Diseño y Desarrollo de Servicios Web – Proyecto

Centro Hoteleria y Turismo

Analisis y Desarrollo de Software, SENA

Ficha 3070301

Aprendiz

Victor Nurel Angulo Chavez

Instructor

Juan Manuel Aldana Zambrano

Repositorio github

30 de noviembre de 2025

Contenido

Tablas

Ilustraciones

Introducción

La interfaz de usuario (UI) y la experiencia del usuario (UX) se ocupan de cómo aparece y funciona un sitio web o una aplicación, el diseño de servicios se centra en el recorrido del usuario, abarca las interacciones digitales y los puntos de contacto humanos, contribuyendo a la satisfacción general.

El diseño y desarrollo de servicios web es un campo fundamental de la ingeniería de software moderna que se centra en la creación de aplicaciones accesibles a través de Internet. Estos servicios permiten la comunicación e interoperabilidad entre diferentes sistemas de software, independientemente de su lenguaje de programación o plataforma subyacente.

Con el desarrollo de esta actividad, se busca realizar la documentación de la API construida.

Objetivo

Realizar los servicios necesarios para cumplir con las características del software.

- Realizar los servicios según requerimientos del proyecto.
- Realizar el API Rest según necesidades del proyecto.
- Realizar las validaciones de verificación.
- Utilizar herramientas de versionamiento para la creación del proyecto.

Diseño y Desarrollo de Servicios Web - Proyecto

Un proyecto de Diseño y Desarrollo de Servicios Web implica crear una plataforma digital completa, centrándose tanto en la experiencia del usuario (UX/UI) como en la funcionalidad técnica. El objetivo es desarrollar una solución web que sea intuitiva, estable y que satisfaga un conjunto específico de requisitos.

¿Qué es PHP?

PHP es un lenguaje de programación de código abierto del lado del servidor, de uso general, diseñado para el desarrollo web que permite crear sitios web dinámicos. Se ejecuta en el servidor antes de enviar el resultado al navegador del usuario (generalmente como HTML).

Servicios Web con PHP

Los servicios web con PHP se crean usando protocolos como SOAP o REST para permitir la comunicación entre diferentes aplicaciones a través de la web. Se utilizan extensiones de PHP para manejar formatos como XML y se aprovechan clases como SoapClient para consumir servicios o SoapServer para crearlos. Para APIs RESTful, se diseñan URIs, se manejan métodos HTTP (GET, POST), y se responde en formatos como JSON o XML.

Servicios en PHP

Los "servicios en PHP" pueden referirse a dos cosas: **servicios web** (APIs que permiten la comunicación entre aplicaciones) o **clases de servicio** (clases PHP que encapsulan lógica de negocio para ser reutilizada). PHP se utiliza para crear servicios web a través de protocolos como SOAP y REST, así como para construir aplicaciones web complejas con frameworks que usan clases de servicio para organizar la funcionalidad.

Herramientas de Versionamiento

Las herramientas de versionamiento, también conocidas como sistemas de control de versiones (VCS), son programas que rastrean y gestionan los cambios en archivos y código para facilitar la

colaboración y mantener un historial completo. Las herramientas más populares son **Git** (distribuido), **Subversion** y **Mercurial**. Otras opciones incluyen CVS (centralizado), y para proyectos de aprendizaje automático, **Pachyderm**.

Las herramientas de versionamiento bajo la que se ha estado implementando el proyecto son Git y GitHub.

Git

(el sistema distribuido para controlar versiones localmente) y plataformas de alojamiento en la nube como GitHub, GitLab y Bitbucket (que permiten compartir, colaborar y respaldar repositorios remotos). Estas herramientas facilitan el rastreo de cambios, la colaboración en equipo, la creación de ramas para experimentar sin afectar el código principal y la recuperación de versiones pasadas.

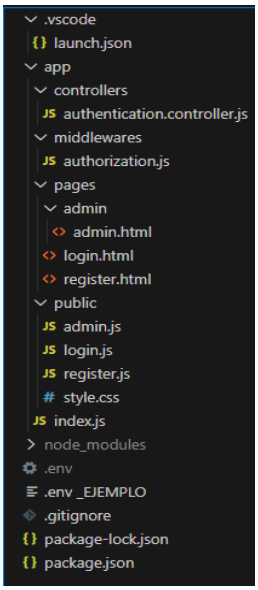
Proyecto en PHP

Un proyecto PHP es una aplicación web o sitio web construido usando el lenguaje de programación PHP, que se ejecuta en el servidor para crear contenido dinámico. Estos proyectos suelen integrar PHP con otras tecnologías como HTML, CSS y JavaScript, y a menudo usan bases de datos como MySQL para almacenar y gestionar información. Puedes desarrollar desde sitios web sencillos hasta aplicaciones complejas con frameworks de PHP.

MySQL

Es el sistema de gestión de bases de datos relacionales (RDBMS) de código abierto más popular del mundo, utilizado para almacenar, organizar y recuperar datos de manera eficiente. Utiliza el lenguaje de programación estándar SQL (Structured Query Language) para la gestión y consulta de datos.

Estructura API Login y Registro Visual Estudio Code



Index.js

JS index.js

En este archivo se levanta el servidos y contiene la ruta del puerto del servidor y de todos los servicios de la API Login y Registro. Y la configuración para la lectura de las extenciones .js y cookieParser.

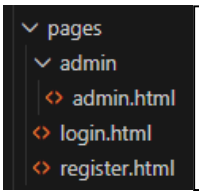
Ilustración 1 *index.js*

index.js

app > index.js > ...

```
1   import  express  from "express"; //permite levantar nuestro servidor
2   import cookieParser from 'cookie-parser';
3   //Fix para __direname
4   import path from 'path';
5   import {fileURLToPath} from 'url';
6   const __dirname = path.dirname(fileURLToPath(import.meta.url));
7   import {methods as authentication} from "./controllers/authentication.controller.js"
8   import {methods as authorization} from "./middlewares/authorization.js";
9
10  //Server
11  const app = express();//instancia de express
12  app.set("port",4000);
13  app.listen(app.get("port"));
14  console.log("Servidor corriendo en puerto",app.get("port"));//indica que esta corriendo el servidor
15
16  //Configuración
17  app.use(express.static(__dirname + "/public"));//para acceder a los arcivos de public como archivos estaticos
18  app.use(express.json()); //para que express lea el json
19  app.use(cookieParser()) //para que express lea las cookies
20
21
22  //Rutas
23  app.get("/",authorization.soloPublico, (req,res)=> res.sendFile(__dirname + "/pages/login.html"));
24  app.get("/register",authorization.soloPublico,(req,res)=> res.sendFile(__dirname + "/pages/register.html"));
25  app.get("/admin",authorization.soloAdmin,(req,res)=> res.sendFile(__dirname + "/pages/admin/admin.html"));
26  app.post("/api/login",authentication.login);
27  app.post("/api/register",authentication.register);
```

Pages



La carpeta pages contiene dos archivos (login.html y register.html) y una subcarpeta admin que contiene el archivo (admin.html), estos archivos son los encargados de definir la interfaz de cada ventana (títulos, párrafos, label, botones, etc).

Ilustración 2*visualización pantalla admin*

admin.html

app > pages > admin > admin.html > ...

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Admin</title>
    <link rel="stylesheet" href="style.css">
    <!--conexión con el scrpt de admin.js...defer hace que se carge luego del body-->
    <script src="admin.js" defer></script>
</head>
<body>
    <main>
        <div class="form-container">
            <hi>Bienvenido al área de administración</hi>
            <button>Cerrar sesión</button>
        </div>
    </main>
</body>
</html>
```

Ilustración 3 *visualización pantalla login*

login.html

app > pages > login.html > html > body

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Iniciar sesion</title>
    <link rel="stylesheet" href="style.css">
    <!--conexión con el scrpt de login.js...el defer hace que se carge luego del body-->
    <script src="login.js" defer></script>
</head>
<body>
    <main>
        <div class="form-container"> <!--creando el formulario-->
            <hi>Iniciar sesión</hi>
            <p>Ingresá tus credenciales para acceder al área de administración.</p>
            <form id="login-form">
                <label for="user" class="sr-only">User</label>
                <input type="text" name="user" id="user" placeholder="Nombre de usuario">
                <label for="password" class="sr-only">Password</label>
                <input type="password" name="password" id="password" placeholder="Contraseña">
                <button type="submit">Iniciar sesión</button> <!--submit enviar el formulario-->
                <p class="error escondido">Error al iniciar sesión</p>
            </form>
            <p>¿Todavía no temes una cuenta? - <a href="/register">Registrate</a></p>
        </div>
    </main>
</body>
</html>
```

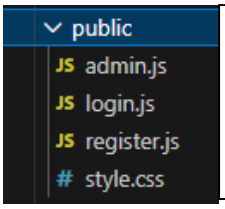
Ilustración 4 *visualización pantalla register*

register.html

app > pages > register.html > html

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Registro</title>
    <link rel="stylesheet" href="style.css">
    <!--conexión con el scrpt de register.js...defer hace que se carge luego del body-->
    <script src="register.js" defer></script>
</head>
<body>
    <main>
        <div class="form-container"> <!--creando el formulario-->
            <hi>Crear cuenta</hi>
            <p>Crear tu cuenta para acceder al área de administración.</p>
            <form id="register-form">
                <label for="user" class="sr-only">User</label>
                <input type="text" name="user" id="user" placeholder="Nombre de usuario">
                <label for="email" class="sr-only">Email</label>
                <input type="email" name="email" id="email" placeholder="Correo electrónico">
                <label for="password" class="sr-only">Password</label>
                <input type="password" name="password" id="password" placeholder="Contraseña">
                <button type="submit">Iniciar sesión</button> <!--submit enviar el formulario-->
                <p class="error escondido">Error al registrarse</p>
            </form>
            <p>¿Ya estás registrado? - <a href="/">Iniciar sesión</a></p>
        </div>
    </main>
</body>
</html>
```

Public



La carpeta public contiene tres archivos .js (admin.js, login.js y register.js) y un archivo (style.css), los archivos .js hacen el fetch para conectar con el backend y el cuerpo de la solicitud. El archivo style.css ordena y personaliza la vista por pantalla.

Ilustración 5 *admin.js*

admin.js

app > public > admin.js > ...

```
document.getElementsByTagName("button")[0].addEventListener("click",()=>{
  document.cookie ='jwt=; Path=/; Expires=Thu, 01 Jan 1970 00:00:01 GMT;';
  document.location.href = "/"
})
```

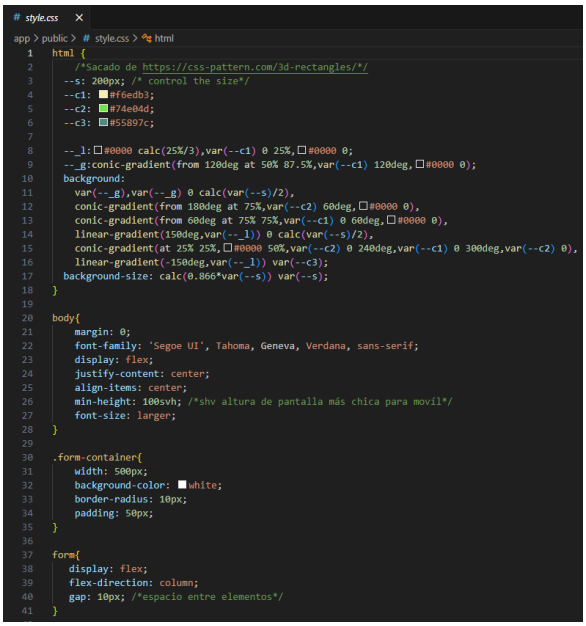
Ilustración 6 *login.js*

```
app > public > JS login.js > ...
1   const mensajeError = document.getElementsByClassName("error")[0] /*mostrar error de login*/
2
3   document.getElementById("login-form").addEventListener("submit",async (e)=>{
4     e.preventDefault();
5     const user = e.target.children.user.value;
6     const password = e.target.children.password.value;
7     const res = await fetch("http://localhost:4000/api/login",{ /*fetch para conectar con el backend*/
8       method:"POST",
9       headers:{
10        "Content-Type":"application/json"
11      },
12      body: JSON.stringify({ /*cuerpo de la solicitus*/
13        user,password
14      })
15    });
16    if(!res.ok) return mensajeError.classList.toggle("escondido",false); /*en caso de error pasa a true*/
17    const resJson = await res.json();
18    if(resJson.redirect){
19      window.location.href = resJson.redirect;
20    }
21  })
```

Ilustración 7 *register.js*

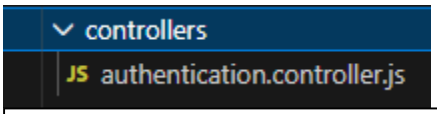
```
app > public > JS register.js > ...
1   const mensajeError = document.getElementsByClassName("error")[0]; /*mostrar error de registro*/
2
3   document.getElementById("register-form").addEventListener("submit",async(e)=>{
4     e.preventDefault();
5     console.log(e.target.children.user.value)
6     const res = await fetch("http://localhost:4000/api/register",{ /*fetch para conectar con el backend*/
7       method:"POST",
8       headers:{
9         "Content-Type" : "application/json"
10      },
11      body: JSON.stringify({ /*cuerpo de la solicitus*/
12        user: e.target.children.user.value,
13        email: e.target.children.email.value,
14        password: e.target.children.password.value
15      })
16    });
17    if(!res.ok) return mensajeError.classList.toggle("escondido",false); /*en caso de error pasa a true*/
18    const resJson = await res.json();
19    if(resJson.redirect){
20      window.location.href = resJson.redirect;
21    }
22  })
23
```

Ilustración 8 *style.css*



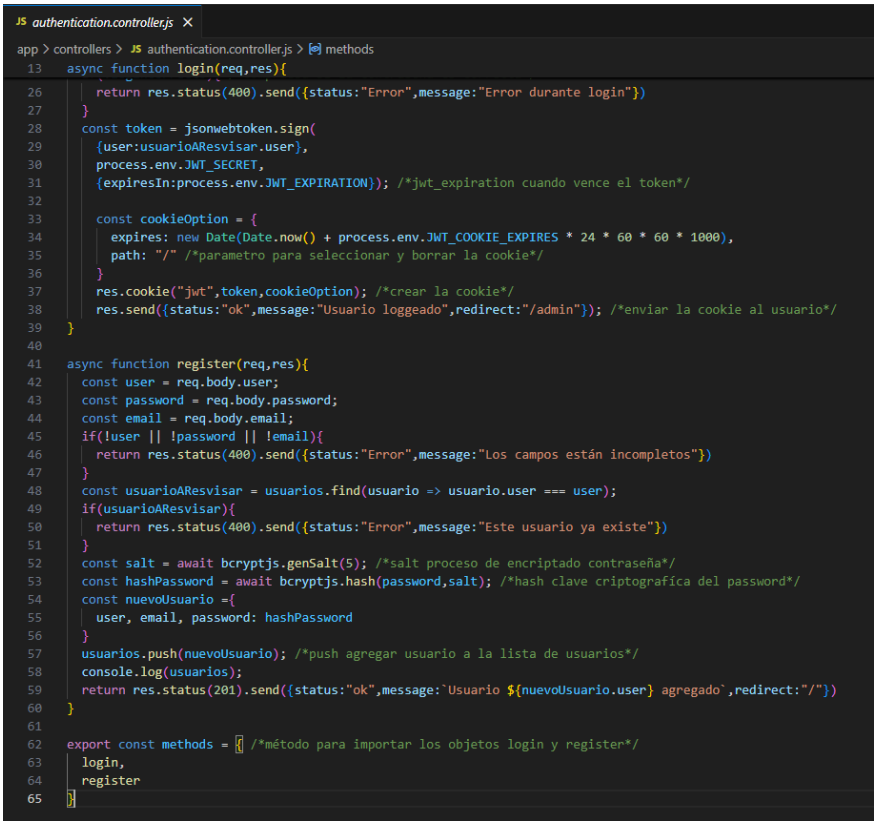
```
app > public > # style.css > html
1   html {
2     /*Sacado de https://css-pattern.com/3d-rectangles/*/
3     --s: 200px; /* control the size*/
4     --c1: #f6edb3;
5     --c2: #74e04d;
6     --c3: #55897c;
7
8     --_l:#0000 calc(25%/3),var(--c1) 0 25%,#0000 0;
9     --_g:conic-gradient(from 120deg at 50% 87.5%,var(--c1) 120deg,#0000 0);
10    background:
11      var(--_g),var(--_g) 0 calc(var(--s)/2),
12      conic-gradient(from 180deg at 75%,var(--c2) 60deg,#0000 0),
13      conic-gradient(from 60deg at 75% 75%,var(--c1) 0 60deg,#0000 0),
14      linear-gradient(150deg,var(--_l)) 0 calc(var(--s)/2),
15      conic-gradient(at 25% 25%,#0000 50%,var(--c2) 0 240deg,var(--c1) 0 300deg,var(--c2) 0),
16      linear-gradient(-150deg,var(--_l)) var(--c3);
17    background-size: calc(0.866*var(--s)) var(--s);
18  }
19
20  body{
21      margin: 0;
22      font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
23      display: flex;
24      justify-content: center;
25      align-items: center;
26      min-height: 100svh; /*shv altura de pantalla más chica para movil*/
27      font-size: larger;
28  }
29
30  .form-container{
31      width: 500px;
32      background-color: white;
33      border-radius: 10px;
34      padding: 50px;
35  }
36
37  form{
38     display: flex;
39     flex-direction: column;
40     gap: 10px; /*espacio entre elementos*/
41  }
```

Controllers



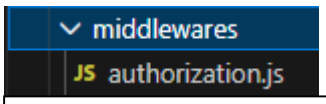
La carpeta controllers posee el archivo authentication.controller.js, que contiene el código que indica los requerimientos y respuestas asociadas a los servicios de registro y login, los parámetros para la selección y borrado de las cookies y su tiempo de expiración, encriptado de contraseñas y el método push para agregar usuario.

Ilustración 9 *authentication.controller.js*



```
authentication.controller.js
app > controllers > authentication.controller.js > methods
13  async function login(req,res){
26      return res.status(400).send({status:"Error",message:"Error durante login"})
27    }
28    const token = jsonwebtoken.sign(
29      {user:usuarioAResvisar.user},
30      process.env.JWT_SECRET,
31      {expiresIn:process.env.JWT_EXPIRATION}); /*jwt_expiration cuando vence el token*/
32
33      const cookieOption = {
34        expires: new Date(Date.now() + process.env.JWT_COOKIE_EXPIRES * 24 * 60 * 60 * 1000),
35        path: "/" /*parametro para seleccionar y borrar la cookie*/
36      }
37      res.cookie("jwt",token,cookieOption); /*crear la cookie*/
38      res.send({status:"ok",message:"Usuario loggeado",redirect:"/admin"}); /*enviar la cookie al usuario*/
39  }
40
41  async function register(req,res){
42    const user = req.body.user;
43    const password = req.body.password;
44    const email = req.body.email;
45    if(!user || !password || !email){
46      return res.status(400).send({status:"Error",message:"Los campos están incompletos"})
47    }
48    const usuarioAResvisar = usuarios.find(usuario => usuario.user === user);
49    if(usuarioAResvisar){
50      return res.status(400).send({status:"Error",message:"Este usuario ya existe"})
51    }
52    const salt = await bcryptjs.genSalt(5); /*salt proceso de encriptado contraseña*/
53    const hashPassword = await bcryptjs.hash(password,salt); /*hash clave criptografíca del password*/
54    const nuevoUsuario ={
55      user, email, password: hashPassword
56    }
57    usuarios.push(nuevoUsuario); /*push agregar usuario a la lista de usuarios*/
58    console.log(usuarios);
59    return res.status(201).send({status:"ok",message:`Usuario ${nuevoUsuario.user} agregado`,redirect:"/"})
60  }
61
62  export const methods = { /*método para importar los objetos login y register*/
63    login,
64    register
65  }
```

Middlewares



La carpeta middlewares posee el archivo authorization.js, que contiene el código que autentifica los datos para poder pasar al siguiente proceso. Y el llamado a los métodos soloAdmin y soloPublic que especifican los espacios con los que se puede interactuar según el servicio de la API en la que se encuentra el usuario.

Ilustración 10 *authorization.js*

authorization.js

app > middlewares > authorization.js > methods

```
import jsonwebtoken from "jsonwebtoken";
import dotenv from "dotenv";
import {usuarios} from "./../controllers/authentication.controller.js";

dotenv.config();

function soloAdmin(req,res,next){ //next indica continuar con el siguiente proceso
  const logueado = revisarCookie(req);
  if(logueado) return next();
  return res.redirect("/")
}

function soloPublico(req,res,next){
  const logueado = revisarCookie(req);
  if(!logueado) return next();
  return res.redirect("/admin")
}

function revisarCookie(req){
  try{
    const cookieJWT = req.headers.cookie.split("; ").find(cookie => cookie.startsWith("jwt=")).slice(4);
    const decodificada = jsonwebtoken.verify(cookieJWT,process.env.JWT_SECRET);
    console.log(decodificada)
    const usuarioAResvisar = usuarios.find(usuario => usuario.user === decodificada.user);
    console.log(usuarioAResvisar)
    if(!usuarioAResvisar){
      return false
    }
    return true;
  }
  catch{
    return false; //try catch verifica errores, no dejando cookie valida
  }
}

export const methods = {
  soloAdmin,
  soloPublico,
}
```

.env

.env
.env _EJEMPLO
.gitignore

Archivo sistema .env (contiene los datos texto de cifrado, expiración de cookies “No se dede compartir”). Archivo .env_EJEMPLO (da un ejemplo del contenido del archivo .env). archivo .gitignore (en su interior muestra los archivos que serán ignorados al subir el proyecto al repositorio en git).

.env

```
JWT_SECRET=textosecretoDECIFRADO
JWT_EXPIRATION=7d
JWT_COOKIE_EXPIRES=1
```

.env _EJEMPLO

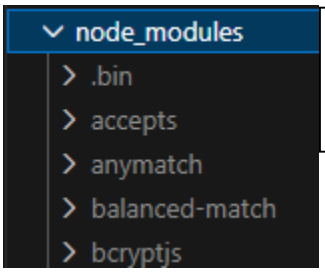
```
JWT_SECRET=
JWT_EXPIRATION=
JWT_COOKIE_EXPIRES=
```

.gitignore

```
.env
node_modules
```

Node_modules

Ilustración 11 *node_modules*

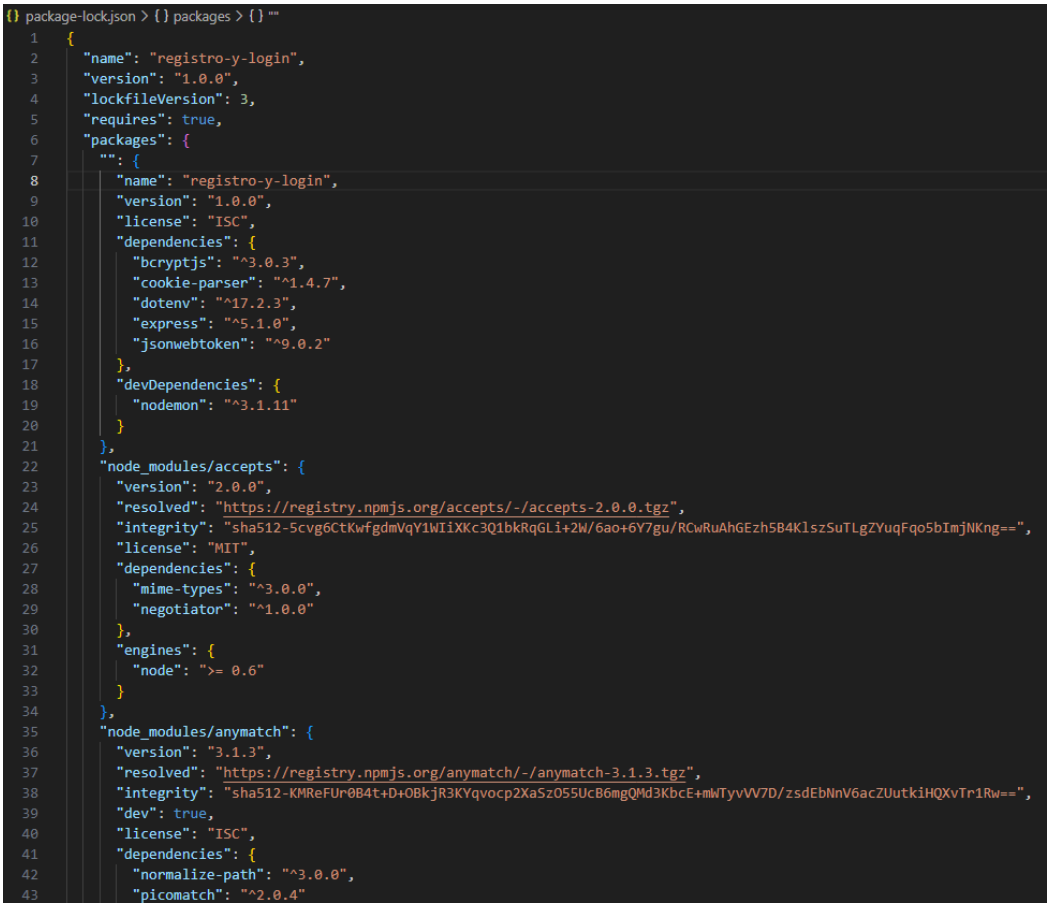


En esta carpeta node_modules, se cargan todas las librerías y dependencias instaladas (express, cookie parser, jsonwebtoken, etc).

Package-lock.json

Hace parte de las librerías y dependencias instaladas con node_modules.

Ilustración 12 *package-lock.json*



Package.json

{} package.json

Paquete que indica que se está dentro de un proyecto en node.js y carga las configuraciones por defecto del programa.

```
{} package.json > ...
{
  "name": "registro-y-login",
  "version": "1.0.0",
  "description": "",
  "main": "app/index.js",
  "type": "module",
  ▷Depurar
  "scripts": {
    "dev": "nodemon --exec node app/index.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bcryptjs": "^3.0.3",
    "cookie-parser": "^1.4.7",
    "dotenv": "^17.2.3",
    "express": "^5.1.0",
    "jsonwebtoken": "^9.0.2"
  },
  "devDependencies": {
    "nodemon": "^3.1.11"
  }
}
```

Documentación API Login y Registro

Login y Registro API

Esta Api se utiliza para acceder al registro en una página, realizar el login y acceder a la ventana admin de la Api y guarda los datos de la cuenta en la memoria del navegador.

Autentificación

La autenticación en la API se realiza mediante un nombre de usuario y contraseña valido registrado en la API.

```
bcryptjs from "bcryptjs"; /*clave criptografica para encriptar la contraseña*/

jsonwebtoken from "jsonwebtoken"; /*crear clave encriptada de autorización y
validación para dar al usuario*/
```

Base URL

La base URL para los End Points de la aplicación Login y Registro es https://localhost:4000 .

Por ejemplo, para acceder al servicio register a través de la Api, la URL completa es una solicitud

GET a http://localhost:4000/register .

Responses

Todas las respuestas devueltas por la API Login y Registro siguen la especificación JSON: API.

Errores

Login y Registro utiliza códigos de estado HTTP comunes para indicar si una solicitud fue

exitosa o hubo un error.

- Los códigos de estado en el rango 2xx indican una respuesta exitosa.
- Los códigos de estado en el rango 4xx indican errores causados por su solicitud, como la falta de envío de los parámetros requeridos.
- Los códigos de estado en el rango 5xx indican que algo salió mal con la aplicación Login y Registro.

http Status Codes

estos son los códigos de estado HTTP comunes que utiliza la API Login y Registro:

- 200 (OK): Su solicitud fue exitosa.
- 201 (Creado): Su solicitud fue exitosa y se creó un nuevo recurso para su cuenta.
- 400 (No existe el usuario): Error durante login.
- 400 (Contraseña incorrecta): Error durante login.
- 400 (Campos incompletos): campos incompletos
- 400 (No se puede procesar): Este usuario ya existe.

Limitación de velocidad

Las solicitudes se asocian a su dirección IP y se reinician cada minuto.

Referencia de puntos finales de API

GET/localhost:4000

Retorna la ventana de login de la API Login y Registro.

URL requerida.

```
url https://localhost:4000
```

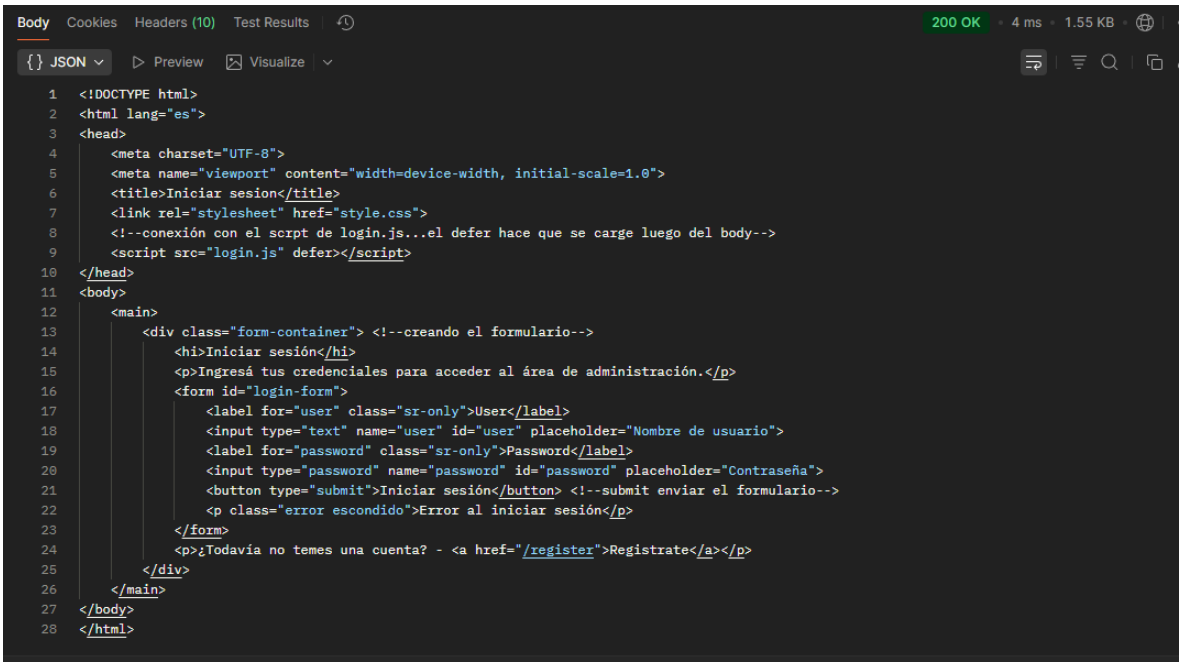
Parámetros requeridos.

Tabla 1 *parámetros get/login*

Parámetro	Tipo	Descripción
Id	String	Nombre de usuario
Id	String	Contraseña

Ejemplo de respuesta satisfactoria. Status: 200 OK

Ilustración 13 *get/login status: 200 ok*



GET/register

Retorna la ventana para el registro de la API Login y Registro.

URL requerida.

`url https://localhost:4000/register`

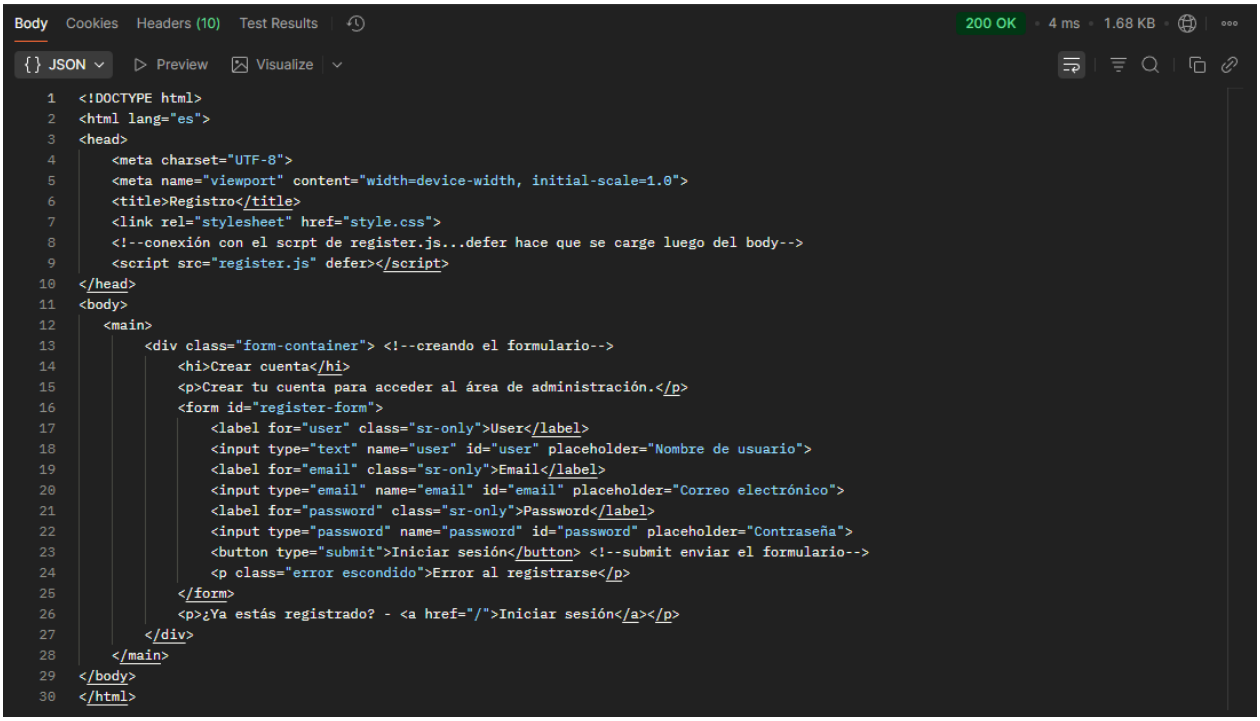
Parámetros requeridos.

Tabla 2 *parámetros get/register*

Parámetro	Tipo	Descripción
Id	String	Nombre de usuario
Id	Email	Correo electrónico
Id	Password	Contraseña

Ejemplo de respuesta satisfactoria Status: 200 OK

Ilustración 14 *get/register status: 200 ok*



POST/register

Creación de usuario

URL requerida

`url https://localhost:4000/register`

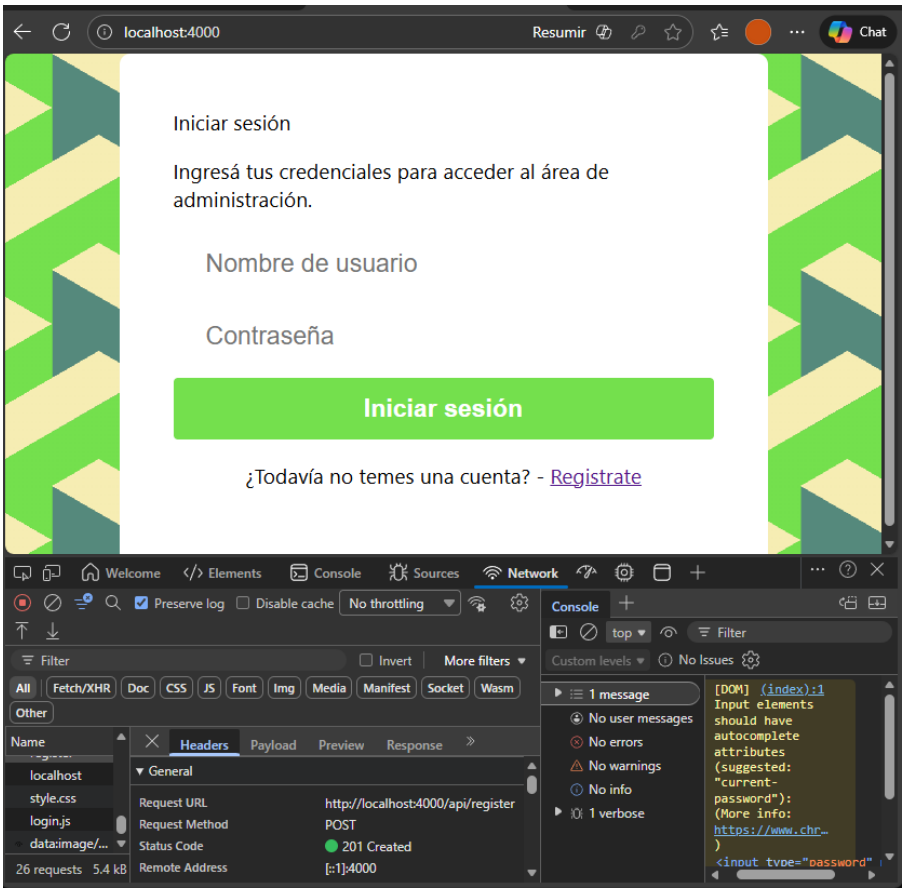
Parámetros requeridos

Tabla 3 *parámetros post/register*

Parámetro	Tipo	Descripción
Id	String	Nombre de usuario
Id	Email	Correo electrónico
Id	Password	Contraseña

Creación satisfactoria Status: 201 Created

Ilustración 15 *Status: 201 created*



La API Login y Registro cargara la ventana de login, donde se ingresan los datos registrados y accede a la ventana de admin.

POST/register

Creación de usuario

URL requerida

```
url https://localhost:4000/register
```

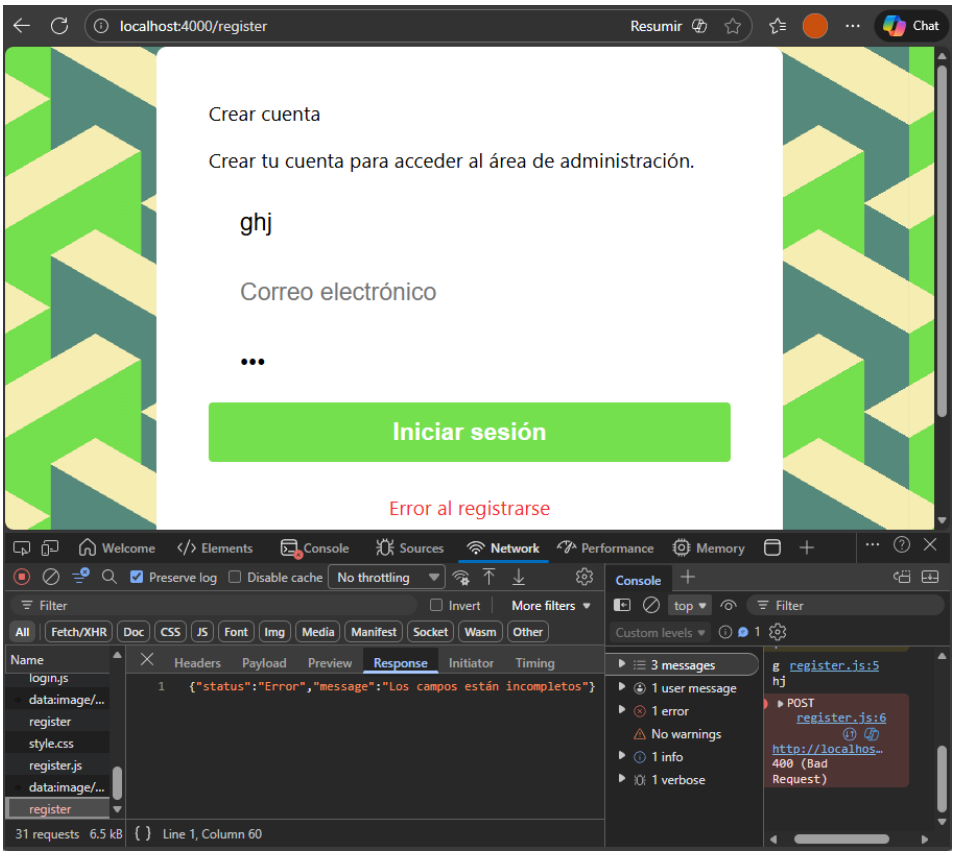
Parámetros requeridos

Tabla 4 *parámetros post/register*

Parámetro	Tipo	Descripción
Id	String	Nombre de usuario
Id	Email	
Id	Password	Contraseña

Campos incompletos Status: error 400 Bad Request

Ilustración 16 *post/register status: 400*



Response: “Los campos están incompletos”

POST/register

Creación de usuario

URL requerida

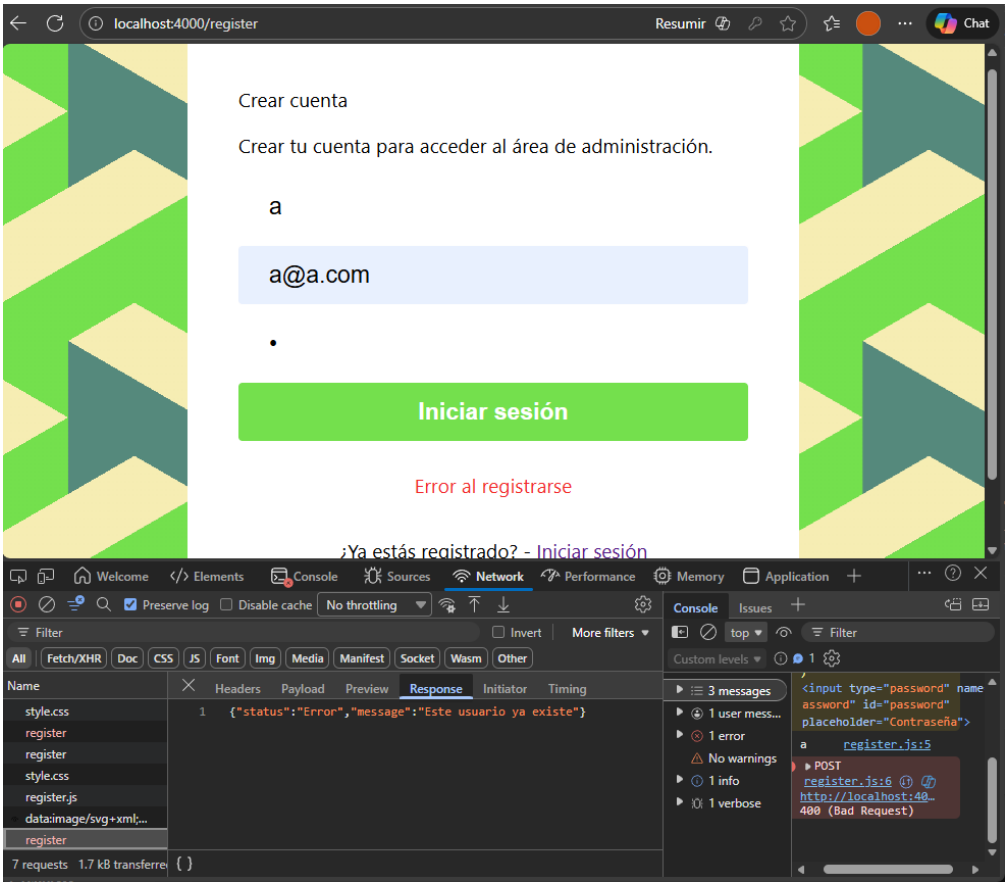
`url https://localhost:4000/register`

Parámetros requeridos

Tabla 5 *parámetros post/register*

Parámetro	Tipo	Descripción
Id	String	Nombre de usuario
Id	Email	Correo electrónico
Id	Password	Contraseña

El usuario ya existe Status: error 400 Bad Request



Response: “Este usuario ya existe” 400(Bad Request)

POST/localhost:4000

Iniciar sesión

URL requerida

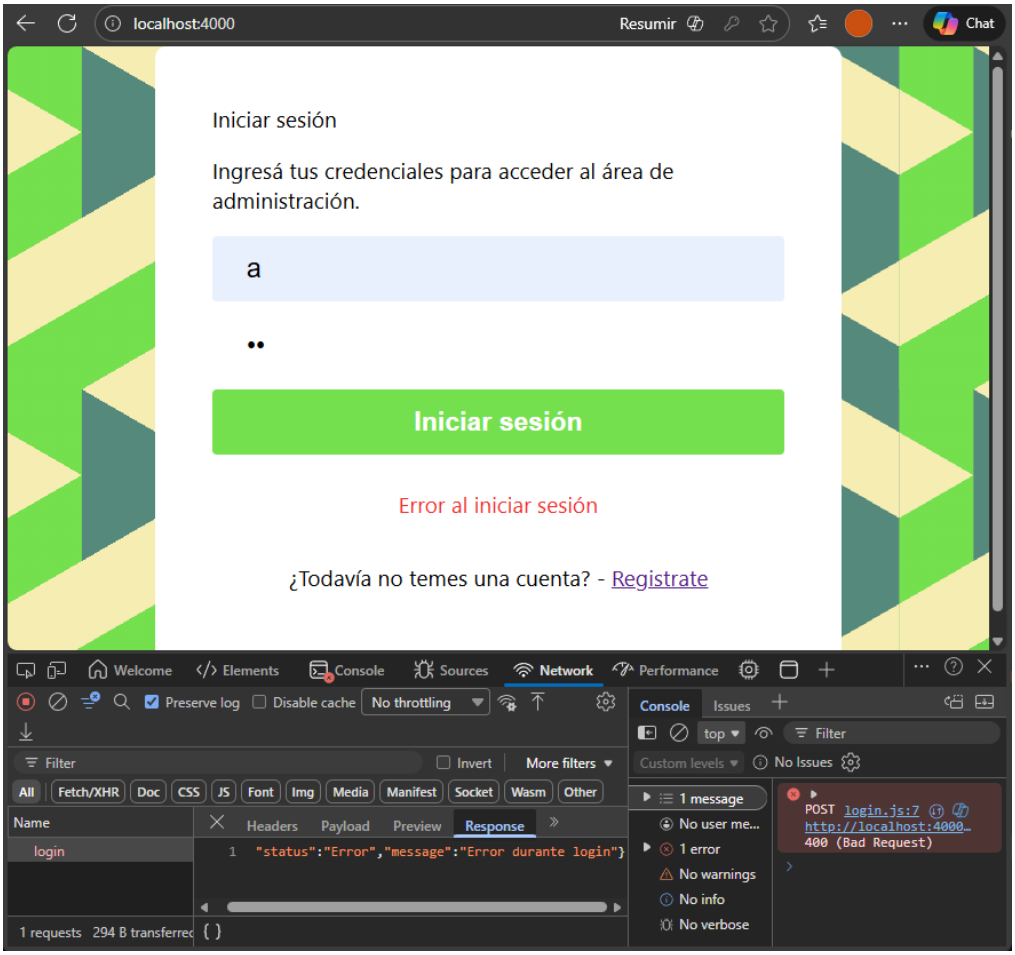
```
url https://localhost:4000
```

Parámetros requeridos

Tabla 6 *parámetros post/login*

Parámetro	Tipo	Descripción
Id	String	Nombre de usuario
Id	Password	Contraseña

Usuario o contraseña invalido Status: error 400 Bad Request



Response: “Error durante login”

Conclusión

Los servicios web son cruciales para permitir que componentes de software creados con distintos lenguajes de programación funcionen juntos sin problemas, como si hubieran sido desarrollados en el mismo lenguaje. Esto es clave para la integración de sistemas y la eficiencia operativa.

Conclusiones

Del desarrollo de la actividad anterior puedo concluir que:

- Se realizaron los servicios según requerimientos del proyecto.
- Se realizó el API Rest según necesidades del proyecto.
- Se realizó las validaciones de verificación.
- Se utilizaron herramientas de versionamiento para la creación del proyecto.

El diseño y desarrollo de servicios web es un proceso crucial y multifacético que integra la estética y la usabilidad (diseño) con la funcionalidad técnica (desarrollo), usando tecnologías como HTML, CSS y JavaScript para crear sitios atractivos y eficientes.

Referencias Web

- What is API Testing? (with Examples) https://www.browserstack.com/guide/what-is-api-testing#:~:text=API%20testing%20plays%20a%20crucial,development%20and%20quality%20assurance%20practices.
- ¿Qué son las pruebas de API? https://www.ibm.com/es-es/think/topics/api-testing#:~:text=Mejorar%20el%20tiempo%20medio%20de,que%20afecten%20a%20los%20usuarios.
- ¿Qué es Postman? Cómo usar Postman para probar APIs https://www.blazemeter.com/blog/how-use-postman-test-apis#:~:text=Postman%20es%20un%20cliente%20API,as%C3%AD%20como%20leer%20sus%20respuestas.