САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Домашняя работа №2

Выполнил:

Кадникова Екатерина

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.

Ход работы

1. Реализация моделей

В рамках ДЗ1 была создана схема данных (см. Рисунок 1).

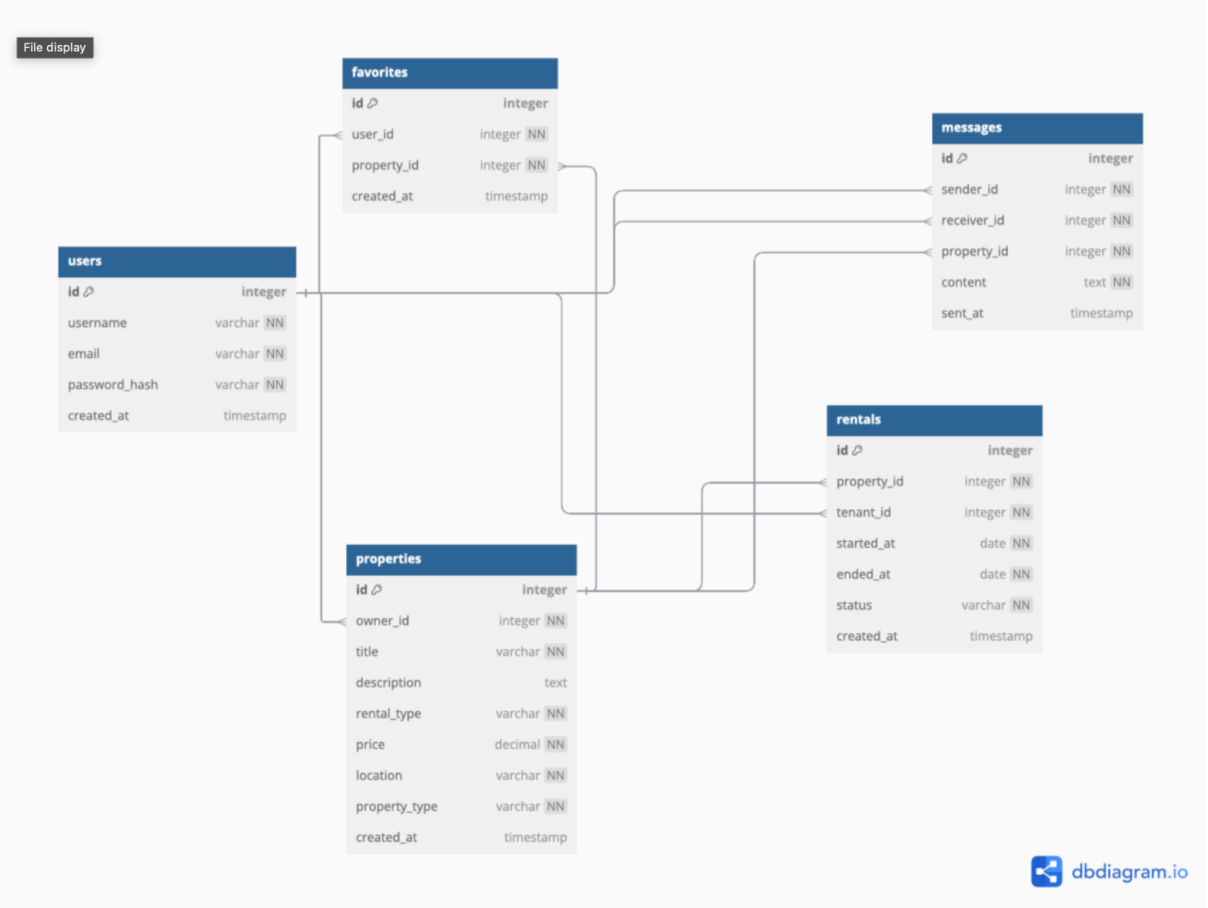

Рисунок 1 - Схема данных

Спроектированные модели были реализованы с помощью TypeORM. Пример реализованной модели - User (см. Листинг 1).

Листинг 1 - Модель User:

```
import { Entity, PrimaryGeneratedColumn, Column, CreateDateColumn,
OneToMany } from "typeorm";
import { Property } from "./Property";
import { Rental } from "./Rental";
import { Message } from "./Message";
import { Favorite } from "./Favorite";

@Entity()
export class User {
   @PrimaryGeneratedColumn()
   id: number;

   @Column({ unique: true })
   username: string;

   @Column({ unique: true })
   email: string;

   @Column()
   password_hash: string;

   @CreateDateColumn()
   created_at: Date;

   @OneToMany(() => Property, property => property.owner)
   properties: Property[];

   @OneToMany(() => Rental, rental => rental.tenant)
   rentals: Rental[];

   @OneToMany(() => Message, message => message.sender)
   sentMessages: Message[];

   @OneToMany(() => Message, message => message.receiver)
   receivedMessages: Message[];

   @OneToMany(() => Favorite, favorite => favorite.user)
   favorites: Favorite[];
}
```

Реализованы поля модели, а также выделены связи модели User с другими моделями.

Аналогичным образом были реализованы оставшиеся модели Favorite, Message, Property, Rental. Также были выделены “справочники” Enum - PropertyType (тип недвижимости), RentalStatus (статус аренды), RentalType (тип аренды).

2. Реализация CRUD-методов

Далее был реализован набор из CRUD-методов для работы с каждой моделью данных средствами Express + TypeScript.

Для каждой модели были реализованы методы для создания, получения, редактирования и удаления. Логика каждого метода реализована в соответствующем контроллере (пример контроллера для пользователя см. на Листинге 2).

Листинг 2 - Контроллер методов для работы с пользователем:

```
import { Request, Response } from "express";
import { AppDataSource } from "../data-source";
import { User } from "../models/User";
import bcrypt from "bcrypt";
import jwt from "jsonwebtoken";
import { JWT_SECRET, JWT_EXPIRES_IN } from "../config/jwt";

const userRepo = AppDataSource.getRepository(User);

export const createUser = async (req: Request, res: Response):
Promise<void> => {
    try {
        const { username, email, password } = req.body;

        const existingUser = await userRepo.findOneBy([{ username }, {
email }]);
        if (existingUser) {
            res.status(400).json({ message: "Username or email already in
use" });
            return;
        }

        const hashedPassword = await bcrypt.hash(password, 10);
        const user = userRepo.create({ username, email, password_hash:
hashedPassword });
        await userRepo.save(user);

        res.status(201).json({ message: "User created successfully", id:
user.id });
    } catch (err) {
        res.status(500).json({ message: "Error creating user", error: err
});
    }
};

export const loginUser = async (req: Request, res: Response):
Promise<void> => {
    try {
        const { usernameOrEmail, password } = req.body;
```

```
            const user = await userRepo.findOneBy([
                { username: usernameOrEmail },
                { email: usernameOrEmail },
            ]);

            if (!user) {
                res.status(404).json({ message: "User not found" });
                return;
            }

            const isValid = await bcrypt.compare(password,
user.password_hash);
            if (!isValid) {
                res.status(401).json({ message: "Invalid password" });
                return;
            }

            const token = jwt.sign({ userId: user.id }, JWT_SECRET, {
                expiresIn: JWT_EXPIRES_IN,
            });

            res.json({ token });
        } catch (err) {
            res.status(500).json({ message: "Login error", error: err });
        }
    };

    export const getAllUsers = async (_: Request, res: Response):
Promise<void> => {
        try {
            const users = await userRepo.find();
            res.json(users);
        } catch (err) {
            res.status(500).json({ message: "Error fetching users", error: err
});
        }
    };

    export const getUserByIdOrEmail = async (req: Request, res: Response):
Promise<void> => {
        try {
            const { identifier } = req.params;

            const user = await userRepo.findOne({
                where: [
                    { id: parseInt(identifier, 10) || 0 },
                    { email: identifier },
                ],
            });

            if (!user) {
                res.status(404).json({ message: "User not found" });
                return;
```

```
            }

            res.json(user);
        } catch (err) {
            res.status(500).json({ message: "Error fetching user", error: err
});
        }
    };

    export const updateUser = async (req: Request, res: Response):
Promise<void> => {
        try {
            const { id } = req.params;
            const { username, email, password_hash: passwordHash } = req.body;

            const user = await userRepo.findOneBy({ id: parseInt(id, 10) });

            if (!user) {
                res.status(404).json({ message: "User not found" });
                return;
            }

            user.username = username ?? user.username;
            user.email = email ?? user.email;
            user.password_hash = passwordHash ?? user.password_hash;

            await userRepo.save(user);
            res.json(user);
        } catch (err) {
            res.status(500).json({ message: "Error updating user", error: err
});
        }
    };

    export const deleteUser = async (req: Request, res: Response):
Promise<void> => {
        try {
            const { id } = req.params;
            const user = await userRepo.findOneBy({ id: parseInt(id, 10) });

            if (!user) {
                res.status(404).json({ message: "User not found" });
                return;
            }

            await userRepo.remove(user);
            res.status(204).send();
        } catch (err) {
            res.status(500).json({ message: "Error deleting user", error: err
});
        }
    };
```

Также были реализованы роутеры со списком эндпоинтов (пример роутера см. на Листинге 3).

Листинг 3 - Роутер userRoutes:

```
import { Router } from "express";
import {
   createUser,
   loginUser,
   getAllUsers,
   getUserByIdOrEmail,
   updateUser,
   deleteUser,
} from "../controllers/userController";

const router = Router();

router.post("/", createUser);
router.post("/login", loginUser);
router.get("/", getAllUsers);
router.get("/:identifier", getUserByIdOrEmail);
router.put("/:id", updateUser);
router.delete("/:id", deleteUser);

export default router;
```

3. Реализация API-эндпоинта для получения пользователя по id/email

Помимо основных методов для работы с моделями был реализован метод для получения пользователя по идентификатору или электронной почты (см. Листинг 4).

Листинг 4 - Метод getUserByIdOrEmail:

```
export const getUserByIdOrEmail = async (req: Request, res: Response):
Promise<void> => {
    try {
        const { identifier } = req.params;

        const user = await userRepo.findOne({
            where: [
                { id: parseInt(identifier, 10) || 0 },
                { email: identifier },
            ],
        });

        if (!user) {
            res.status(404).json({ message: "User not found" });
            return;
        }
```

```
            res.json(user);
        } catch (err) {
            res.status(500).json({ message: "Error fetching user", error: err
});
        }
    };
```

Метод возвращает пользователя на основе введенных данных для поиска - идентификатора (в случае ввода числового значения) или электронной почты (в обратном случае).

Таким образом при выполнении запросов http://localhost:3000/users/1 или http://localhost:3000/users/katya@mail.ru будет получен одинаковый верный результат:

```
    {
        "id": 1,
        "username": "katya",
        "email": "katya@mail.ru",
                                                    "password_hash":
"$2b$10$iIZb6E4kGZXp5uKvBjL9G.gVZVfwK2gZz060ydPoDiHUd0dL.1HqW",
        "created_at": "2025-04-19T16:57:50.382Z"
    }
```

Вывод

В рамках работы были реализованы модели данных в соответствие со схемой из ДЗ1, реализованы CRUD-методы для всевозможной работы со всеми данными, а также дополнительный эндпоинт для расширенного поиска пользователя.