САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Домашняя работа №2

Выполнила:

Платонова Александра

Группа K3339

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 данныхсредствами Express + TypeScript;
- реализовать API-эндпоинт для получения пользователя по id/email;

Ход работы

В ходе выполнения работы были реализованы модели данных, спроектированные в ДЗ1, программный код представлен в приложении А. Для реализации CRUD-методов был создан базовый репозиторий:

```
export class BaseRepository<T> {
  protected repository: Repository<T>;

  constructor(repository: Repository<T>) {
    this.repository = repository;
  }

  async findAll(options?: FindManyOptions<T>): Promise<T[]> {
    return this.repository.find(options);
  }

  async findOne(id: number, options?: FindOneOptions<T>): Promise<T | null> {
    return this.repository.findOne({ where: { id }, ...options } as any);
  }

  async create(data: Partial<T>): Promise<T> {
    const entity = this.repository.create(data);
    return this.repository.save(entity);
  }

  async update(id: number, data: Partial<T>): Promise<T | null> {
    await this.repository.update(id, data);
    return this.findOne(id);
  }

  async delete(id: number): Promise<void> {
    await this.repository.delete(id);
  }
}
```

Далее, основываясь на базовом репозитории, были добавлены реализации для каждой модели, они представлены в приложении А, листинг А. 2.

Для получения пользователя по id/email также был реализован репозиторий:

```
export class ResidentRepository {
  private repository: Repository<Resident>;

  constructor() {
    this.repository = AppDataSource.getRepository(Resident);
  }
  async findById(id: number): Promise<Resident | null> {
    return this.repository.findOne({
      where: { id },
      relations: ['check_ins']
    });
  }
  async findByEmail(email: string): Promise<Resident | null> {
    return this.repository.findOne({
      where: { email },
      relations: ['check_ins']
    });
  }
  async findByIdOrEmail(idOrEmail: number | string): Promise<Resident | null> {
    if (typeof idOrEmail === 'number') {
      return this.findById(idOrEmail);
    } else {
      return this.findByEmail(idOrEmail);
    }
  }
}
```

Далее был создан контроллер, для обработки http-запросов.

```
export class ResidentController {
  private residentRepository = new ResidentRepository();

  async getResident(req: Request, res: Response) {
    try {
      const { id, email } = req.query;

      if (!id && !email) {
        return res.status(400).json({ error: 'введите id или email' });
      }
```

```
      let resident: Resident | null = null;

      if (id) {
        const residentId = parseInt(id as string);
        if (isNaN(residentId)) {
          return res.status(400).json({ error: 'Неверный формат id' });
        }
        resident = await this.residentRepository.findById(residentId);
      } else if (email) {
        resident = await this.residentRepository.findByEmail(email as string
);
      }

      if (resident) {
        return res.json(resident);
      } else {
        return res.status(404).json({ error: 'Пользователь не найден' });
      }
    } catch (error) {
      console.error('Error fetching resident:', error);
      return res.status(500).json({ error: 'Internal server error' });
    }
  }
}
```

И, непосредственно, энд-поинт для получения данных:

```
const router = express.Router();
const residentController = new ResidentController();

// Ex: GET /api/residents?id=123 или GET /api/residents?email=user@example.com
router.get('/residents', residentController.getResident);
```

Вывод:

В ходе выполнения работы были реализованы модели, спроектированные ранее, создан базовый репозиторий для crud-операция и написаны реализации этого репозитория для каждой сущности. Также был создан API-эндпоинт для получения пользователя по id/email в качестве передаваемых параметром в запросе.

ПРИЛОЖЕНИЕ А

Программный код

Описание классов сущностей базы данных приведено в листинге А.1.

Листинг А.1. – Сущности БД

```
@Entity()
export class Address {
  @PrimaryGeneratedColumn()
  id: number;
  @Column({ length: 50 })
  city_district: string;
  @Column({ length: 100 })
  street: string;
  @Column({ length: 20 })
  zip_code: string;
  @OneToOne(() => Hostel, (hostel) => hostel.address)
  hostel: Hostel;
}
@Entity()
export class Hostel {
  @PrimaryGeneratedColumn()
  id: number;
  @Column({ length: 100, default: 'name of the hostel' })
  name: string;
  @Column()
  house_num: number;
```

```
  @Column()
  building: number;
  @OneToOne(() => Address, (address) => address.hostel)
  @JoinColumn()
  address: Address;
  @ManyToOne(() => Organization, (organization) => organization.hostels)
  organization: Organization;
  @OneToMany(() => Room, (room) => room.hostel)
  rooms: Room[];
}
@Entity()
export class Room {
  @PrimaryGeneratedColumn()
  id: number;
  @Column()
  floor: number;
  @Column()
  beds: number;
  @Column('float')
  area: number;
  @Column()
  busy_beds: number;

  @ManyToOne(() => Hostel, (hostel) => hostel.rooms)
  hostel: Hostel;
```

```
  @OneToMany(() => CheckInOut, (checkIn) => checkIn.room)
  checkIns: CheckInOut[];
}

@Entity()
export class CheckInOut {
  @PrimaryGeneratedColumn()
  id: number;
  @Column()
  doc_num: number;
  @Column({ type: 'date', default: () => 'CURRENT_DATE' })
  date_of_issue: Date;
  @Column({ type: 'text', nullable: true })
  comment: string;
  @Column({ type: 'text', nullable: true })
  check_out_reason: string;
  @Column({ type: 'date' })
  date_of_checkout: Date;
  @Column({ length: 100 })
  doc_name: string;
  @Column({ type: 'date', default: () => 'CURRENT_DATE' })
  date_of_start: Date;
  @ManyToOne(() => Resident, (resident) => resident.checkIns)
  resident: Resident;
  @ManyToOne(() => Room, (room) => room.checkIns)
```

```
  room: Room;
  @OneToOne(() => Payment, (payment) => payment.checkInOut)
  payment: Payment;
}
@Entity()
export class Payment {
  @PrimaryGeneratedColumn()
  id: number;
  @Column('float')
  amount: number;
  @Column({ length: 50 })
  status: 'p' | 'np' | 'pp';
  @Column({ type: 'date', default: () => 'CURRENT_DATE' })
  date_pay: Date;
  @ManyToOne(() => CheckInOut, (checkInOut) => checkInOut.payment)
  checkInOut: CheckInOut;
}
@Entity()
export class Resident {
  @PrimaryGeneratedColumn()
  id: number;
  @Column({ unique: true })
  registration_number: string;
  @Column({ length: 500 })
  full_name: string;
```

```
@Column({ length: 100, unique: true })
  email: string;
  @Column({ default: false })
  has_children: boolean;
  @Column({ length: 20, unique: true })
  passport_number: string;
  @Column({ type: 'date' })
  passport_issue_date: Date;
  @ManyToOne(() => PassportOffice, (office) => office.residents)
  passport_issuer: PassportOffice;
  @OneToMany(() => CheckInOut, (checkIn) => checkIn.resident)
  check_ins: CheckInOut[];
}
```

Листинг А.2. – Реализация репозиториев

```
import { Address } from '../entities/Address';
import { BaseRepository } from './BaseRepository';
import { Repository } from 'typeorm';
import { AppDataSource } from '../data-source';

export class AddressRepository extends BaseRepository<Address> {
  constructor() {
    super(AppDataSource.getRepository(Address));
  }

  async findByZipCode(zipCode: string): Promise<Address[]> {
    return this.repository.find({ where: { zip_code: zipCode } });
  }
}
import { Hostel } from '../entities/Hostel';
import { BaseRepository } from './BaseRepository';
import { Repository } from 'typeorm';
import { AppDataSource } from '../data-source';

export class HostelRepository extends BaseRepository<Hostel> {
```

```
  constructor() {
    super(AppDataSource.getRepository(Hostel));
  }

  // Дополнительные методы специфичные для Hostel
  async findByName(name: string): Promise<Hostel[]> {
    return this.repository.find({ where: { name } });
  }

  async findByOrganization(organizationId: number): Promise<Hostel[]> {
    return this.repository.find({
      where: { organization: { id: organizationId } },
      relations: ['organization']
    });
  }
}
```

```
import { Room } from '../entities/Room';
import { BaseRepository } from './BaseRepository';
import { Repository } from 'typeorm';
import { AppDataSource } from '../data-source';

export class RoomRepository extends BaseRepository<Room> {
  constructor() {
    super(AppDataSource.getRepository(Room));
  }

  async findByHostel(hostelId: number): Promise<Room[]> {
    return this.repository.find({
      where: { hostel: { id: hostelId } },
      relations: ['hostel']
    });
  }

  async findAvailableRooms(): Promise<Room[]> {
    return this.repository.find({
      where: { free_place: MoreThan(0)  }
  }
}
```

```
import { CheckInOut } from '../entities/CheckInOut';
import { BaseRepository } from './BaseRepository';
import { Repository } from 'typeorm';
import { AppDataSource } from '../data-source';

export class CheckInOutRepository extends BaseRepository<CheckInOut> {
```

```
  constructor() {
    super(AppDataSource.getRepository(CheckInOut));
  }
  async findByResident(residentId: number): Promise<CheckInOut[]> {
    return this.repository.find({
      where: { resident: { id: residentId } },
      relations: ['resident']
    });
  }

  async findByRoom(roomId: number): Promise<CheckInOut[]> {
    return this.repository.find({
      where: { room: { id: roomId } },
      relations: ['room']
    });
  }

  async findActiveCheckIns(): Promise<CheckInOut[]> {
    return this.repository.find({
      where: { date_of_checkout: MoreThan(new Date()) },
      relations: ['resident', 'room']
    });
  }
}
```

```
import { Payment } from '../entities/Payment';
import { BaseRepository } from './BaseRepository';
import { Repository } from 'typeorm';
import { AppDataSource } from '../data-source';

export class PaymentRepository extends BaseRepository<Payment> {
  constructor() {
    super(AppDataSource.getRepository(Payment));
  }

  async findByStatus(status: 'p' | 'np' | 'pp'): Promise<Payment[]> {
    return this.repository.find({ where: { status } });
  }

  async findByCheckInOut(checkInOutId: number): Promise<Payment[]> {
    return this.repository.find({
      where: { checkInOut: { id: checkInOutId } },
      relations: ['checkInOut']
    });
  }
```