САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа №1

Выполнила:

Платонова Александра

Группа К3339

Проверил:
Добряков Д. И.

Санкт-Петербург

2025 г.

Задача

Требуется разработать шаблон проекта (boilerplate) с использованием следующих технологий:

- Express.js
- TypeORM
- TypeScript

Структура проекта должна включать четкое разделение на следующие компоненты:

- модели (Models) – сущности базы данных, определяемые с помощью TypeORM.
- контроллеры (Controllers) – модули, обрабатывающие бизнес-логику и HTTP-запросы.
- роуты (Routes) – маршруты API, связывающие endpoints с соответствующими контроллерами.

Описание предметной области:

Прикладное программное обеспечение деятельности отдела заселения муниципальных общежитий администрации города. В ведении администрации города находится несколько десятков общежитий. Раньше они принадлежали предприятиям города, а теперь, после банкротства предприятий, все эти общежития переданы муниципальным властям. В последние годы бесплатные квартиры гражданам города практически не предоставляются, а количество малоимущих жителей, нуждающихся в жилье, растет. Хоть как-то улучшить жилищные условия этой категории граждан позволяет наличие муниципальных общежитий. Получить четкую картину их заселения позволит данное программное обеспечение. База данных отдела содержит информацию об общежитиях, комнатах общежитий и проживающих.

Ход работы

На первом этапе была спроектирована схема базы данных приложения, соответствующая третьей нормальной форме. На рисунке 1 представлена инфологическая модель базы данных.

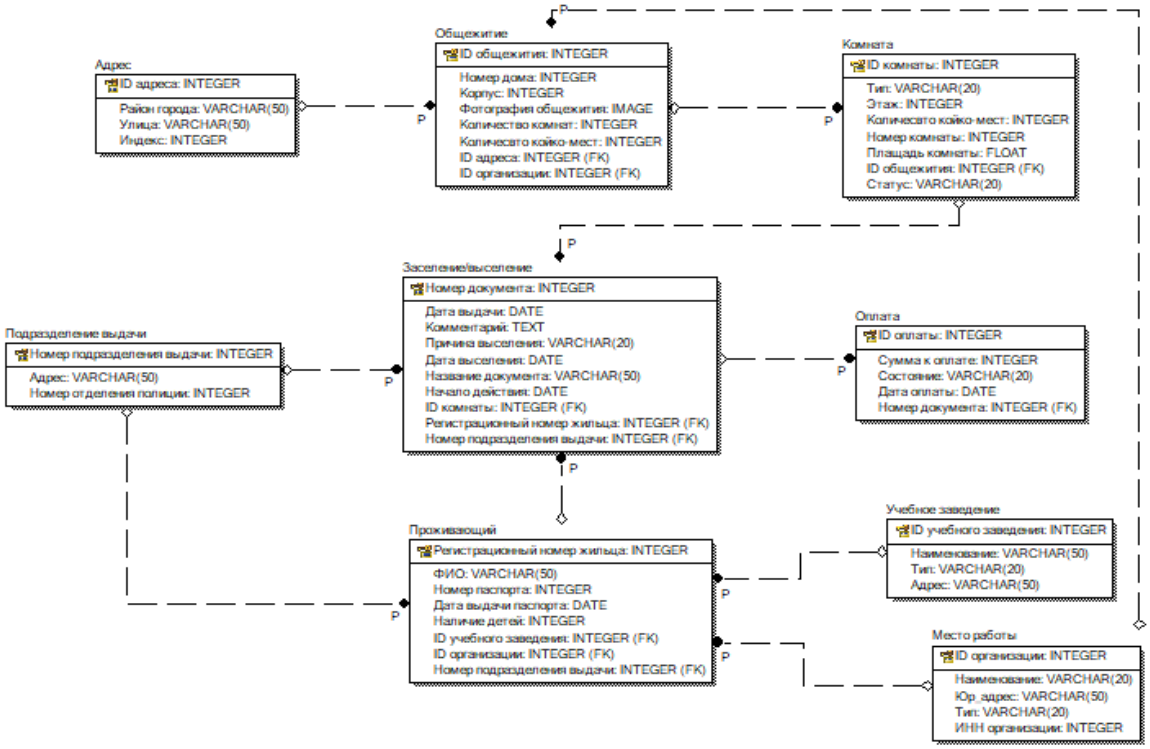

Рисунок 1 – Схема инфологической модели базы данных в нотации IDEF1X

Далее была проведена настройка окружения: инициализирован Node.js проект с использованием TypeScript, установлены и настроены основные зависимости: Express.js, TypeORM и вспомогательные библиотеки, настроена конфигурация TypeScript, создана базовая структура каталогов проекта.

На следующем этапе была разработана модульная структура проекта с четким разделением на компоненты, определены основные слои приложения: модели, контроллеры, роуты.

Затем была реализована база данных. Созданы сущности предметной области на TypeORM:

- организации
- адреса зданий
- общежития
- комнаты
- проживающие
- записи о заселении/выселении
- платежи

Были определены связи между сущностями (один-ко-многим, многие-к-одному).

Далее производилась реализация бизнес-логики, а именно:

- реализованы контроллеры для обработки HTTP-запросов
- для каждой сущности созданы CRUD-операции

Программный код представлен в приложении А.

Вывод

В ходе выполнения лабораторной работы был разработан шаблон веб-сервиса для автоматизации процессов заселения и выселения в муниципальных общежитиях. Основной акцент был сделан на создание надежного и масштабируемого проекта, который может быть использован как основа для будущей системы управления жилым фондом. На первом этапе проведено исследование предметной области, в ходе которого были выявлены ключевые бизнес-процессы и требования к системе. На основе проведенного анализа разработана комплексная модель данных, учитывающая все аспекты работы с общежитиями. Для реализации модели выбрана связка Express.js + TypeORM + TypeScript, обеспечивающая

высокую производительность и надежность. Все поставленные задачи были выполнены.

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;
}
```

Описание контроллеров приведено в листинге А.2.

Листинг А.2. – Контроллеры

```
export class AddressController {
  private addressRepository = AppDataSource.getRepository(Address);
  async create(req: Request, res: Response) {
    const address = this.addressRepository.create(req.body);
    await this.addressRepository.save(address);
    res.status(201).json(address);
```

```
  }
}
export class RoomController {
  private roomRepository = AppDataSource.getRepository(Room);
  async getAll(req: Request, res: Response) {
    const rooms = await this.roomRepository.find({
      where: { hostel: { id: parseInt(req.query.hostel_id as string) } },
      relations: ['hostel']
    });
    res.json(rooms);
  }
  async update(req: Request, res: Response) {
    await this.roomRepository.update(req.params.id, req.body);
    const room = await this.roomRepository.findOneBy({ id: parseInt(req.params.id) });
    res.json(room);
  }
}
export class CheckInOutController {
  private checkInRepository = AppDataSource.getRepository(CheckInOut);
  async create(req: Request, res: Response) {
    const checkIn = this.checkInRepository.create(req.body);
    await this.checkInRepository.save(checkIn);
    res.status(201).json(checkIn);
  }
  async getAll(req: Request, res: Response) {
```

```
    const checkIns = await this.checkInRepository.find({
      where: {
        resident: { id: parseInt(req.query.resident_id as string) },
        room: { id: parseInt(req.query.room_id as string) }
      },
      relations: ['resident', 'room']
    });
    res.json(checkIns);
  }
}
export class PaymentController {
  private paymentRepository = AppDataSource.getRepository(Payment);
  async create(req: Request, res: Response) {
    const payment = this.paymentRepository.create(req.body);
    await this.paymentRepository.save(payment);
    res.status(201).json(payment);
  }
  async updateStatus(req: Request, res: Response) {
    await this.paymentRepository.update(req.params.id, { status: req.body.status });
    const payment = await this.paymentRepository.findOneBy({ id: parseInt(req.params.id) });
    res.json(payment);
  }
}
```

Описание роутов для всех сущностей представлено на листинге А.3.

Листинг А.3. – Роуты

```
// Организации
router.get('/organizations', organizationController.getAll.bind(organizationController));
router.post('/organizations', organizationController.create.bind(organizationController));
router.get('/organizations/:id', organizationController.getById.bind(organizationController));

// Общежития
router.get('/hostels', hostelController.getAll.bind(hostelController));
router.get('/hostels/:id', hostelController.getById.bind(hostelController));
router.post('/hostels', hostelController.create.bind(hostelController));

// Адреса
router.post('/addresses', addressController.create.bind(addressController));
router.get('/addresses/:id', addressController.getById.bind(addressController));

// Комнаты
router.get('/rooms', roomController.getAll.bind(roomController));
router.put('/rooms/:id', roomController.update.bind(roomController));
router.post('/rooms', roomController.create.bind(roomController));

// Проживающие
router.get('/residents', residentController.getAll.bind(residentController));
router.post('/residents', residentController.create.bind(residentController));
router.put('/residents/:id', residentController.update.bind(residentController));
router.get('/residents/:id', residentController.getById.bind(residentController));
```

```
// Заселения/выселения
router.get('/check-ins', checkInOutController.getAll.bind(checkInOutController));
router.post('/check-ins', checkInOutController.create.bind(checkInOutController));
router.put('/check-ins/:id', checkInOutController.update.bind(checkInOutController));
router.delete('/check-ins/:id', checkInOutController.delete.bind(checkInOutController));

// Платежи
router.get('/payments', paymentController.getAll.bind(paymentController));
router.post('/payments', paymentController.create.bind(paymentController));
router.patch('/payments/:id/status',
paymentController.updateStatus.bind(paymentController));
router.get('/payments/:id', paymentController.getById.bind(paymentController));
```