САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Домашняя работа №2

Выполнил:

Рыбинская Злата

Группа

К3344

Проверил:
Добряков Д. И.

Санкт-Петербург

2026 г.

1 Постановка задачи

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

2 Ход работы

В соответствии с рекомендациями была обновлена ERD-модель (рис.1):

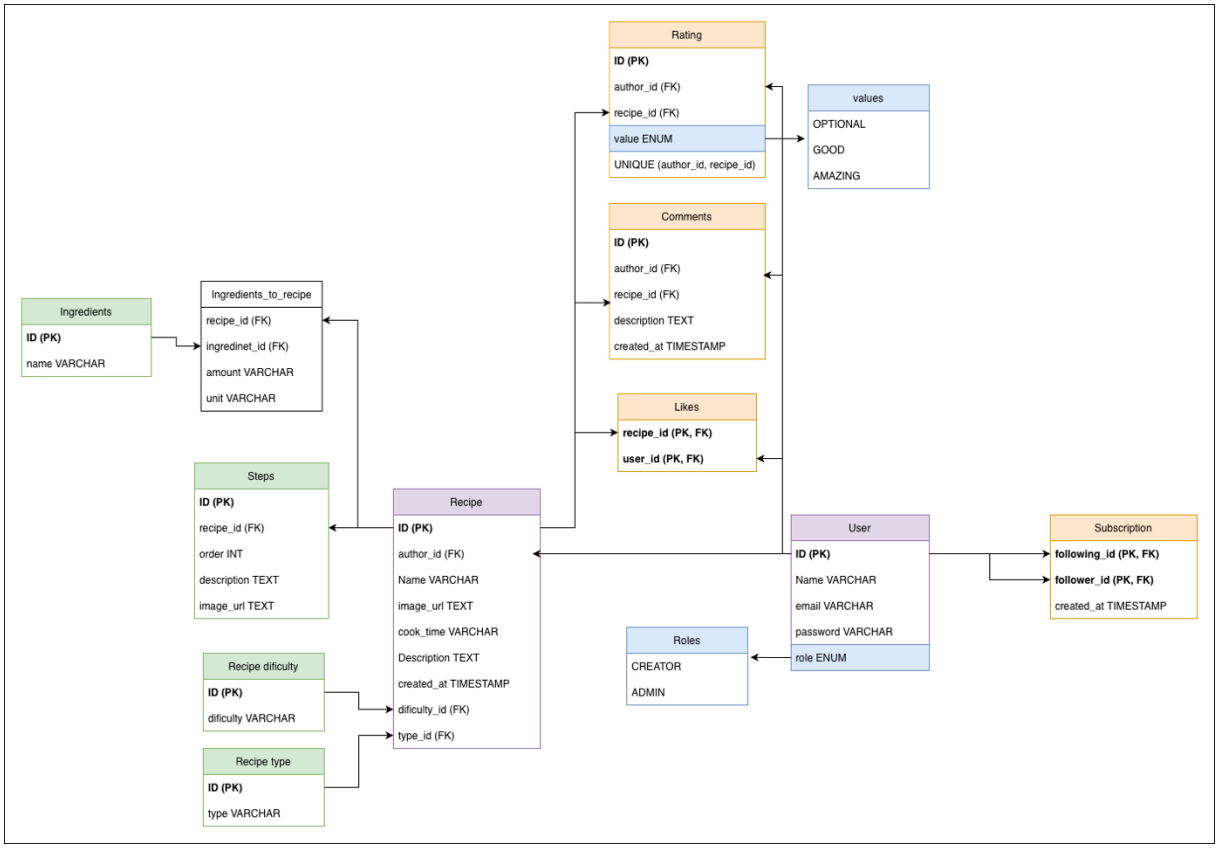

Рисунок 1 – обновленная ERD-модель

Разработанный Swagger полностью отражает спроектированную ERD-модель. В базе данных реализованы следующие основные сущности:

1. User

2. Recipe
3. Ingredient
4. Ingredients_to_recipe (связующая таблица many-to-many)
5. Step
6. Comment
7. Rating
8. Likes
9. Subscription
10. Recipe difficulty
11. Recipe type

Каждая сущность ER-диаграммы представлена в API соответствующими ресурсами:

ERD-сущность	**Отражение в API**
User	*/users, /auth/**
Recipe	*/recipes*
Ingredient	*/ingredients, /recipes/{id}/ingredients*
Step	*/recipes/{id}/steps*
Comment	*/recipes/{id}/comments*
Likes	*/recipes/{id}/like*
Rating	*/recipes/{id}/rating*
Subscription	*/users/{id}/subscribe*

Связи many-to-many (ингредиенты, лайки, подписки) реализованы через вложенные ресурсы, что соответствует REST-подходу и структуре БД.

Ограничения уникальности, присутствующие в БД (например, уникальность пары (author_id, recipe_id) в Rating и (user_id, recipe_id) в Likes), отражены в API через возвращаемый код ошибки 409 Conflict.

Таким образом, API полностью соответствует логической и физической модели данных.

Для реализации выбран формат REST API, так как:

- Ресурсно-ориентированная архитектура соответствует структуре ERD.

- Простота масштабирования.

- Стандартизированные HTTP-методы:

- REST хорошо поддерживается NestJS и Swagger.

Таким образом, REST является наиболее логичным и архитектурно корректным выбором.

Описание эндпоинтов представлено далее:

POST /auth/register
Регистрация нового пользователя.
Тело запроса:
- name (string)
- email (string)
- password (string)

Ответ 201:
- объект User

Ошибки:
- 400 – некорректные данные

POST /auth/login
Авторизация пользователя.
Тело запроса:
- email (string)
- password (string)

Ответ 200:
- access_token (JWT)

Ошибки:
- 401 – неверные учетные данные

GET /users
Возвращает список пользователей.
Ответ 200:
- массив объектов User

GET /users/{id}
Возвращает пользователя по идентификатору.
Ответ 200:
- объект User

Ошибки:
- 404 – пользователь не найден

GET /recipes
Получение списка рецептов с возможностью фильтрации:

Параметры query:

- type_id
- difficulty_id

Ответ 200:

- массив Recipe

POST /recipes

Создание рецепта.

Тело запроса:

- name
- description
- cook_time
- difficulty_id
- type_id

Ответ 201:

- созданный объект Recipe

Ошибки:

- 401 – требуется авторизация

GET /recipes/{id}

Получение рецепта по ID.

Ответ 200:

- Recipe

Ошибки:

- 404 – рецепт не найден

PUT /recipes/{id}

Полное обновление рецепта.

Тело запроса:

- все поля CreateRecipeRequest

Ответ 200:

- обновленный Recipe

Ошибки:

- 403 – недостаточно прав
- 404 – рецепт не найден

PATCH /recipes/{id}

Частичное обновление рецепта.

Тело запроса:

- любые поля из UpdateRecipeRequest

Ответ 200:

- обновленный Recipe

Ошибки:

- 403 – недостаточно прав
- 404 – рецепт не найден

PATCH используется для изменения отдельных полей без полной замены объекта.

DELETE /recipes/{id}

Удаление рецепта.

Ответ 204
Ошибки:
- 403 – недостаточно прав
- 404 – рецепт не найден

GET /ingredients
Возвращает список ингредиентов.

POST /ingredients
Создание нового ингредиента.

GET /recipes/{id}/ingredients
Получение ингредиентов рецепта.
Ответ 200:
- массив RecipeIngredient

POST /recipes/{id}/ingredients
Добавление ингредиента к рецепту.
Тело запроса:
- ingredient_id
- amount
- unit

Ответ 201

GET /recipes/{id}/steps
Получение шагов рецепта.
Ответ 200:
- массив Step

POST /recipes/{id}/steps
Создание шага.
Тело запроса:
- order
- description
- image_url (опционально)

Ответ 201
Последовательность шагов определяется полем order.

GET /recipes/{id}/comments
Получение комментариев к рецепту.

POST /recipes/{id}/comments
Создание комментария.
Тело запроса:
- text

Ответ 201

POST /recipes/{id}/like
Поставить лайк рецепту.
Ответ 201

Ошибки:
- 409 – лайк уже существует

DELETE /recipes/{id}/like
Удалить лайк.
Ответ 204

POST /recipes/{id}/rating
Создание или обновление рейтинга.
Тело запроса:
- value (ENUM)

Ответ 200 :
- объект Rating

Ограничение уникальности реализовано на уровне базы данных (один пользователь – один рейтинг на рецепт).

POST /users/{id}/subscribe
Подписка на автора.
Ответ 201
Ошибки :
- 409 – уже подписан

DELETE /users/{id}/subscribe
Отписка от автора.
Ответ 204

3 Вывод

В ходе проектирования был разработан и формализован REST API сервиса обмена рецептами, полностью соответствующий ERD-модели базы данных. Все сущности предметной области – пользователи, рецепты, ингредиенты, шаги, комментарии, лайки, рейтинг и подписки – отражены в виде ресурсов и связаны через корректно спроектированные эндпоинты. Реализованы связи один-ко-многим и многие-ко-многим, соблюдены ограничения уникальности, а также использованы стандартизированные HTTP-методы в соответствии с семантикой операций (GET, POST, PUT, PATCH, DELETE).

Документирование API с использованием спецификации OpenAPI 3.0.3 (Swagger) позволило формально описать структуру запросов и ответов, возможные коды ошибок и схемы данных. Это обеспечивает прозрачность взаимодействия между клиентской и серверной частями, упрощает

тестирование и повышает масштабируемость системы. Разработанный API обладает логической целостностью, расширяемостью и соответствует современным архитектурным принципам построения веб-сервисов.