







Frontender[1.0] JavaScript - CRUD операции, mock api, взаимодействие с сервером

 YouTube	https://youtu.be/la_CRmcTsMQ
 Telegram	https://t.me/Dmitry_Kolotilshikov
 Github	https://github.com/DmitryKolotilshikov/
 Boosty	https://boosty.to/dmitry_ko
# Номер урока	50



Полезные ссылки:

- <https://learn.javascript.ru/fetch>
- <https://jsonplaceholder.typicode.com>
- <https://jsonplaceholder.typicode.com/guide>
- <https://mockapi.io/projects>



Schema

Define Resource schema, it will be used to generate mock data.

id	Object ID	
name	Faker.js	name.fullName
avatar	Faker.js	image.image
city	Faker.js	address.city



On GET /users
\$mockData
On GET /users/:id
\$mockData
On POST /users
\$mockData
On PUT /users/:id
\$mockData
On DELETE /users/:id
\$mockData

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <script src="script.js" defer></script>
  <title>CRUD, mock api</title>
  <style>
    img {
      width: 100px;
      height: 100px;
      object-fit: cover;
    }
    section {
      padding: 0 5px;
      border: 1px solid black;
      text-align: center;
    }
  </style>
</head>
<body>
  <header class="header">
    <button id="btn-create">create a new user</button>
    <br>
    <button id="btn-edit">edit existing user</button>
    <br>
    <button id="btn-remove">remove existing user</button>
  </header>

  <div id="user-container"></div>
</body>
</html>
```

```

/*
  CRUD операции

  C - CREATE  [POST]
  R - READ    [GET]
  U - UPDATE  [PATCH, PUT]
  D - DELETE  [DELETE]

  PATCH используется для частичных обновления ресурса, а PUT — для замены всего ресурса
  (в уроке используем PUT, так как mock api не поддерживает PATCH)
  (НО у mock api PUT может имитировать метод PATCH)

  https://mockapi.io
*/

const createUserBtn = document.querySelector("#btn-create");
const editUserBtn = document.querySelector("#btn-edit");
const removeUserBtn = document.querySelector("#btn-remove");
const userContainer = document.querySelector("#user-container");

const MOCK_API_URL = "ваш mock api url с уникальным идентификатором";

let users = [];

createUserBtn.addEventListener("click", () => createNewUser())
editUserBtn.addEventListener("click", () => editExistingUser())
removeUserBtn.addEventListener("click", () => removeExistingUser())

// ----- Отрисовка пользователей -----
renderUsers = () => {
  userContainer.innerHTML = "";

  users.forEach(user => {
    const userWrapper = document.createElement("section");
    const userName = document.createElement("h3");
    const userCity = document.createElement("p");
    const userAvatar = document.createElement("img");

    userName.textContent = `Name: ${user.name}`;
    userCity.textContent = `City: ${user.city}`;
    userAvatar.src = user.avatar;
    userWrapper.append(userName, userCity, userAvatar);

    userContainer.append(userWrapper);
  })
}

// ----- Удаление существующего пользователя -----
const removeExistingUser = async () => {
  try {
    const ID = "6"; // в реальности всегда динамичное

    const response = await fetch(`${MOCK_API_URL}/${ID}`, {
      method: 'DELETE',
    });

    if (response.status === 404) {

```

```

        throw new Error(`${ID} не найден`);
    }

    const removedUser = await response.json();

    users = users.filter((user) => user.id !== removedUser.id);
    renderUsers();

    console.log("ПОЛЬЗОВАТЕЛЬ УСПЕШНО УДАЛЕН");
} catch (error) {
    console.error("ОШИБКА при удалении пользователя: ", error.message)
}
}

// ----- Изменение существующего пользователя -----
const editExistingUser = async () => {
    try {
        const ID = "1"; // в реальности всегда динамичное

        const response = await fetch(`${MOCK_API_URL}/${ID}`, {
            method: 'PUT',
            body: JSON.stringify({
                name: "Marina",
                city: "Tashkent",
                avatar: "https://avatars.mds.yandex.net/i?id=6444bd82bce43803b8ad0601c12a80e7-5230955-images-thumbs&n=13"
            }),
            headers: {
                "Content-Type": "application/json",
            }
        });
        const editedUser = await response.json();

        users = users.map((user) => {
            if (user.id === editedUser.id) {
                return editedUser;
            }
            return user;
        })
        renderUsers();

        console.log("ПОЛЬЗОВАТЕЛЬ УСПЕШНО ОТРЕДАКТИРОВАН");
    } catch (error) {
        console.error("ОШИБКА при редактировании пользователя: ", error.message)
    }
}

// ----- Создание нового пользователя -----
const createNewUser = async () => {
    try {
        const response = await fetch(MOCK_API_URL, {
            method: 'POST',
            body: JSON.stringify({
                name: "Valentina",
                city: "Tokio",
                avatar: "https://avatars.mds.yandex.net/i?id=7d3c8e0a5e3e1ea0705bdd6c0139af4b6767cc57-10852819-images-thumbs&n=13"
            }),
        });
    }

```

```

        headers: {
            "Content-Type": "application/json",
        }
    });
    const newCreatedUser = await response.json();

    users.unshift(newCreatedUser);
    renderUsers();

    console.log("НОВЫЙ ПОЛЬЗОВАТЕЛЬ УСПЕШНО СОЗДАН");
} catch (error) {
    console.error("ОШИБКА создания нового пользователя: ", error.message)
}
}

// ----- Получение всех пользователей -----
const getUsersAsync = async () => {
    try {
        console.log("СТАРТ ПРОЦЕССА")

        const response = await fetch(MOCK_API_URL);
        users = await response.json();

        renderUsers();
    } catch (error) {
        console.error("ПОЙМАННАЯ ОШИБКА: ", error.message)
    } finally {
        console.log("ФИНИШ ПРОЦЕССА")
    }
}

getUsersAsync();

```