САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Практическая работа 2

Выполнила:

Фролова Кристина

Группа К3339

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

Вариант: Сервис для аренды недвижимости

1. Реализация моделей

Были выделены следующие модели: User, Rules, Room, Rental, Property, Photo, Message, Living, Flat, CountryHouse, Comfort, Category, Advertisement.

Также были созданные необходимые enum-файлы: AdvertisementStatus, LivingType, RentType, RentalStatus, RoomType.

Пример реализованной модели AdvertisementEntity предоставлен на рисунке 1.

Все оставшиеся модели были созданы по аналогии с показанной сущностью.

```
@Entity({name: 'advertisements'}) Show usages
export class AdvertisementEntity extends  BaseEntity {

    @PrimaryGeneratedColumn()
    id: number;

    @ManyToOne(() : typeof UserEntity  => UserEntity)
    @JoinColumn({name: "owner_id"})
    owner: UserEntity;

    @Column({type: "text"})
    title: string;

    @Column({type: "text"})
    description: string;

    @ManyToOne(() : typeof CategoryEntity  => CategoryEntity)
    @JoinColumn({name: "category_id"})
    category: CategoryEntity;

    @Column({
        type: "enum",
        enum: RentType,
        name: "rent_type"
    })
    rentType: RentType;

    @Column({
        type: "enum",
        enum: AdvertisementStatus,
        default: AdvertisementStatus.PENDING
    })
    status: AdvertisementStatus;

    @CreateDateColumn({name: "created_at"})
    createdAt: Date;

    @Column({type: "decimal", name: "price_per_period"})
    pricePerPeriod: number;

    @Column({type: "decimal", name: "commision"})
    commission: number;

    @Column({type: "decimal", name: "deposit"})
    deposit: number;

    @OneToMany(() : typeof PhotoEntity  => PhotoEntity, photo : PhotoEntity  => photo.advertisement)
    photos: PhotoEntity[];

    @OneToMany(() : typeof MessageEntity  => MessageEntity, message : MessageEntity  => message.advertisement)
    messages: MessageEntity[];

    @OneToOne(() : typeof RulesEntity  => RulesEntity, rules : RulesEntity  => rules.advertisement)
    rules: RulesEntity;
}
```

Рисунок 1 - AdvertisementEntity

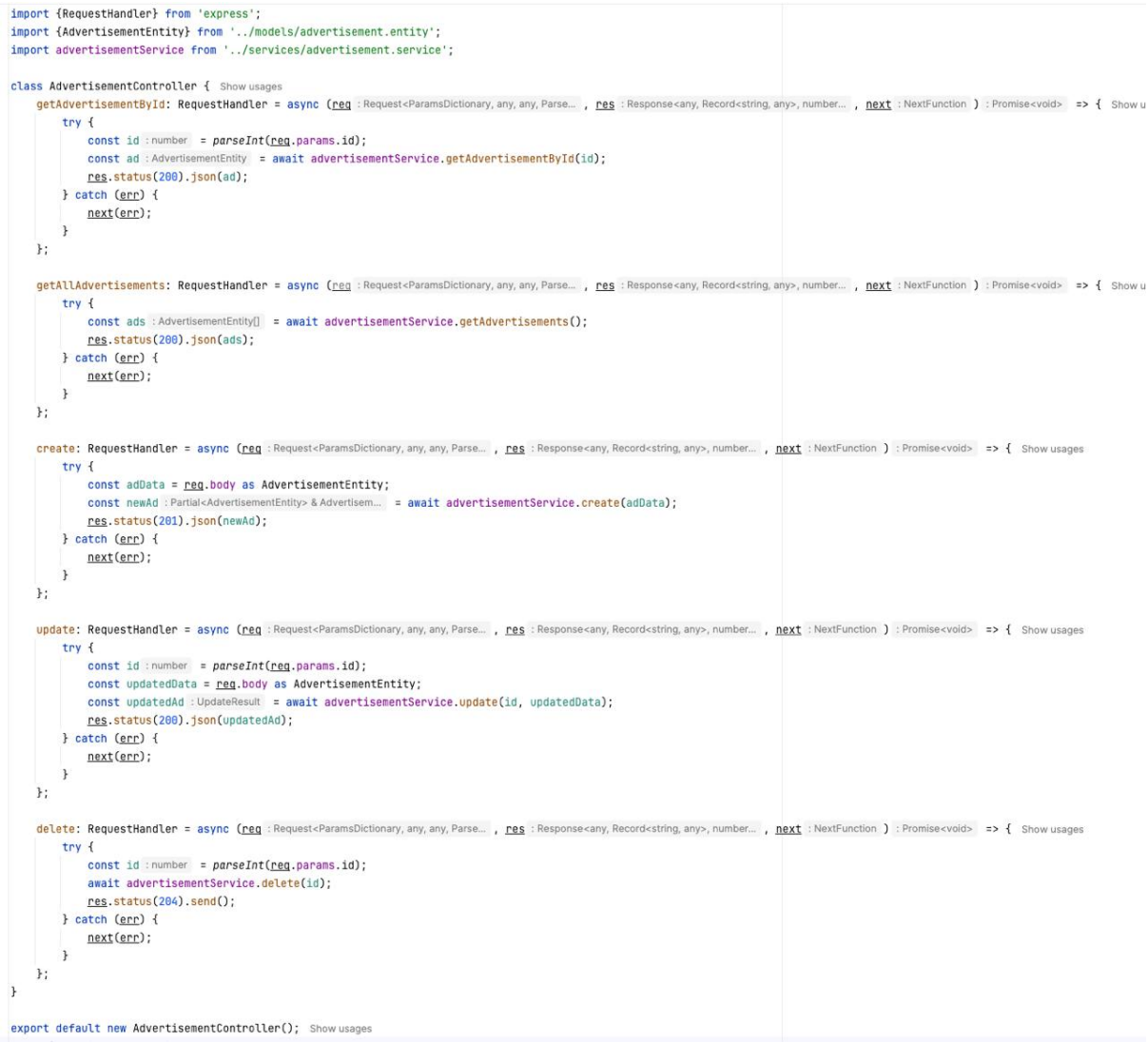
2. Реализовать набор из CRUD-методов для работы с моделями данных средствами Express + TypeScript;

Для каждой модели был создан сервис, в котором определены CRUD-методы, и в котором идёт работа непосредственно с репозиториями. Далее, для тех моделей, которым по бизнес-логике нашего приложения это необходимо, были созданы контроллеры и роутеры. Помимо этого было создано кастомное исключение EntityNotFoundError, которое может работать с любой сущностью, и выбрасывается в случае неудачного поиска объекта по какому-либо из признаков. Также был создан errorHandler, который как раз работает с возникшими ошибками, и обрабатывает их, чтобы клиент получал корректный формат ошибки.

Пример связки service + controller + router для сущности Advertisement представлен на рисунках 2, 3, 4

```
import dataSource from "../config/data-source";
import {AdvertisementEntity} from "../models/advertisement.entity";
import EntityNotFoundError from "../errors/entity-not-found.error";

class AdvertisementService {  Show usages

    private repository : Repository<AdvertisementEntity>  = dataSource.getRepository(AdvertisementEntity);

    async create(advertisement: Partial<AdvertisementEntity>) : Promise<Partial<AdvertisementEntity> & Ad...  {  Show usages
        return await this.repository.save(advertisement);
    }

    async getAdvertisementById(id: number) : Promise<AdvertisementEntity>  {  Show usages
        const advertisement : AdvertisementEntity  = await this.repository.findOneBy({id: id});
        if (!advertisement) {
            throw new EntityNotFoundError(AdvertisementEntity, id, "id");
        }
        return await this.repository.findOneBy({id: id});
    }

    async getAdvertisements() : Promise<AdvertisementEntity[]>  {  Show usages
        return await this.repository.find();
    }

    async delete(id: number) : Promise<DeleteResult>  {  Show usages
        return await this.repository.delete(id);
    }

    async update(id: number, advertisement: Partial<AdvertisementEntity>) : Promise<UpdateResult>  {  Show usages
        const advertisementEntity : AdvertisementEntity  = await this.getAdvertisementById(id);
        return await this.repository.update(id, advertisement);
    }
}

export default new AdvertisementService();  Show usages
```

Рисунок 2 – AdvertisementService



```
import {RequestHandler} from 'express';
import {AdvertisementEntity} from '../models/advertisement.entity';
import advertisementService from '../services/advertisement.service';

class AdvertisementController {  Show usages
    getAdvertisementById: RequestHandler = async (req : Request<ParamsDictionary, any, any, Parse... , res : Response<any, Record<string, any>, number... , next : NextFunction ) : Promise<void> => {  Show u
        try {
            const id : number  = parseInt(req.params.id);
            const ad : AdvertisementEntity  = await advertisementService.getAdvertisementById(id);
            res.status(200).json(ad);
        } catch (err) {
            next(err);
        }
    };

    getAllAdvertisements: RequestHandler = async (req : Request<ParamsDictionary, any, any, Parse... , res : Response<any, Record<string, any>, number... , next : NextFunction ) : Promise<void> => {  Show u
        try {
            const ads : AdvertisementEntity[]  = await advertisementService.getAdvertisements();
            res.status(200).json(ads);
        } catch (err) {
            next(err);
        }
    };

    create: RequestHandler = async (req : Request<ParamsDictionary, any, any, Parse... , res : Response<any, Record<string, any>, number... , next : NextFunction ) : Promise<void> => {  Show usages
        try {
            const adData = req.body as AdvertisementEntity;
            const newAd : Partial<AdvertisementEntity> & Advertisem...  = await advertisementService.create(adData);
            res.status(201).json(newAd);
        } catch (err) {
            next(err);
        }
    };

    update: RequestHandler = async (req : Request<ParamsDictionary, any, any, Parse... , res : Response<any, Record<string, any>, number... , next : NextFunction ) : Promise<void> => {  Show usages
        try {
            const id : number  = parseInt(req.params.id);
            const updatedData = req.body as AdvertisementEntity;
            const updatedAd : UpdateResult  = await advertisementService.update(id, updatedData);
            res.status(200).json(updatedAd);
        } catch (err) {
            next(err);
        }
    };

    delete: RequestHandler = async (req : Request<ParamsDictionary, any, any, Parse... , res : Response<any, Record<string, any>, number... , next : NextFunction ) : Promise<void> => {  Show usages
        try {
            const id : number  = parseInt(req.params.id);
            await advertisementService.delete(id);
            res.status(204).send();
        } catch (err) {
            next(err);
        }
    };
}

export default new AdvertisementController();  Show usages
```

Рисунок 3 – AdvertisementController

```
import { Router } from 'express';
import advertisementController from '../controllers/advertisement.controller';

const router : Router  = Router();

router.get('/:id', advertisementController.getAdvertisementById);
router.get('/', advertisementController.getAllAdvertisements);
router.post('/', advertisementController.create);
router.put('/:id', advertisementController.update);
router.delete('/:id', advertisementController.delete);

export default router;  Show usages
```

Рисунок 4 - AdvertisementRouter

3. Реализовать API-эндпоинт для получения пользователя по id/email.

Эта задача выполнена по аналогии со структурой из прошлого задания. Для сущности User был сделан отдельный сервис, контроллер и роутер. В случае, если нет User с заданным id или email, то выбрасывается исключение EntityNotFoundError, а клиент получает 404 статус ошибки с подробным описанием.

Реализация service, controller и router представлены на рисунках 5, 6, 7.

```
class UserService {  Show usages

    private repository : Repository<UserEntity>  = dataSource.getRepository(UserEntity);

    async getUserById(id: number) : Promise<UserEntity>  {  Show usages
        const user : UserEntity  = await this.repository.findOneBy({id: id});
        if (!user) {
            throw new EntityNotFoundError(UserEntity, id, "id");
        }
        return user;
    }

    async getUserByMail(mail: string) : Promise<UserEntity>  {  Show usages
        const user : UserEntity  = await this.repository.findOneBy({mail: mail})
        if (!user) {
            throw new EntityNotFoundError(UserEntity, mail, "mail");
        }
        return user;
    }
}

export default new UserService();  Show usages
```

Рисунок 5 – UserService

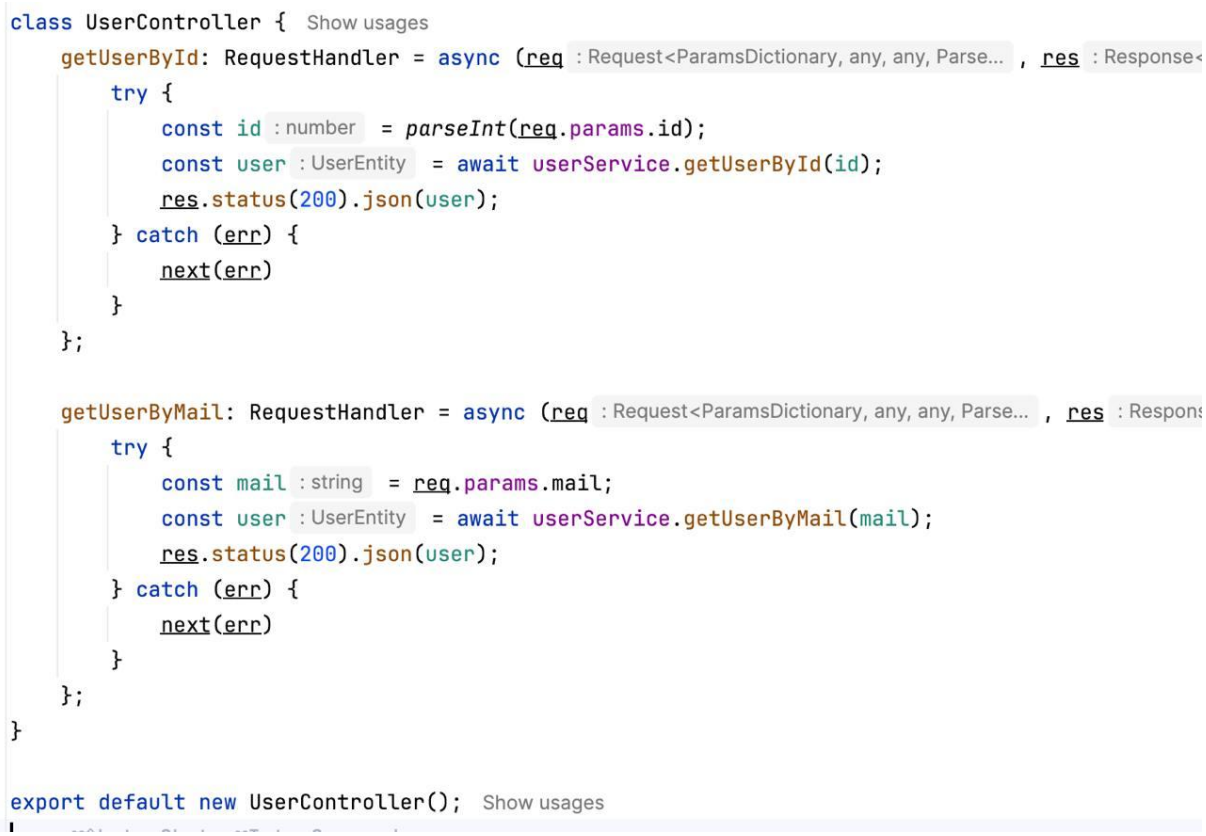


Рисунок 6 - UserController

Рисунок 7 – UserService

Вывод

В рамках работы были созданы модели, спроектированные в рамках ДЗ1, для моделей были созданы сервисы, в которых содержатся CRUD-методы, работа которых реализована при помощи TypeORM; контроллеры и роутеры. Было написано кастомное исключение а также обработчик ошибок. Вся работа велась при помощи express и TypeScript.