САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Фронтенд-разработка

Отчёт

Лабораторная работа №2.

Взаимодействие с внешним API.

Выполнил:

Колмогорова А.С.

K3342

Проверил:

Добряков Д. И.

Санкт-Петербург

2025 г.

СОДЕРЖАНИЕ

1 Задача

В рамках второй лабораторной работы необходимо было к наработкам из лабораторной №1 привязать реализацию мокового API с помощью JSON-сервера (обеспечить регистрацию пользователей и авторизацию).

Вариант: **Сервис для обмена рецептами и кулинарных блогов**

Основные составляющие:

- Вход
- Регистрация
- Личный кабинет пользователя (сохраненные рецепты, публикации)
- Поиск рецептов с фильтрацией по типу блюда, сложности, поиск по названию
- Страница рецепта с фото, пошаговыми инструкциями и видео
- Социальные функции (комментарии, лайки, подписки на кулинаров)

2 Ход работы

Первым делом необходимо было разобраться с моковыми API, выбрать вариант реализации. Я остановилась на npm json-server-auth (https://www.npmjs.com/package/json-server-auth), изучила документацию и дополнительный пример с лекции.

Скопировав все файлы из ЛР1, в директории для второй лабораторной запустила

```
npm install -D json-server json-server-auth
```

Затем создала `db.json`, определив изначальную структуру коллекцией:

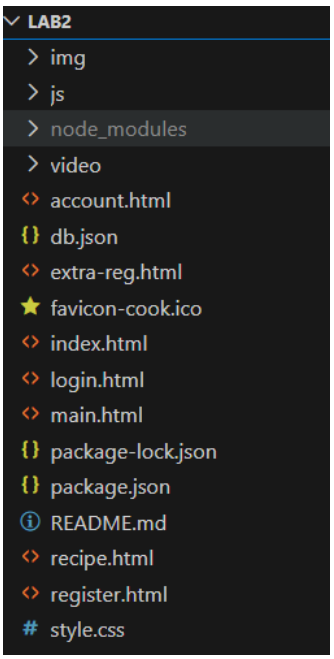
```
{
  "users": []
}
```

При попытке запустить сервер возникли ошибки, которые удалось решить по следующему алгоритму:

- `npm uninstall json-server json-server-auth`
- `npm install json-server@0.17.4 json-server-auth@2.1.0`
- добавить в `package.json`:
- `"scripts": {`
  `"server": "json-server db.json -m ./node_modules/json-server-auth",`
  `"start": "node server.js"}`
- запустить сервер: `npm run server`

Теперь по адресу: `http://localhost:3000` работает сервер.

Итоговая структура проекта выглядит следующим образом:



Разницы с лабораторной №1 в плане html-содержимого практически нет, добавлена еще одна страница: extra-reg, в ней у пользователя запрашиваются дополнительные данные для заполнения профиля индивидуальными данными.

Наиболее объемной задачей являлось переписывание логики функций с использованием запросов к API.

2.1 Регистрация и авторизация (register&login.js)

Одна из ключевых частей, реализованных с помощью запросов к API:

- эндпоинт: `'POST'/register` через await запрос
  - создает новую запись пользователя, добавляет в db.json в коллекцию users, пароль хэшируется
  - получаем accessToken (JWT токен для аутентификации), запоминаем в localStorage, для дальнейшей проверки и сопоставления
- эндпоинт: `'PATCH'/users/{id}`
  - для добавления дополнительной информации пользователя

- эндпоинт: `'POST'/login`
    - осуществляется вход в аккаунт, с проверкой данных, даже получаем accessToken

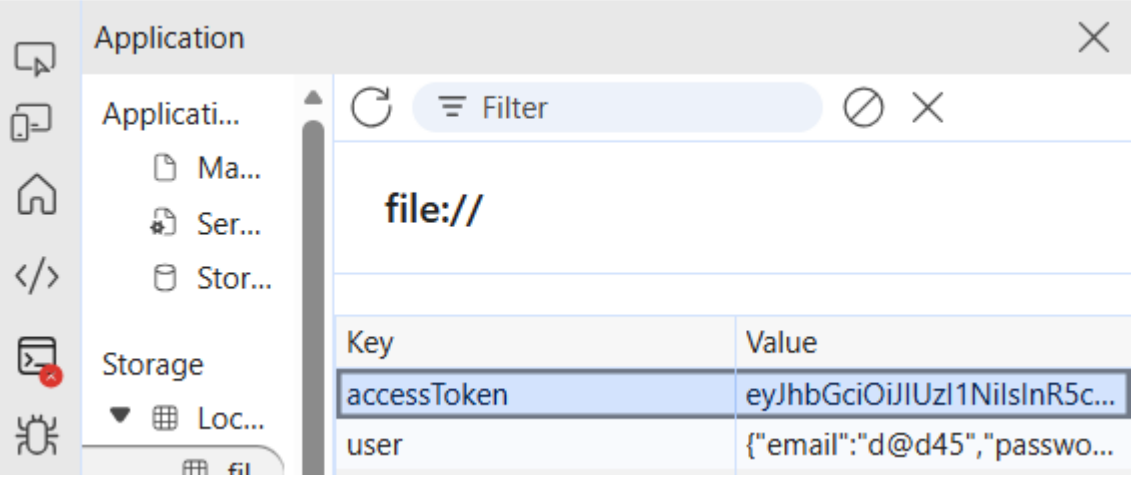


Рисунок 1 – Хранение accessToken в localStorage

2.2 Рецепты (account.js)

Для реализации функцию и логики в аккаунте пользователя, как уже было сказано, из коллекции db.json/users подтягивается информация о текущем пользователе. При создании собственных рецептов, они также дополняются в качестве свойства юзера.

По пунктам:

- эндпоинт: `GET /users/{id}` – загружает профиль
    - получает актуальные данные пользователя с сервера
    - обновляет информацию в localStorage
    - отображает профиль на странице
- эндпоинт: `PATCH /users/{id}`
    - изменяет данные пользователя (name, username, bio) и синхронизирует изменения с сервером
    - также обновляет localStorage
- эндпоинт: `GET /publicRecipes` – работа с базой рецептов
    - получает все публичные рецепты

- фильтрует по сохраненным id из localStorage и отображает соответствующие рецепты

2.3 Страница регистрации (main.js + recipe.js)

- эндпоинт: `GET /publicRecipes` – загружает все публичные рецепты
  - реализована фильтрация на клиенте по типу/сложности/поиску
  - отображение карточек рецептов

3 Вывод

В рамках выполнения лабораторной работы были изучены возможности и особенности мокового API в качестве JSON-server. Рассмотрены и имплементированы GET/POST/PATCH запросы для реализации лабораторной работы.