

**Дисциплина:** Технологии программирования

**Выполнил(а):** Мулдабекова Амина

**Преподаватель:** Рашидинов Дамир

**Лабораторная работа №9**

## **Цель работы:**

Освоить работу с асинхронными запросами в веб-приложениях, научиться получать и обрабатывать данные из внешних REST API, отображать их в динамическом интерфейсе и обрабатывать ошибки сети.

## **Краткий теоретический обзор:**

Fetch API — это современный интерфейс JavaScript, который позволяет выполнять HTTP-запросы к внешним ресурсам (серверу, API и т.д.) асинхронно, то есть без перезагрузки страницы.

Асинхронность реализуется через промисы (Promises) и ключевые слова async/await, что делает код более читаемым.

Основные особенности:

* Позволяет отправлять и получать данные в формате JSON.
* Работает через промисы, что упрощает обработку ошибок.
* Используется для взаимодействия клиента с сервером в реальном времени.

Пример синтаксиса:

fetch('https://example.com/api')

.then(response => response.json())

.then(data => console.log(data))

.catch(error => console.error('Ошибка:', error));

## **Контрольные вопросы:**

**1. Какие современные технологии лежат в основе развития информационных систем?**

Современные ИС строятся на основе облачных вычислений, больших данных (Big Data), искусственного интеллекта (AI), Интернета вещей (IoT) и блокчейн-технологий.

**2. В чём суть технологии «облачных вычислений»?**

Облачные вычисления позволяют хранить и обрабатывать данные на удалённых серверах, предоставляя пользователю доступ через интернет. Это снижает затраты на оборудование и делает работу гибкой.

**3. Какие преимущества и риски несёт использование искусственного интеллекта?**

Преимущества: автоматизация процессов, повышение точности решений, анализ больших данных.

Риски: потеря рабочих мест, ошибки алгоритмов, этические и конфиденциальные проблемы.

**4. Как Интернет вещей изменяет повседневную жизнь?**

IoT позволяет объединять устройства в единую сеть — от бытовой техники до автомобилей. Это повышает удобство, безопасность и эффективность управления ресурсами.

**5. Почему блокчейн называют «технологией доверия»?**

Потому что данные в блокчейне защищены криптографически и хранятся в распределённой сети — их невозможно изменить без согласия всех участников.

**6. Какие аспекты безопасности необходимо учитывать при использовании ИТ в образовании и науке?**

Важно защищать персональные данные, соблюдать авторские права, предотвращать несанкционированный доступ и использовать надёжные каналы связи при работе с научными базами и образовательными платформами.

## **Листинг программы (HTML + JavaScript)**

<!DOCTYPE html>

<html lang="ru">

<head>

<meta charset="UTF-8">

<title>Лабораторная №9 — Асинхронные запросы</title>

<style>

body { font-family: Arial; margin: 40px; background-color: #fafafa; }

table { border-collapse: collapse; width: 100%; margin-top: 15px; }

th, td { border: 1px solid #ccc; padding: 10px; text-align: left; }

th { background-color: #f2f2f2; }

#loader { color: blue; font-weight: bold; margin-bottom: 10px; }

button { margin-top: 15px; padding: 8px 15px; background-color: #007BFF; color: white; border: none; border-radius: 5px; cursor: pointer; }

</style>

</head>

<body>

<h2>Список пользователей</h2>

<div id="loader"></div>

<table id="userTable">

<thead>

<tr><th>ID</th><th>Имя</th><th>Email</th><th>Город</th></tr>

</thead>

<tbody></tbody>

</table>

<button id="reload">Обновить данные</button>

<script>

async function loadUsers() {

const loader = document.getElementById('loader');

const tbody = document.querySelector('#userTable tbody');

loader.textContent = 'Загрузка данных...';

tbody.innerHTML = '';

try {

const response = await fetch('https://jsonplaceholder.typicode.com/users');

if (!response.ok) throw new Error('Ошибка сети');

const users = await response.json();

users.forEach(u => {

const row = `

<tr>

<td>${u.id}</td>

<td>${u.name}</td>

<td>${u.email}</td>

<td>${u.address.city}</td>

</tr>`;

tbody.insertAdjacentHTML('beforeend', row);

});

loader.textContent = '';

} catch (error) {

loader.textContent = 'Ошибка при получении данных.';

console.error(error);

}

}

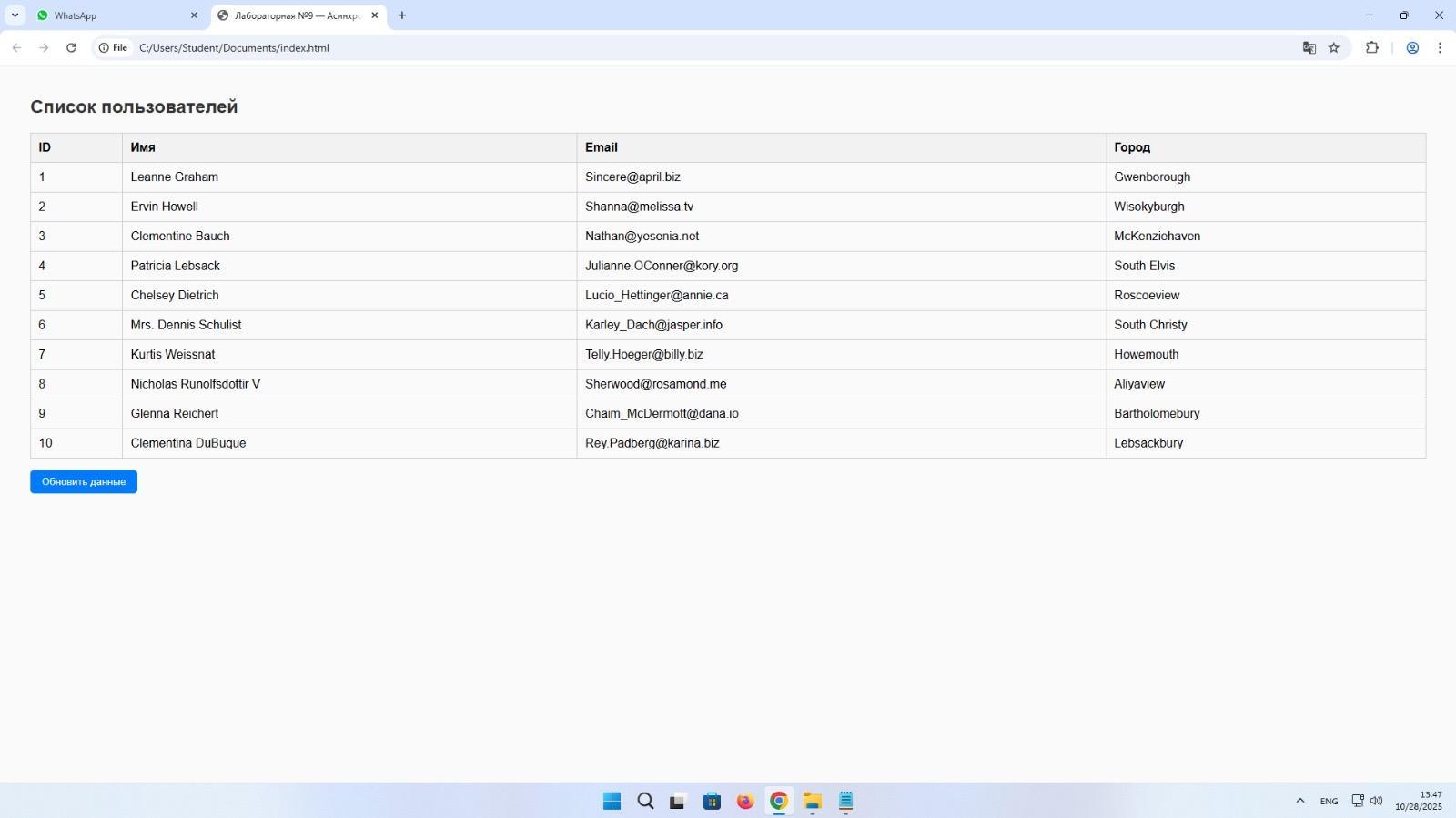
document.getElementById('reload').addEventListener('click', loadUsers);

loadUsers();

</script>

</body>

</html>



## **Объяснение кода, как работает и тд:**

## **1.**

## **<!DOCTYPE html>**

## Это объявление типа документа.

## Оно сообщает браузеру, что страница написана в стандарте HTML5 — это нужно, чтобы браузер правильно понимал и отображал код.

## **2.**

## **<html lang="ru"> ... </html>**

## Всё содержимое сайта находится внутри тегов <html> и </html>.

## Атрибут lang="ru" указывает, что страница написана на русском языке — это помогает браузеру и поисковикам (и экранным читалкам).

## **3.**

## **<head> ... </head>**

## В этом разделе нет видимого содержимого, но он содержит важную информацию о странице — настройки, название, стили и кодировку.

### ➤

### **<meta charset="UTF-8">**

## Указывает, что текст на странице закодирован в формате UTF-8 — чтобы русские буквы отображались корректно и не превращались в «кракозябры».

### ➤

### **<title>Лабораторная №9 — Асинхронные запросы (Fetch API)</title>**

## Это название вкладки, которое ты видишь в браузере (вверху).

### ➤

### <style> ... </style>

## Здесь описывается внешний вид страницы (CSS).

## **4.**

## **Стили (CSS)**

## CSS управляет тем, как всё выглядит — цвета, шрифты, рамки, отступы и т. д.

## Разберём самые важные элементы:

### body

## body {

## font-family: Arial, sans-serif;

## margin: 40px;

## background-color: #fafafa;

## }

## **задаёт шрифт Arial,**

## **делает отступ по краям 40 пикселей,**

## **устанавливает светло-серый фон страницы.**

### **h2**

## h2 { color: #333; }

## — делает заголовок тёмно-серым.

### **table,th,td**

## table {

## border-collapse: collapse;

## width: 100%;

## margin-top: 15px;

## }

## th, td {

## border: 1px solid #ccc;

## padding: 10px;

## text-align: left;

## }

## th {

## background-color: #f2f2f2;

## }

## **border-collapse: collapse; — убирает двойные границы.**

## **width: 100% — таблица растягивается на всю ширину.**

## **th, td — добавляют рамку, отступы и выравнивание.**

## **th имеет серый фон, чтобы отличать заголовки от данных.**

### **#loader**

## #loader {

## color: #007BFF;

## font-weight: bold;

## margin-bottom: 10px;

## }

## Оформление текста “Загрузка данных…” — синий и жирный.

### **button**

## button {

## margin-top: 15px;

## padding: 8px 15px;

## background-color: #007BFF;

## color: white;

## border: none;

## border-radius: 5px;

## cursor: pointer;

## }

## button:hover {

## background-color: #0056b3;

## }

## Создаёт красивую синюю кнопку с белым текстом и закруглёнными углами.

## **button:hover делает её темнее, когда наводишь курсор.**

## **5.**

## **<body> ... </body>**

## Это то, что видно на странице.

### ➤

### **<h2>Список пользователей</h2>**

## Заголовок.

### ➤

### <div id="loader"></div>

## Пустой контейнер, куда JavaScript позже вставит текст «Загрузка данных…» или «Ошибка при получении данных».

### **➤ Таблица**

## <table id="userTable">

## <thead>

## <tr>

## <th>ID</th>

## <th>Имя</th>

## <th>Email</th>

## <th>Город</th>

## </tr>

## </thead>

## <tbody></tbody>

## </table>

## **<thead> — заголовки колонок.**

## **<tbody> — пока пустой. → JavaScript позже добавит туда строки пользователей.**

### **➤ Кнопка**

## <button id="reload">Обновить данные</button>

## При нажатии — страница заново загрузит пользователей (через JS).

## **6. JavaScript (**

## <script> ... </script>

## )

## Здесь написана логика программы — как страница получает данные и обновляется.

### **Объявление функции**

## async function loadUsers() { ... }

## Функция называется loadUsers и помечена как async, значит — она работает асинхронно и внутри можно использовать await (ожидание ответа сервера без «заморозки» страницы).

### **Получаем ссылки на нужные элементы**

## const loader = document.getElementById('loader');

## const tbody = document.querySelector('#userTable tbody');

## Теперь мы можем изменять текст в блоке #loader и добавлять строки в таблицу.

### **Показываем, что идёт загрузка**

## loader.textContent = 'Загрузка данных...';

## tbody.innerHTML = '';

## Очищаем старую таблицу и пишем, что данные загружаются.

### **Пробуем сделать запрос**

## **const response = await fetch('https://jsonplaceholder.typicode.com/users');**

## → fetch() отправляет HTTP-запрос на указанный URL.

## → await ждёт ответа (асинхронно).

## Если запрос неудачный (ошибка сети, сервер не ответил):

## **if (!response.ok) throw new Error('Ошибка сети');**

## → выбрасывается исключение, и программа перейдёт в блок catch.

### **Превращаем ответ в данные**

## **const users = await response.json();**

## Ответ приходит в формате JSON (список пользователей).

## Эта строка превращает его в массив объектов JavaScript.

## **Создаём строки таблицы**

## users.forEach(u => {

## const row = `

## <tr>

## <td>${u.id}</td>

## <td>${u.name}</td>

## <td>${u.email}</td>

## <td>${u.address.city}</td>

## </tr>

## `;

## tbody.insertAdjacentHTML('beforeend', row);

## });

## **Для каждого пользователя создаётся строка <tr> и добавляется в таблицу:**

## **${u.id} — идентификатор**

## **${u.name} — имя**

## **${u.email} — почта**

## **${u.address.city} — город**

## **Скрываем надпись о загрузке**

## **loader.textContent = '';**

## После успешной загрузки надпись «Загрузка данных…» убирается.

## **Обрабатываем ошибки**

## **} catch (error) {**

## **loader.textContent = 'Ошибка при получении данных.';**

## **console.error(error);**

## **}**

## Если что-то пошло не так — выводим сообщение и лог ошибки в консоль.

## 

### **Привязываем кнопку**

## **document.getElementById('reload').addEventListener('click', loadUsers);**

## Теперь при клике на кнопку запускается функция loadUsers() снова — обновление данных.

### **Загружаем пользователей сразу при открытии страницы**

## **loadUsers();**

## Эта строчка вызывает функцию один раз при загрузке сайта, чтобы сразу показать пользователей без нажатия на кнопку.

## **Итого (коротко)**

## **Когда ты открываешь страницу:**

## Появляется надпись «Загрузка данных…».

## JS делает запрос на сайт jsonplaceholder.typicode.com.

## Получает список пользователей в формате JSON.

## Добавляет их в таблицу.

## Если что-то не работает — выводит сообщение об ошибке.

## При нажатии «Обновить данные» всё повторяется заново.

## **Вывод:**

В ходе лабораторной работы я изучила принципы выполнения асинхронных запросов с помощью Fetch API. Научилась получать данные с внешнего сервера и динамически отображать их на странице без перезагрузки.

Закрепила навыки обработки ошибок и работы с JSON-данными.

Освоила базовые приёмы интеграции REST API в веб-приложения.