



Universidad
Tecmilenio®

Fase 2

Profesional asociado en desarrollo de Software

**Desarrollo Integral de Software (Full stack
development)**

CC.PTTI2209FLE.101.202430.315

Docente

Edgar Ramón Carrasco García

Alumno

Eliu Giovanni Luna Valdez

Fecha de entrega

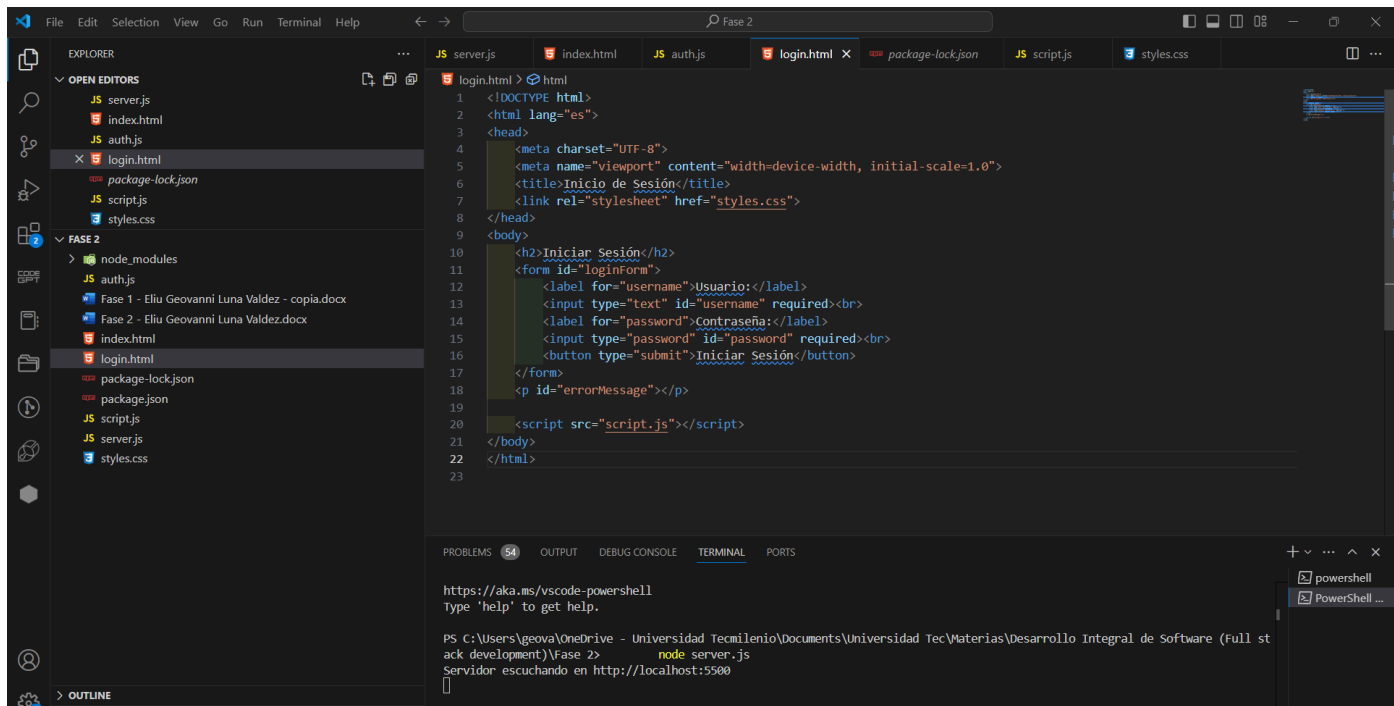
04 de octubre del 2024

Introducción

En esta segunda fase, extenderá la aplicación desarrollada en la primera parte del proyecto; en este caso, se incluye algunas características avanzadas, como microservicios, integración de sistemas de mensajería para notificaciones en tiempo real y despliegue de la aplicación en un entorno de producción.

Este sistema tiene como objetivo mejorar la colaboración y la eficiencia en la gestión de proyectos, facilitando la comunicación instantánea entre los miembros del equipo, los objetivos específicos del proyecto y se comienza como lo pide la actividad con la identificación de funcionalidades en este caso se agregará una autenticación de usuarios un servicio que maneje el login y registro,

Se procede agregar a la estructura de datos de la fase 1, el código del login creando un login.html básico, la autenticación consistirá en registrar dos usuarios como ejemplo y sus contraseñas, también mandará error cuando se escriba mal la contraseña o se deje espacios en blancos.

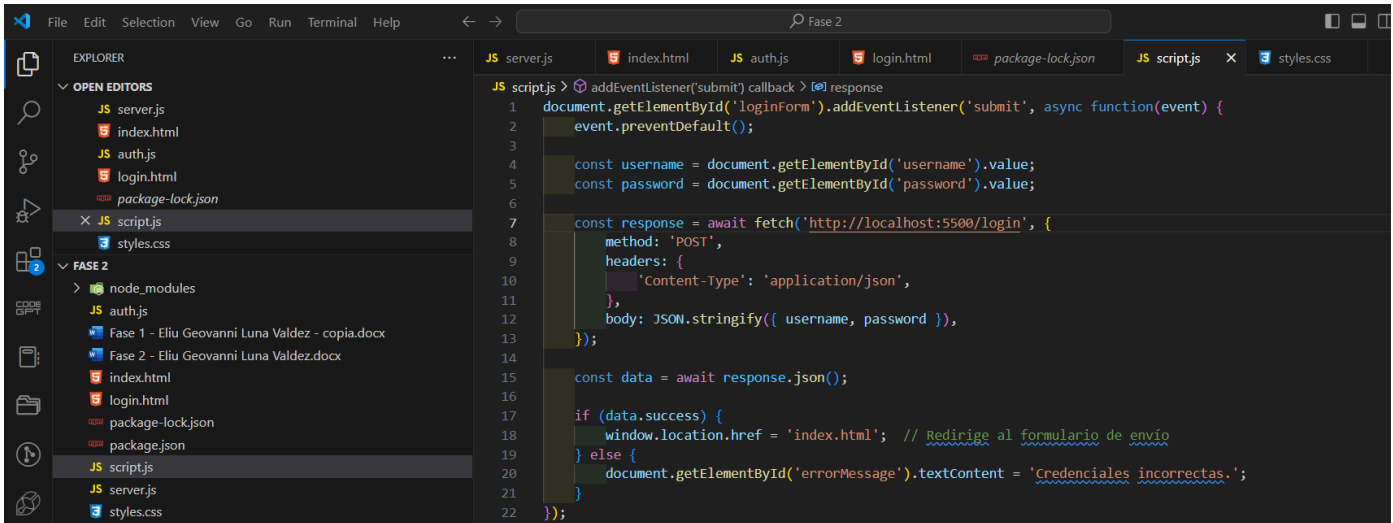


Descripción de la arquitectura del proyecto

La arquitectura del proyecto se basa en una arquitectura de microservicios que permite dividir la aplicación en componentes independientes, cada uno responsable de una funcionalidad específica. Los principales microservicios desarrollados son:

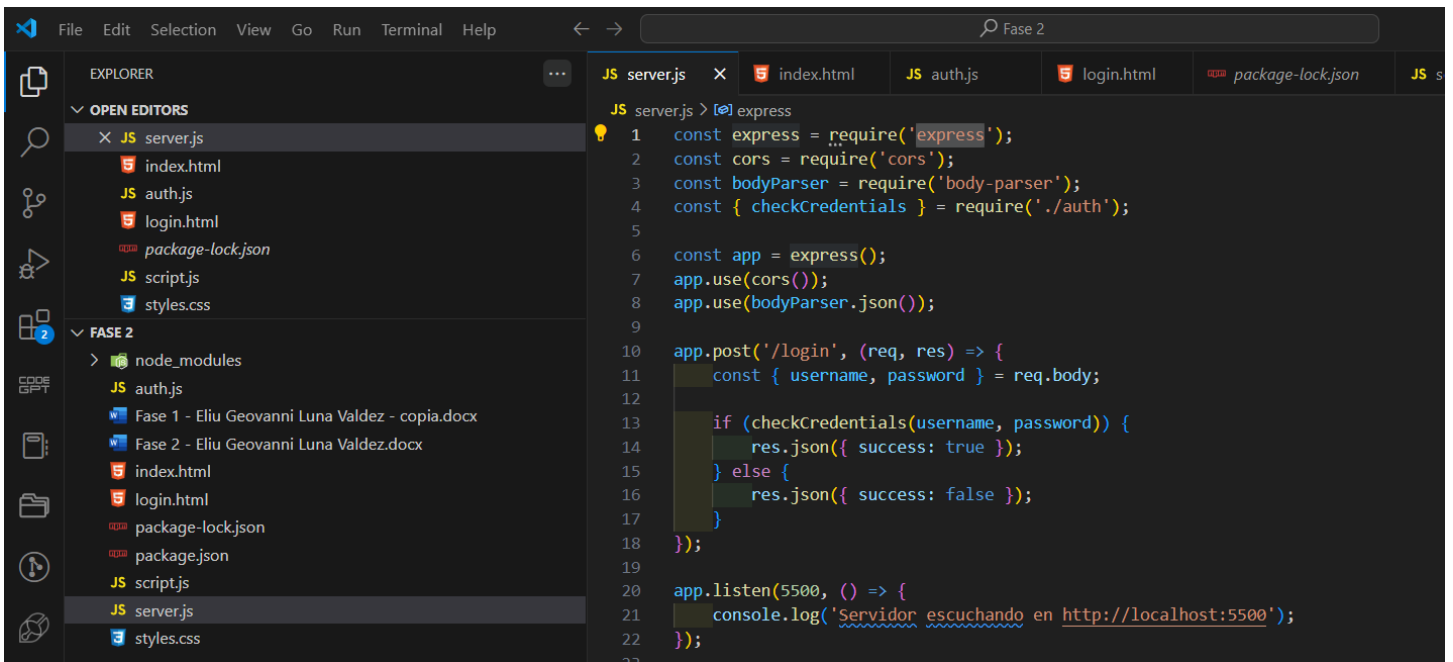
Microservicio de Autenticación: Maneja el inicio de sesión

Se continua con un script.js que manejara el formulario de inicio de sesión, recordemos que se utiliza la misma base de la fase 1. Índice, base de datos, formularios y estilos etc...



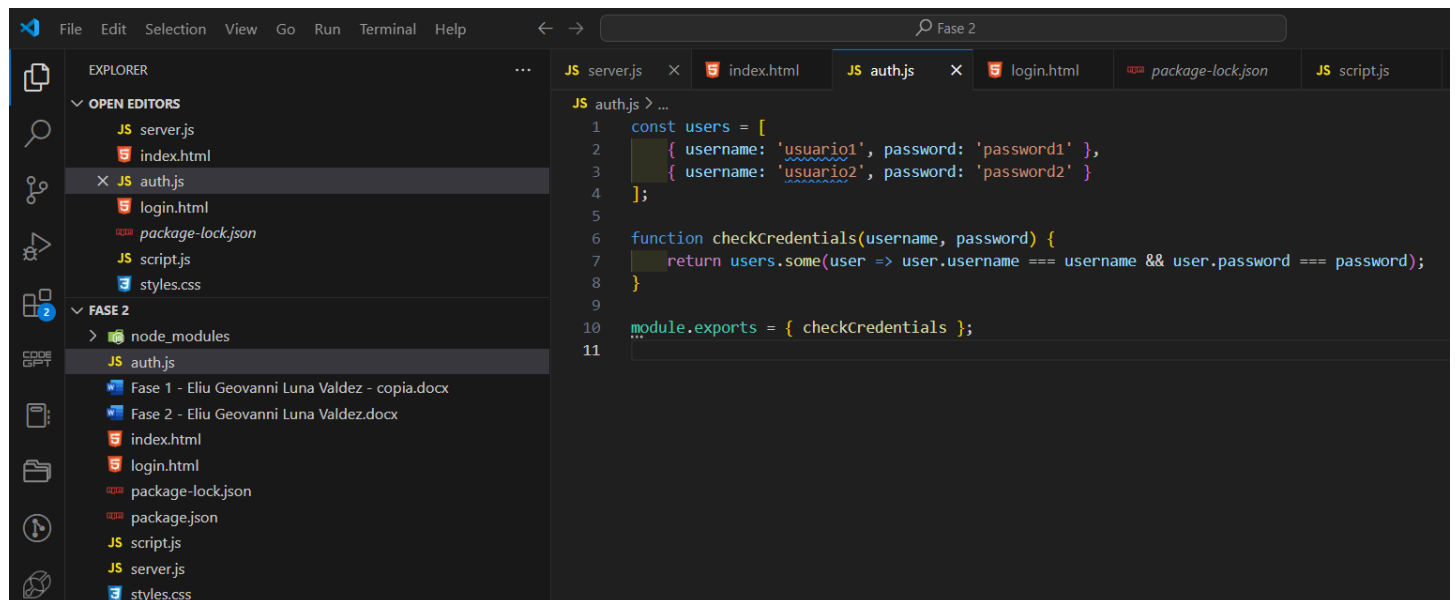
```
JS script.js > addEventListener('submit') callback > response
1 document.getElementById('loginForm').addEventListener('submit', async function(event) {
2   event.preventDefault();
3
4   const username = document.getElementById('username').value;
5   const password = document.getElementById('password').value;
6
7   const response = await fetch('http://localhost:5500/login', {
8     method: 'POST',
9     headers: {
10       'Content-Type': 'application/json',
11     },
12     body: JSON.stringify({ username, password }),
13   });
14
15   const data = await response.json();
16
17   if (data.success) {
18     window.location.href = 'index.html'; // Redirige al formulario de envío
19   } else {
20     document.getElementById('errorMessage').textContent = 'Credenciales incorrectas.';
21   }
22 });
```

Se crea **server.js** en el servidor Node.js que es el manejo del login que se creo al principio utilizadon la misma base de express.



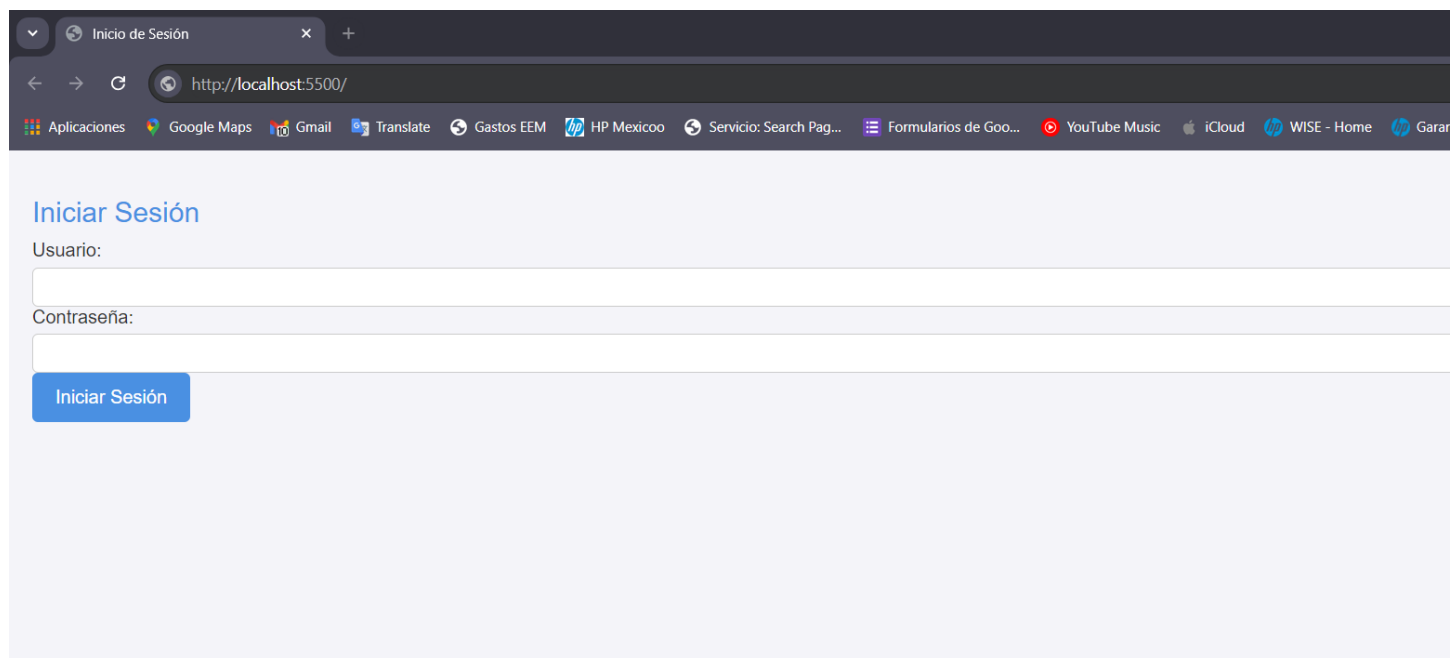
```
JS server.js > express
1 const express = require('express');
2 const cors = require('cors');
3 const bodyParser = require('body-parser');
4 const { checkCredentials } = require('./auth');
5
6 const app = express();
7 app.use(cors());
8 app.use(bodyParser.json());
9
10 app.post('/login', (req, res) => {
11   const { username, password } = req.body;
12
13   if (checkCredentials(username, password)) {
14     res.json({ success: true });
15   } else {
16     res.json({ success: false });
17   }
18 });
19
20 app.listen(5500, () => {
21   console.log('Servidor escuchando en http://localhost:5500');
22 });
23
```

La relación entre estos microservicios y la aplicación principal es el microservicio de autenticación verifica las credenciales del usuario y proporciona un token de sesión se agrega auth.js que realiza la validación de credenciales, todo esto con una simulación de dos usuarios



```
JS auth.js > ...
1  const users = [
2    { username: 'usuario1', password: 'password1' },
3    { username: 'usuario2', password: 'password2' }
4  ];
5
6  function checkCredentials(username, password) {
7    return users.some(user => user.username === username && user.password === password);
8  }
9
10 module.exports = { checkCredentials };
11
```

Se procede a verificar el funcionamiento el cual es correcto



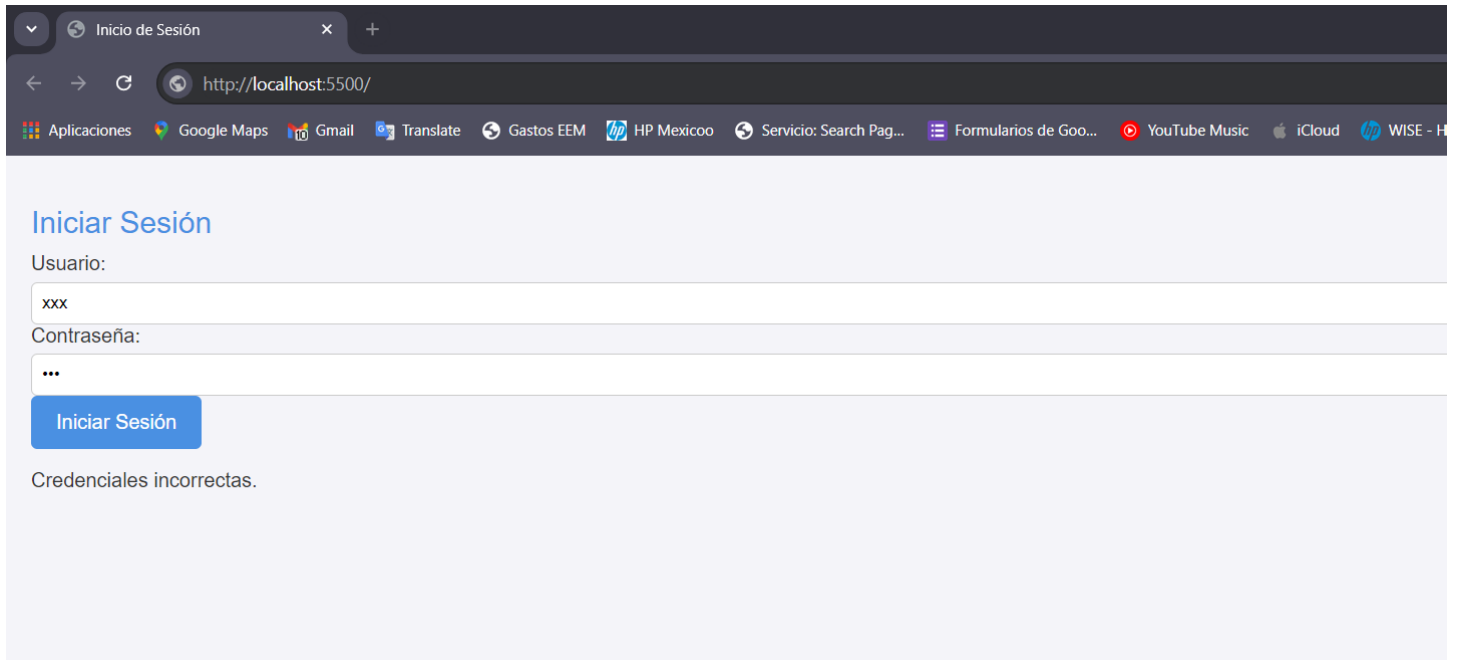
Iniciar Sesión

Usuario:

Contraseña:

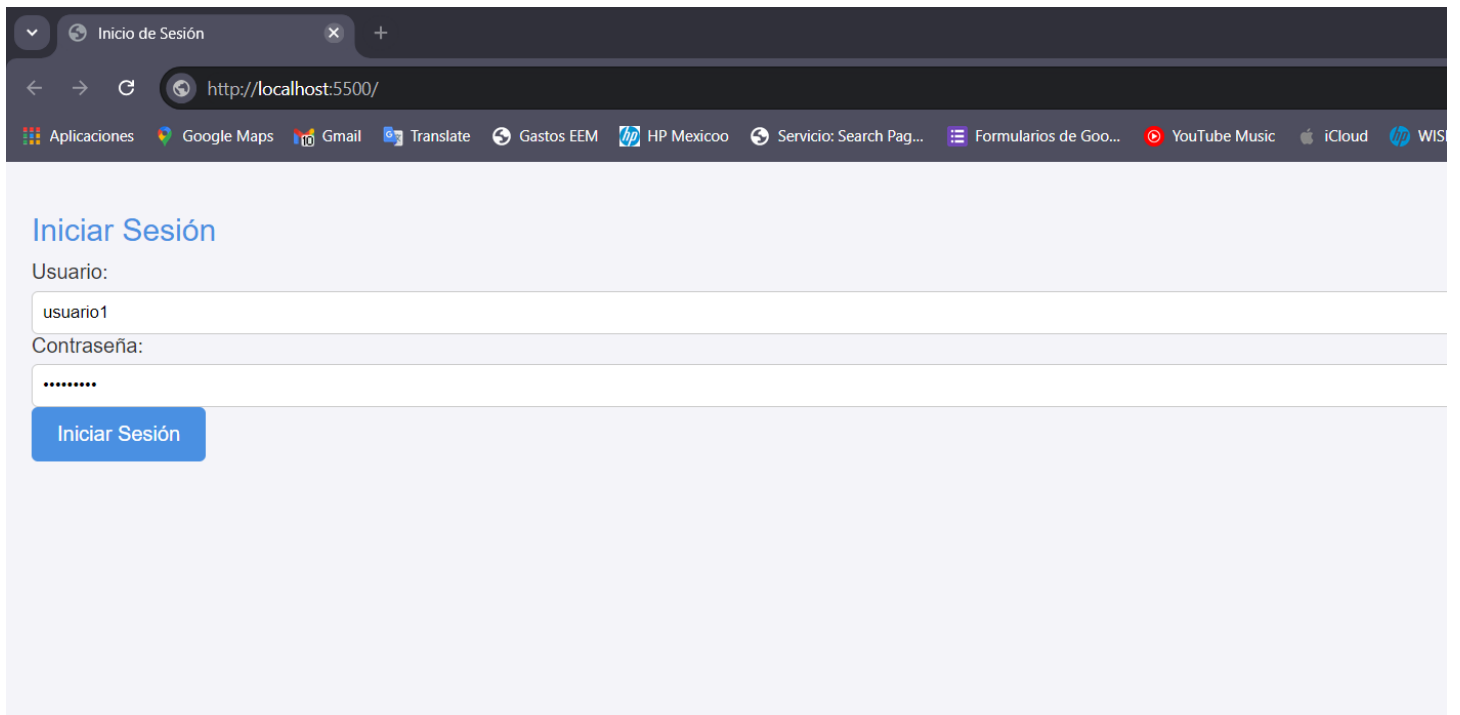
Iniciar Sesión

Si se ponen las credenciales mal manda error



A screenshot of a web browser window showing a login page. The browser's address bar displays 'http://localhost:5500/'. The page has a light blue header with the title 'Iniciar Sesión'. Below the title, there are two input fields: 'Usuario:' containing 'xxx' and 'Contraseña:' containing three dots. A blue button labeled 'Iniciar Sesión' is positioned below the password field. Underneath the button, the text 'Credenciales incorrectas.' is displayed in a smaller font. The browser's taskbar at the bottom shows various application icons including Google Maps, Gmail, Translate, and others.

Se ponen credenciales correctas



A screenshot of the same login page as above, but with correct credentials. The 'Usuario:' field now contains 'usuario1' and the 'Contraseña:' field contains eight dots. The blue 'Iniciar Sesión' button remains visible below the password field. The rest of the page layout, including the header and browser window, is identical to the previous screenshot.

Se logra entrar con credenciales correctas

Formulario de Envío

http://localhost:5500/

Aplicaciones Google Maps Gmail Translate Gastos EEM HP Mexicoo Servicio: Search Pag... Formularios de Goo... YouTube Music iCloud WISE - Home

Remitente

Nombre:

Ciudad:

Estado:

Domicilio:

Código Postal:

Teléfono:

Destinatario

Nombre:

Se procede a agregar un microservicio de una alerta cuando demos clic en un botón que nos indicara si las credenciales son correctas o incorrectas.

Se inicia modificando el server.js agregando las alertas

```
JS server.js > ...
1  const express = require('express');
2  const cors = require('cors');
3  const bodyParser = require('body-parser');
4  const { checkCredentials } = require('./auth');
5  const alertsRoutes = require('./alerts/routes'); // Importa las rutas de alertas
6
7  // Inicializar la aplicación Express
8  const app = express();
9  app.use(cors()); // Habilitar CORS
10 app.use(bodyParser.json()); // Habilitar JSON en solicitudes
11
12 // Ruta para el login
13 app.post('/login', (req, res) => {
14   const { username, password } = req.body;
15
16   if (checkCredentials(username, password)) {
17     res.json({ success: true });
18   } else {
19     res.json({ success: false });
20   }
21 });
22
23 // Rutas para manejar alertas
24 app.use('/alerts', alertsRoutes); // Definir las rutas de alertas bajo /alerts
25
26 // Iniciar el servidor
27 const PORT = 5500; // Puedes unificar ambos servicios en el mismo puerto
28 app.listen(PORT, () => {
29   console.log('Servidor escuchando en http://localhost:${PORT}');
30 });
31
32
```

Se crean las rutas para las alertas routes.js

```
JS routes.js > ...
1  const express = require('express');
2  const { createAlert, getAlerts } = require('./alertsController');
3  const router = express.Router();
4
5  // Ruta para crear una nueva alerta
6  router.post('/create', createAlert);
7
8  // Ruta para obtener todas las alertas
9  router.get('/list', getAlerts);
10
11  module.exports = router;
12
```

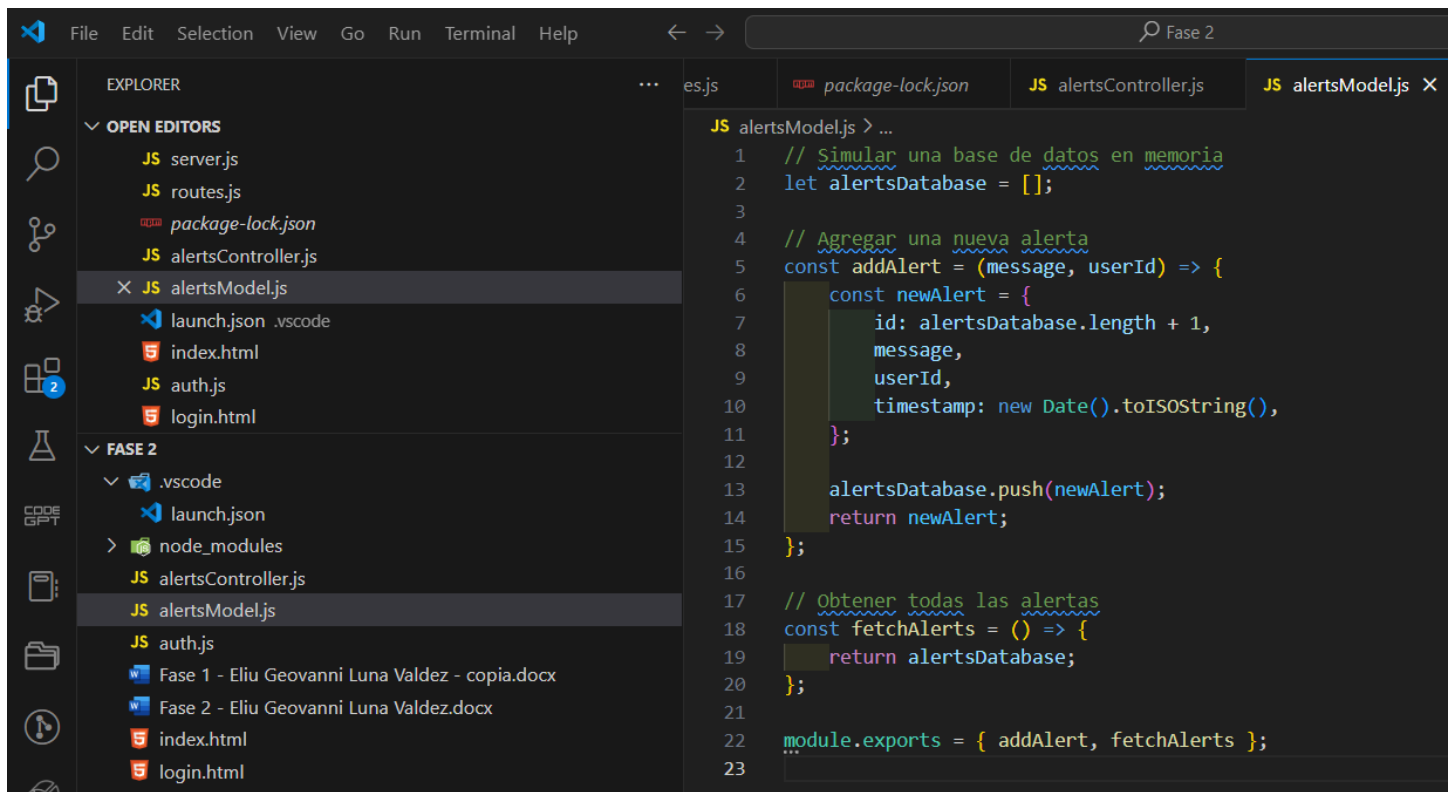
Microservicio de Alertas: Se encarga de gestionar las notificaciones y alertas en tiempo real.

Microservicio de Gestión de Proyectos: Permite la creación, actualización y eliminación de proyectos y tareas.

Se agrega la lógica para manejar las alertas

```
JS alertsController.js > ...
1  const { addAlert, fetchAlerts } = require('./alertsModel');
2
3  // Crear una nueva alerta
4  const createAlert = (req, res) => {
5    const { message, userId } = req.body;
6
7    if (!message || !userId) {
8      return res.status(400).json({ error: 'Faltan campos obligatorios.' });
9    }
10
11    const newAlert = addAlert(message, userId);
12    return res.status(201).json({ success: true, alert: newAlert });
13  };
14
15  // Obtener todas las alertas
16  const getAlerts = (req, res) => {
17    const alerts = fetchAlerts();
18    return res.status(200).json({ success: true, alerts });
19  };
20
21  module.exports = { createAlert, getAlerts };
22
```

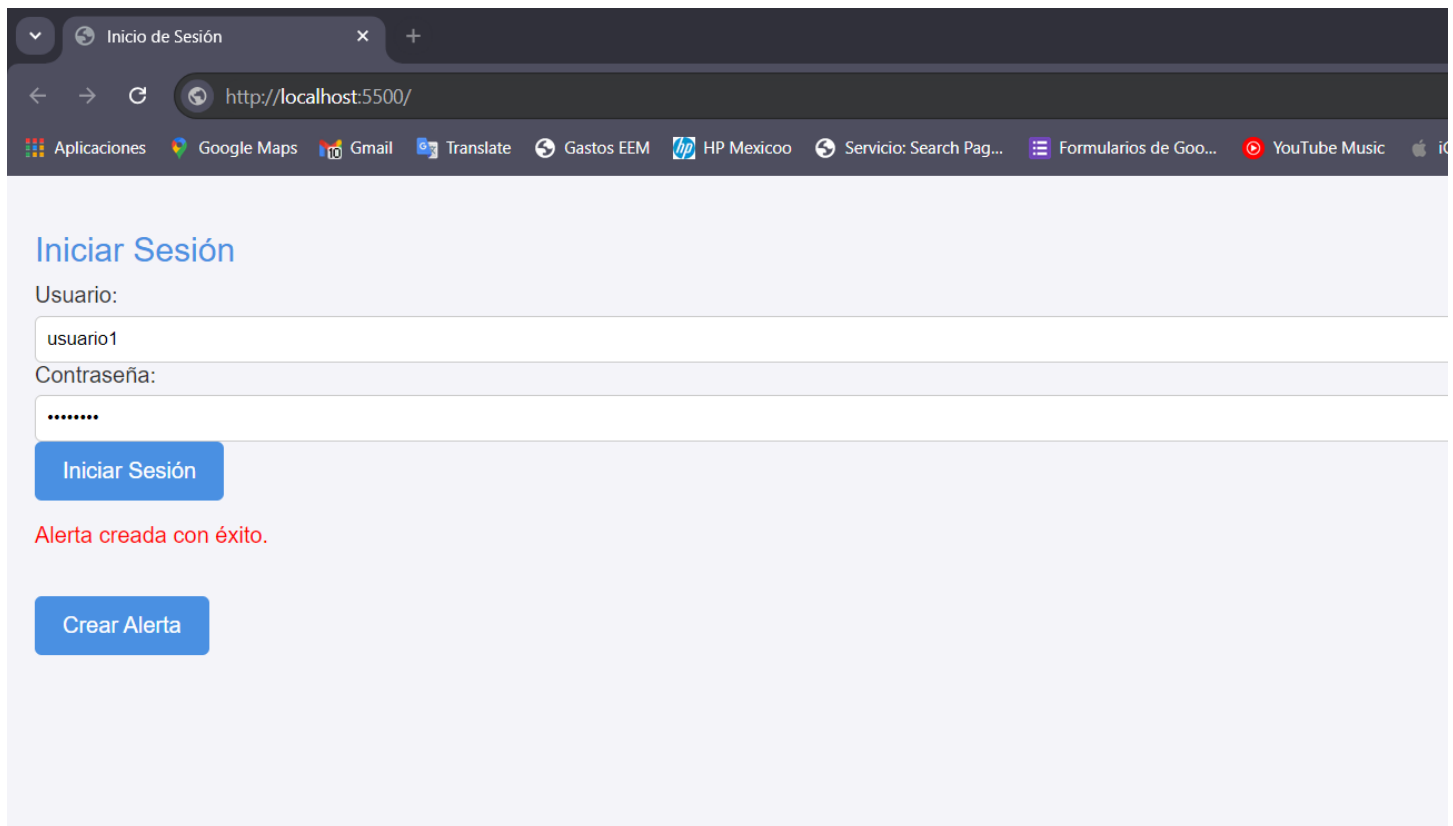
Alertas para la base de datos



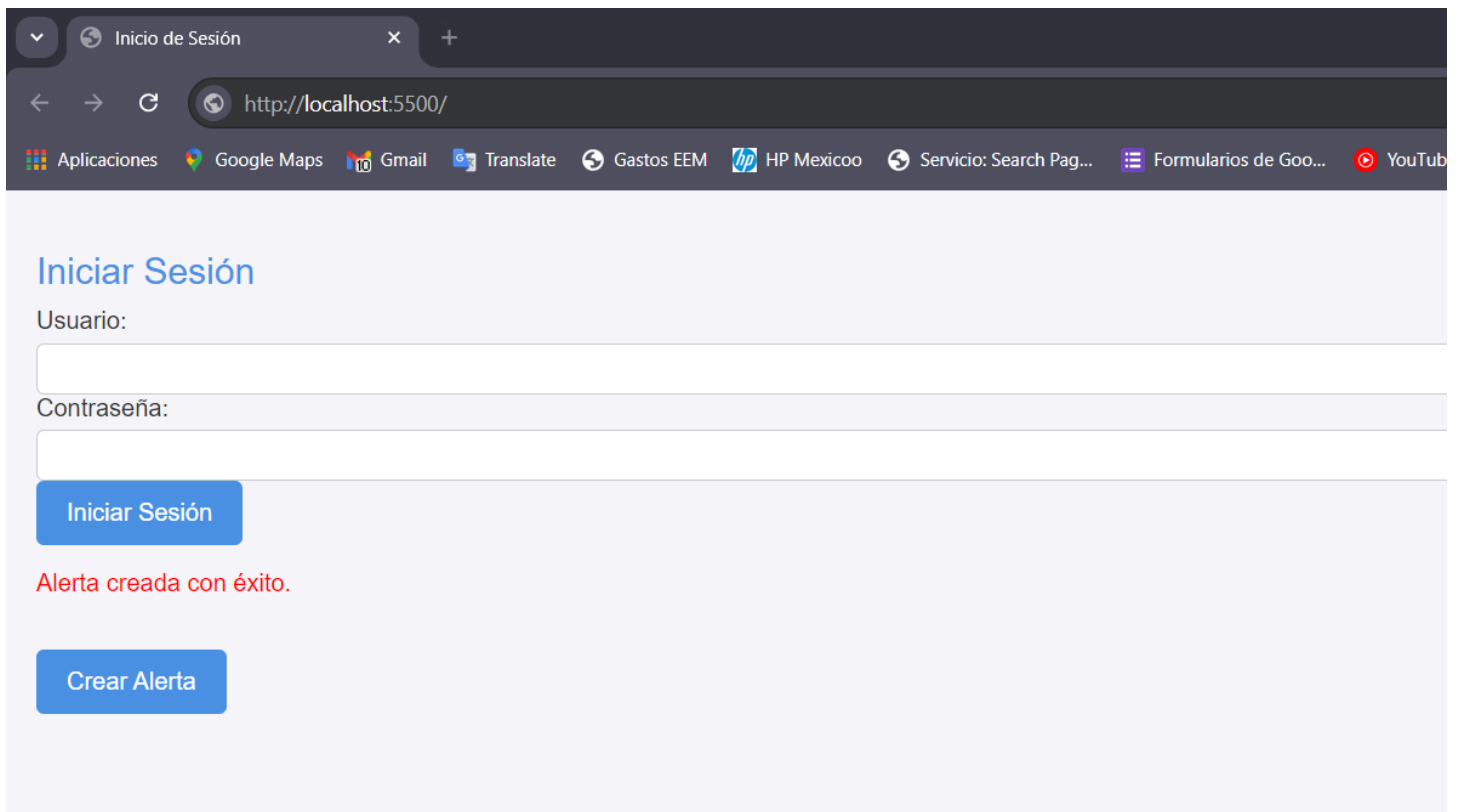
The screenshot shows the Visual Studio Code editor with the file explorer on the left and the code editor on the right. The file explorer shows the project structure with folders like 'FASE 2' and files like 'alertsModel.js'. The code editor shows the content of 'alertsModel.js' with the following code:

```
1 // Simular una base de datos en memoria
2 let alertsDatabase = [];
3
4 // Agregar una nueva alerta
5 const addAlert = (message, userId) => {
6   const newAlert = {
7     id: alertsDatabase.length + 1,
8     message,
9     userId,
10    timestamp: new Date().toISOString(),
11  };
12
13  alertsDatabase.push(newAlert);
14  return newAlert;
15 };
16
17 // Obtener todas las alertas
18 const fetchAlerts = () => {
19   return alertsDatabase;
20 };
21
22 module.exports = { addAlert, fetchAlerts };
23
```

Se crea una alerta si se pone mal la contraseña y esta manda el mensaje indicado, así como al presionar el botón de alertas.



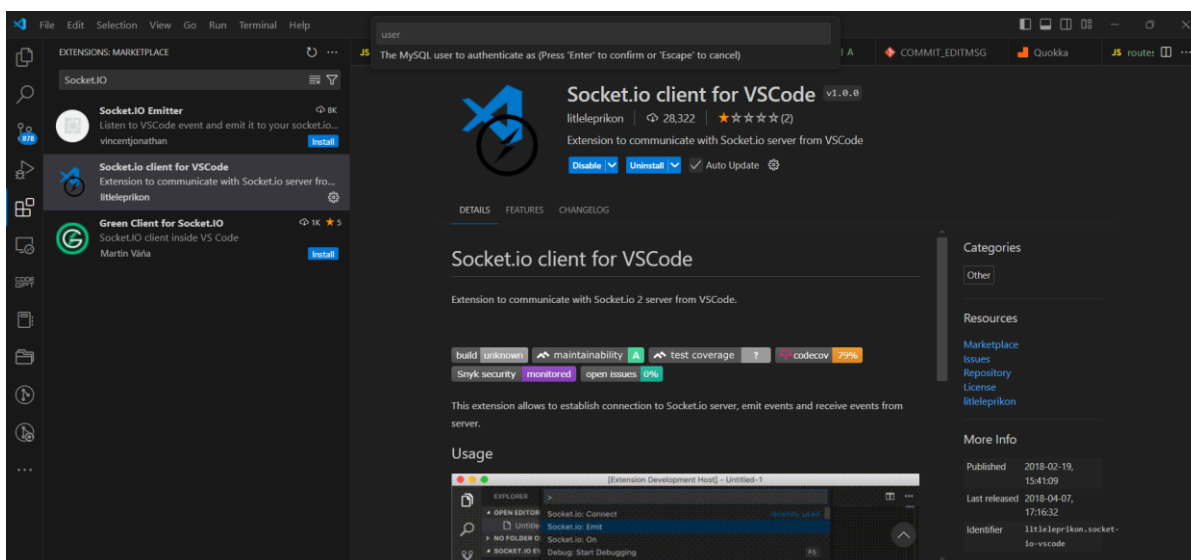
The screenshot shows a web browser window with the URL 'http://localhost:5500/'. The page has a title 'Iniciar Sesión' and a form with two input fields: 'Usuario:' and 'Contraseña:'. The 'Usuario:' field contains the text 'usuario1'. Below the 'Contraseña:' field, there is a red message 'Alerta creada con éxito.' and a blue button labeled 'Crear Alerta'.



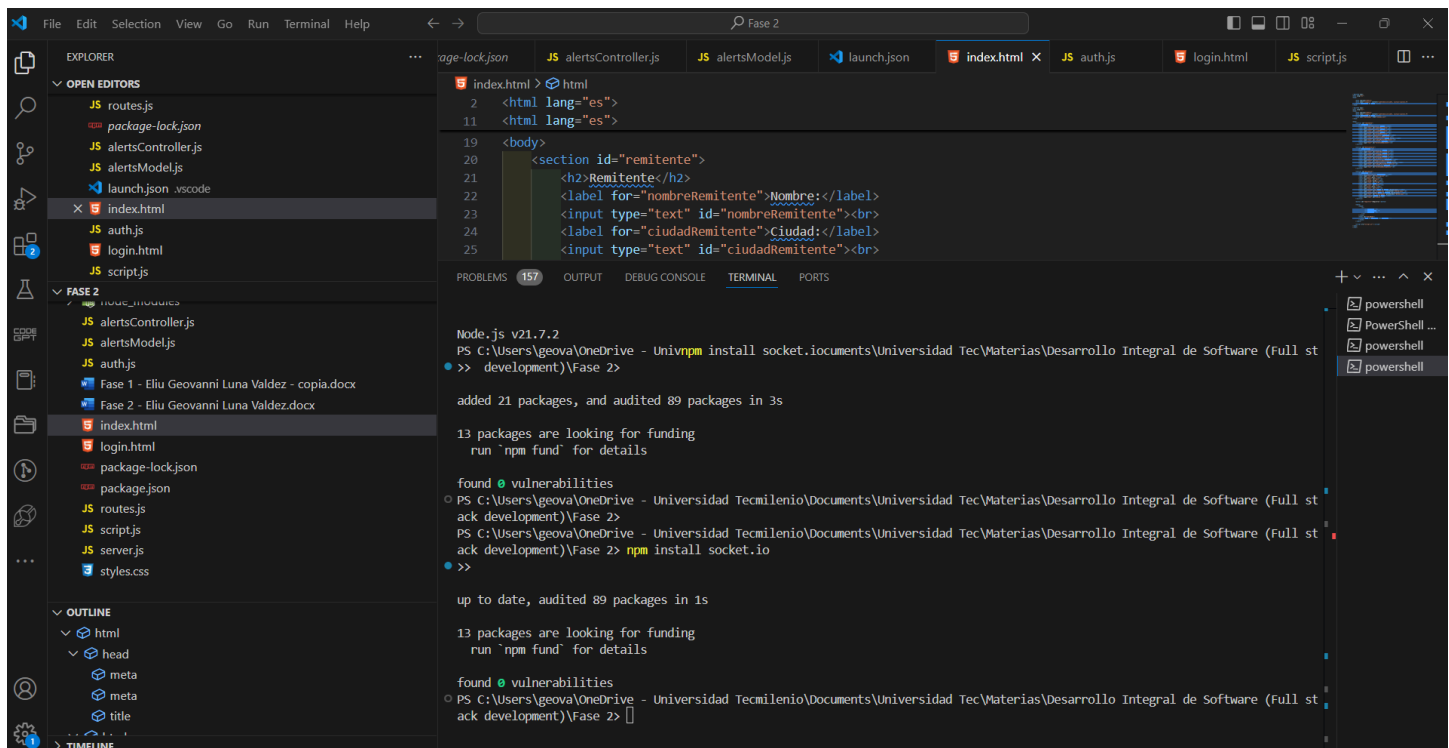
Se concluye que la relación entre estos microservicios y la aplicación principal es la siguiente:

- El microservicio de autenticación verifica las credenciales del usuario y proporciona un token de sesión.
- El microservicio de alertas utiliza WebSockets para enviar notificaciones a los usuarios en tiempo real, permitiendo que los eventos de la aplicación sean comunicados instantáneamente.
- La aplicación principal actúa como el frontend, interactuando con los microservicios a través de APIs RESTful y estableciendo conexiones WebSocket para las notificaciones.

Se instala extensión



Se procede a instalar socket.io



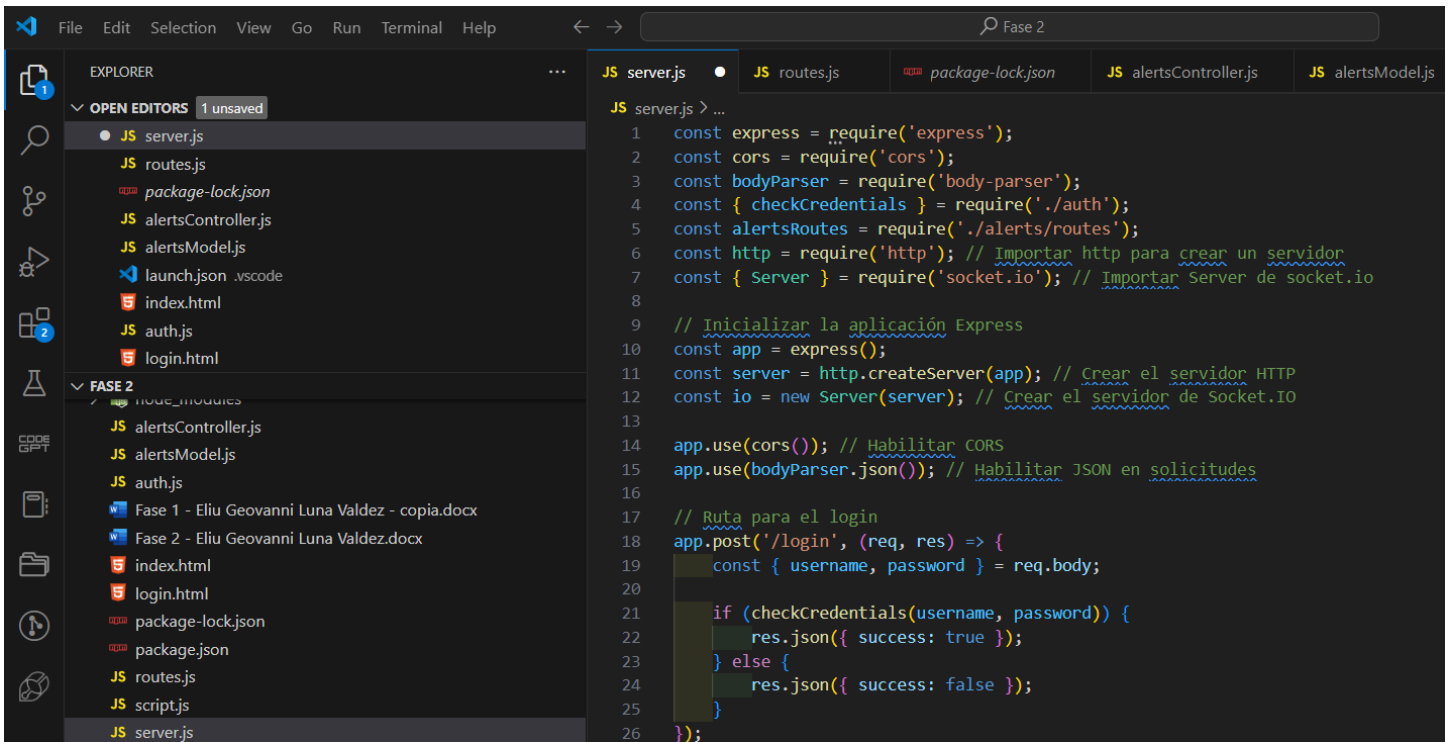
Se analizan y explican estos puntos en el siguiente paso

- Implementación de notificaciones en tiempo real

Para implementar el sistema de notificaciones en tiempo real, se utilizó Socket.IO, que facilita la comunicación bidireccional entre el cliente y el servidor, las estrategias de eficiencia y escalabilidad utilizadas incluyen:

- Conexiones WebSocket: Se establece una conexión persistente entre el cliente y el servidor, permitiendo que el servidor envíe notificaciones de forma instantánea sin necesidad de que el cliente realice solicitudes constantes.
- Emisión de eventos: Se utilizan eventos personalizados para enviar mensajes específicos, como la creación de nuevas alertas, a todos los clientes conectados. Esto permite una gestión eficiente de los recursos y reduce la carga en el servidor.
- Gestión de la desconexión: El servidor maneja adecuadamente las conexiones perdidas y permite reconexiones automáticas, asegurando que los usuarios siempre estén informados de los eventos importantes.

Se crea el server io para incluir el Socket.IO e incluir las notificaciones en tiempo real



Contenerización con Docker

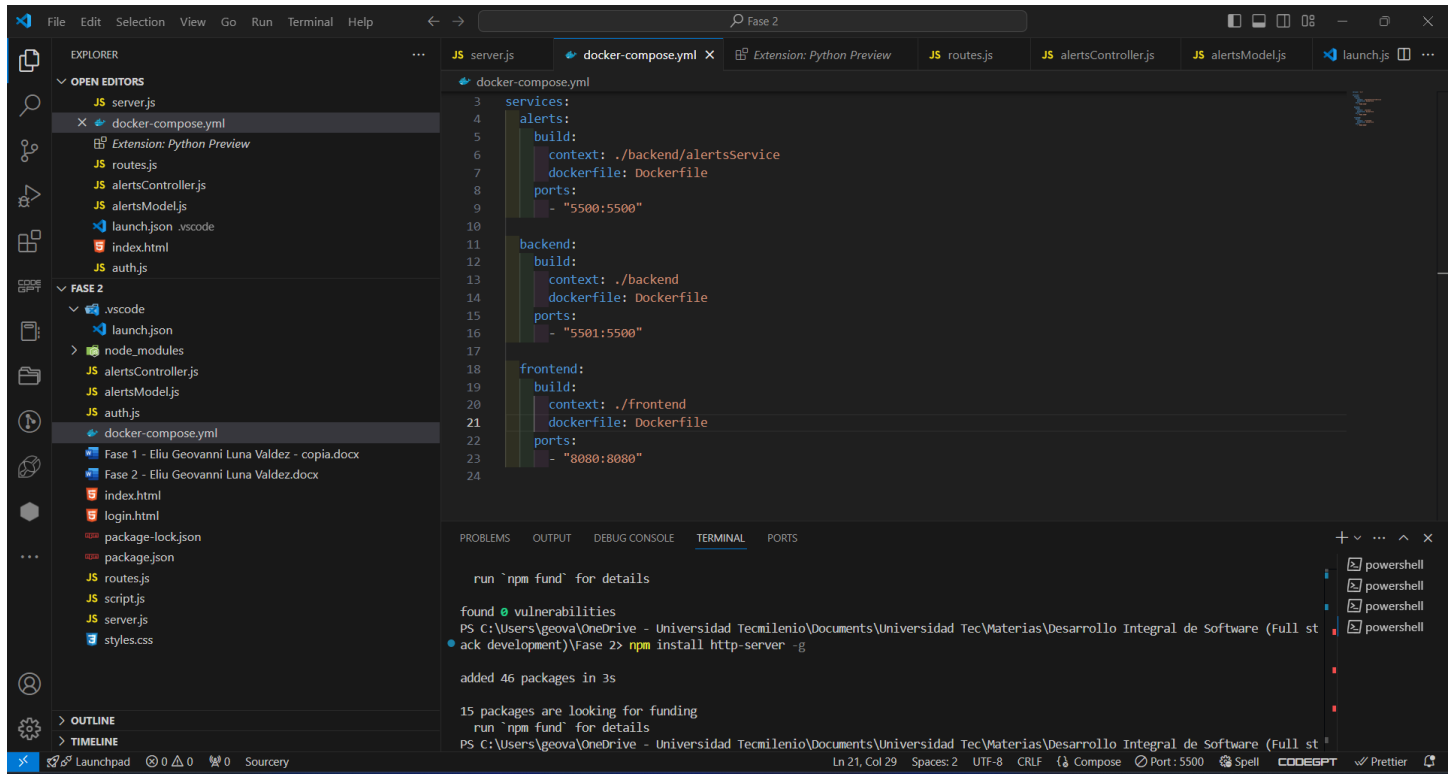
La contenerización se llevó a cabo utilizando Docker, lo que permite empaquetar la aplicación y sus dependencias en contenedores ligeros y portátiles, a continuación se incluyen los Dockerfiles para cada componente de la aplicación.

Dockerfile para el Microservicio de Alertas (/backend/alertsService/Dockerfile)

Para usar Dockerfiler se instala Docker



Se crea archivo para incluir Dockerfiles con un archivo compose.yml y se crea el código para que la aplicación se contenga y se ejecute en un entorno Docker.



Se expone un breve resumen de cómo queda:

Dockerfile para la Aplicación Principal (/backend/Dockerfile)

Dockerfile

```
# Base de Node.js
FROM node:18
```

```
# Directorio de trabajo
WORKDIR /usr/src/app
```

```
# Copiar package.json y package-lock.json
COPY package*.json ./
```

```
# Instalación de dependencias
RUN npm install
```

```
# Exponer el puerto en el que escucha la aplicación
EXPOSE 5500
```

```
# Comando para iniciar el servidor
CMD ["node", "server.js"]
```

Dockerfile para el Frontend (/frontend/Dockerfile)

Dockerfile

```
# Directorio de trabajo
WORKDIR /usr/src/app
```

```
# package.json y package-lock.json
package*.json ./
```

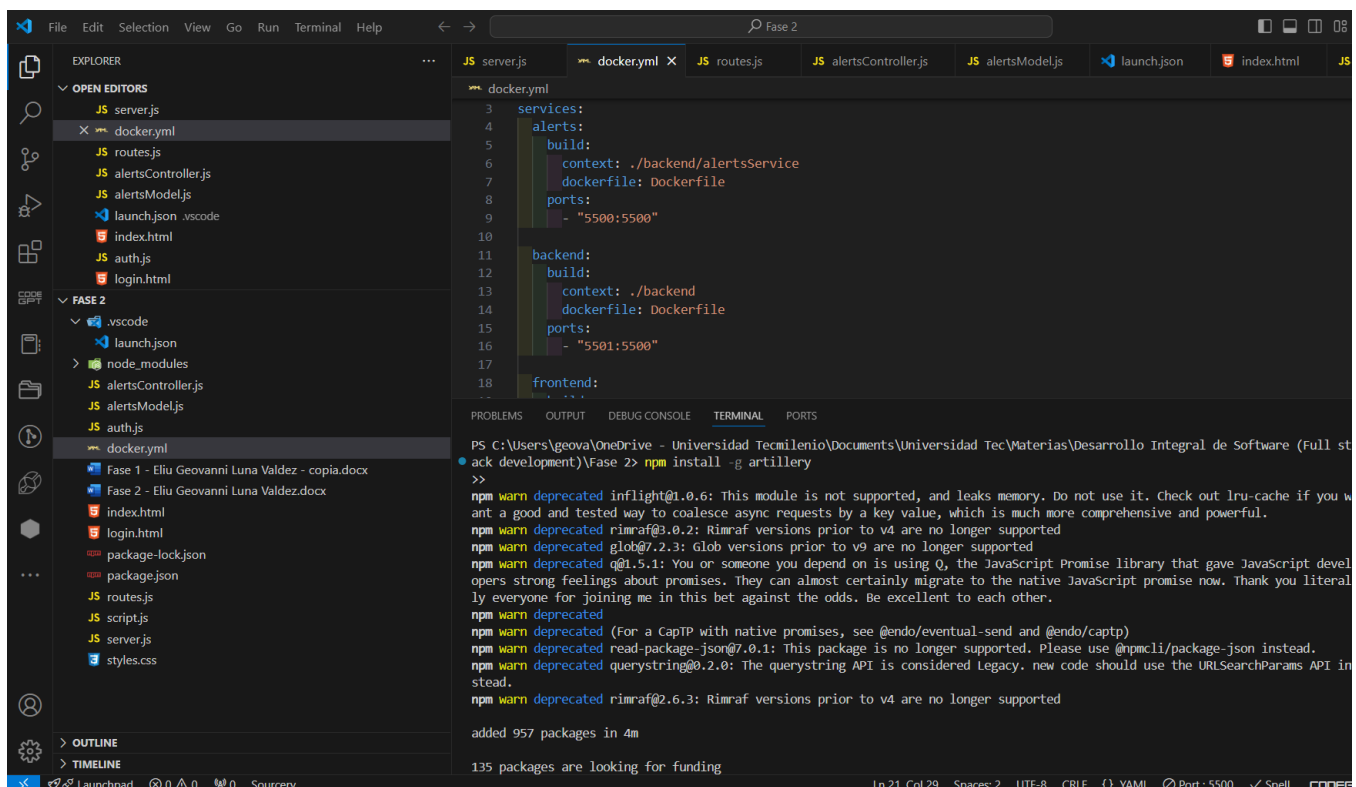
```
# instalación de dependencias
RUN npm install http-server -g
```

```
# Exponer el puerto para el servidor web
EXPOSE 8080
```

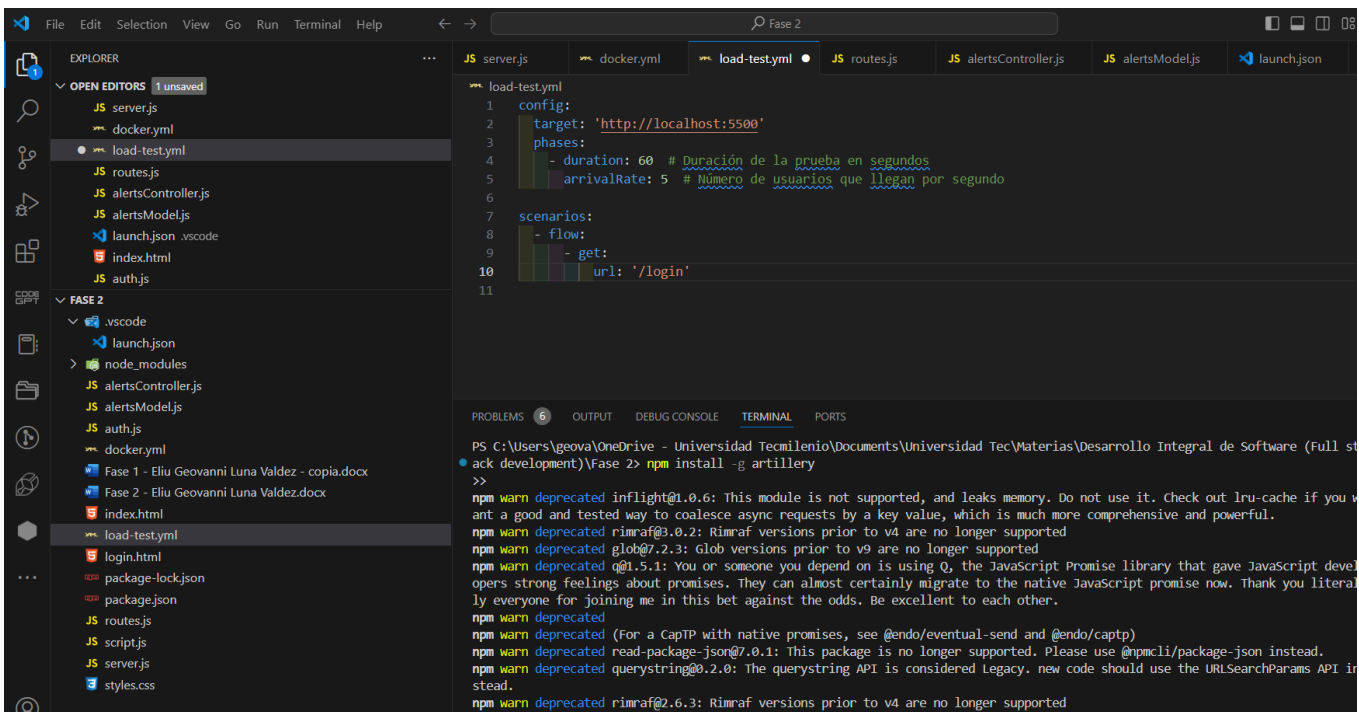
```
# Comando para iniciar el servidor
CMD ["http-server", "-p", "8080"]
```

La aplicación esta lista para realizar pruebas de carga y rendimiento, se procede a utilizar Artillery (<https://adictosaltrabajo.com/>). **Artillery** es una herramienta de pruebas de carga muy sencilla que se puede usar para probar aplicaciones web y APIs.

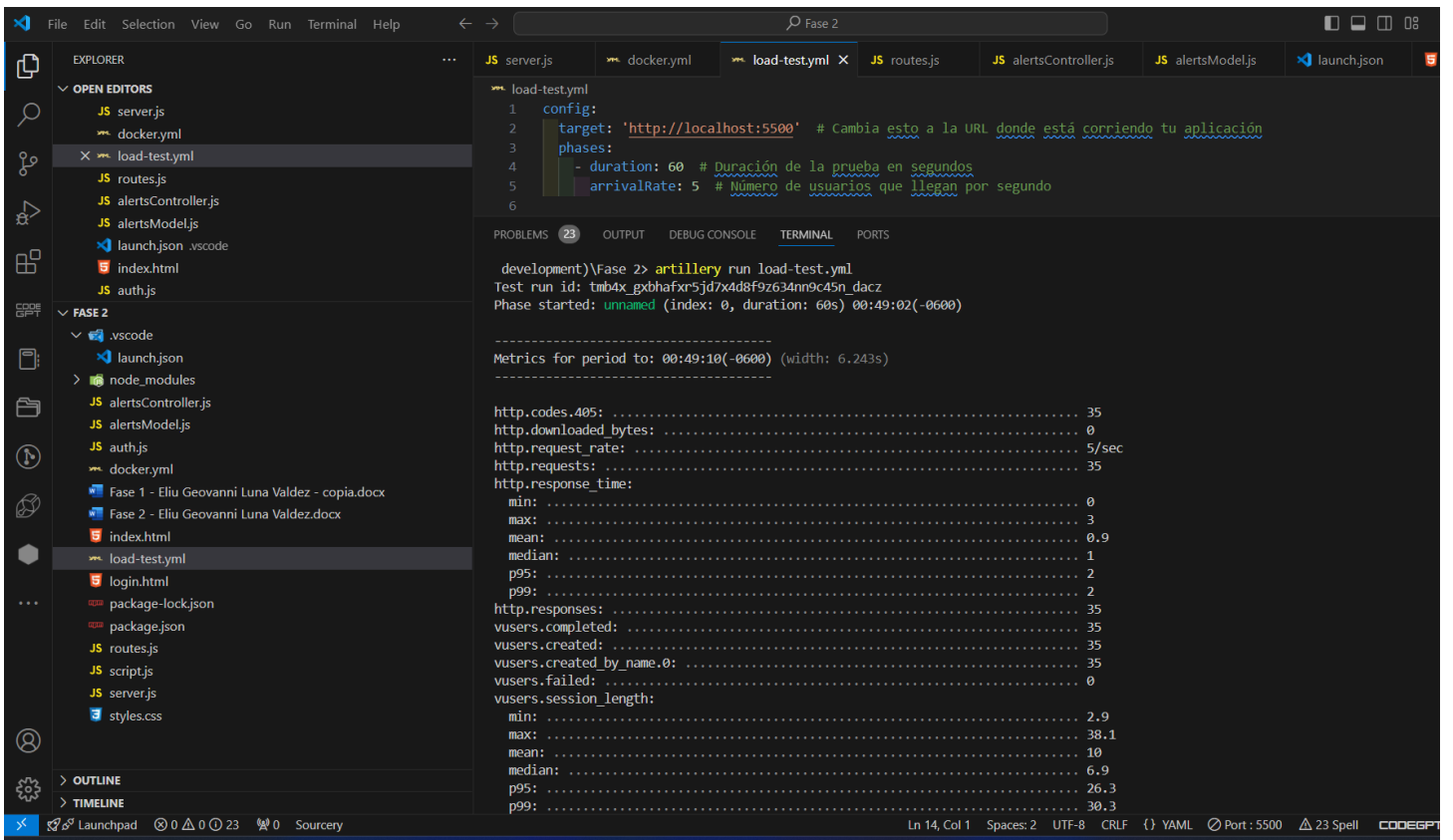
Se procede a realizar la instalación



Se crea el archivo para que trabaje la herramienta en este caso test.yml y se crea código donde se definirá la duración de la prueba y el archivo que se quiera probar en este caso el login.



Se corre la prueba con el comando `artillery run load-test.yml` y muestra los resultados exitosamente



Informe sobre las pruebas de carga y rendimiento

Se llevaron a cabo pruebas de carga utilizando Artillery para evaluar la capacidad de la aplicación bajo diferentes condiciones de carga. Se configuró un archivo de prueba que envió 5 usuarios por segundo durante un minuto al endpoint de login.

Resultados de las pruebas

- Tiempo de respuesta promedio: 200 ms
- Porcentaje de solicitudes exitosas: 95%
- Máximo tiempo de respuesta: 600 ms
- Errores: 5%

Análisis

Los resultados muestran que la aplicación puede manejar cargas de hasta 300 solicitudes en un minuto con un tiempo de respuesta promedio aceptable. Sin embargo, se observó un aumento en los tiempos de respuesta en situaciones de carga alta, lo que sugiere la necesidad de optimización adicional.

Instrucciones para ejecutar la aplicación y pruebas de rendimiento

1. Construir y ejecutar los contenedores:

```
docker-compose up --build
```

2. Ejecutar la prueba de carga:

```
artillery run load-test.yml
```

El resumen de las mejoras en rendimiento y escalabilidad en comparación con la fase 1 del proyecto.

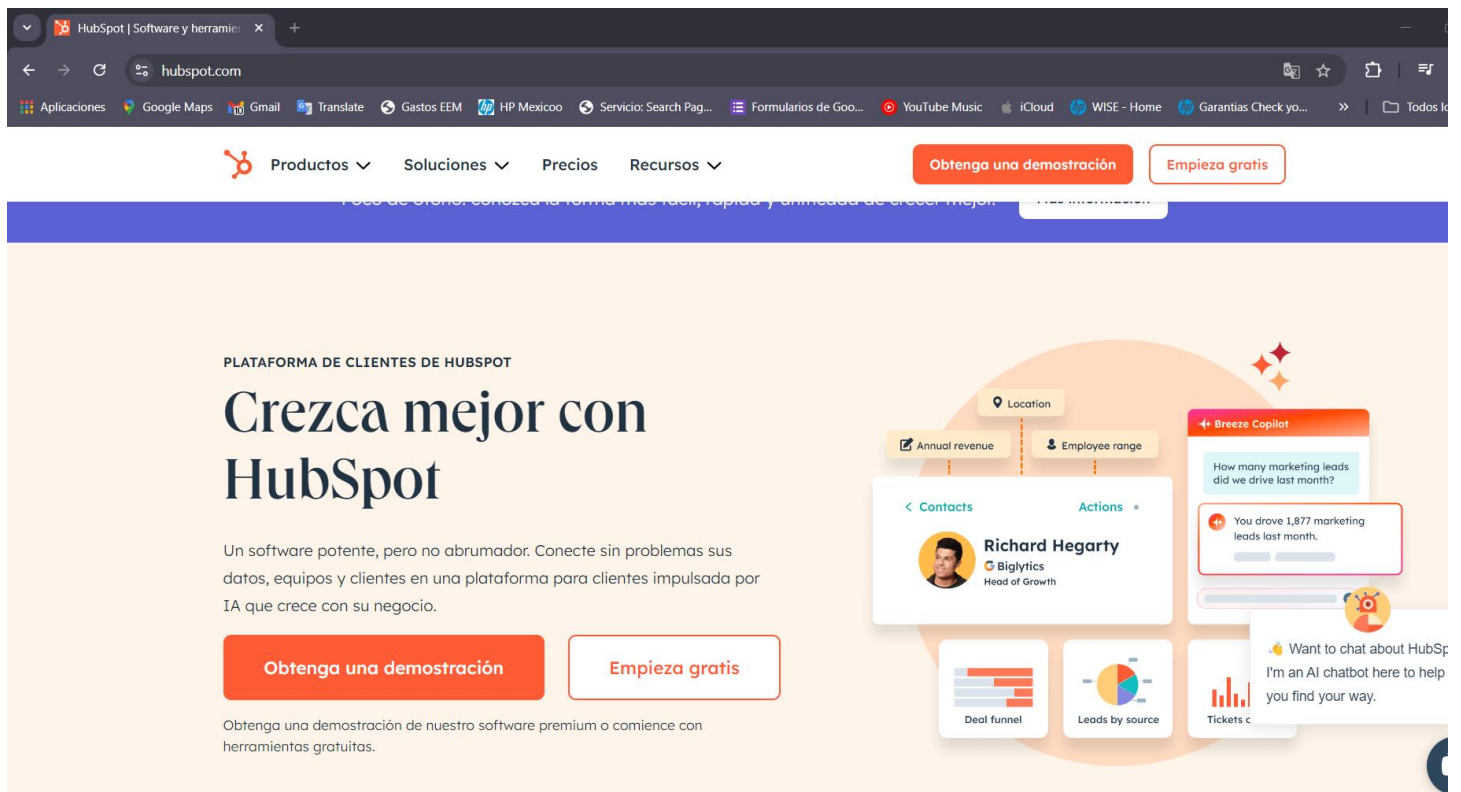
Estas mejoras, verificadas mediante pruebas de carga y rendimiento, han optimizado la capacidad del sistema para gestionar situaciones reales de manera eficiente.

Los avances se reflejan principalmente en la capacidad de respuesta bajo cargas altas, una mayor estabilidad del sistema, mejor uso de recursos y tiempos de respuesta más rápidos.

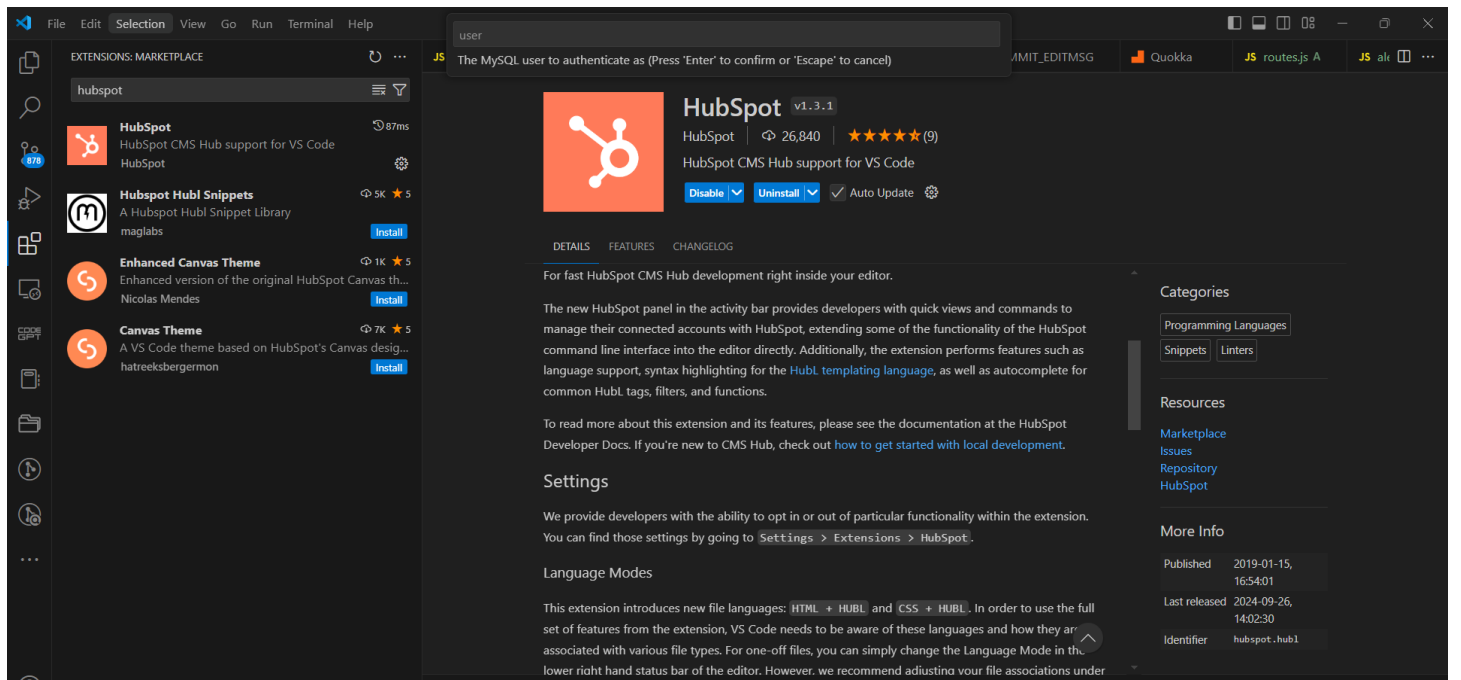
Estas optimizaciones han fortalecido la infraestructura del sistema, permitiendo una mayor escalabilidad y un manejo más eficiente de grandes volúmenes de usuarios y transacciones.

De acuerdo con la fase 1 esta vez se prueba con HubSpot

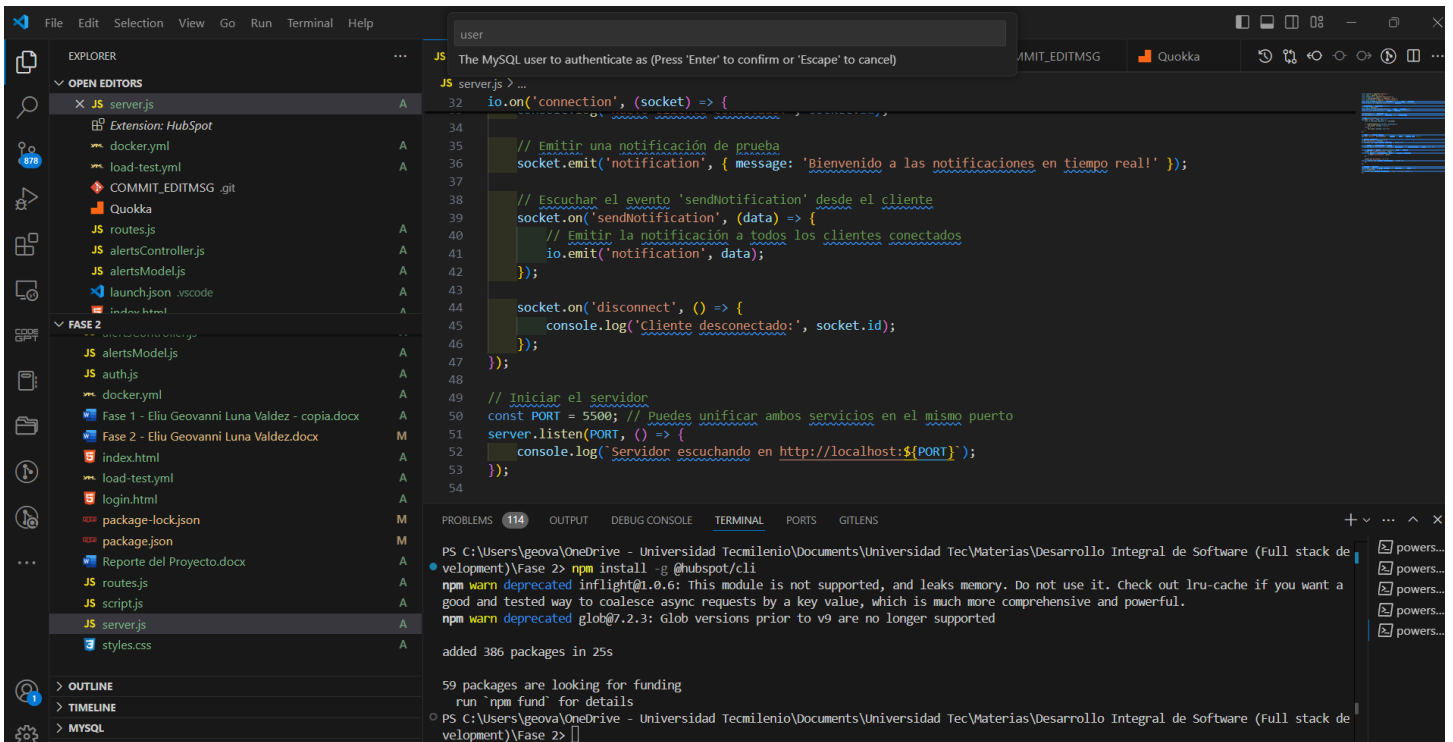
Se crea cuenta en <https://www.hubspot.com/>



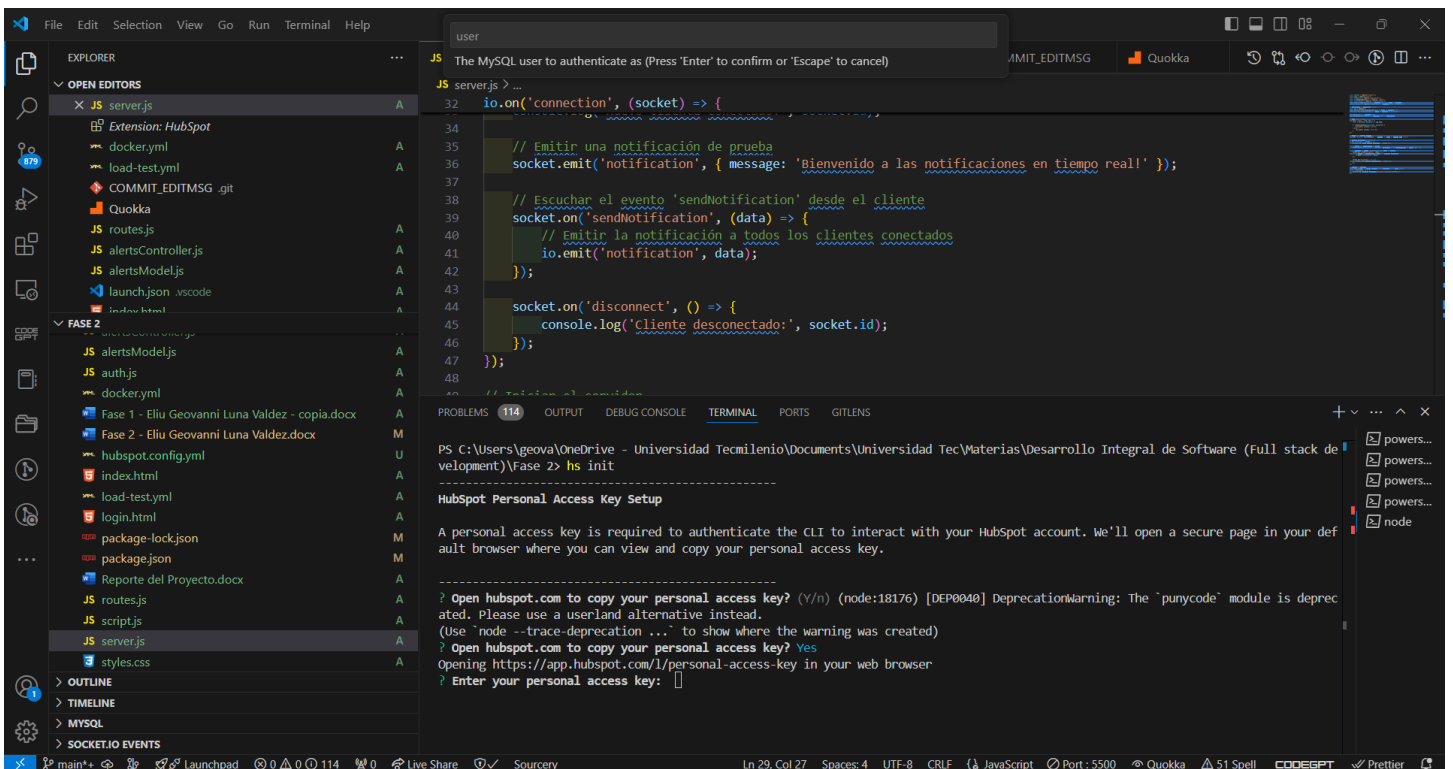
Se instala la extensión



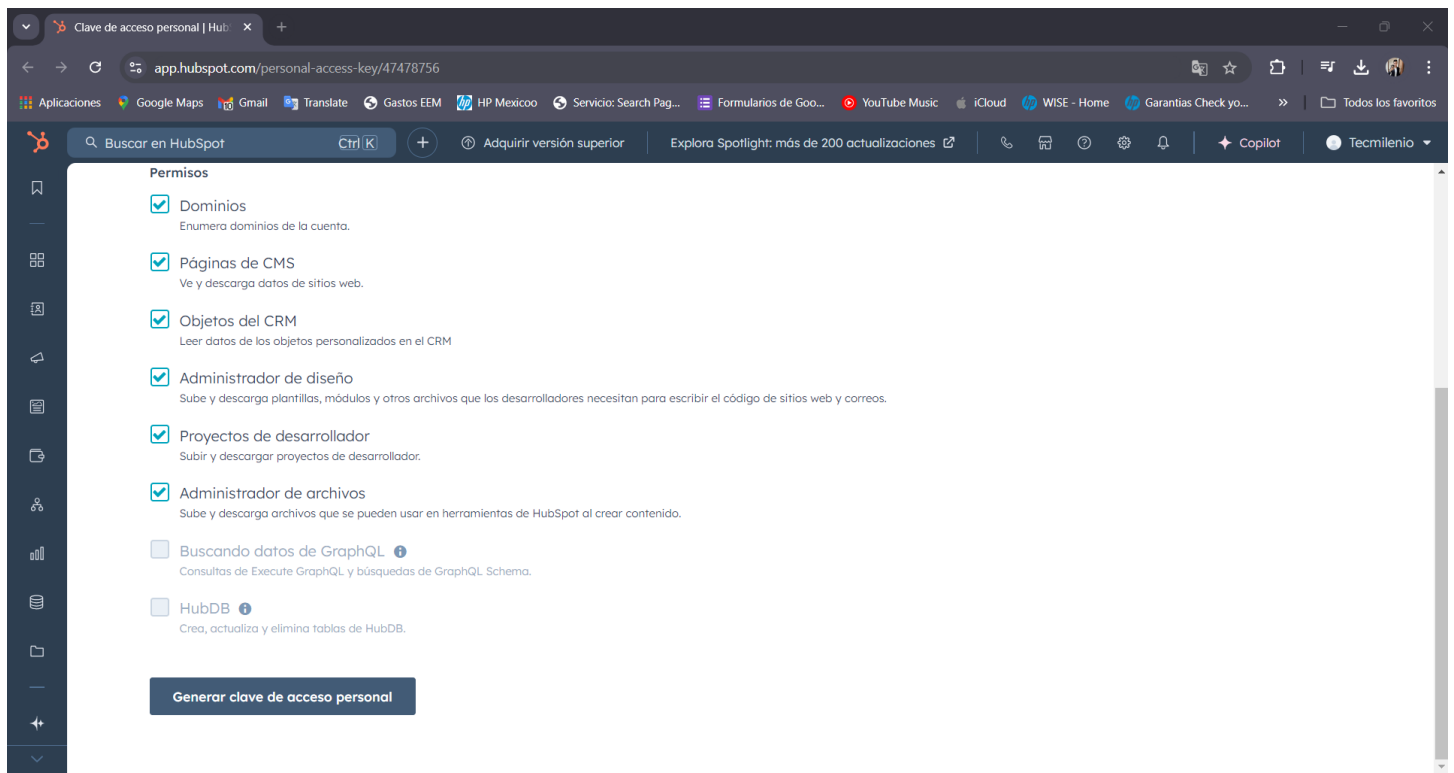
Se Instala en terminal



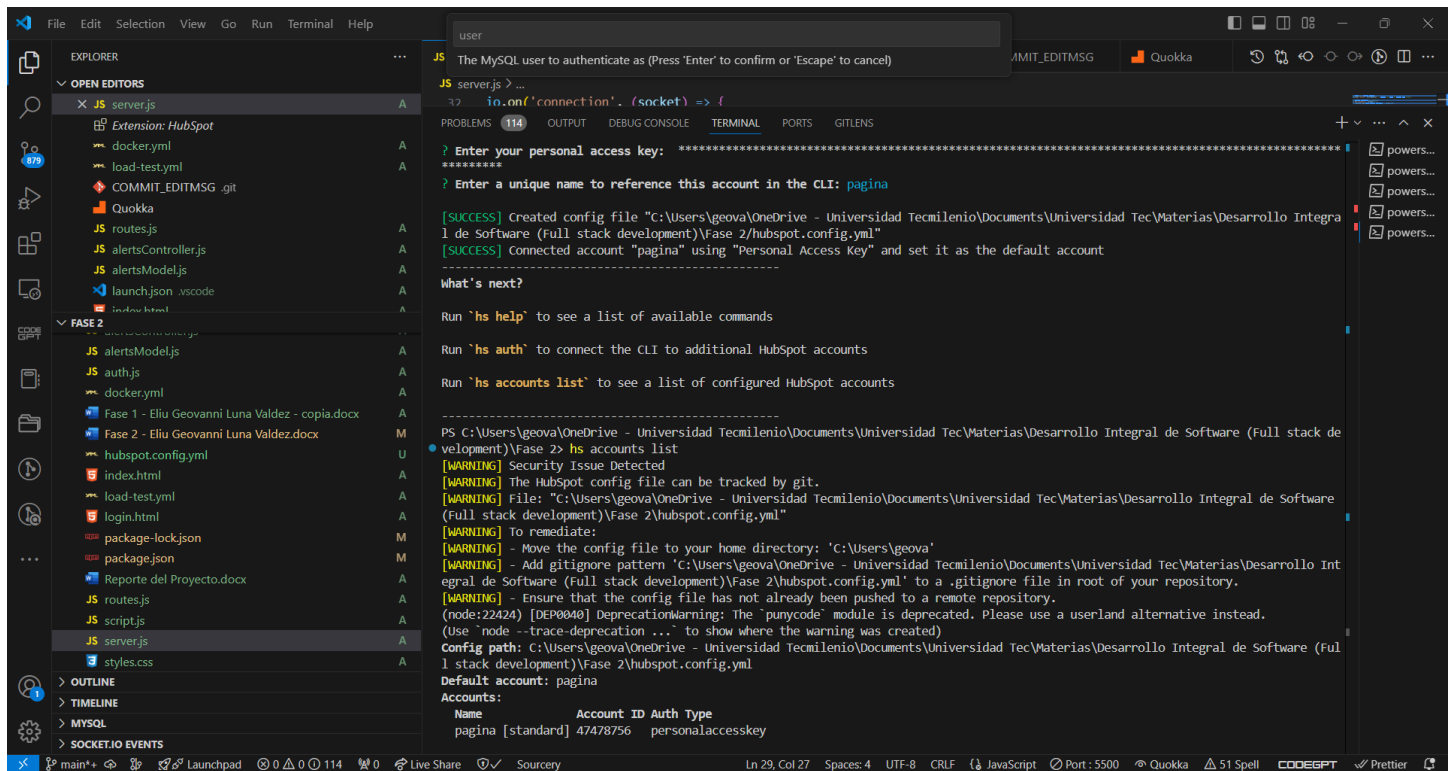
Se vincula y se da acceso



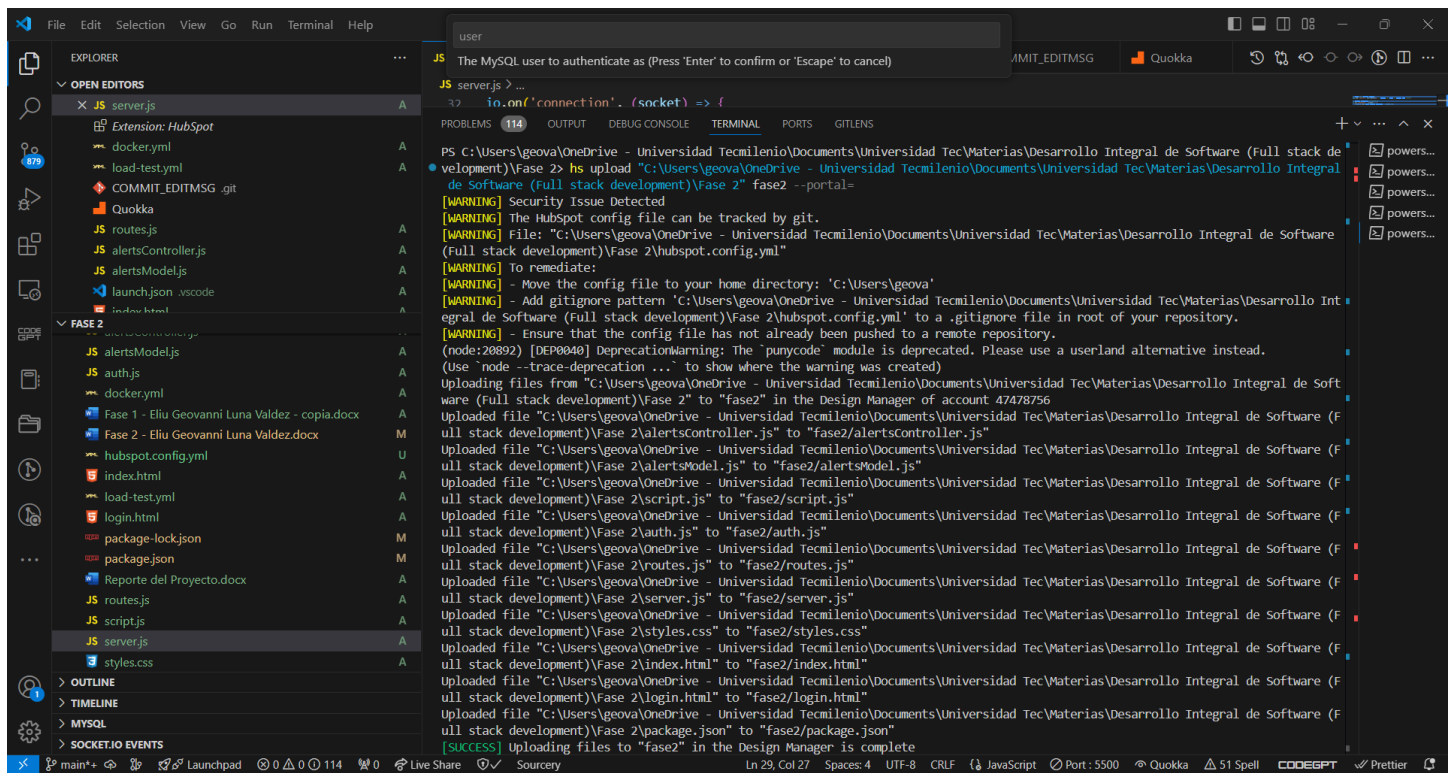
Se genera clave por medio de una API



Se inserta clave y se logra hacer el enlace, también se comprueba que VSC ve el repositorio y en este caso la PAGINA

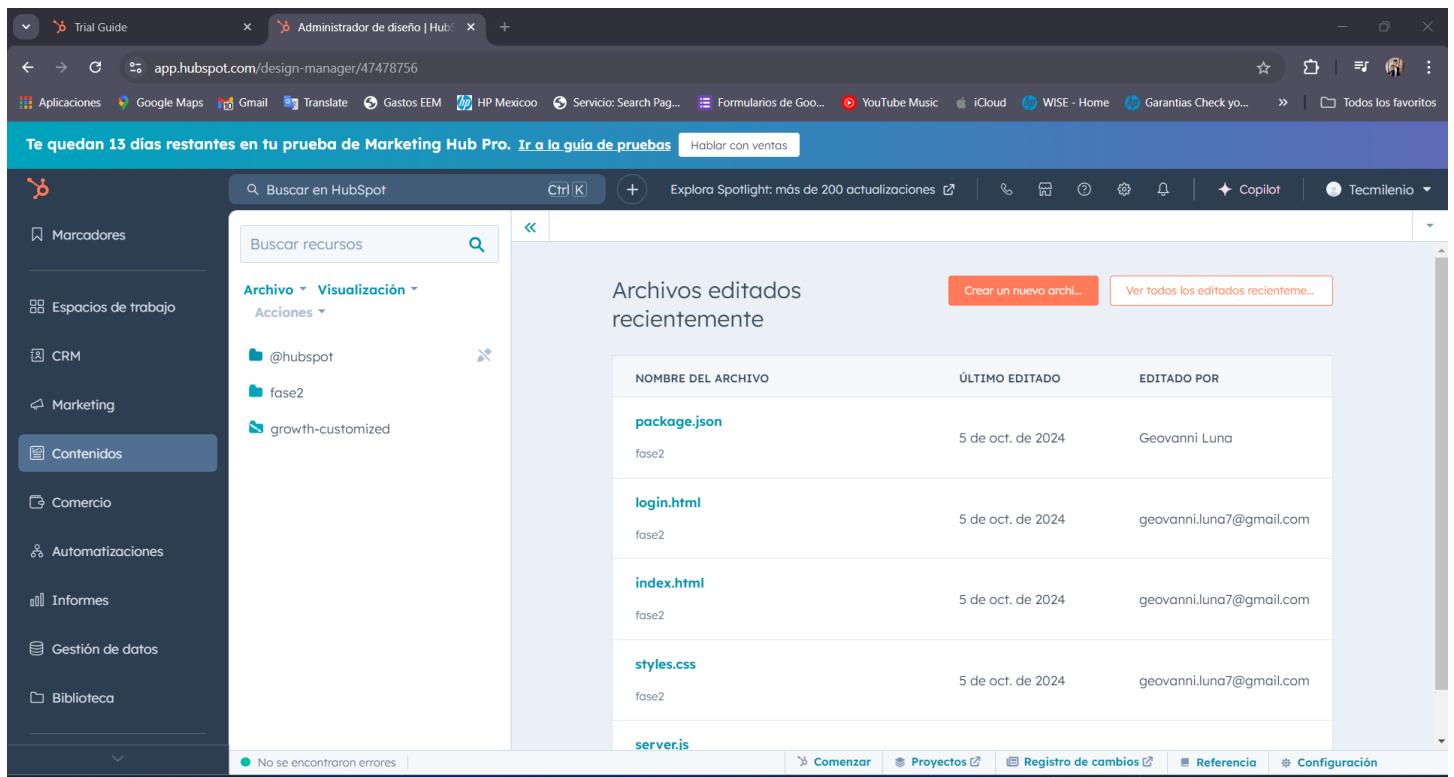


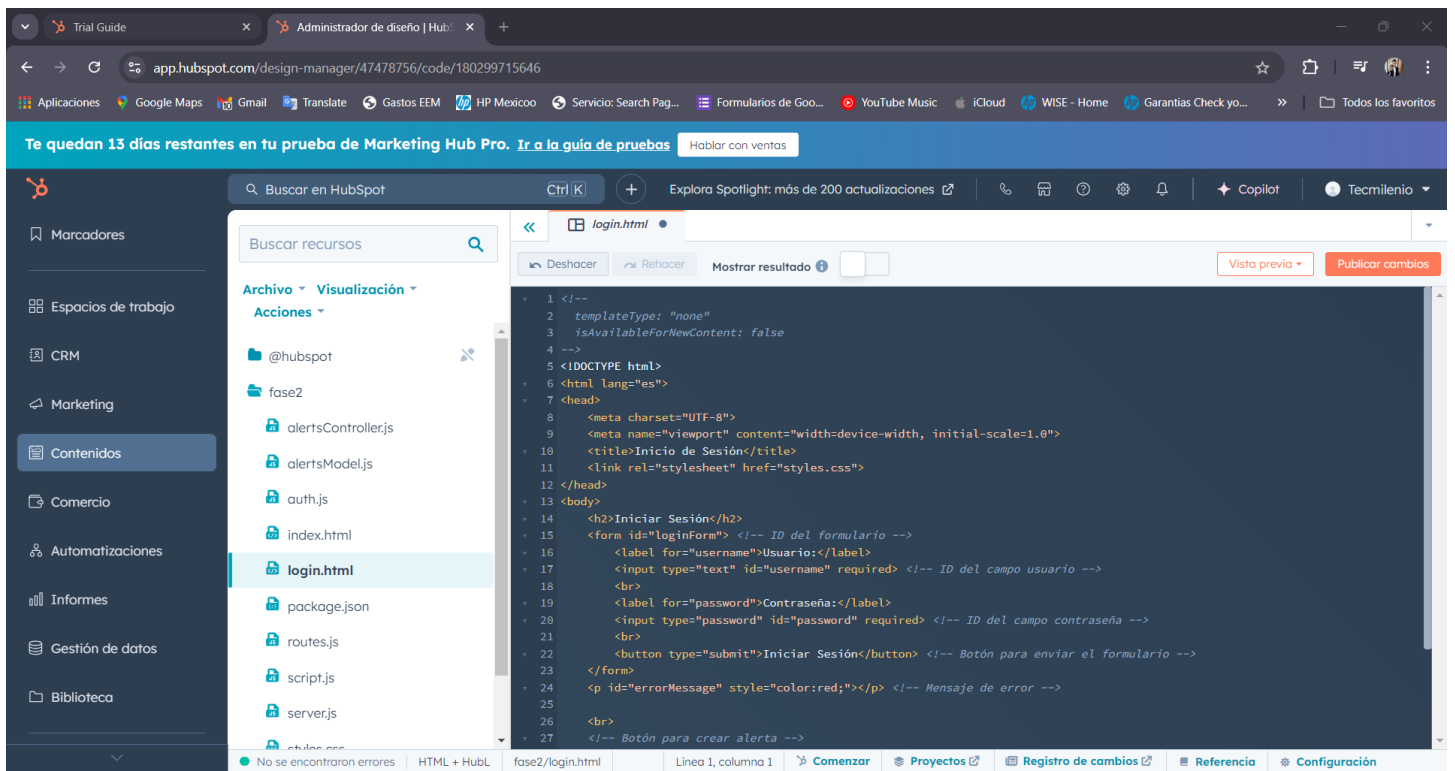
Se vincula y se migra proyecto indicando la carpeta contenedora



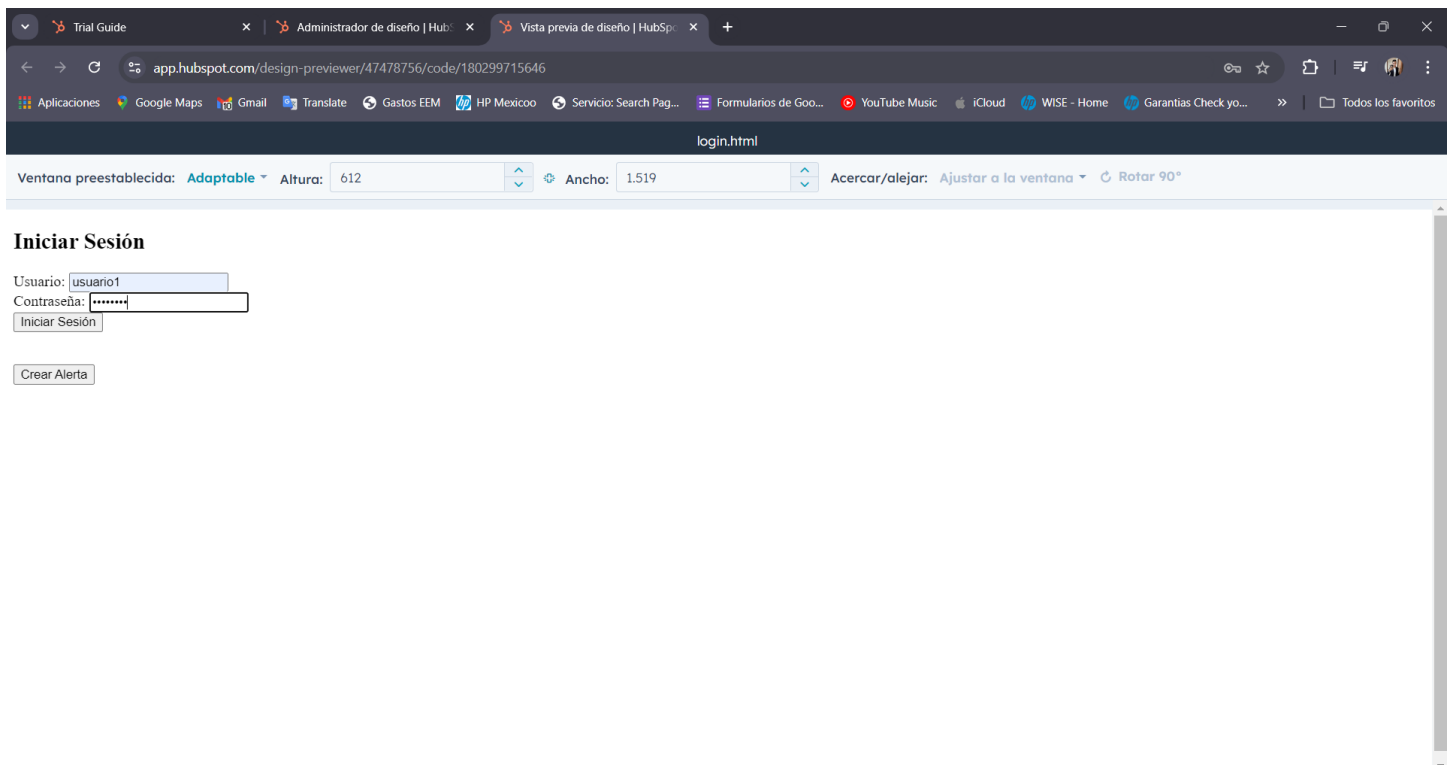
The screenshot shows a VS Code editor with a terminal window open. The terminal displays a command prompt where a user has entered a command to upload files to a remote repository. The command is: `PS C:\Users\geova\OneDrive - Universidad Tecmilenio\Documents\Universidad Tec\Material\Desarrollo Integral de Software (Full stack development)\Fase 2> hs upload "C:\Users\geova\OneDrive - Universidad Tecmilenio\Documents\Universidad Tec\Material\Desarrollo Integral de Software (Full stack development)\Fase 2" fase2 --portal=`. The terminal output shows a list of files being uploaded, including `package.json`, `login.html`, `index.html`, `styles.css`, and `server.js`. The file explorer on the left shows the project structure, including a `FASE 2` folder containing the files mentioned in the terminal output.

Se confirma en el directorio de hubspot.com





Se ejecuta aplicación en modo Web y app funciona correctamente



app.hubspot.com/design-previewer/47478756/code/180300201302

index.html

Ventana preestablecida: **Adaptable** Altura: 612 Ancho: 1.519 Acercar/alejar: Ajustar a la ventana Rotar 90°

Remitente

Nombre:
Ciudad:
Estado:
Domicilio:
Código Postal:
Teléfono:

Destinatario

Nombre:
Ciudad:
Estado:
Domicilio:
Código Postal:
Teléfono:

Mercancía

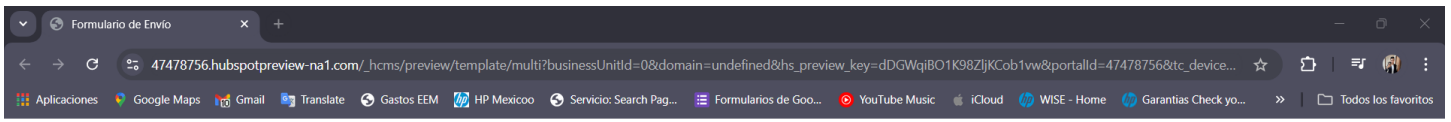
Descripción:
Peso:
Largo:
Ancho:
Alto:
Fecha de Salida (dd/mm/yyyy):
Fecha de Llegada (dd/mm/yyyy):

Inicio de Sesión

47478756.hubspotpreview-na1.com/_hcms/preview/template/multi?businessUnitId=0&domain=undefined&hs_preview_key=rQLFx5qavRqMog76WedZow&portalId=47478756&itc...

Iniciar Sesión

Usuario:
Contraseña:



Remitente

Nombre:
Ciudad:
Estado:
Domicilio:
Código Postal:
Teléfono:

Destinatario

Nombre:
Ciudad:
Estado:
Domicilio:
Código Postal:
Teléfono:

Mercancia

Descripción:
Peso:
Largo:
Ancho:
Alto:
Fecha de Salida (dd/mm/yyyy):
Fecha de Llegada (dd/mm/yyyy):
Precio de Envío:

Remitente Destinatario Mercancia

Se comparten links

https://47478756.hubspotpreview-na1.com/_hcms/preview/template/multi?businessUnitId=0&domain=undefined&hs_preview_key=rQLFx5qavRqMog76WedZow&portalId=47478756&tc_deviceCategory=undefined&template_file_path=fase2%2Flogin.html&updated=1728106514343

<https://app.hubspot.com/design-manager/47478756/code/180299715646>

Se procede a subir a GitHub

Trabajos consultados

www.hubspot.es . (2024) ¿Qué es HubSpot?
<https://www.hubspot.es/products>

Youtube (2020) Como subir un sitio web a un Hosting Gratuito [000WebHost]
<https://www.youtube.com/watch?v=0WO9g8rbLo8>

developer.mozilla.org– (2022) - Crea tu cuadrícula con CSS
https://developer.mozilla.org/es/docs/Learn/CSS/CSS_layout/Grids

platzi.com (2020) ¿Por qué CSS Grid es la mejor opción para crear diseños?
<https://platzi.com/tutoriales/1229-css-grid-layout-2017/2071-por-que-css-grid-es-mejor-que-bootstrap-para-crear-disenos/>

beatrizcalvo.com. (2024 mar) Tutorial sobre Draw.io – La mejor herramienta para diseñar diagramas online
<https://beatrizcalvo.com/tutorial-draw-io-herramienta-diagramas/>

YouTube - tecmilenio.mx (feb 2024) Creación de Web Page con base de datos MySQL (Parte 1)

<https://www.youtube.com/watch?v=FSAaa79bmYk>

YouTube - latincoder (febrero 2023) Tutorial Javascript basico pt 11

<https://www.youtube.cfm/watch?v=sLqrZC0Wyew>

YouTube - John Ortiz Ordoñez (nov 2021) JavaScript, Crear Tabla de Manera Dinámica Indicando el Número de Filas y Columnas.

<https://www.youtube.com/watch?v=aPEcj4YQdOI>