САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет по домашнему заданию №2

Выполнил: Гольцман Глеб

Группа K3344

Проверил: Добряков Давид Ильич

Санкт-Петербург

2026 г.

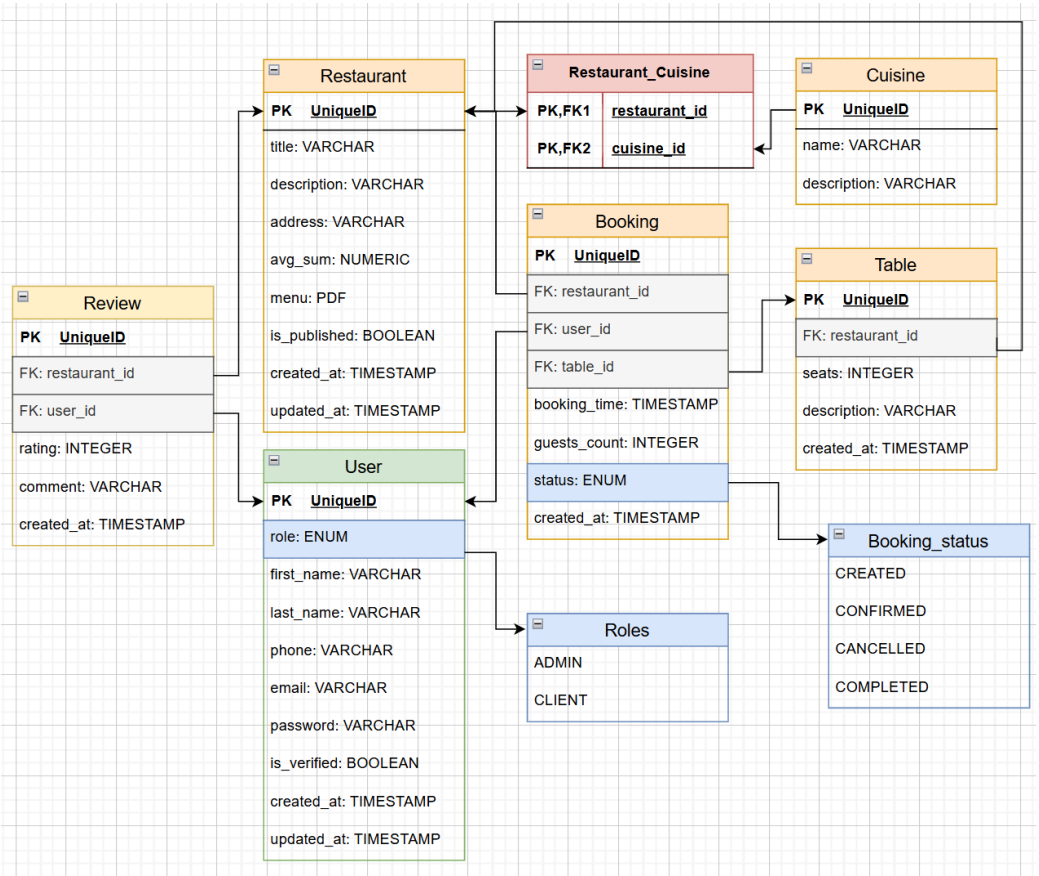
Задача

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

На защите ожидаю увидеть swagger со спроектированным API и обсудить ваши методы.

Ход работы

В качестве предметной области я выбрал приложение для бронирования столиков в ресторанах.



1. Функциональные и нефункциональный требования

Функциональные требования: управление пользователями, ресторанами, столами, кухнями, бронированиями, отзывами(создание, получение, обновление, удаление). Поддержка пагинации, возврат типизированных ошибок.

Нефункциональные требования: формат данных JSON, API должно быть типизировано и документировано через OpenAPI 3.0,

2. Формат реализации API

REST API выбран как стандарт для web-приложений: прост в интеграции, поддерживается большинством фронтенд-фреймворков.

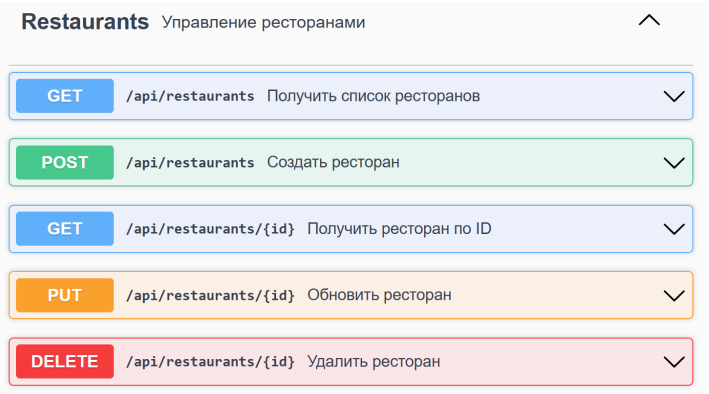
3. Эндпоинты API

Были использованы стандартные HTTP методы для доступа ко всем endoint: GET, POST, PUT, DELETE. На каждый запрос и ответ сформированы отдельные модели для того, чтобы не передавать лишние данные по сети и не связывать entity и dto. Пример методов через TypeSpec, OpenApi, Swagger.

```
import "@typespec/http";
using Http;

@route("api/restaurants/")
@tag("Restaurant ")
interface RestaurantApi{
      @get list(...PaginationQuery): RestaurantResponse[];
      @get @route("/{id}") getById(@path id: string):
      RestaurantResponse | NotFoundError;
      @post create(...AuthHeader, @body body: CreateRestaurantDto):
      RestaurantResponse | UnauthorizedError;
      @put @route("/{id}") update(...AuthHeader, @path id: string, @body body:
      UpdateRestaurantDto): RestaurantResponse | NotFoundError;
      @delete @route("/{id}") delete(...AuthHeader, @path id: string): void;
}
```

```
/api/restaurants:
  get:
    tags: [Restaurants]
    summary: Получить список ресторанов
    parameters:
      - $ref: '#/components/parameters/AuthHeader'
      - $ref: '#/components/parameters/page'
      - $ref: '#/components/parameters/size'
    responses:
      '200':
        content:
          application/json:
            schema:
              type: array
              items: { $ref: '#/components/schemas/RestaurantResponse' }
      '401': { $ref: '#/components/schemas/UnauthorizedError' }
```



Общий класс обшибок

```yaml
ErrorResponse:
  type: object
  properties:
    code:
      type: integer
      description: Код ошибки
    message:
      type: string
      description: Сообщение ошибки
    details:
      type: array
      description: Дополнительная информация об ошибке
      items:
        type: object
        properties:
          field:
            type: string
          issue:
            type: string
```

```
import "@typespec/http";
using Http;

@error
model Error {
  code: int32;
  message: string;
}
```

Общие типовые ошибки

```
model UnauthorizedError extends Error {
  code: 401;
  message: "Unauthorized: Invalid credentials or session expired";
}

model ForbiddenError extends Error {
  code: 403;
  message: "Forbidden: You do not have permission";
}

model NotFoundError extends Error {
  code: 404;
  message: "Not Found: Resource doesn't exist";
}
```

```yaml
NotFoundError:
  allOf:
    - $ref: '#/components/schemas/ErrorResponse'
  example:
    code: 404
    message: "Resource not found"
UnauthorizedError:
  allOf:
    - $ref: '#/components/schemas/ErrorResponse'
  example:
    code: 401
    message: "Unauthorized: Invalid credentials or session expired"
```

Тело ответа

```yaml
CreateRestaurantDto:
  type: object
  properties:
    title: { type: string }
    description: { type: string }
    address: { type: string }
    avgCheck: { type: number, format: float }
    cuisineIds:
      type: array
      items: { type: string }
  required: [title, description, address, avgCheck, cuisineIds]

UpdateRestaurantDto:
  type: object
  properties:
    title: { type: string }
    description: { type: string }
    address: { type: string }
    avgCheck: { type: number, format: float }
    cuisineIds:
      type: array
      items: { type: string }
```

CreateRestaurantDto {

title*	string
description*	string
address*	string
avgCheck*	number($float)
cuisineIds*	[string]

}

UpdateRestaurantDto {

title	string
description	string
address	string
avgCheck	number($float)
cuisineIds	[string]

}

Вывод

В ходе выполнения домашнего задания №2 были выделены функциональные и нефункциональные требования api, был выбран формат реализации, при помощи инструмента typespen и автогенерации opanApi докуменации реализован openApi.yaml документ с описанием всех эндпоинтов, ошибок и ответов приложения.