

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ  
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Дисциплина:** Бэк-энд разработка

**Отчет**

**Домашняя работа №2 «Проектирование API»**

**Выполнила:**

**Клапнева Диана**

**Группа К3342**

**Проверил:  
Добряков Д. И.**

**Санкт-Петербург**

**2026 г.**

## Задача

Задание:

1. Опишите все функциональные и нефункциональные требования к вашему бэкенд-приложению
2. Выберите формат реализации API, которого вы будете придерживаться: REST API или Backend For Frontend
3. Спроектируйте все эндпоинты вашего API в формате OpenAPI-схемы с учётом реализованной диаграммы БД
4. Опишите тело каждого запроса и тело каждого ответа, продумайте возможные ошибки, возвращаемые из API

Необходимо сделать отчёт по шаблону

## Ход работы

**Шаг 1:** На данный момент я ограничусь возможностью создавать, изменять и удалять новые записи пользователей и объектов недвижимости, а также новых сделок и сообщений. Также мы реализуем поиск недвижимости по фильтрам, как было указано в задании изначально.

**Шаг 2:** Будем держаться формата REST API.

**Шаг 3:** Приведем ERD-схему в формат http-схемы. Промежуточные таблицы (UserToEstate и т.д.) я интегрировала в ключевые ресурсы – User, Estate, Deal, Message, Session.

Промежуточный результат:

```
model Deal_role {  
    landlord: "landlord",  
    tenant: "tenant",  
}  
  
model User {  
    id: int32;  
    first_name: string;  
    middle_name?: string;  
    last_name: string;  
    deal_role?: Deal_role;  
    email: string;  
    password: string;  
    is_verified: boolean;  
    created_at: utcDateTime;  
    updated_at: utcDateTime;  
    estates_ids?: int32[];  
    deals_ids?: int32[];  
    session_ids?: int32[];  
}  
  
model Estate {  
    id: int32;  
    user_id: int32;  
    address: string;  
    type: Estate_type;  
    room_amount: int32;  
    description?: string;  
    price: float64;  
    image_path?: string;  
    bath_type?: Bath_type;  
    fridge?: boolean;  
    washing_machine?: boolean;  
    internet?: boolean;  
    tv?: boolean;  
    furnished_rooms?: boolean;  
    furnished_kitchen?: boolean;  
    is_verified: boolean;  
    is_available: boolean;  
    created_at: utcDateTime;  
    updated_at?: utcDateTime;  
}  
  
model Estate_type {  
    apartment: "apartment",  
    cottage: "cottage",  
    room: "room"  
}  
  
model Bath_type {  
    shower: "shower",  
    bathtub: "bathtub"  
}  
  
model Deal {  
    id: int32;  
    landlord_id: int32;  
    tenant_id: int32;  
    period: string;  
    deal_status: Deal_status;  
    estates: Estate[];  
    is_published: boolean;  
    created_at: utcDateTime;  
    updated_at?: utcDateTime;  
}  
  
model Deal_Status {  
    pending: "pending",  
    active: "active",  
    closed: "closed"  
}  
  
model Session {  
    id: int32;  
    user_id: int32;  
    created_at: utcDateTime;  
    updated_at?: utcDateTime;  
}  
  
model Message {  
    id: int32;  
    user_id: int32;  
    session_id: int32;  
    message: string;  
    attachment_file_path?: string;  
}
```

**Шаг 4:** по заданию к проекту должна быть реализована фильтрация недвижимости, сделаем модель фильтров для Estate:

```
//фильтры для estate
model EstateFilterParams {
    @query
    user_id?: int32;
    @query
    type?: Estate_type;
    @query
    @minValue(0)
    min_price?: float64;
    @query
    @maxValue(10000000)
    max_price?: float64;
    @query
    min_rooms?: int32;
    @query
    max_rooms?: int32;
    @query
    has_fridge?: boolean;
    @query
    has_washing_machine?: boolean;
    @query
    has_internet?: boolean;
    @query
    has_tv?: boolean;
    @query
    furnished?: boolean;
    @query
    is_available?: boolean;
    @query
```

**Шаг 5:** сформируем тела запросов и ответов уже сейчас, ведь они используются в эндпоинтах. Нужно, чтобы можно было создать, удалить и изменить новые записи для User, Estate, Message и Deal, а также фильтровать Estate по всем признакам.

Запросы для создания (пример)

```
//тела запросов - создание новых записей
model CreateUserRequest {
    first_name: string;
    middle_name: string;
    last_name: string;
    deal_role?: Deal_role;
    email: string;
    password: string;
}

model CreateEstateRequest {
    user_id: int32;
    address: string;
    type: Estate_type;
    room_amount: int32;
    description?: string;
    price: float64;
    image_path?: string;
    bath_type?: Bath_type;
    fridge?: boolean;
    washing_machine?: boolean;
    internet?: boolean;
    tv?: boolean;
    furnished_rooms?: boolean;
    furnished_kitchen?: boolean;
}
```

Запросы для изменения (пример)

```
//тела запросов - обновляем
```

```
model UpdateUserRequest {
    first_name?: string;
    middle_name?: string;
    last_name?: string;
    deal_role?: Deal_role;
    email?: string;
    password?: string;
    is_verified?: boolean;
}

model UpdateEstateRequest {
    user_id?: int32;
    address?: string;
    type?: Estate_type;
    room_amount?: int32;
    description?: string;
    price?: float64;
    image_path?: string;
    bath_type?: Bath_type;
    fridge?: boolean;
    washing_machine?: boolean;
    internet?: boolean;
    tv?: boolean;
    furnished_rooms?: boolean;
    furnished_kitchen?: boolean;
    is_verified?: boolean;
    is_available?: boolean;
}
```

**Шаг 6:** создадим эндпоинты – для всех ключевых ресурсов будут методы Get, Post Delete и Patch. Вот они для сделки, например:

```
//deal эндпоинты
@route("/deals")
interface Deals {
    @post
    createDeal(
        | @body body: CreateDealRequest
    ): {
        @statusCode statusCode: 201;
        @body deal: Deal;
    } | ValidationException | NotFoundError;

    @patch
    @route("/{dealId}")
    updateDeal(
        | @path dealId: int32,
        | @body body: UpdateDealRequest
    ): Deal | NotFoundError | ValidationException;

    @delete
    @route("/{dealId}")
    deleteDeal(
        | @path dealId: int32
    ): {
        @statusCode statusCode: 204;
    } | NotFoundError | ForbiddenError;
}
```

Поиск для Estate использует параметры(фильтры), которые мы прописали ранее.

```
//estate эндпоинты
@route("/estates")
interface Estates {
    @get
    listEstates(
        | ...EstateFilterParams
    ): Estate[] | ValidationException;
```

**Шаг 7:** подумаем об ошибках – на данный момент актуальны будут 422 (validation – когда мы не ввели обязательное поле, или ввели неправильный тип данных), 403 – forbidden (запретим, кому захотим) и 404 – нужного ресурса может не быть.

```
//ошибки
model NotFoundError {
    @statusCode statusCode: 404;
    error: {
        code: "NOT_FOUND";
        message: string;
    };
}

model ValidationError {
    @statusCode statusCode: 422;
    error: {
        code: "VALIDATION_ERROR";
        message: string;
        details?: {
            field: string;
            message: string;
        }[];
    };
}

model ForbiddenError {
    @statusCode statusCode: 403;
    error: {
        code: "FORBIDDEN";
        message: string;
    };
}
```

## Вывод

Данная домашняя работа позволяет спланировать структуру арі проекта и понять на практике, как в принципе формируются запросы клиента к серверу. Благодаря этой работе я более детально представляю себе будущую архитектуру приложения.