САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа №2

Выполнил:

Ананьев Никита

К3340

Проверил:
Добряков Д. И.

Санкт-Петербург

2025 г.

Задача

Написать свой boilerplate на express + TypeORM + typescript.

Ход работы

Согласно варианту об аренде недвижимости и составленной в ДЗ1 схеме (см. рис. 1), с помощью TypeORM были реализованы следующие модели:

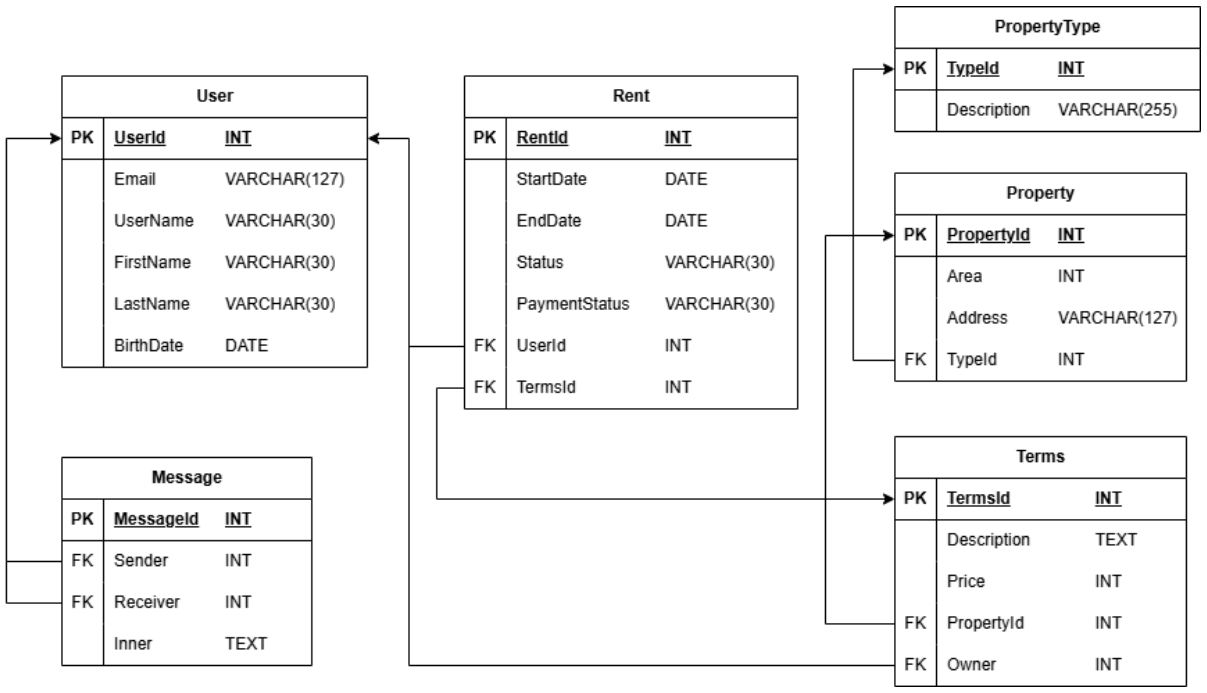

Рисунок 1 - схема отношений сущностей согласно варианту

Модель юзера (см. рис. 2):

```
@Entity()
@Index(["firstName", "lastName"])
export class User {
    @PrimaryGeneratedColumn()
    id!: number

    @Index()
    @Column({ unique: true })
    email!: string

    @Column()
    password!:string;

    @Column()
    firstName!: string

    @Column({ nullable: true })
    lastName?: string

    @Column()
    birthDate!: Date

    @CreateDateColumn()
    registerDate!: Date

    @OneToMany(() => Rent, (rent) => rent.renting)
    rents?: Rent[]

    @OneToMany(() => Property, (prop) => prop.owner)
    properties?: Property[]

    @OneToMany(() => Message, (msg) => msg.sender)
    sentMessages?: Message[]
```

Рисунок 2 - реализация модели юзера

Чтобы весь отчет не состоял из скриншотов кода, буду приводить примеры реализации связанные с юзером, остальные аналогичные, за исключением некоторых изменений.

Помимо модели юзера, были реализованы:

- Модель аренды
- Модель сообщения
- Модель недвижимости
- Модель типа недвижимости

От сущности Terms отказался в своей реализации, т.к. она оказалась излишней.

Далее были реализованы сервисы для работы с бизнес-логикой (см. рис. 3 и рис. 4):

```typescript
import { Repository, ObjectLiteral, FindManyOptions } from "typeorm";
import { NotFoundError } from "../errors/NotFoundError";


export abstract class BaseService<Model extends ObjectLiteral, DTO> {
    constructor(protected readonly repository: Repository<Model>){}

    protected abstract toDto(model: Model): DTO

    async findAll(options?: FindManyOptions<Model>): Promise<DTO[]> {
        const foundModels = await this.repository.find(options)
        return foundModels.map((m: Model) => this.toDto(m))
    }

    async findById(id: number, relations: string[]): Promise<DTO | null> {
        const model: Model | null = await this.repository.findOne({ where: {id: id} as any, relations: relations })
        if (!model)
            throw new NotFoundError("Can't find record with this id")
        return this.toDto(model)
    }

    // async update()
}

```

Рисунок 3 - Базовый класс сервиса

```typescript
export interface IUserService {
    register(registerInfo: CreateUserDto): Promise<ResponseUserDto>;
    login(loginInfo: LoginDto): Promise<void>;
    findAll(options?: FindManyOptions<User>): Promise<ResponseUserDto[]>
    findById(id: number, relations: string[]): Promise<ResponseUserDto | null>
}


@Service('user.service')
export class UserService extends BaseService<User, ResponseUserDto> implements IUserService {
    constructor(protected readonly repository: Repository<User> = AppDataSource.getRepository(User)) {
        super(repository)
    }

    protected toDto(model: User): ResponseUserDto {
        return UserMapper.toDto(model)
    }

    async register(registerInfo: CreateUserDto): Promise<ResponseUserDto> {
        let user: User = UserMapper.toModel(registerInfo)
        try {
            user = await this.repository.save(user)
            return this.toDto(user)
        } catch (error: any) {
            if (error.code == '23505')
                throw new UserAlreadyExistsError("This email is already taken")
            throw new CreationError(error)
        }
    }

    async login(loginInfo: LoginDto): Promise<void> {
        const user: User | null = await this.repository.findOneBy({ email: loginInfo.email })
        if (!user)
            throw new NotFoundError("Can't find user with this email")
        // TODO: token creation
    }
}
```

Рисунок 4 - реализация сервиса юзера

Аналогично были созданы сервисы и для других сущностей.

Для обработки http запросов были добавлены хэндлеры (или иначе - контроллеры) Пример реализации базового контроллера и контроллера для юзеров см. на рис. 5 и рис. 6.

```
19  export abstract class BaseHandler {
20      public readonly router: Router = Router()
21      protected abstract initRoutes(): void
22
23      constructor(){
24          this.initRoutes()
25      }
26
27      protected success(response: Response, data: any, status: HttpCodes) {
28          response.status(status).json({
29              data
30          })
31      }
32
33      protected error(response: Response, status: HttpCodes, message: string) {
34          response.status(status).json({
35              message: message
36          })
37      }
38  }
39
```

Рисунок 5 - базовый хэндлер

```
8   export class UserHandler extends BaseHandler {
9       private readonly service : IUserService
10
11      constructor(service : IUserService) {
12          super()
13          this.service = service
14      }
15
16      protected initRoutes(): void {
17          this.router.get("/",  this.getUserList.bind(this))
18          this.router.get("/:id", this.getUserById.bind(this))
19          this.router.post("/", this.createUser.bind(this))
20          this.router.patch("/:id", this.updateUser.bind(this))
21      }
22
23      async getUserList(req: Request, res: Response) {
24          const userDtos = await this.service.findAll()
25          this.success(res, userDtos, HttpCodes.OK)
26      }
27
28      async getUserById(req: Request, res: Response) {
29          const userDto: ResponseUserDto | null = await this.service.findById(Number(req.params.id), ["rents", "properties"])
30          if (!userDto)
31              this.error(res, HttpCodes.NOT_FOUND, "User not found")
32          this.success(res, userDto, HttpCodes.OK)
33      }
34
35      async createUser(req: Request, res: Response) {
36          try {
37              const userDto = await this.service.register(req.body)
38              console.log(userDto)
39              this.success(res, userDto, HttpCodes.CREATED)
40          } catch (error: any) {
41              if (error instanceof UserAlreadyExistsError)
42                  this.error(res, HttpCodes.CONFLICT, "Email already taken")
43              console.log(error)
44              this.error(res, HttpCodes.INTERNAL_SERVER_ERROR, "Server error")
45          }
46      }
```

Рисунок 6 - хэндлер для юзера

В конечном счете все хэндлеры были добавлены к экземпляру express сервера (см. рис. 7):

```
export class App {
    private readonly app: express.Application

    constructor() {
        AppDataSource
            .initialize()
            .then(() => {
                console.log("Data Source has been initialized!")
            })
            .catch((err) => {
                console.error("Error during Data Source initialization:", err)
            })
        this.app = express()
        this.app.use(express.json());
        this.initHandlers()
    }

    private initHandlers() {
        const userService = new UserService(AppDataSource.getRepository(User))
        const userHandler = new UserHandler(userService)

        const propertyService = new PropertyService(AppDataSource.getRepository(Property))
        const propertyHandler = new PropertyHandler(propertyService)

        const rentService = new RentService(AppDataSource.getRepository(Rent))
        const rentHandler = new RentHandler(rentService)

        const messageService = new MessageService(AppDataSource.getRepository(Message))
        const messageHandler = new MessageHandler(messageService)

        this.app.use("/users", userHandler.router)
        this.app.use("/properties", propertyHandler.router)
        this.app.use("/rents", rentHandler.router)
        this.app.use("/messages", messageHandler.router)
    }

    public listen(port: number) {
        this.app.listen(port, () => {
          console.log(`Server running on port ${port}`);
        });
    }
}
```

Вывод

Средствами TypeORM и express удалось реализовать хорошо структурированный и масштабируемый boilerplate приложения на тему аренды недвижимости.