**Байсариев Шынгыс Ис 24-21**

**Тема:** Использование Fetch API для работы с данными пользователей.  
**Цель:** Освоить применение Fetch API для получения, обработки и отображения данных на веб-странице с помощью JavaScript.

**теоретический обзор Fetch API**

**Fetch API** — это современный интерфейс JavaScript для отправки HTTP-запросов и получения ответов от сервера.  
Он используется для взаимодействия между **клиентом (браузером)** и **сервером**, заменяя устаревший XMLHttpRequest.

**🔹 Основные понятия и термины:**

* **HTTP-запрос (Request)** — обращение к серверу с целью получения или отправки данных.
* **HTTP-ответ (Response)** — результат запроса, который возвращает сервер.
* **Promise (промис)** — объект, представляющий результат асинхронной операции. Позволяет обрабатывать данные после их загрузки.
* **Асинхронность** — выполнение кода без блокировки основной программы (страница не «зависает» при ожидании ответа).
* **Методы Fetch API** — GET, POST, PUT, DELETE (аналогичные CRUD-операциям в базах данных).
  + GET — получить данные.
  + POST — отправить новые данные.
  + PUT — обновить существующие.
  + DELETE — удалить.
* **.then()** — метод обработки результата запроса, когда данные успешно получены.
* **.catch()** — обработка ошибок (например, если сервер недоступен).
* **async / await** — современный синтаксис для работы с асинхронными функциями, делает код более читаемым.

**Структура программы**

1. **HTML-часть**
   * Создаёт каркас страницы: заголовок, контейнер для карточек пользователей и кнопку «Обновить».
   * Используется тег <div> для отображения списка пользователей, <button> — для обновления данных.
2. **CSS-часть**
   * Отвечает за визуальное оформление карточек пользователей: цвета, шрифты, тени, анимации и выравнивание элементов.
   * Добавляет эффект наведения (hover), скругления углов и плавное появление карточек.
3. **JS-часть (JavaScript)**
   * **Функции генерации данных:** создают случайные имена, города, e-mail и посты.
   * **Асинхронная функция loadUsers()** — имитирует загрузку данных (или получение с сервера) и отображает карточки на странице.
   * **DOM-манипуляции:** с помощью insertAdjacentHTML или innerHTML создаются новые элементы на странице.
   * **Обработка событий:** по клику на кнопку вызывается функция обновления пользователей.

**Основные элементы логики программы**

| **Элемент** | **Назначение** |
| --- | --- |
| getRandomItem() | Возвращает случайный элемент из массива (имя, город и т.д.). |
| generateRandomUser() | Формирует один объект пользователя с данными (имя, e-mail, город, id, аватар). |
| loadUsers() | Основная функция, которая загружает и выводит пользователей на страницу. |
| document.getElementById() | Получает элемент HTML по его id для работы с ним. |
| innerHTML | Позволяет вставить HTML-код внутрь элемента. |
| insertAdjacentHTML() | Добавляет сгенерированные карточки без перезаписи старых элементов. |
| addEventListener() | Отслеживает нажатие кнопки для обновления данных. |
| try...catch | Обработка возможных ошибок при загрузке данных. |

**Выводы**

В ходе выполнения работы:

* Изучен механизм работы **Fetch API** и принципы **асинхронного программирования**.
* Освоены методы обработки данных через **Promise** и конструкции async/await.
* Реализовано динамическое создание контента (карточек пользователей) средствами JavaScript.
* Отработано взаимодействие между HTML, CSS и JS для формирования современного интерфейса.

**Основная сложность:** корректное использование шаблонных строк и асинхронных функций.  
**Решение:** применение обратных кавычек ` для вставки данных и использование await для удобного получения результата.

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <title>Users — Fetch API</title>

  <style>

    \* { box-sizing: border-box; margin: 0; padding: 0; }

    body {

      font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;

      background: linear-gradient(135deg, #f0f4f8, #d9e2ec);

      min-height: 100vh;

      display: flex;

      justify-content: center;

      align-items: center;

      flex-direction: column;

      padding: 30px;

    }

    h2 {

      color: #333;

      margin-bottom: 20px;

      text-align: center;

    }

    #loader {

      color: #007acc;

      font-weight: bold;

      font-size: 1.2em;

      margin-bottom: 15px;

      text-align: center;

    }

    .cards-container {

      display: flex;

      flex-wrap: wrap;

      justify-content: center;

      gap: 20px;

      max-width: 1000px;

    }

    .card {

      background: #fff;

      border-radius: 12px;

      box-shadow: 0 5px 15px rgba(0,0,0,0.1);

      padding: 20px;

      width: 230px;

      text-align: center;

      transition: transform 0.3s, box-shadow 0.3s;

      animation: fadeIn 0.6s ease;

    }

    .card:hover {

      transform: translateY(-5px);

      box-shadow: 0 10px 20px rgba(0,0,0,0.2);

    }

    .card img {

      width: 80px;

      height: 80px;

      border-radius: 50%;

      margin-bottom: 15px;

    }

    .card h3 {

      font-size: 1.1em;

      margin-bottom: 5px;

      color: #007acc;

    }

    .card p {

      font-size: 0.9em;

      color: #555;

      margin-bottom: 6px;

    }

    button {

      margin-top: 25px;

      padding: 12px 20px;

      background-color: #007acc;

      color: white;

      font-size: 1em;

      border: none;

      border-radius: 6px;

      cursor: pointer;

      transition: background-color 0.3s;

    }

    button:hover {

      background-color: #005fa3;

    }

    @keyframes fadeIn {

      from { opacity: 0; transform: translateY(20px); }

      to { opacity: 1; transform: translateY(0); }

    }

  </style>

</head>

<body>

  <h2>Users</h2>

  <div id="loader"></div>

  <div class="cards-container" id="cardsContainer"></div>

  <button id="reload">Refresh Users</button>

  <script>

    const firstNames = ["John","Emma","Oliver","Sophia","Liam","Ava","Noah","Mia","Ethan","Isabella"];

    const lastNames = ["Smith","Johnson","Williams","Brown","Jones","Garcia","Miller","Davis","Martinez","Lopez"];

    const posts = [

      "Python is best!",

      "Learning JavaScript is fun.",

      "Working on a new project.",

      "I love programming.",

      "Fetch API makes life easier!"

    ];

    const cities = [

      "New York", "Los Angeles", "Chicago", "Houston", "San Francisco",

      "Seattle", "Boston", "Denver", "Miami", "Austin",

      "Dallas", "Atlanta", "Las Vegas", "San Diego", "Portland"

    ];

    function getRandomNumber(max = 100) {

      return Math.floor(Math.random() \* max) + 1;

    }

    function getRandomItem(arr) {

      return arr[Math.floor(Math.random() \* arr.length)];

    }

    function generateRandomUser() {

      const firstName = getRandomItem(firstNames);

      const lastName = getRandomItem(lastNames);

      const username = `${firstName} ${lastName}`;

      const email = `${firstName.toLowerCase()}.${lastName.toLowerCase()}@example.com`;

      const post = getRandomItem(posts);

      const city = getRandomItem(cities);

      const avatar = `https://i.pravatar.cc/150?img=${getRandomNumber(70)}`;

      const id = getRandomNumber(9999);

      return { username, email, post, avatar, city, id };

    }

    async function loadUsers() {

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

      const container = document.getElementById('cardsContainer');

      loader.textContent = 'Loading users...';

      container.innerHTML = '';

      try {

        const users = Array.from({length: 6}, generateRandomUser);

        users.forEach(user => {

          const card = `

            <div class="card">

              <img src="${user.avatar}" alt="Avatar">

              <h3>${user.username}</h3>

              <p><strong>ID:</strong> ${user.id}</p>

              <p><strong>Email:</strong> ${user.email}</p>

              <p><strong>City:</strong> ${user.city}</p>

              <p>${user.post}</p>

            </div>

          `;

          container.insertAdjacentHTML('beforeend', card);

        });

        loader.textContent = '';

      } catch (error) {

        loader.textContent = 'Error loading data.';

        console.error(error);

      }

    }

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

    loadUsers();

  </script>

</body>

</html>

