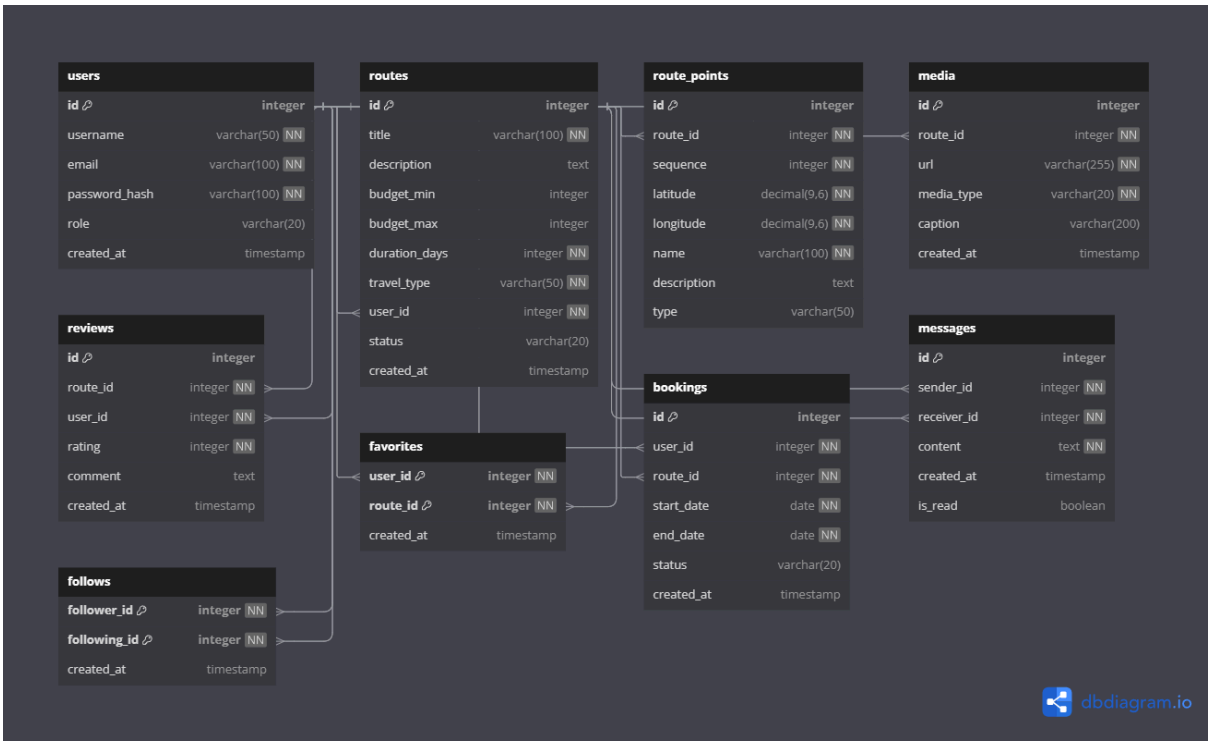
2025 г.

Задача

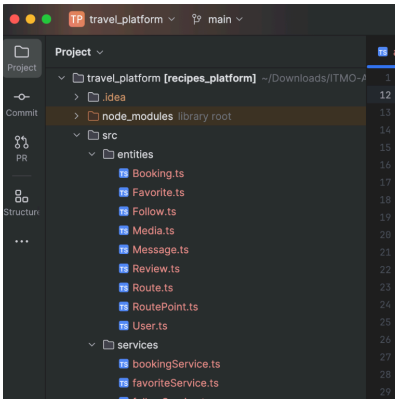
1. Реализовать все модели данных, спроектированные в рамках ДЗ1
2. Реализовать набор из CRUD-методов для работы с моделями данных средствами Express + TypeScript
3. Реализовать API-эндпоинт для получения пользователя по id/email

Ход работы

В рамках ДЗ1 была спроектирована следующая схема данных:



В ходе выполнения второй работы я пошел последовательно по каждому пункту.
В папке entities были созданы все модели данных:



Для примера прикладываю скрипт реализации сущности Route:

Route.ts

```ts
import { Entity, PrimaryGeneratedColumn, Column, ManyToOne, OneToMany } from "typeorm";
import { User } from "./User";
import { RoutePoint } from "./RoutePoint";
import { Media } from "./Media";
import { Review } from "./Review";
import { Favorite } from "./Favorite";
import { Booking } from "./Booking";

@Entity() Show usages
export class Route {
    @PrimaryGeneratedColumn()
    id: number;

    @Column({ length: 100 })
    title: string;

    @Column('text')
    description: string;

    @Column('int')
    budget_min: number;

    @Column('int')
    budget_max: number;
```

```ts
export class Route {
    @Column('int')
    duration_days: number;

    @Column({ length: 50 })
    travel_type: string;

    @Column({ default: 'draft' })
    status: string;

    @Column('timestamp', { default: () : string => 'CURRENT_TIMESTAMP' })
    created_at: Date;

    @ManyToOne(() : typeof User => User, user : User => user.routes)
    user: User;

    @OneToMany(() : typeof RoutePoint => RoutePoint, point : RoutePoint => point.route)
    points: RoutePoint[];

    @OneToMany(() : typeof Media => Media, media : Media => media.route)
    media: Media[];

    @OneToMany(() : typeof Review => Review, review : Review => review.route)
    reviews: Review[];
```

Аналогично были спроектированы и прочие сущности.

Набор из CRUS-методов находится в папке services, не отходя от Route, прикрепил реализацию routeService

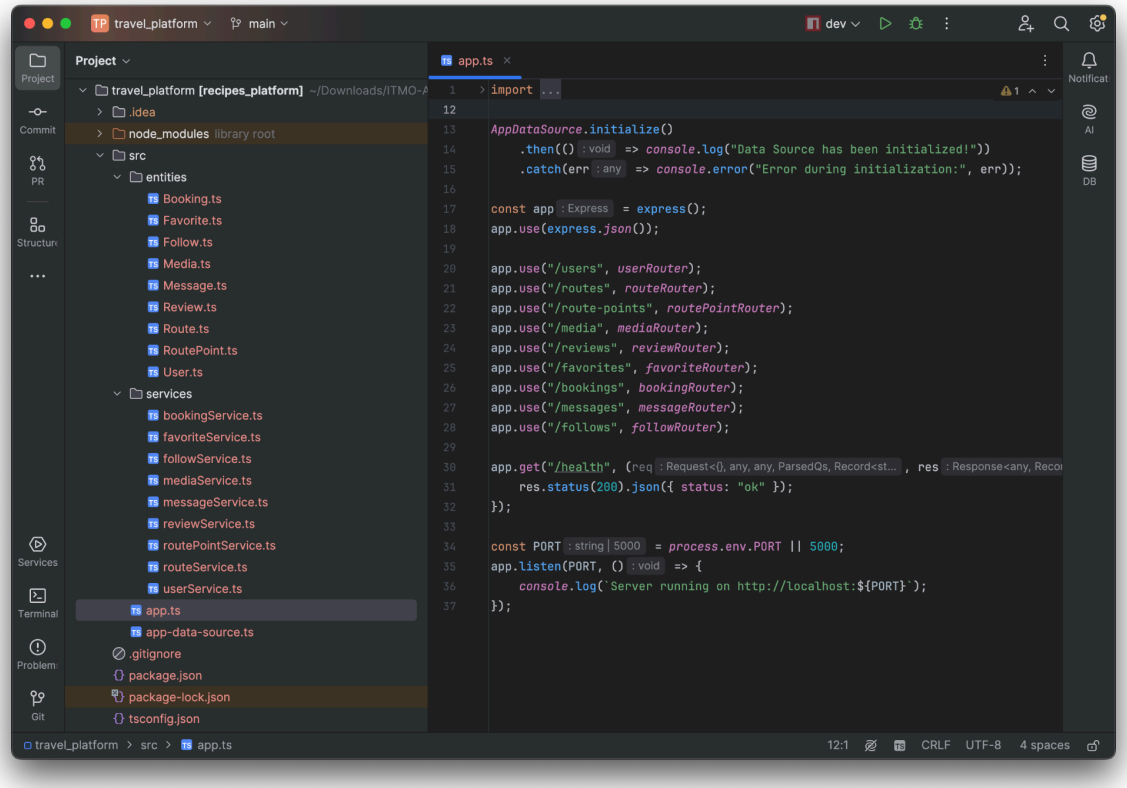
```ts
import express, { Request, Response } from "express";
import { AppDataSource } from "../app-data-source";
import { Route } from "../entities/Route";

const router : Router  = express.Router();

router.get("/", async (req: Request, res: Response) : Promise<void>  => {
    const routes : Route[]  = await AppDataSource.getRepository(Route).find({
        relations: ["user", "points", "media"]
    });
    res.json(routes);
});

router.get("/:id", async (req: Request, res: Response) : Promise<void>  => {
    const route : Route | null  = await AppDataSource.getRepository(Route).findOne({
        where: { id: parseInt(req.params.id) },
        relations: ["user", "points", "media", "reviews"]
    });
    res.json(route);
});

router.post("/", async (req: Request, res: Response) : Promise<void>  => {
    const route : Route[]  = AppDataSource.getRepository(Route).create(req.body);
    const results : Route[]  = await AppDataSource.getRepository(Route).save(route);
```

```ts
router.post("/", async (req: Request, res: Response) : Promise<void>  => {
    res.status(201).json(results);
});

router.put("/:id", async (req: Request, res: Response) : Promise<void>  => {
    const route : Route | null  = await AppDataSource.getRepository(Route).findOneBy({
        id: parseInt(req.params.id)
    });

    if (route) {
        AppDataSource.getRepository(Route).merge(route, req.body);
        const results : Route  = await AppDataSource.getRepository(Route).save(route);
        res.json(results);
    } else {
        res.status(404).json({ message: "Route not found" });
    }
});

router.delete("/:id", async (req: Request, res: Response) : Promise<void>  => {
    const results : DeleteResult  = await AppDataSource.getRepository(Route).delete(req.params.id);
    res.json(results);
});

export default router;  Show usages
```

Для получения пользователя по id/email был реализован API-эндпоинт

```typescript
import express, { Request, Response } from "express";
import { AppDataSource } from "../app-data-source";
import { User } from "../entities/User";

const router : Router = express.Router();

router.get("/", async (req: Request, res: Response) : Promise<void> => {
    const users : User[] = await AppDataSource.getRepository(User).find();
    res.json(users);
});

router.get("/:id", async (req: Request, res: Response) : Promise<void> => {
    const user : User | null = await AppDataSource.getRepository(User).findOneBy({
        id: parseInt(req.params.id)
    });
    res.json(user);
});

router.get("/email/:email", async (req: Request, res: Response) : Promise<void> => {
    const user : User | null = await AppDataSource.getRepository(User).findOneBy({
        email: req.params.email
    });
    res.json(user);
});
```

Таким образом, получилось создать следующую структуру:



Вывод

В ходе работы научился работать с новыми средствами и освоил TypeORM. Также составил четкую структуру для дальнейшей работы.