

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ  
УНИВЕРСИТЕТ ИТМО**

**Дисциплина: Бэк-энд разработка**

**Отчет по домашнему заданию №2**

**Выполнил: Гольцман Глеб**

**Группа К3344**

**Проверил: Добряков Давид Ильич**

**Санкт-Петербург**

**2026 г.**

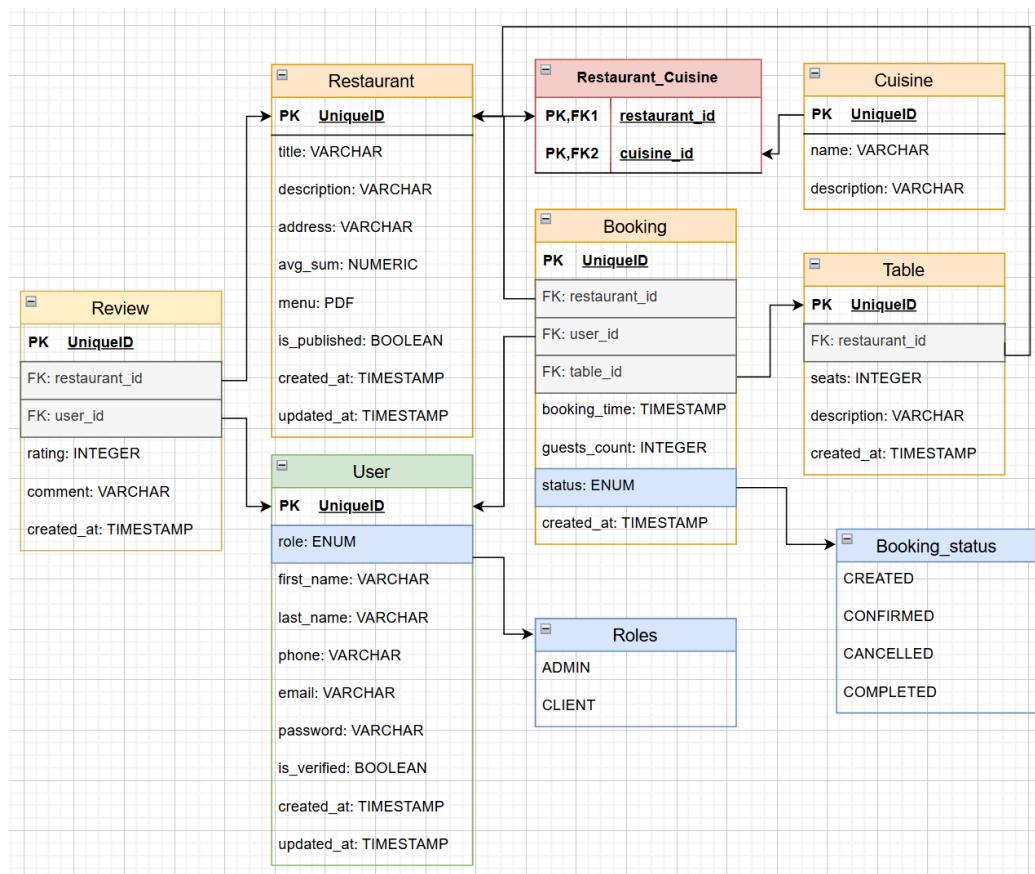
## Задача

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 столов 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 endpoint: 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.

The screenshot shows two side-by-side code snippets. On the left is a portion of a TypeSpec file for a `RestaurantApi`. On the right is the corresponding `openapi: 3.0.0` schema for the `/api/restaurants` endpoint.

**TypeSpec (Left):**

```
1 import "@typespec/http";
2 using Http;
3
4 @route("api/restaurants/")
5 @tag("Restaurant ")
6 interface RestaurantApi{
7     @get list(...PaginationQuery): RestaurantResponse[];
8     @get @route("/{id}") getById(@path id: string):
9         RestaurantResponse | NotFoundError;
10    @post create(...AuthHeader, @body body: CreateRestaurantDto):
11        RestaurantResponse | UnauthorizedError;
12    @put @route("/{id}") update(...AuthHeader, @path id: string, @body body:
13        UpdateRestaurantDto): RestaurantResponse | NotFoundError;
14    @delete @route("/{id}") delete(...AuthHeader, @path id: string): void;
15 }
```

**OpenAPI 3.0 Schema (Right):**

```
447 /api/restaurants:
448   get:
449     tags: [Restaurants]
450     summary: Получить список ресторанов
451     parameters:
452       - $ref: '#/components/parameters/AuthHeader'
453       - $ref: '#/components/parameters/page'
454       - $ref: '#/components/parameters/size'
455     responses:
456       '200':
457         content:
458           application/json:
459             schema:
460               type: array
461               items: { $ref: '#/components/schemas/RestaurantResponse' }
462       '401': { $ref: '#/components/schemas/UnauthorizedError' }
```

The screenshot shows a Swagger UI interface for a "Restaurants" API. It lists the following endpoints:

- GET /api/restaurants** Получить список ресторанов
- POST /api/restaurants** Создать ресторан
- GET /api/restaurants/{id}** Получить ресторан по ID
- PUT /api/restaurants/{id}** Обновить ресторан
- DELETE /api/restaurants/{id}** Удалить ресторан

## Общий класс ошибок

```
66  ErrorResponse:
67    type: object
68    properties:
69      code:
70        type: integer
71        description: Код ошибки
72      message:
73        type: string
74        description: Сообщение ошибки
75      details:
76        type: array
77        description: Дополнительная информация об ошибке
78        items:
79          type: object
80          properties:
81            field:
82              type: string
83            issue:
84              type: string
```

```
1  import "@typespec/http";
2  using Http;
3
4  @error
5  model Error {
6    code: int32;
7    message: string;
8 }
```

## Общие типовые ошибки

```
10  model UnauthorizedError extends Error {
11    code: 401;
12    message: "Unauthorized: Invalid credentials or session expired";
13  }
14
15  model ForbiddenError extends Error {
16    code: 403;
17    message: "Forbidden: You do not have permission";
18  }
19
20  model NotFoundError extends Error {
21    code: 404;
22    message: "Not Found: Resource doesn't exist";
23 }
```

```
86  NotFoundError:
87  allOf:
88    - $ref: '#/components/schemas/ErrorResponse'
89  example:
90    code: 404
91    message: "Resource not found"
92
93  UnauthorizedError:
94  allOf:
95    - $ref: '#/components/schemas/ErrorResponse'
96  example:
97    code: 401
98    message: "Unauthorized: Invalid credentials or session expired"
```

## Тело ответа

```
198 CreateRestaurantDto:
199   type: object
200   properties:
201     title: { type: string }
202     description: { type: string }
203     address: { type: string }
204     avgCheck: { type: number, format: float }
205     cuisineIds:
206       type: array
207       items: { type: string }
208     required: [title, description, address, avgCheck, cuisineIds]
209
210 UpdateRestaurantDto:
211   type: object
212   properties:
213     title: { type: string }
214     description: { type: string }
215     address: { type: string }
216     avgCheck: { type: number, format: float }
217     cuisineIds:
218       type: array
219       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 арі, был выбран формат реализации, при помощи инструмента typespen и автогенерации opanApi документации реализован openApi.yaml документ с описанием всех эндпоинтов, ошибок и ответов приложения.