САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Фронт-энд разработка

Отчет по теме: «Иммитация работы с API. Моковое API»

Лабораторная работа №2

Выполнил:

Суворин И. А.

Группа 1.1

Проверил:
Добряков Д. И.

Санкт-Петербург

2025 г.

Целью данной лабораторной работы является работа с моковым API, работа с JSON сервером и работа с fetch, написание запросов.

В качестве основы для ЛР2 была взята обновленная ЛР1 – веб-приложение, сервис для обмена рецептов и кулинарных блогов.

Реализация мокового API

Для имитации серверной части приложения был использован JSON Server. Он юыл представлен db.json. В нем описаны основные сущности:

— Recipes – список рецептов перенесенный из ЛР1, в дальнейшем пополняющийся рецептами пользователей;

— Users – список пользователей, которые будут зарегистрированы на сайте.

Благодаря fetch далее в работе будут выполнены следующие запросы:

— DELETE-запрос – удаление данных,

— Get-запрос – получение данных,

— POST-запрос – добавление данных.

Установка сервера проходит через команду «npm install -g json-server», написанную в терминале. Запуск сервера начинается благодаря «json-server --watch db.json --port 3001», где необходимо указать источник данных и номер порта.

Реализованные запросы

1. GET-запрос. Получение всех рецептов на сервере.

```
fetch("http://localhost:3001/recipes")
        .then(response => response.json())
        .then(recipes => {
            container.innerHTML = "";
            showRecipes(recipes);
        })
        .catch(error => {
            console.log("Ошибка загрузки рецептов:", error);
            container.innerHTML = "Не удалось загрузить рецепты";
        });
```

fetch() – создание запроса на сервер для выполнение HTTP-запроса (`"http://localhost:3001/recipes"`). Так мы получаем Promise, который в случае успеха разрешается в объект «response» - это не данные а просто «обертка» над ответом сервера: статус, заголовки, тело ответа, которое еще нужно прочитать. Первый .then дает нам этот

объект Response. Мы вызываем «response json » (необязательно этот метод чтения), потому что тело ответа можно прочитать только 1 раз. Это асинхронная операция, которая возвращает еще один Promise. Второй Promise дает нам реальные данные. Это 2 разные асинхронные операции. В fetch 2 последовательных асинхронных действия. «showRecipes» - функция, в которую передается массив рецептов (recipes), полученный от сервера. Далее пользователь в консоли может получить уведомление в случае ошибки.

2. GET-запрос. Получение рецептов для фильтрации

```
    fetch("http://localhost:3001/recipes")
        .then(response => response.json())
        .then(allRecipes => {
            let result = [];
            for (let i = 0; i < allRecipes.length; i++) {
                let r = allRecipes[i];
                let good = true;

                if (text !== "" && r.title.toLowerCase().indexOf(text) ===
-1 &&
                    (r.short && r.short.toLowerCase().indexOf(text) === -
1)) {
                    good = false;
                }

                if (diff !== "" && r.difficulty !== diff) {
                    good = false;
                }

                if (good) result.push(r);
            }

            let container = document.getElementById("recipesList");
            container.innerHTML = "";
            showRecipes(result);
        })
        .catch(error => {
            console.log("Ошибка фильтра:", error);
        });
```

Идет создание запроса на сервер через HTTP при помощи fetch. Далее получаем ответ (первый then), потом читаем его (второй then). Так как это асинхронная операция, то мы должны заранее подготовить ответ, куда положим результат (let result =). Цикл для всех рецептов, отбираем по индексу. Так мы смотрим на количество рецептов, полученных с сервера. (r) – ссылка на текущий объект рецепта. (good) – флагЮ который говорит о том, что нам подходит рецепт. Далее идет фильтрация по тексту введенному пользователем в поиске. Фильтрация по сложности. Добавление рецепта в

результат поиска, если он соответствует всем условиям.

3. GET-запрос. Получение одного рецепта по id.

```
fetch("http://localhost:3001/recipes")
        .then(response => {
            if (!response.ok) throw new Error("Не удалось загрузить список
рецептов");
            return response.json();
        })
        .then(recipes => {

            // Ищем рецепт, приводя id к строке
            let recipe = recipes.find(r => String(r.id) === id);

            if (!recipe) {
                container.innerHTML = '<div class="alert alert-
danger">Рецепт с id ' + id + ' не найден</div>';
                return;
            }

            let html = '';

            html += '<h1>' + (recipe.title || 'Без названия') + '</h1>';
            if (recipe.short) html += '<p class="text-muted">' +
recipe.short + '</p>';

            let imgSrc = recipe.image || 'assets/img/example.jpg';
            html += '<img src="' + imgSrc + '" class="img-fluid rounded
mb-4" alt="' + (recipe.title || '') + '"
onerror="this.src=\'assets/img/example.jpg\'">';

            html += '<div class="row"><div class="col-lg-8">';

            html += '<h4>Ингредиенты</h4>';
            if (Array.isArray(recipe.ingredients) &&
recipe.ingredients.length > 0) {
                html += '<ul class="list-group mb-4">';
                recipe.ingredients.forEach(ing => {
                    let trimmed = ing.trim();
                    if (trimmed) html += '<li class="list-group-item">' +
trimmed + '</li>';
                });
                html += '</ul>';
            } else {
                html += '<p class="text-muted">Ингредиенты не
указаны</p>';
            }
```

```
            html += '<h4>Шаги приготовления</h4>';
            if (Array.isArray(recipe.steps) && recipe.steps.length > 0) {
                html += '<ol class="list-group list-group-numbered mb-
4">';
                recipe.steps.forEach(step => {
                    let trimmed = step.trim();
                    if (trimmed) html += '<li class="list-group-item">' +
trimmed + '</li>';
                });
                html += '</ol>';
            } else {
                html += '<p class="text-muted">Шаги не указаны</p>';
            }

            html += '</div><div class="col-lg-4"><div class="card"><div
class="card-body">';
            html += '<p><strong>Время:</strong> ' + (recipe.time || '—') +
' минут</p>';
            html += '<p><strong>Сложность:</strong> ' + (recipe.difficulty
|| '—') + '</p>';
            if (recipe.author) html += '<p><strong>Автор:</strong> ' +
recipe.author + '</p>';
            if (recipe.published) html +=
'<p><strong>Опубликовано:</strong> ' + recipe.published + '</p>';
            html += '</div></div></div></div>';

            container.innerHTML = html;
        })
        .catch(error => {
            console.error("Ошибка:", error);
            container.innerHTML = '<div class="alert alert-danger">Ошибка
загрузки: ' + error.message + '</div>';
        });
```

Идет подключение к серверу через HTTP, через fetch. Далее получаем ответ (первый `then`), потом читаем его (второй `then`). Далее происходит поиск рецепта по id. В db.json каждый рецепт имеет свой уникальный id. Далее идет проверка: найден ли рецепт. Если нет, то выводит сообщение об ошибку через alert и останавливает выполнение функции. Место, о котором мы должны были подумать заранее (`let html`), где постепенно будет собираться HTML для конкретного рецепта, будут использоваться такие же критерии как и у рецептов в db.json. По итогу получается готовая страница рецепта (`container innerHTML = html`).

4. GET-запрос. Авторизация пользователя.

```
    fetch("http://localhost:3001/users?username=" + username +
"&password=" + password)
        .then(response => response.json())
        .then(users => {
            if (users.length > 0) {
                // Вход успешен
                localStorage.setItem("myuser", username);
                updateHeader();
                document.getElementById("loginBlock").classList.add("d-
none");
                document.getElementById("profileBlock").classList.remove("
d-none");
                document.getElementById("userName").innerText = username;
                document.getElementById("addRecipeBlock").classList.remove
("d-none");
                showMyRecipes();
                alert("Добро пожаловать!");
            } else {
                // Регистрируем нового
                let newUser = { username: username, password: password };
            alert("Ошибка сервера");
        });
```

Данный запрос проверяет, существует ли такой пользователь с указанным логином и паролем. Если да, то он считается авторизированным и сохраняется в LocalStorage (`myuser - ключ, username- значение`). Обновляем шапку (`updateHeader `), чтобы было видно название профиля.

5. POST-запрос. Регистрация нового пользователя.

```
                fetch("http://localhost:3001/users", {
                    method: "POST", // POST запрос
                    body: JSON.stringify(newUser),
                    headers: {
                        "Content-Type": "application/json"
                    }
                })
                .then(response => response.json())
                .then(data => {
                    localStorage.setItem("myuser", username);
                    updateHeader();
                    document.getElementById("loginBlock").classList.add("d
-none");
                    document.getElementById("profileBlock").classList.remo
ve("d-none");
                    document.getElementById("userName").innerText =
username;
```

```
                    document.getElementById("addRecipeBlock").classList.re
move("d-none");
                    showMyRecipes();  // сразу показываем после
регистрации
                    alert("Вы зарегистрированы и вошли!");
                })
                .catch(error => {
                    alert("Ошибка регистрации");
                });
            }
        })
        .catch(error => {
```

Если нет такого пользователя, то идет его добавление по формату «имя и пароль». Создается объект нового пользователя, который и будет отправлен на сервер. Data гарантирует нам пользователя.

6. GET-запрос. Получение рецептов текущего пользователя.

```
    fetch("http://localhost:3001/recipes?author=" + user)
        .then(response => response.json())
        .then(recipes => {
            container.innerHTML = "";
            if (recipes.length === 0) {
                container.innerHTML = '<p class="text-muted">У тебя пока
нет своих рецептов</p>';
                return;
            }

            for (let i = 0; i < recipes.length; i++) {
                let r = recipes[i];
                let card =
                    '<div class="col-md-6 mb-3">' +
                        '<div class="card">' +
                            '<div class="card-body">' +
                                '<h5>' + r.title + '</h5>' +
                                '<p class="small text-muted">' + (r.short
|| '') + '</p>' +
                                '<small>Время: ' + r.time + ' мин • ' +
r.difficulty + '</small>' +
                                '<div class="mt-3">' +
                                    '<a href="recipe.html?id=' + r.id + '"
class="btn btn-sm btn-outline-primary">Открыть</a>' +
                                    ' <button class="btn btn-sm btn-
outline-danger" onclick="deleteRecipe(\'' + r.id + '\')">Удалить</button>'
+
                                '</div>' +
                            '</div>' +
                        '</div>' +
                    '</div>';
```

```
                container.innerHTML += card;
            }
        })
        .catch(error => {
            container.innerHTML = "Ошибка загрузки";
        });
```

Получает только те рецепты, которые были добавлены пользователем. Собирает его из формы, которую пользователь заполняет, чтобы поделиться рецептом.

7. POST-запрос. Добавление нового рецепта.

```
fetch("http://localhost:3001/recipes", {
        method: "POST",
        body: JSON.stringify(newRecipe),
        headers: {
            "Content-Type": "application/json"
        }
    })
    .then(response => response.json())
    .then(data => {
        alert("Рецепт добавлен!");
        document.getElementById("addRecipeForm").reset();
        showMyRecipes();  // сразу обновляем список своих рецептов
    })
    .catch(error => {
        alert("Ошибка при добавлении");
    });
```

Данный запрос добавляет новый рецепт на сервер согласно образцу, который лежит в (newRecipe).

DELETE-запрос. Удаление добавленного рецепта от пользователя.

```
    fetch("http://localhost:3001/recipes/" + id, {
        method: "DELETE"
    })
```

Удаляет рецепт по его id на сервере

Результаты работы

Рисунок 1 – стоковый db.json

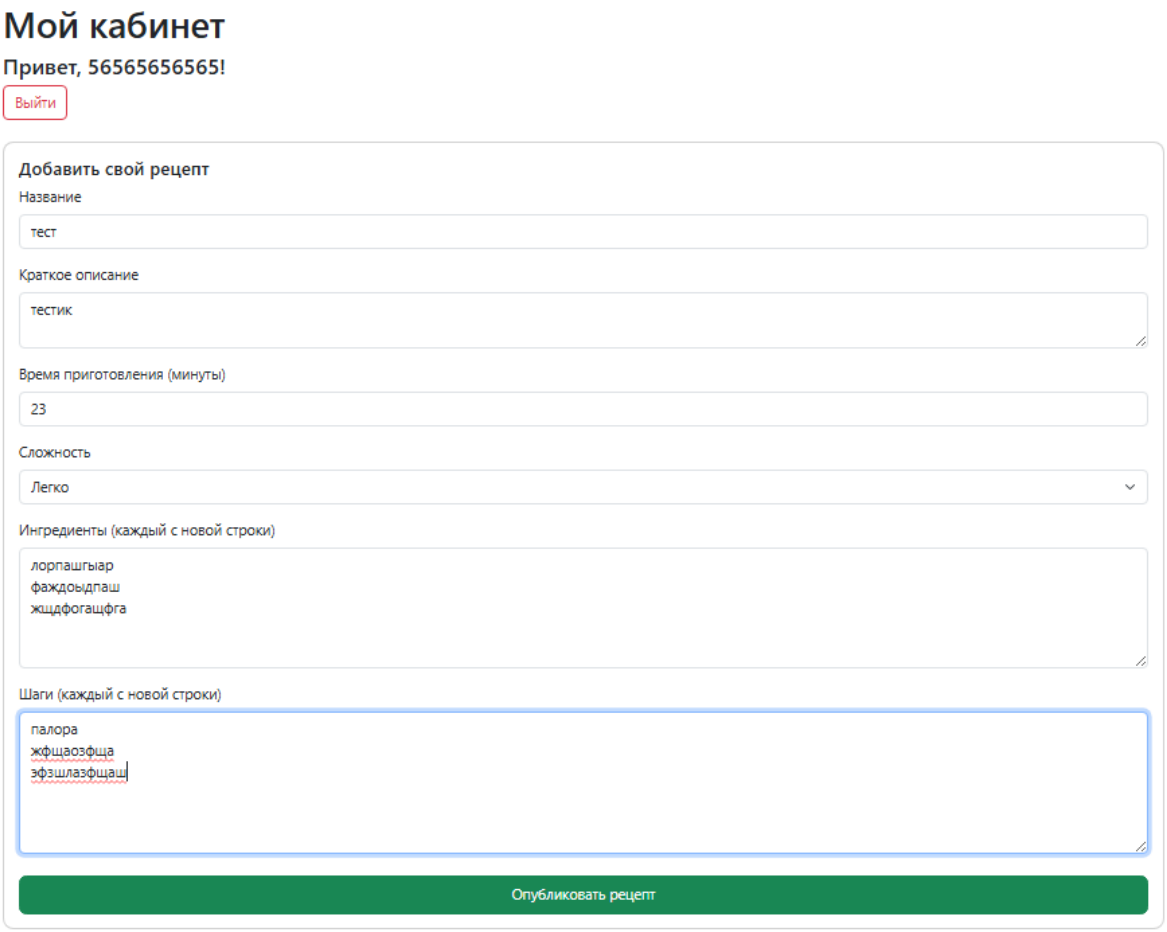

Рисунок 2 – добавление нового пользователя и рецепта в db.json

Рисунок 3 – результаты добавления

ВЫВОД

В ходе выполнения ЛР2 была проведена работа с моковым API по средствам fetch. В результате работы было написано несколько запросов, которые передают основную идею веб-приложения с добавлением новых пользователей и рецептов, чтением старых рецептов (те, которые лежат нв JSON Server), удалением новых рецептов.