САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Практическая/Лабораторная работа

Выполнил:

Габов Михаил

К3340

Проверил:
Добряков Д. И.

Санкт-Петербург

2022 г

1. Обзор Задания

Задача заключалась в создании приложения с использованием Express.js, TypeScript и TypeORM. Основные требования включали:

- Реализацию всех моделей данных, спроектированных в рамках предыдущего задания (ДЗ1).
- Создание набора CRUD-методов (Create, Read, Update, Delete) для каждой модели.
- Реализацию специального API-эндпоинта для поиска и получения пользователя по его id или email..

2. Реализация Моделей Данных

Все спроектированные модели данных были реализованы как сущности (Entities) TypeORM. Связи между моделями, такие как "один ко многим" (@OneToMany) и "многие к одному" (@ManyToOne), настроены с использованием соответствующих декораторов. Каскадное удаление (onDelete: "CASCADE") применяется для поддержания целостности данных при удалении связанных сущностей.

Список реализованных моделей:

- **User**: Представляет пользователя системы. Содержит поля id, username, email, password, bio, avatarUrl и createdAt. Имеет связи с моделями Recipe, Comment, Like, Favorite и Follow.
- **Recipe**: Описывает рецепт. Включает поля id, title, description, difficulty и createdAt. Связан с пользователем (User), шагами приготовления (RecipeStep), ингредиентами (RecipeIngredient), комментариями (Comment), лайками (Like) и избранными (Favorite).
- **Ingredient**: Модель ингредиента с полями id и name.
- **RecipeIngredient**: Промежуточная таблица для реализации связи "многие ко многим" между Recipe и Ingredient. Содержит дополнительное поле amount для указания количества ингредиента.
- **RecipeStep**: Описывает отдельный шаг в рецепте. Включает поля id, stepNumber, description, imageUrl и связана с Recipe.
- **Comment**: Модель комментария к рецепту. Содержит id, content, createdAt

и связана с User и Recipe.
- **Like**: Фиксирует лайк пользователя к рецепту. Связывает User и Recipe.
- **Favorite**: Фиксирует добавление рецепта в избранное. Связывает User и Recipe.
- **Follow**: Реализует механику подписок между пользователями. Содержит идентификаторы подписчика (followerId) и того, на кого подписываются (followingId).

Все модели сконфигурированы в источнике данных AppDataSource для работы с TypeORM.

3. Реализация CRUD-методов и API-эндпоинтов

Для каждой основной модели данных реализован полный набор CRUD-операций. Маршрутизация построена с помощью express.Router, где каждый ресурс (например, /users, /recipes) вынесен в отдельный файл.

Примеры реализованных эндпоинтов:

Модель	HTTP Метод	Путь	Описание	Контроллер
User	POST	/users	Создание нового пользователя.	createUser
	GET	/users	Получение списка всех пользователей.	getAllUsers
	GET	/users/:id	Получение пользователя по id.	getUserById
	PUT	/users/:id	Обновление данных пользователя.	updateUser
	DELETE	/users/:id	Удаление пользователя.	deleteUser
	POST	/users/login	Аутентификация пользователя.	loginUser

Recipe	POST	/recipes	Создание нового рецепта.	createRecipe
	GET	/recipes	Получение списка рецептов с фильтрацией.	getRecipes
	GET	/recipes/:id	Получение рецепта по id.	getRecipeById
	PUT	/recipes/:id	Обновление рецепта.	updateRecipe
	DELETE	/recipes/:id	Удаление рецепта.	deleteRecipe
Comment	POST	/comments	Создание комментария (требует аутентификации).	createComment
	GET	/comments	Получение комментариев по recipeId.	getCommentsByRecipe
	DELETE	/comments/:id	Удаление комментария (требует аутентификации).	deleteComment
Like	POST	/likes	Добавление лайка.	createLike
	DELETE	/likes	Удаление лайка по userId и recipeId.	deleteLike
Favorite	POST	/favorites	Добавление в избранное.	createFavorite
	DELETE	/favorites	Удаление из избранного по userId и recipeId.	deleteFavorite
Follow	POST	/follows	Создание подписки.	createFollow

	DELETE	/follows	Отмена подписки по followerId и followingId.	deleteFollow

Для защищенных маршрутов, таких как создание комментария или удаление своего профиля, используется middleware-функция protect, которая проверяет JWT-токен аутентификации.

4. Реализация API-эндпоинта для получения пользователя по id/email

Отдельное требование по реализации эндпоинта для поиска пользователя было выполнено.

- **Эндпоинт**: GET /users/search/by
- **Контроллер**: searchUser в файле src/controllers/userController.ts.

Логика работы:

1. Эндпоинт принимает GET-запрос по пути /users/search/by.
2. Он может принимать один из двух query-параметров: id или email.
3. Если ни один из параметров не предоставлен, сервер возвращает ошибку 400 Bad Request.
4. В зависимости от предоставленного параметра, контроллер searchUser использует метод findOneBy репозитория User для поиска пользователя в базе данных либо по id, либо по email.
5. Если пользователь найден, его данные возвращаются в формате JSON с кодом 200 OK.
6. Если пользователь не найден, возвращается статус 404 Not Found.