# ACTIVIDAD 3 - ARQUITECTURA CLIENTE SERVIDOR

Iván David Caviedes León.

Facultad de Ingeniería, Corporación Universitaria Iberoamericana.

202231: Ingeniería de Software

Diana Pinilla

5 de abril 2021.

## EXPLICACIÓN DEL PROYECTO

La idea del proyecto es crear un **CRUD** de datos basado en usuarios, en el servidor se crearon diferentes funcionalidades para el manejo de los datos de cliente para poder acceder a las dichas funcionalidades se acceden por medio de endpoints que es el modo de comunicación con el cliente dichos endpoints pueden tener diferentes métodos http para manejo algunos de los métodos son:

- GET
- POST
- PUT
- DELETE

El cliente envía los datos en forma de estructura JSON para su procesado en el servidor el servidor manipula los datos y devuelve al cliente una respuesta http que pueden ser desde el 200 para indicar que es una respuesta correcta o 500 que puede ser error interno del servidor etc.

En el cliente se realiza el diseño que ver a un usuario en el cual podrá interactuar con los endpoints del servidor enviando datos y recibiéndolos, el cliente tendrá como tarea mostrar datos y mostrar los errores surgidos en el proceso.

## ALGUNAS CAPTURAS DE PANTALLA REALIZADAS AL PROYECTO

### CONFIGURACION

```
const express = require("express");
const bodyParser = require('body-parser');
const usuarios = require('./usuarios');
const cors = require('cors');

const app = express();

app.use(cors());
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({
    extended: true
}));
```

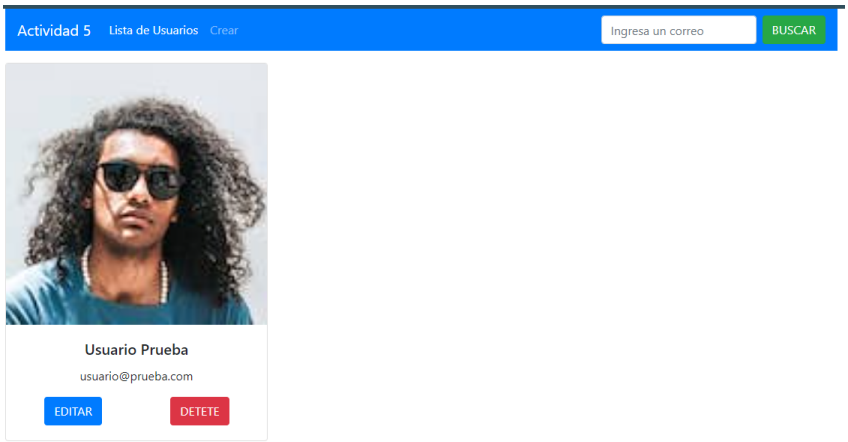
## CREACIÓN DE LOS ENDPOINTS

```
app.get('/usuarios', (req, res) => {
    res.send(usuarios);
})

app.post('/usuarios', (req, res) => {
    const usuario = req.body;
    let nImagen = Math.floor(Math.random() * (12 - 1) + 1);
    let newUsuario = { ...usuario, id: usuarios.length + 1, avatar: `https://reqres.in/img/faces/${nImagen}-image.jpg` }
    usuarios.push(newUsuario);
    res.send(newUsuario);
})
💡
app.get('/usuarios/:id', (req, res) => {
    const email = req.params.id;
    { req.params.id }
    const usuario = usuarios.find(usuario => usuario.email == email);
    if (usuario) res.send([usuario]);
    else res.status(404).send({ mensagem: 'usuario no encontrado' });
})

app.put('/usuarios/:id', (req, res) => {
    const id = req.params.id;
    const { email, first_name, last_name, avatar } = req.body;
    const usuario = usuarios.find(usuario => usuario.id == id);
    if (usuario) {
        usuario.email = email;
        usuario.first_name = first_name;
        usuario.last_name = last_name;
        usuario.avatar = avatar;
        res.send(usuario);
    } else res.status(404).send({ mensage: 'usuario no encontrado' });
})
```

## LEVANTAMIENTO DE SERVIDOR

```
app.listen(4000, () => {
    console.log("Server is running on port 3000");
});
```

## Por el lado del cliente se realizaron las siguientes vistas



## CONEXIÓN CON EL SERVIDOR POR MEDIO DE FETCH

```
const GetUsersByEmail = async (email) => {
  if (email === "") {
    GetUsers()
  }
  else {
    console.log("entro")
    const requestOptions = {
      method: 'GET',
      headers: { 'Content-Type': 'application/json' }
    };
    fetch(`http://localhost:4000/usuarios/${email}`, requestOptions)
      .then(response => response.json())
      .then(data => {
        setusuarios(data)
      });
  }
}
```

## MODELO DEL LADO DEL CLIENTE

El modelo cliente es un computador el cual accede a un servidor o a los servicios a través de internet o una red interna, las aplicaciones clientes realizan peticiones a una o varias aplicaciones servidores

**Tecnologías usadas**

**NODEJS:** Es un entorno de ejecución de multiplataforma esta basado en el lenguaje de programación **JAVASCRIPT esta** diseñado para crear aplicaciones network escalables

**REACT:** Es una biblioteca JavaScript para crear interfaces de usuario, está construido en torno a hacer funciones, que toman las actualizaciones de estado de la página y que se traduzcan en una representación virtual de la página resultante.

**FETCH:** Proporciona una interfaz JavaScript para acceder y manipular partes del canal HTTP, tales como peticiones y respuestas.

## MODELO DEL LADO DEL SERVIDOR

**Tecnologías usadas**

**NODEJS:** Es un entorno de ejecución de multiplataforma está basado en el lenguaje de programación **JAVASCRIPT está** diseñado para crear aplicaciones network escalables además te permite programar la parte Backend.

**EXPRESSJS:** Es un marco de aplicación web de Backend para Node.js, Está diseñado para crear aplicaciones web y API. Se le ha llamado el marco de servidor estándar de facto para Node.js.

**BODYPARSER:** Analice los cuerpos de las solicitudes entrantes controlada por el usuario, todas las propiedades y valores de este objeto no son de confianza y deben validarse antes de confiar

**CORS:** Intercambio de Recursos de Origen Cruzado, es una característica de seguridad del navegador que restringe las solicitudes HTTP de origen cruzado que se inician desde secuencias de comandos que se ejecutan en el navegador

**ENLACE DE GITHUB**

**https://github.com/ITSKY152/Actividad-Arquitectura**

**ENLACE DEL VIDEO EXPLICATIVO**

**https://youtu.be/Q_pDT9V1Tq0**

## BIBLIOGRAFÍA


https://www.nextu.com/blog/que-es-y-como-funciona-react-js/

https://docs.aws.amazon.com/es_es/apigateway/latest/developerguide/how-to-cors.html

https://www.linuxito.com/docs/el-modelo-cliente-servidor.pdf

https://nodejs.org/es/

https://es.wikipedia.org/wiki/Node.js

https://es.wikipedia.org/wiki/React

https://es.reactjs.org/docs/getting-started.html

https://expressjs.com/es/

https://developer.mozilla.org/es/docs/Web/API/Fetch_API/Using_Fetch

https://www.npmjs.com/package/body-parser