САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Домашняя работа №2

Выполнил:

Гуторова Инна

Группа К3341

Проверил:
Добряков Д. И.

Санкт-Петербург

2025 г.

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

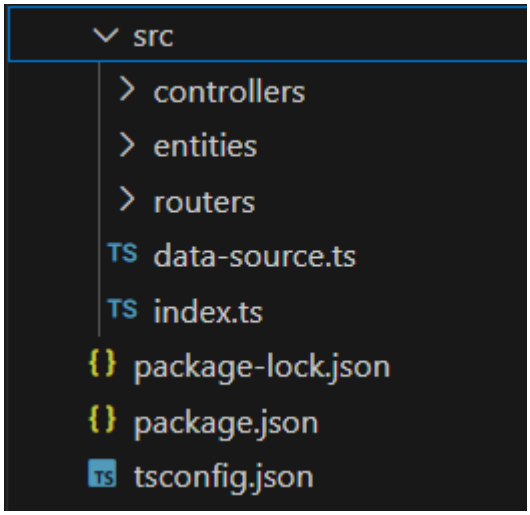
Вариант

Платформа для организации путешествий

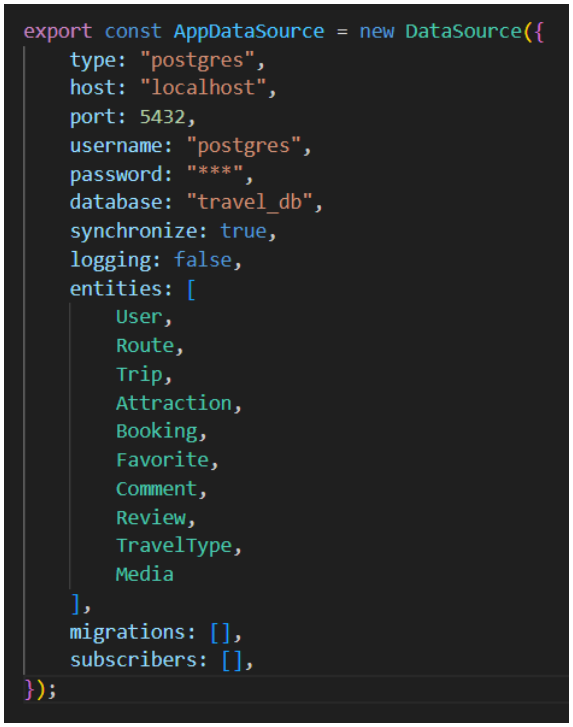
- Вход/Регистрация
- Личный кабинет путешественника (избранные маршруты, бронирования)
- Поиск маршрутов с фильтрацией по бюджету, продолжительности поездки, типу отдыха
- Карточка маршрута с описанием достопримечательностей, отзывами туристов, фото и видео
- Социальные функции (рекомендации, отзывы, общение с другими туристами)

Ход работы

Структура проекта выглядит следующим образом:



В файле data-source происходит подключение к базе данных postgre.



```
export const AppDataSource = new DataSource({
    type: "postgres",
    host: "localhost",
    port: 5432,
    username: "postgres",
    password: "***",
    database: "travel_db",
    synchronize: true,
    logging: false,
    entities: [
        User,
        Route,
        Trip,
        Attraction,
        Booking,
        Favorite,
        Comment,
        Review,
        TravelType,
        Media
    ],
    migrations: [],
    subscribers: [],
});
```

Далее были реализованы все сущности, описанные в домашней работе №1, а именно: User, Route, Trip, Attraction, Booking, Favorite, Comment, Review, Media, TravelType.

Пример: comment

```
import { Entity, PrimaryGeneratedColumn, Column, ManyToOne } from "typeorm";
import User from "./User";
import Route from "./Route";

@Entity()
export default class Comment {
    @PrimaryGeneratedColumn()
    comment_id: number;

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

    @ManyToOne(() => Route, route => route.comments)
    route: Route;

    @Column('text')
    comment_text: string;

    @Column({ type: 'date' })
    date: Date;
}
```

Для каждой сущности создан отдельный контроллер, обрабатывающий CRUD операции (и дополнительно получение пользователя по email). Пример метода контроллера: получение комментария по id

```
export const getCommentById = async (req: Request, res: Response) => {
  try {
    const commentRepository = AppDataSource.getRepository(Comment);
    const comment = await commentRepository.findOne({
      where: { comment_id: parseInt(req.params.id) },
      relations: ['user', 'route']
    });
    comment ? res.json(comment) : res.status(404).json({ message: 'Comment not found' });
  } catch (error: any) {
    res.status(500).json({ message: error.message });
  }
};
```

Далее для каждой сущности создается и роутер.

Пример для комментария:

```ts
import { Router } from 'express';
import {
  createComment,
  getComments,
  getCommentById,
  updateComment,
  deleteComment
} from '../controllers/commentController';

const commentRouter = Router();

commentRouter.post('/', createComment);
commentRouter.get('/', getComments);
commentRouter.get('/:id', getCommentById);
commentRouter.put('/:id', updateComment);
commentRouter.delete('/:id', deleteComment);

export default commentRouter;
```

И все вместе они собираются в index.ts:

```ts
import userRouter from './routers/userRouter';
import routeRouter from './routers/routeRouter';
import tripRouter from './routers/tripRouter';
import attractionRouter from './routers/attractionRouter';
import bookingRouter from './routers/bookingRouter';
import favoriteRouter from './routers/favoriteRouter';
import commentRouter from './routers/commentRouter';
import reviewRouter from './routers/reviewRouter';
import travelTypeRouter from './routers/travelTypeRouter';
import mediaRouter from './routers/mediaRouter';

const app = express();
const PORT = 3000;

app.use(express.json());

AppDataSource.initialize()
  .then(() => console.log("Database connected!"))
  .catch(err => console.error("Database connection error:", err));

app.use('/users', userRouter);
app.use('/routes', routeRouter);
app.use('/trips', tripRouter);
app.use('/attractions', attractionRouter);
app.use('/bookings', bookingRouter);
app.use('/favorites', favoriteRouter);
app.use('/comments', commentRouter);
app.use('/reviews', reviewRouter);
app.use('/travel-types', travelTypeRouter);
app.use('/media', mediaRouter);
```

Вывод

В ходе работы был разработан бэкенд для туристического сервиса на Node.js и Express. Используя TypeORM, был создан API с основными CRUD-операциями для пользователей, маршрутов и бронирований.

Тестирование через Postman показало, что все функции работают корректно: можно создавать пользователей, добавлять маршруты и бронирования и т.д. Получилась рабочая основа, к которой в будущем можно добавить авторизацию, поиск по фильтрам и другие полезные функции для туристического сервиса.