

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Бэк-энд разработка

Отчет

Домашнее задание №2

Выполнила:
Лапшина Екатерина
Алексеевна
Группа К3340

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. Модели данных (Entities)

Реализованы следующие сущности:

User (Пользователь)

```
@Entity("users")
export class User {
  @PrimaryGeneratedColumn("increment")
  UserID!: number;

  @Column({ type: "varchar", length: 150, unique: true })
  username!: string;

  @Column({ type: "varchar", length: 150 })
  first_name!: string;

  @Column({ type: "varchar", length: 150 })
  last_name!: string;

  @Column({ type: "varchar", length: 254, unique: true })
  email!: string;

  @Column({ type: "varchar", length: 128 })
  password!: string;

  @Column({ type: "boolean", default: false })
  is_staff!: boolean;

  @Column({ type: "boolean", default: true })
  is_active!: boolean;

  @Column({ type: "boolean", default: false })
  is_superuser!: boolean;

  @Column({ type: "datetime", nullable: true })
  last_login!: Date;

  @CreateDateColumn()
  date_joined!: Date;

  @UpdateDateColumn()
  updated_at!: Date;

  @OneToMany(() => Contract, contract => contract.AgentID)
  agentContracts!: Contract[];

  @OneToMany(() => Contract, contract => contract.ClientID)
  clientContracts!: Contract[];
}
```

Apartment (Квартира)

```

@Entity("apartments")
export class Apartment {
    @PrimaryGeneratedColumn("increment")
    ApartmentID!: number;

    @Column({ type: "int" })
    Number!: number;

    @Column({ type: "int" })
    Square!: number;

    @Column({ type: "varchar", length: 255, nullable: true })
    Description!: string;

    @Column({ type: "varchar", length: 255, nullable: true })
    Photo!: string;

    @Column({ type: "int" })
    Cost!: number;

    @ManyToOne(() => Building, building => building.Apartments)
    Building!: Building;

    @OneToMany(() => Contract, contract => contract.ApartmentID)
    Contracts!: Contract[];
}

```

Contract (Контракт)

```

@Entity("contracts")
export class Contract {
    @PrimaryGeneratedColumn("increment")
    ContractID!: number;

    @ManyToOne(() => User, user => user.agentContracts)
    @JoinColumn({ name: "AgentID" })
    AgentID!: User;

    @ManyToOne(() => User, user => user.clientContracts)
    @JoinColumn({ name: "ClientID" })
    ClientID!: User;

    @ManyToOne(() => Apartment, apartment => apartment.Contracts)
    @JoinColumn({ name: "ApartmentID" })
    ApartmentID!: Apartment;

    @Column({
        type: "varchar",
        length: 1,
        default: ContractStatus.PENDING,
        enum: ContractStatus
    })
    Status!: ContractStatus;

    @Column({ type: "date", nullable: true })
    startDate!: Date;

    @Column({ type: "date", nullable: true })
    endDate!: Date;
}

```

Building (Здание)

```

@Entity("buildings")
export class Building {
  @PrimaryGeneratedColumn("increment")
  BuildingID!: number;

  @Column({ type: "varchar", length: 100 })
  City!: string;

  @Column({ type: "varchar", length: 100 })
  Street!: string;

  @Column({ type: "varchar", length: 100 })
  Number!: string;

  @Column({ type: "varchar", length: 100, nullable: true })
  Type!: string;

  @Column({ type: "varchar", length: 255, nullable: true })
  Description!: string;

  @Column({ type: "varchar", length: 255, nullable: true })
  Photo!: string;

  @OneToMany(() => Apartment, apartment => apartment.Building)
  Apartments!: Apartment[];
}

```

2. API Endpoints

Реализованы следующие RESTful API endpoints:

Аутентификация

- **POST /api/auth/login** - Вход в систему с получением JWT токена

Пользователи

- **GET /api/users** - Получить всех пользователей
- **GET /api/users/agents** - Получить список агентов
- **GET /api/users/clients** - Получить список клиентов
- **GET /api/users/:id** - Получить пользователя по ID
- **POST /api/users** - Регистрация нового пользователя
- **PUT /api/users/:id** - Обновить данные пользователя
- **DELETE /api/users/:id** - Удалить пользователя

Квартиры

- **GET /api/apartments** - Получить все квартиры (с фильтрацией)
- **GET /api/apartments/:id** - Получить квартиру по ID
- **POST /api/apartments** - Создать новую квартиру
- **PUT /api/apartments/:id** - Обновить данные квартиры
- **DELETE /api/apartments/:id** - Удалить квартиру

Здания

- **GET /api/buildings** - Получить все здания
- **GET /api/buildings/:id** - Получить здание по ID
- **POST /api/buildings** - Создать новое здание
- **PUT /api/buildings/:id** - Обновить данные здания
- **DELETE /api/buildings/:id** - Удалить здание

Контракты

- **GET /api/contracts** - Получить все контракты

- GET /api/contracts/:id - Получить контракт по ID
- POST /api/contracts - Создать новый контракт
- PUT /api/contracts/:id - Обновить контракт
- DELETE /api/contracts/:id - Удалить контракт

3. Middleware и безопасность

Аутентификация

Реализован middleware для проверки JWT токенов:

```
export const authMiddleware = async (req: AuthRequest, res: Response, next: NextFunction) => {
  try {
    const token = req.header("Authorization")?.replace("Bearer ", "");

    if (!token) {
      return res.status(401).json({ message: "Access denied. No token provided." });
    }

    const decoded = jwt.verify(token, process.env.JWT_SECRET || "fallback-secret") as any;
    const userRepository = AppDataSource.getRepository(User);
    const user = await userRepository.findOne({ where: { UserID: decoded.userId } });

    if (!user) {
      return res.status(401).json({ message: "Invalid token." });
    }

    req.user = user;
    next();
  } catch (error) {
    res.status(401).json({ message: "Invalid token." });
  }
};
```

4. Функциональность

Вход и регистрация

- Реализован POST /api/auth/login для входа
- Реализован POST /api/users для регистрации
- JWT токены для аутентификации

Личный кабинет

- Получение информации о пользователе через GET /api/users/:id
- Связь с контрактами через отношения в базе данных

Поиск недвижимости

- GET /api/apartments поддерживает фильтрацию
- Возможность фильтрации по цене, площади, зданию

Управление объектами

- CRUD операции для квартир и зданий
- Загрузка фотографий через статические файлы

История сделок

- Контракты связаны с пользователями и квартирами
- Статусы контрактов: PENDING, ACTIVE, FINISHED

Вывод

В ходе выполнения домашнего задания был успешно реализован RESTful API для сервиса аренды недвижимости на базе Express.js и TypeScript.

Достигнутые результаты:

1. Реализована система аутентификации с JWT токенами
2. Создана регистрация и вход пользователей
3. Реализован личный кабинет с информацией о пользователе
4. Реализован поиск недвижимости с возможностью фильтрации
5. Создана система управления объектами недвижимости
6. Реализована система контрактов для отслеживания сделок
7. Реализована безопасность (хеширование паролей, middleware аутентификации)

Проект готов к использованию и может быть расширен дополнительным функционалом, таким как система сообщений между пользователями, более сложная система фильтрации или интеграция с внешними сервисами.