

Programación Web Avanzada

Diego Saavedra

Aug 23, 2024

Table of contents

1	Bienvenido	11
1.1	Descripción de la Asignatura:	11
1.2	Contribución de la Asignatura:	11
1.3	Resultado de Aprendizaje de la Carrera: (Unidad de Competencia)	11
1.4	Objetivo de la Asignatura: (Unidad de Competencia)	11
1.5	Resultado de Aprendizaje de la Asignatura: (Elemento de Competencia)	12
I	Unidad 1: Introducción al Desarrollo Web	13
2	Deploy desde Github	14
2.1	Crear un repositorio en Github	14
2.1.1	Clonar el repositorio	14
2.1.2	Crear un proyecto	15
2.2	Github Pages	17
2.2.1	Habilitar Github Pages	17
2.3	Vercel	23
2.3.1	Crear un proyecto en Vercel	23
3	Creación de Aplicaciones Web	28
4	HTML	29
4.1	¿Qué es HTML?	29
4.2	¿Cómo funciona HTML?	29
4.3	Ejemplo de HTML	29
4.4	Ventajas de HTML	30
4.5	Desventajas de HTML	30
4.6	Ejemplo de uso de HTML	30
4.7	Conclusión	31
5	CSS	32
5.1	¿Qué es CSS?	32
5.2	¿Cómo funciona CSS?	32
5.3	Ejemplo de CSS	32
5.4	Ventajas de CSS	33
5.5	Desventajas de CSS	33
5.6	Ejemplo de uso de CSS	34
5.7	Conclusión	35
6	JavaScript	36
6.1	¿Qué es JavaScript?	36
6.2	¿Cómo funciona JavaScript?	36

6.3	Ejemplo de JavaScript	36
6.4	Ventajas de JavaScript	37
6.5	Desventajas de JavaScript	37
6.6	Ejemplo de uso de JavaScript	37
6.7	Conclusión	38
7	Single Page Applications (SPAs)	39
7.1	¿Qué es una Single Page Application (SPA)?	39
7.2	¿Cómo funciona una SPA?	39
7.3	Ejemplo de SPA	39
7.4	Ventajas de SPAs	40
7.5	Desventajas de SPAs	40
7.6	Ejemplo de uso de SPAs con React	41
7.7	Conclusión	41
7.8	Ajax	42
7.9	¿Qué es Ajax?	42
7.10	¿Cómo funciona Ajax?	42
7.11	¿Por qué usar Ajax?	42
7.12	Ejemplo de uso de Ajax	42
7.13	Ventajas de Ajax	43
7.14	Desventajas de Ajax	43
7.15	Ejemplo de uso de Ajax con jQuery	44
7.16	Ejemplo de uso de Ajax con Fetch API	44
7.17	Ejemplo de uso de Ajax con XMLHttpRequest y Promesas	45
7.18	Ejemplo de uso de Ajax con Axios	46
7.19	Conclusión	47
8	JSON	48
8.1	¿Qué es JSON?	48
8.2	¿Cómo funciona JSON?	48
8.3	Ejemplo de JSON	48
8.4	Ventajas de JSON	49
8.5	Desventajas de JSON	49
8.6	Ejemplo de uso de JSON en JavaScript	49
8.7	Conclusión	50
9	REST	51
9.1	¿Qué es REST?	51
9.2	Principios de REST	51
9.3	Ejemplo de REST	51
9.4	Ventajas de REST	52
9.5	Desventajas de REST	52
9.6	Ejemplo de uso de REST en JavaScript	52
9.7	Ejemplo del uso de una API REST con Thunder Client	53
9.8	Conclusión	53
10	GraphQL	54
10.1	¿Qué es GraphQL?	54
10.2	Principios de GraphQL	54

10.3	Ejemplo de GraphQL	54
10.4	Ventajas de GraphQL	55
10.5	Desventajas de GraphQL	55
10.6	Ejemplo de uso de GraphQL en JavaScript	55
10.7	Ejemplo de uso de GraphQL en React	56
10.8	Conclusión	57
11	WebSockets	58
11.1	¿Qué son los WebSockets?	58
11.2	¿Cómo funcionan los WebSockets?	58
11.3	Ejemplo de uso de WebSockets	58
11.4	Ventajas de WebSockets	59
11.5	Desventajas de WebSockets	59
11.6	Ejemplo de uso de WebSockets con Socket.IO	60
11.7	Conclusión	60
12	Progressive Web Application (PWA)	62
12.1	¿Qué es una Progressive Web Application (PWA)?	62
12.2	Características de una PWA	62
12.3	Ventajas de una PWA	62
12.4	Desventajas de una PWA	63
12.5	Ejemplo de una PWA con React	63
12.6	Ejemplo de una PWA con Angular	63
12.7	Conclusión	64
13	Conclusiones	65
II	Laboratorios	66
14	Laboratorio Ajax	67
14.1	Requisitos	67
14.2	Instrucciones	67
15	Deploy	71
16	Reto	72
16.1	Conclusión	72
17	¿Dónde utilizamos Ajax?	73
18	Laboratorio de Railway	74
18.1	Objetivo	74
18.2	Instrucciones	74
18.3	Desarrollo	74
18.4	Conclusiones	80
19	Ejercicio	81
20	Recursos	83

21 Laboratorio Jinga 2	84
21.1 Objetivo	84
21.2 Requisitos	84
21.3 Desarrollo	84
22 Resultado	94
23 Ejercicios	95
24 Conclusiones	96
25 Referencias	97
26 Laboratorio de Ruby on Rails	98
26.1 ¿Qué es Ruby on Rails?	98
26.2 Instalación de Ruby on Rails	98
27 Creación de una nueva aplicación Rails	100
27.1 Parte 1: Configuración del Proyecto	100
27.1.1 Creación del Proyecto:	100
27.1.2 Navega a la Carpeta del Proyecto:	100
27.1.3 Inicia el Servidor:	100
27.2 Parte 2: Generación del Modelo	101
27.2.1 Conceptos Básicos	101
27.2.2 Generación del Modelo:	101
27.2.3 Ejecutar la Migración:	101
27.3 Parte 3: Generación del Controlador	101
27.3.1 Conceptos Básicos	101
27.3.2 Generación del Controlador:	101
27.4 Parte 4: Rutas	102
27.4.1 Conceptos Básicos	102
27.4.2 Configurar las Rutas:	102
27.5 Parte 5: Vistas	102
27.5.1 Conceptos Básicos	102
27.5.2 Crear las Vistas:	102
27.5.3 Controlador y Acción index:	103
27.6 Parte 6: CRUD - Create	103
27.6.1 Conceptos Básicos	103
27.6.2 Formulario para Crear un Nuevo Ítem:	103
27.6.3 Acción new y create en el Controlador:	103
27.7 Parte 7: CRUD - Read	104
27.7.1 Conceptos Básicos	104
27.7.2 Vista show:	104
27.8 Parte 8: CRUD - Update	105
27.8.1 Conceptos Básicos	105
27.8.2 Formulario para Editar un Ítem:	105
27.8.3 Acción edit y update en el Controlador:	105
27.8.4 Parte 9: CRUD - Delete	106
27.8.5 Conceptos Básicos	106

27.8.6	Acción destroy en el Controlador:	106
27.9	Parte 10: Pruebas y Verificación	108
27.9.1	Conceptos Básicos	108
27.9.2	Verificar la Funcionalidad:	108
27.10	Pruebas Manuales:	108
27.11	Parte 11: Estilo y Mejoras	109
27.11.1	Conceptos Básicos	109
27.11.2	Agregar Estilo:	109
28	Actividad	111
29	Conclusión	115
30	Laboratorio Firebase Cloud Functions REST API CRUD	116
30.1	Descripción	116
30.2	Requisitos	116
30.3	Desarrollo	116
30.3.1	Paso 1: Crear un proyecto en Firebase	116
30.3.2	Paso 2: Inicializar un proyecto de Node.js con Firebase	119
30.3.3	Crear nuestro CRUD en nuestro proyecto de Firebase	126
30.3.4	Paso 3: Autenticar nuestra función con Firebase	128
30.3.5	Paso 4: Crear un documento en Firestore	132
30.3.6	Paso 5: Desplegar nuestra aplicación en Firebase	139
30.4	Conclusiones	140
30.5	Referencias	140
31	Laboratorio de Simple Object Access Protocol (SOAP)	141
31.1	Objetivo	141
31.2	Materiales	141
31.3	Conceptos básicos.	141
31.3.1	SOAP	141
31.3.2	WSDL	141
31.3.3	Spring Boot	142
31.3.4	Maven	142
31.3.5	JAXB	142
31.3.6	Marshalling	142
31.3.7	Unmarshalling	142
31.4	Introducción	142
31.5	Instrucciones	144
31.5.1	Crear un cliente SOAP	145
31.5.2	¿Qué es Unmarchalling y Marchalling?	154
31.5.3	Ejecutar la aplicación	156
31.6	Crear el Controlador	158
31.7	Ejecutar la aplicación	160
31.7.1	Probar el endpoint de suma	161
31.7.2	Probar el endpoint de resta	161
31.8	Reto	161
31.9	Referencias	163

32 Laboratorio de API Rest con Spring Boot.	164
32.1 Objetivo	164
32.2 Requisitos	164
32.3 Dependencias	164
32.4 Conceptos a aprender	164
32.4.1 API Rest	164
32.4.2 API Restful	164
32.4.3 Rest	165
32.5 Crear un proyecto Spring Boot	165
32.6 Configurar la conexión a la base de datos	166
33 Configuración de la Base de Datos	167
33.0.1 Crear un contenedor de Docker con MySQL	167
33.1 Crear Entidades	169
33.2 Probar el Servidor	171
33.3 Creación del API Rest	174
33.4 Probar el API Rest	187
34 Reto	190
35 Conclusiones	191
36 Referencias	192
37 Laboratorio de Dev Containers	193
37.1 Objetivo	193
37.2 Requerimientos	193
37.3 Procedimiento	193
37.3.1 Instalar la extensión de Docker Containers	194
37.3.2 Crear un Docker Container	194
37.3.3 Probar el Dev Contanier	198
37.4 Reto	200
37.5 Conclusiones	201
38 Laboratorio: Implementación de POO en TypeScript	202
38.1 Objetivo	202
38.2 Paso 1: Instalación de TypeScript	202
38.3 Estructura del Proyecto:	202
38.4 Paso 2: Definición de Clases y Encapsulamiento	203
38.5 Paso 3: Herencia	204
38.6 Paso 4: Polimorfismo	204
38.7 Paso 5: Implementación en el Archivo Principal	206
38.8 Paso 6: Compilar y Ejecutar	207
39 Conclusiones	208
40 Reto	209

41 Laboratorio: Creación de una Plataforma de Gestión de Cursos con Next.js y TypeScript	210
41.1 Objetivo	210
41.2 Paso 1: Crear la Aplicación Next.js	210
41.3 Paso 2: Crear la Estructura del Proyecto	210
41.4 Paso 3: Modelo (models/course.ts)	211
41.5 Paso 4: Vista (views/courseView.tsx)	211
41.6 Paso 5: Controlador (controllers/courseController.ts)	212
41.7 Paso 6: Integración en Next.js (pages/courses.tsx)	213
41.8 Paso 7: Página de Inicio (pages/index.tsx)	213
42 Extra	215
42.1 Paso 8: Estilos CSS	215
43 Conclusión	217
44 Reto	218
45 Laboratorio: Creación de un CMS Monolítico con FastAPI	219
45.1 Objetivo:	219
45.2 Requisitos Previos:	219
45.3 Pasos del Laboratorio:	219
45.4 Configurar la Estructura del Proyecto:	220
45.5 Crear los Endpoints de la API:	221
45.6 Pruebas:	223
45.7 Pruebas	224
45.7.1 Crear una Solicitud en Thunder Client	224
45.7.2 Configuración de la Solicitud	225
45.7.3 Cuerpo:	225
45.7.4 Revisar la Respuesta	225
45.8 Reto	226
46 Tutorial para la Creación de Microservicios con Docker y Docker Compose	227
46.1 Requisitos Previos	228
46.2 Estructura del Proyecto	228
46.3 Paso 1: Crear el Microservicio de Autenticación	229
46.4 Paso 2: Crear el Microservicio de Búsqueda	229
46.5 Paso 3: Microservicio de Comentarios	229
46.6 Paso 4: Crear el Microservicio de Gestión de Post	230
46.7 Paso 5: Crear el Archivo docker-compose.yml General	230
46.8 Paso 6: Levantar los Microservicios	231
46.9 Reto	232
46.10 Recursos	232
46.11 Conclusión	233
47 Laboratorio de Implementación de un Sistema de Búsqueda e Indexación de Datos parte 1	234
47.1 Objetivo:	234
47.2 Descripción:	234

47.3	Conceptos Cubiertos:	235
47.4	Paso 1: Configuración del Entorno con Docker	235
47.4.1	1.1. Crear el archivo Dockerfile para Node.js/NestJS :	235
47.4.2	1.2. Crear el archivo docker-compose.yml:	235
47.5	Paso 2: Crear la Aplicación NestJS	237
47.5.1	2.1. Crear una nueva aplicación NestJS	237
47.5.2	2.2. Instalar Dependencias de Elasticsearch	237
47.6	Paso 3: Implementar el Motor de Búsqueda	237
47.6.1	3.1. Configurar el Módulo de Elasticsearch	237
47.7	3.2. Crear el Servicio de Elasticsearch	237
47.7.1	3.3. Crear un Controlador para Manejar las Búsquedas	238
47.8	Paso 4: Probar la Aplicación	239
47.8.1	4.1 Levanta los servicios con Docker Compose:	239
47.9	Paso 5: Configurar el Índice de Elasticsearch	239
47.9.1	5.1. Crear el Índice	239
47.10	Paso 6: Probar la Aplicación	239
47.10.1	6.1. Verificar el Estado de los Contenedores	239
47.10.2	6.2. Probar Elasticsearch	240
47.10.3	6.3. Probar la Aplicación NestJS	241
47.11	Revisar los Logs de los Contenedores	242
47.12	Pruebas Adicionales	242
47.13	Interfaz de Usuario:	243
48	Reto	244
49	Conclusión	245
50	Laboratorio: Análisis de Texto con Python - Detección de Bigramas y Colocaciones.	246
50.1	Objetivo	246
50.2	Requisitos Previos	246
50.3	Instalación de Bibliotecas Necesarias	247
50.4	Desarrollo del Laboratorio	247
51	Conceptos Básicos	248
51.1	Tokenización	248
51.2	Bigrams	248
51.3	Colocaciones	248
52	Desarrollo del Laboratorio	249
52.1	Paso 0: Creamos el Notebook	249
52.2	Paso 1: Importar Bibliotecas	249
52.3	Paso 2: Cargar y Preparar el Corpus	249
52.4	Paso 3: Tokenización	250
52.5	Paso 4: Detección de Bigramas con NLTK	250
52.6	Paso 5: Detección de Colocaciones con Gensim	250
52.7	Paso 6: Análisis de Colocaciones	251
53	Guardar el modelo de bigramas	252

54 Implementación de una aplicación web	253
54.1 Paso 0: Creación de la aplicación web:	253
54.2 Paso 1: Crear una Interfaz de Usuario	253
54.3 Paso 2: Procesar el Texto Ingresado	253
54.4 Paso 3: Mostrar las Colocaciones Detectadas	254
54.5 Paso 4: Ejecutar la Aplicación	254
55 Probar la aplicación con estos ejemplos	255
56 Reto	256
57 Recursos	257
58 Conclusiones	258
59 Laboratorio: Introducción a los Sistemas de Recomendación	259
59.1 Objetivo	259
59.2 Conceptos clave	259
59.3 Requisitos previos	260
59.3.1 Creación de un entorno virtual	260
59.3.2 Instalación de Redis	261
59.3.3 Instalación de Redis	261
59.4 Introducción	261
59.5 Paso 0: Instalación de Redis con docker compose.	261
59.6 Paso 1: Obtener datos de productos de la API de Fake Store	262
59.7 Paso 2: Procesar datos de productos y construir un sistema de recomendación	262
59.7.1 Ejecución del código	263
59.8 Paso 3: Mejorar el sistema de recomendación	264
59.9 Aplicación de mejoras	264
59.9.1 Ejecución del código	265
60 Implementación de recomendaciones en un sistema web	266
60.0.1 Ejecución de la aplicación web	267
61 Probar la aplicación con Thunder Client	268
61.0.1 Ejemplo	268
61.1 Reto	268
61.2 Agregar .gitignore	269
61.3 Agregar README.md	272
61.4 Inicializamos el repositorio git	273
61.5 Recursos	274
61.6 Conclusión	274

1 Bienvenido

¡Bienvenido a la Asignatura de Desarrollo Web Avanzado!

1.1 Descripción de la Asignatura:

El desarrollo web es el proceso de construir aplicaciones web y sitios web para hospedarlos en el internet o una intranet, su desarrollo requiere el conocimiento de lenguajes como HTML, CSS, JavaScript, Java, SQL, entre otras que permitan construir aplicaciones web dinámicas, modulares, seguras, con el uso de frameworks de front-end y back-end.

1.2 Contribución de la Asignatura:

La asignatura contribuye al resultado de aprendizaje del nivel y es parte sustancial de la formación profesional, los componentes son la solución a problemas orientados a la integración de diferentes aplicaciones e infraestructura tecnológica existente en las organizaciones, bajo el sustento de la programación web. Adicionalmente, al considerarse como cátedra integradora aporta y se nutre de las asignaturas de praxis profesional del nivel: Ingeniería de Usabilidad, Programación Web, Sistemas de Bases de datos

1.3 Resultado de Aprendizaje de la Carrera: (Unidad de Competencia)

Formar profesionales en Ingeniería de Software capaces de desarrollar sistemas informáticos mediante el uso de metodologías, herramientas y estándares, demostrando creatividad, eficiencia, eficacia y responsabilidad profesional; con el propósito de optimizar procesos, generar fuentes de empleo y contribuir en la mejora de la economía y competitividad de los sectores productivos del País.

1.4 Objetivo de la Asignatura: (Unidad de Competencia)

Diseñar y desarrollar aplicaciones Web modernas, integrando componentes UI con servicios web básicos utilizando bibliotecas y frameworks para Front-End y Back-End

1.5 Resultado de Aprendizaje de la Asignatura: (Elemento de Competencia)

Relaciona y demuestra el dominio en el desarrollo de páginas y aplicaciones web mediante la presentación de proyectos que expongan los conocimientos adquiridos a lo largo del semestre. Comprender la ciencia detrás de los sistemas de información basados en la Web y los principios de ingeniería de los sistemas de Información Web.

Investiga, modela, diseña y desarrolla páginas y aplicaciones móviles web que unifique el contenido presentado en la materia. Efectúa análisis sobre los componentes software necesarios en proyectos de desarrollo de sistemas Web para con ello lograr la implementación recuperación de la información web y extracción de la información aplicado en la web para análisis de información.

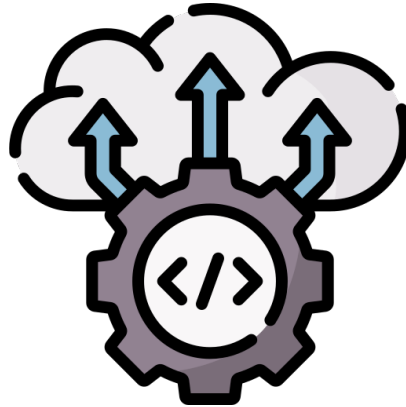
Muestra participación activa como parte de trabajo colaborativo en el desarrollo de casos de estudio.

Integrar los conceptos utilizados en el diseño, desarrollo y el despliegue de aplicaciones basadas en la Web mostrando participación activa como parte de trabajo colaborativo en diferentes casos de estudios.

Part I

Unidad 1: Introducción al Desarrollo Web

2 Deploy desde Github



En este tutorial usted aprenderá a realizar un deploy de una aplicación desde el repositorio de Github hacia un servicio como Github Pages, Vercel, Netlify, etc.

2.1 Crear un repositorio en Github

1. Ingresar a Github y crear un nuevo repositorio.
2. Darle un nombre al repositorio y seleccionar si será público o privado.
3. Crear el repositorio.

2.1.1 Clonar el repositorio

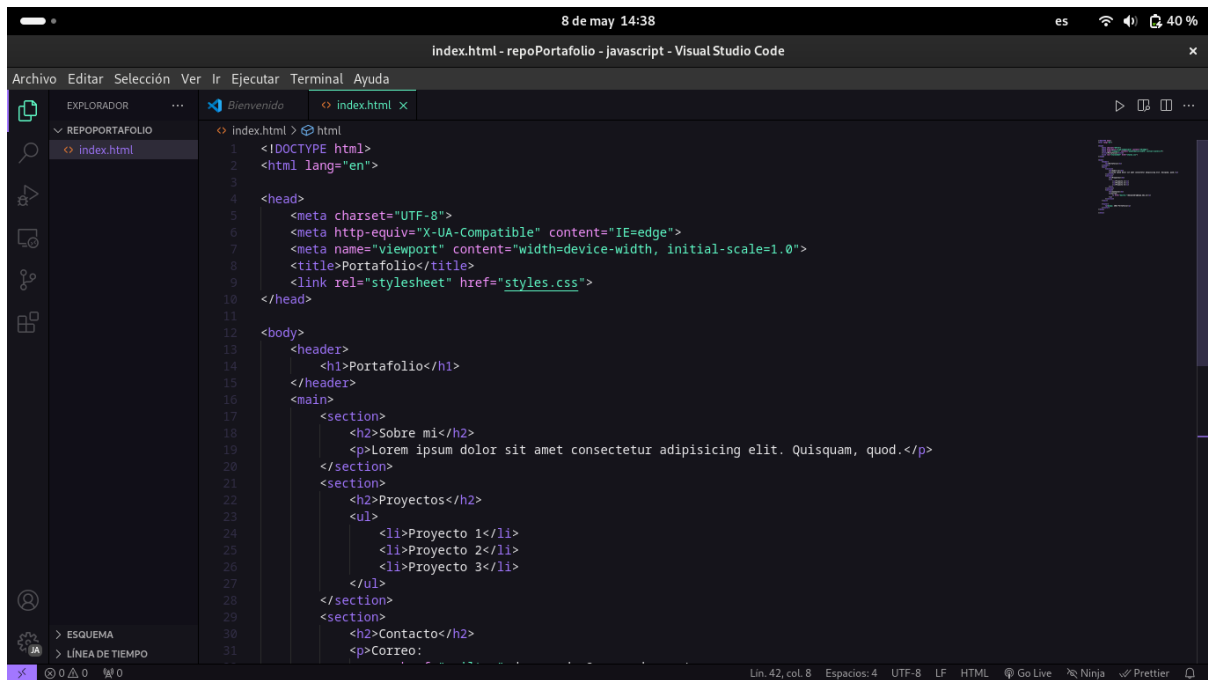
1. Copiar la URL del repositorio.
2. Abrir la terminal y ejecutar el siguiente comando:

```
git clone <URL>
```

2.1.2 Crear un proyecto

En esta práctica vamos a utilizar tecnologías Html5, Css3 y JavaScript para crear un Portafolio Personal.

1. Crear un archivo **index.html** y agregar el siguiente código:



The screenshot shows the Visual Studio Code editor interface. The title bar indicates the file is 'index.html - repoPortafolio - javascript - VisualStudio Code'. The Explorer sidebar on the left shows a folder named 'REPOPORTAFOLIO' containing 'index.html'. The main editor area displays the following HTML code:

```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta http-equiv="X-UA-Compatible" content="IE=edge">
7   <meta name="viewport" content="width=device-width, initial-scale=1.0">
8   <title>Portafolio</title>
9   <link rel="stylesheet" href="styles.css">
10 </head>
11
12 <body>
13   <header>
14     <h1>Portafolio</h1>
15   </header>
16   <main>
17     <section>
18       <h2>Sobre mi</h2>
19       <p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Quisquam, quod.</p>
20     </section>
21     <section>
22       <h2>Proyectos</h2>
23       <ul>
24         <li>Proyecto 1</li>
25         <li>Proyecto 2</li>
26         <li>Proyecto 3</li>
27       </ul>
28     </section>
29     <section>
30       <h2>Contacto</h2>
31       <p>Correo:
```

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Portafolio</title>
  <link rel="stylesheet" href="styles.css">
</head>

<body>
  <header>
    <h1>Portafolio</h1>
  </header>
  <main>
    <section>
      <h2>Sobre mi</h2>
      <p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Quisquam, quod.</p>
    </section>
```

```

    <section>
      <h2>Proyectos</h2>
      <ul>
        <li>Proyecto 1</li>
        <li>Proyecto 2</li>
        <li>Proyecto 3</li>
      </ul>
    </section>
    <section>
      <h2>Contacto</h2>
      <p>Correo:
        <a href="mailto:">dmsaavedra@espe.edu.ec</a>
      </p>
    </section>
  </main>

  <footer>
    <p>&copy; 2024 Portafolio</p>
  </footer>
</body>

</html>

```

2. Crear un archivo **styles.css** y agregar el siguiente código:

```

body {
  font-family: Arial, sans-serif;
  margin: 0;
  padding: 0;
}

header {
  background-color: #333;
  color: white;
  text-align: center;
  padding: 1rem;
}

main {
  padding: 1rem;
}

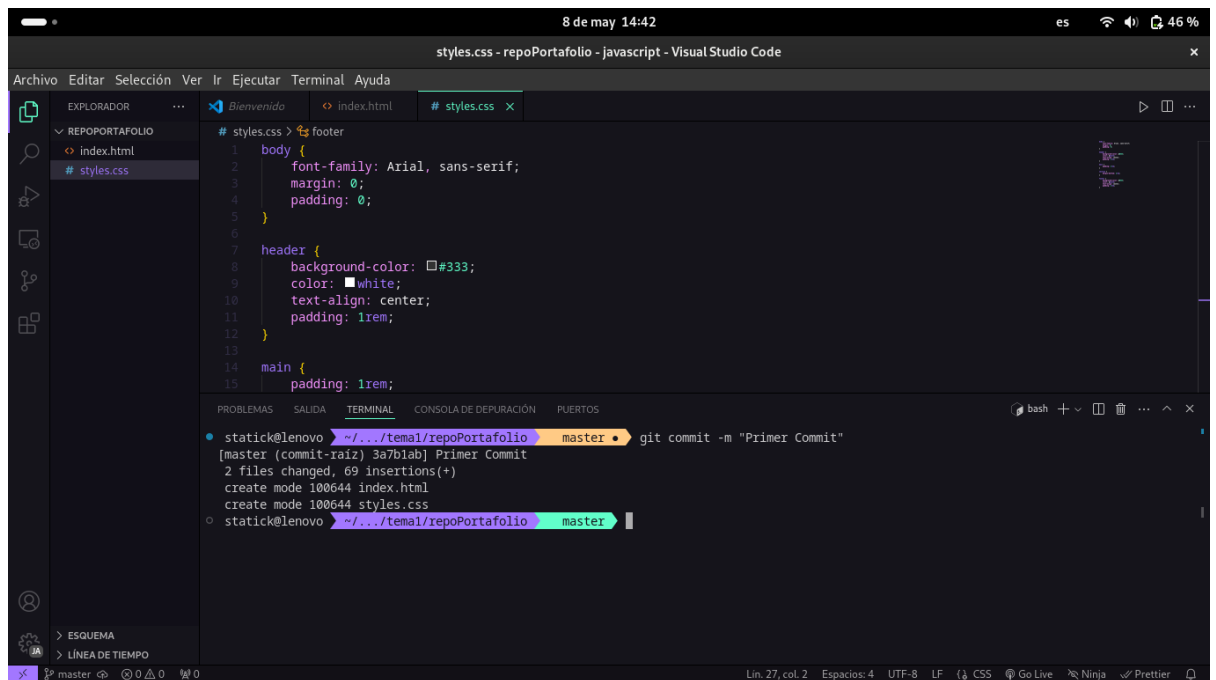
section {
  margin-bottom: 1rem;
}

footer {

```

```
background-color: #333;  
color: white;  
text-align: center;  
padding: 1rem;  
}
```

3. Agregar los archivos al repositorio:



The screenshot shows the Visual Studio Code interface. The main editor displays the content of 'styles.css' with the following CSS code:

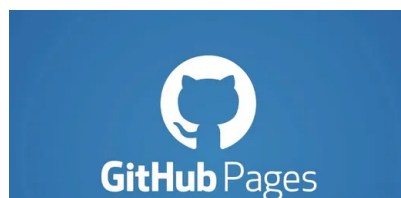
```
# styles.css > footer  
1 body {  
2   font-family: Arial, sans-serif;  
3   margin: 0;  
4   padding: 0;  
5 }  
6  
7 header {  
8   background-color: #333;  
9   color: white;  
10  text-align: center;  
11  padding: 1rem;  
12 }  
13  
14 main {  
15  padding: 1rem;  
16 }
```

The terminal window at the bottom shows the execution of the following commands:

```
static@lenovo ~/.../temal/repoPortafolio master  
● static@lenovo ~/.../temal/repoPortafolio master → git commit -m "Primer Commit"  
[master (commit-raiz) 3a7b1ab] Primer Commit  
2 files changed, 69 insertions(+)  
create mode 100644 index.html  
create mode 100644 styles.css  
○ static@lenovo ~/.../temal/repoPortafolio master
```

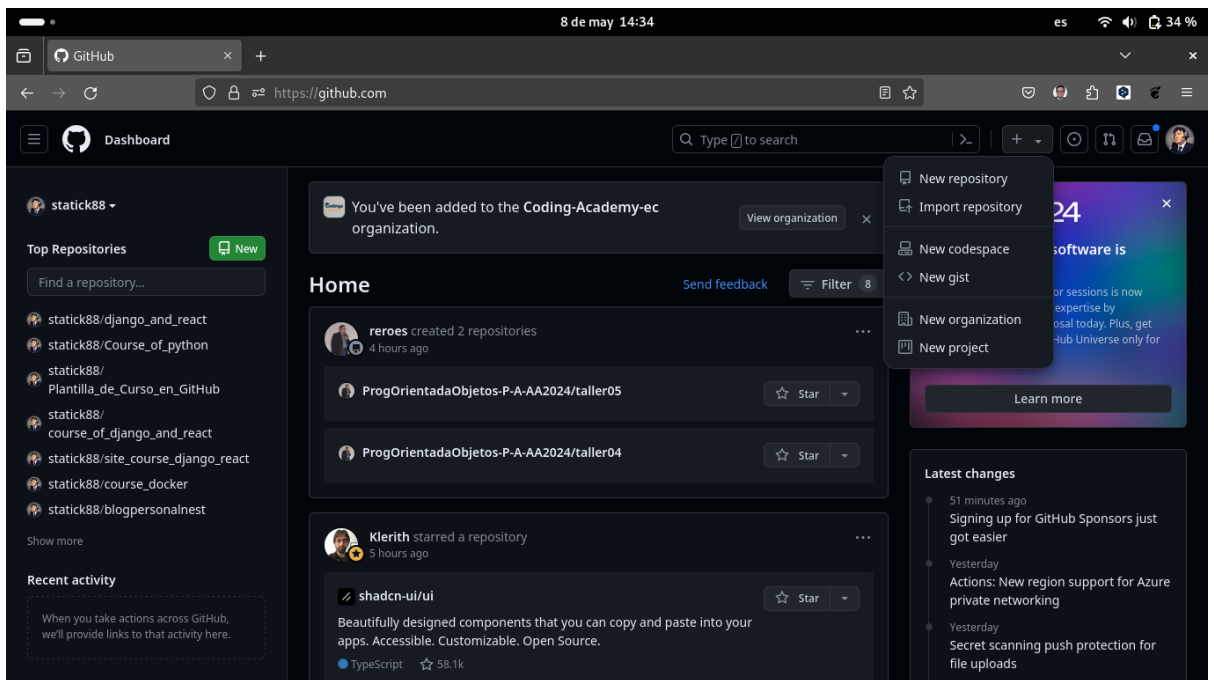
```
git init  
git add .  
git commit -m "Primer commit"
```

2.2 Github Pages

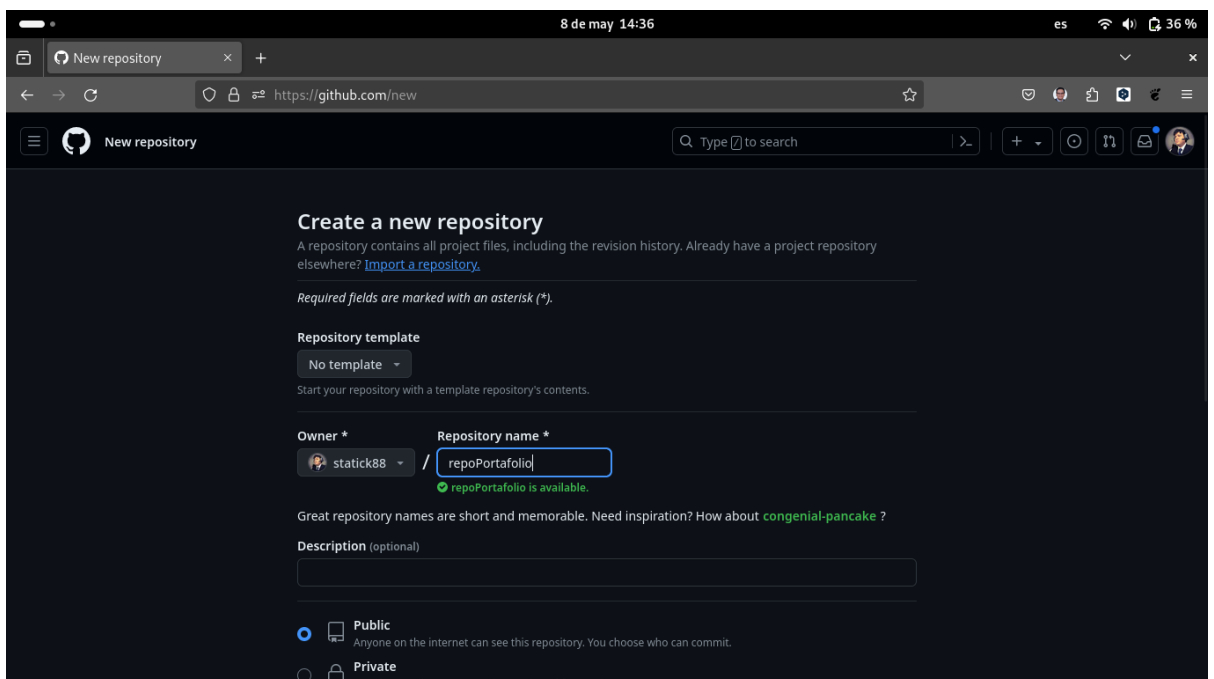


2.2.1 Habilitar Github Pages

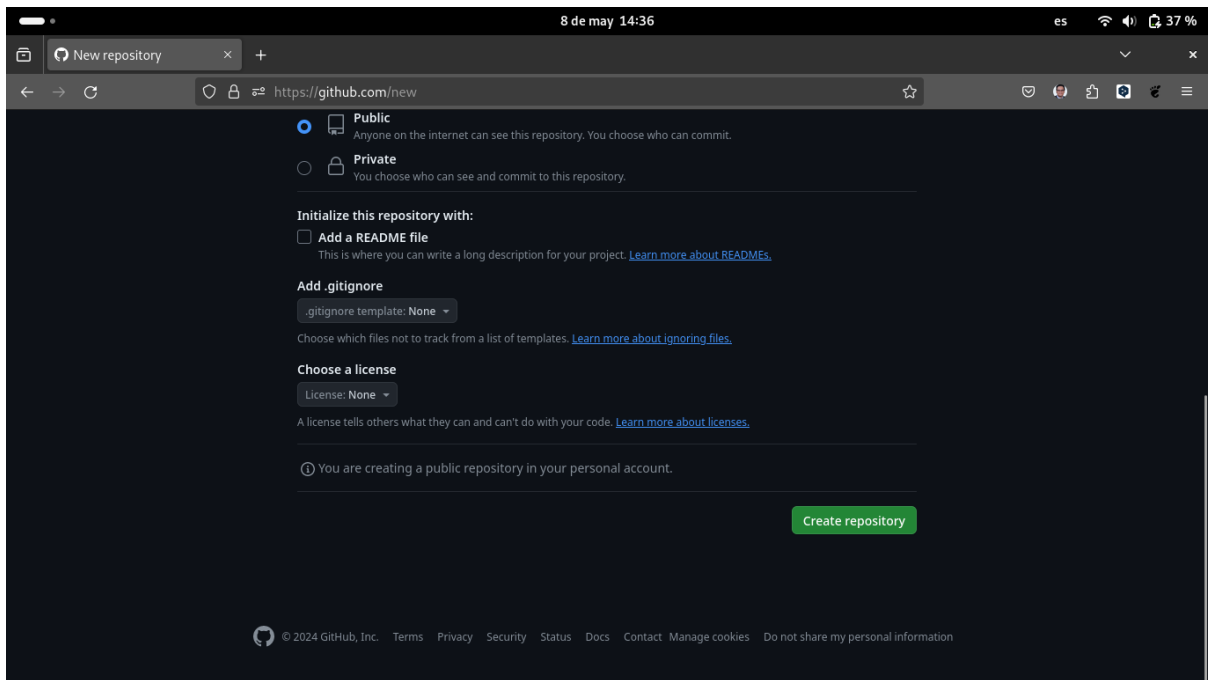
1. Primero es necesario que creamos el repositorio.



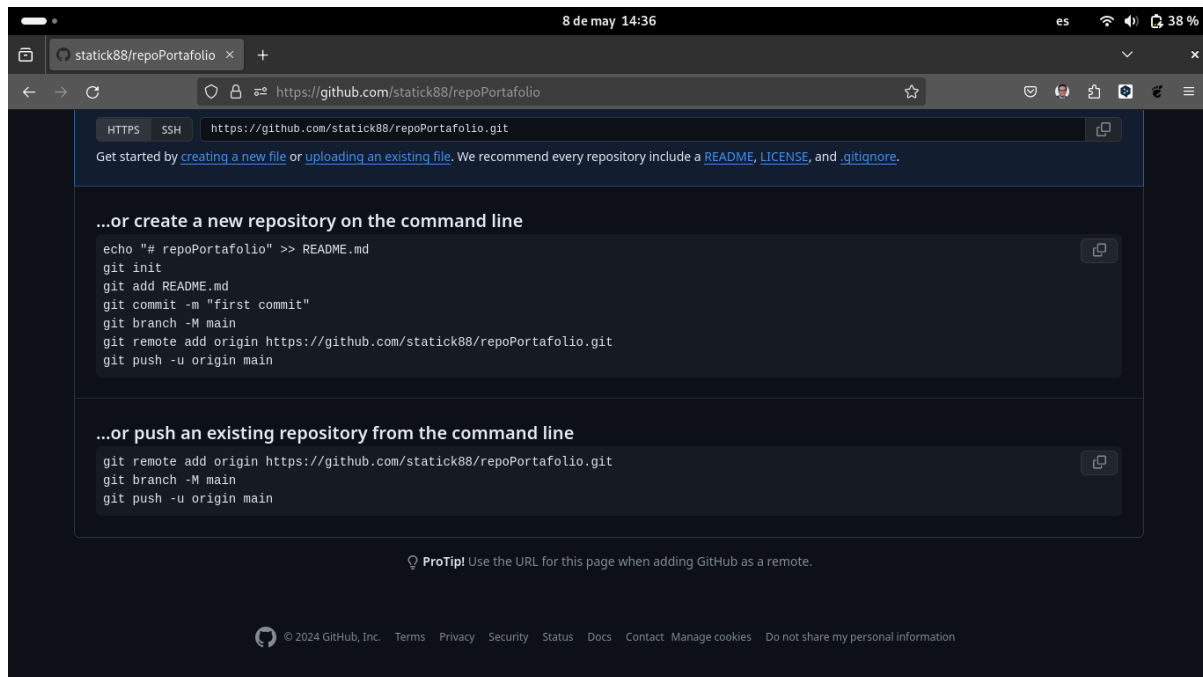
2. Asignamos un nombre.



3. Damos clic al botón **Create Repositorio**.



4. Vinculamos el repositorio local con el repositorio en línea.



5. Ahora vinculamos nuestro repositorio local con el repositorio en línea.

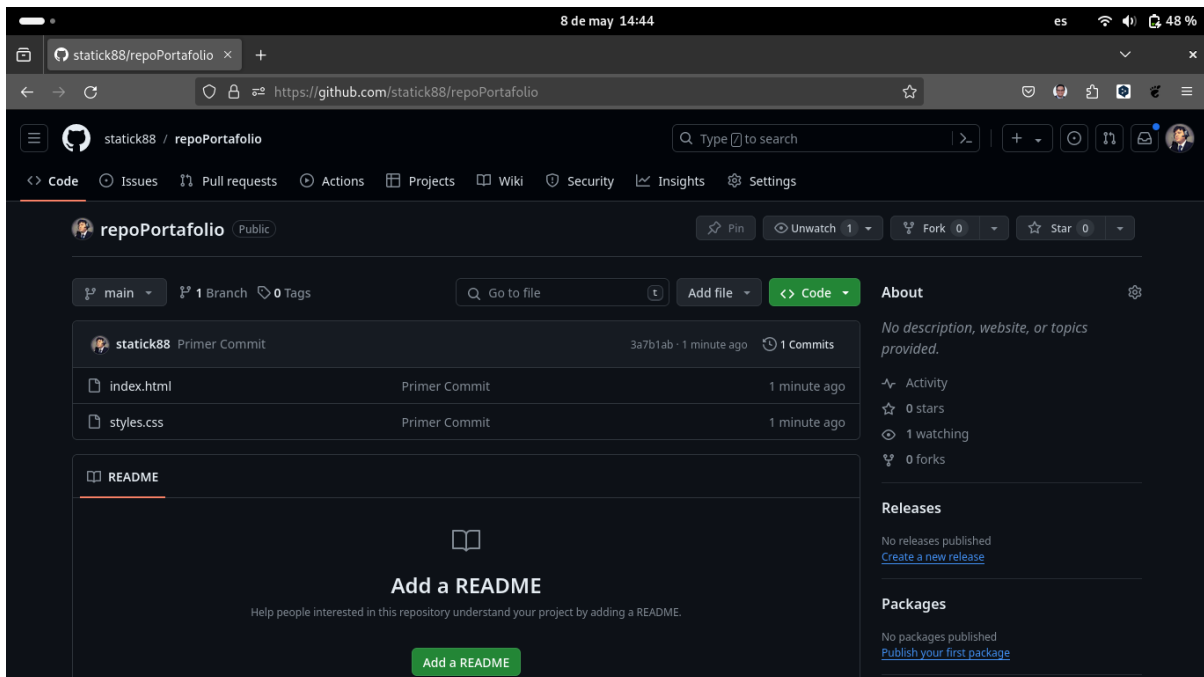
The screenshot shows the Visual Studio Code interface. The editor displays a CSS file named `styles.css` with the following content:

```
# styles.css > footer
1 body {
2   font-family: Arial, sans-serif;
3   margin: 0;
4   padding: 0;
5 }
6
7 header {
8   background-color: #333;
9   color: white;
10  text-align: center;
11  padding: 1rem;
12 }
13
14 main {
15  padding: 1rem;
16 }
```

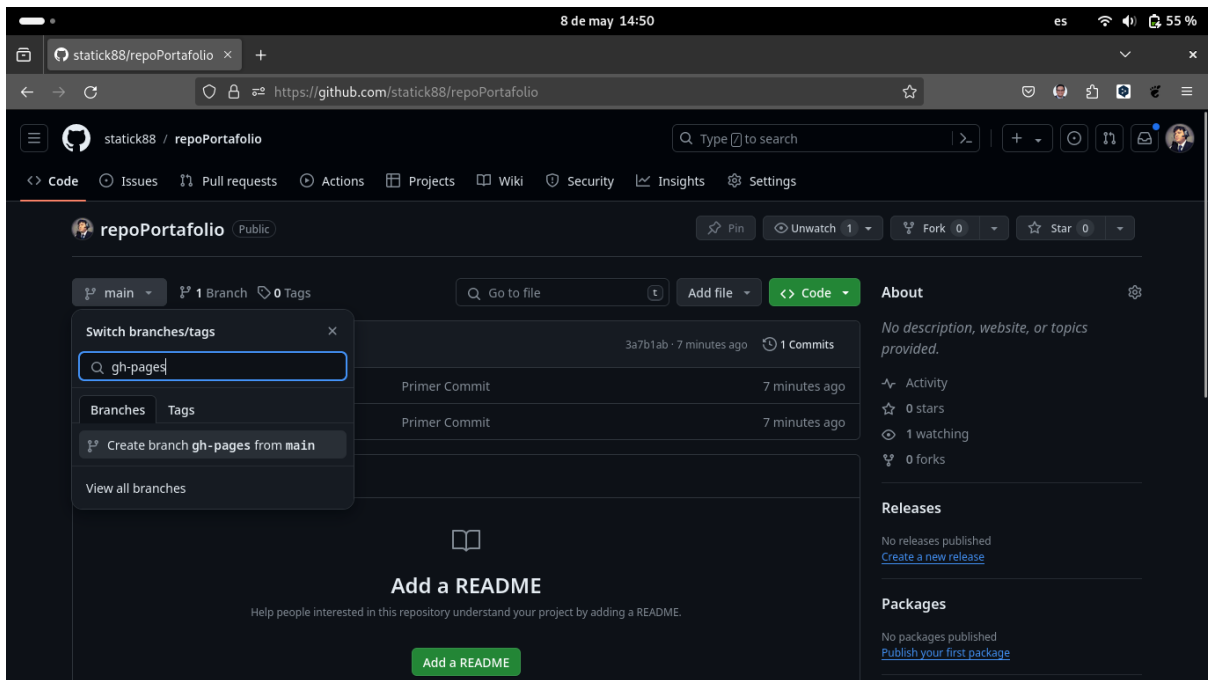
The terminal window shows the following commands and output:

```
create mode 100644 styles.css
static@lenovo ~/.../temal/repoPortafolio master git remote add origin https://github.com/static88/repoPortafolio.git
git branch -M main
git push -u origin main
Enumerando objetos: 4, listo.
Contando objetos: 100% (4/4), listo.
Compresión delta usando hasta 8 hilos
Comprimiendo objetos: 100% (4/4), listo.
Escribiendo objetos: 100% (4/4), 837 bytes | 837.00 KiB/s, listo.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/static88/repoPortafolio.git
 * [new branch]    main -> main
rama 'main' configurada para rastrear 'origin/main'.
static@lenovo ~/.../temal/repoPortafolio main
```

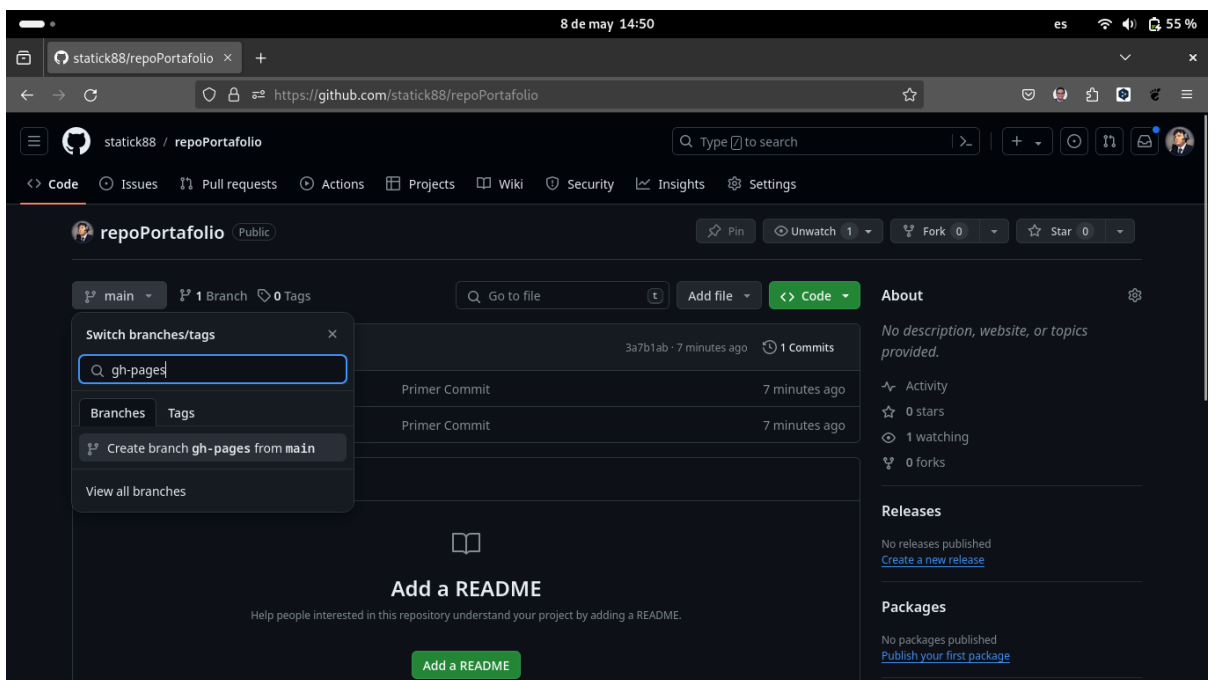
6. Actualizamos el navegador web.



7. Nos dirigimos a la seccion de las Ramas y buscamos la rama **gb-pages**, de la misma forma al no encontrarla damos clic en la opción **Create branch gh-pages from main**.

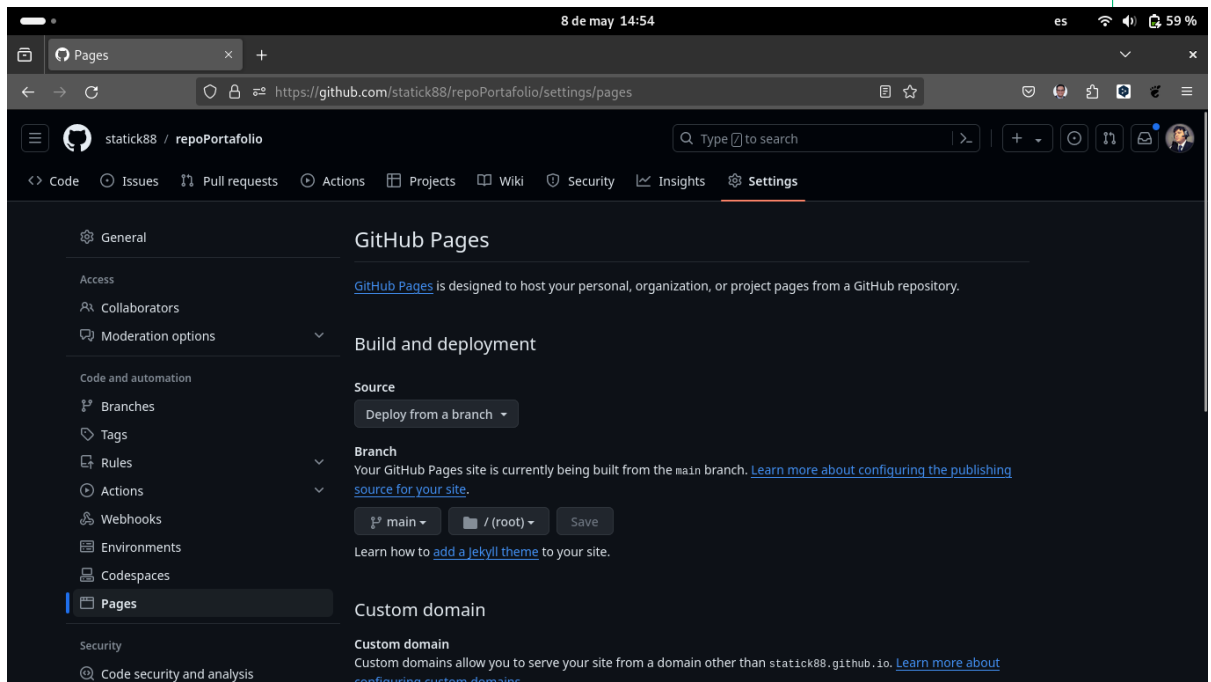


8. Verificamos la url generada para acceder a nuestra página web en la sección de **Settings**.

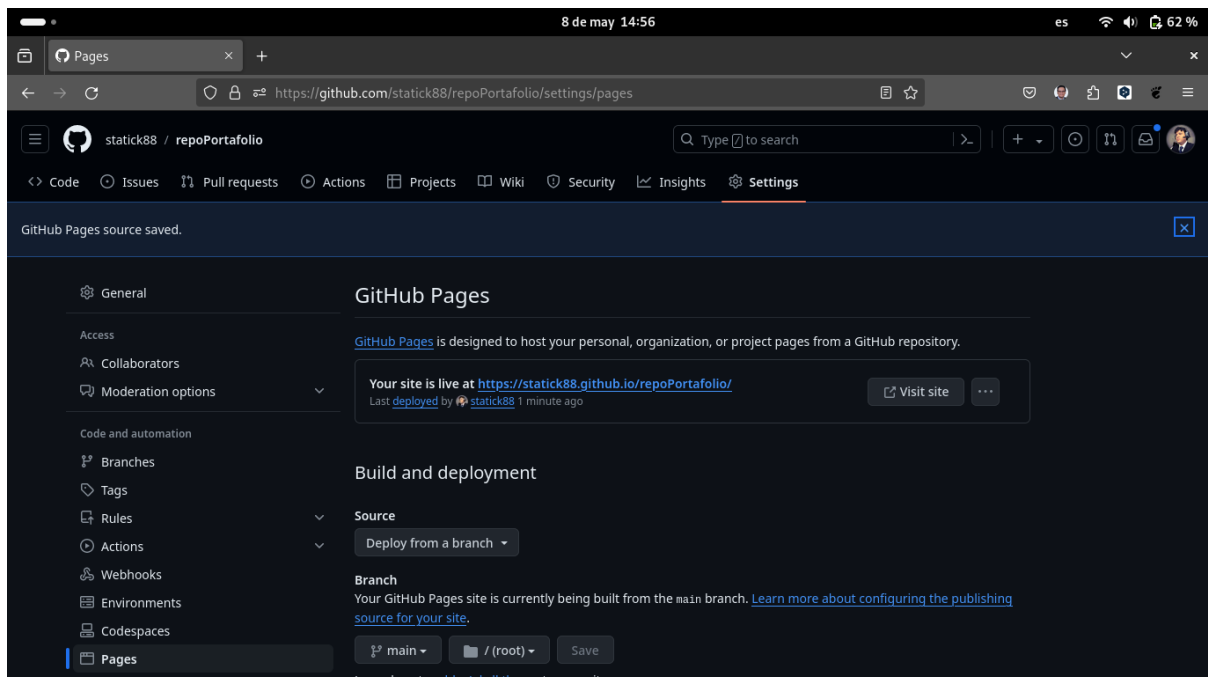


💡 Tip

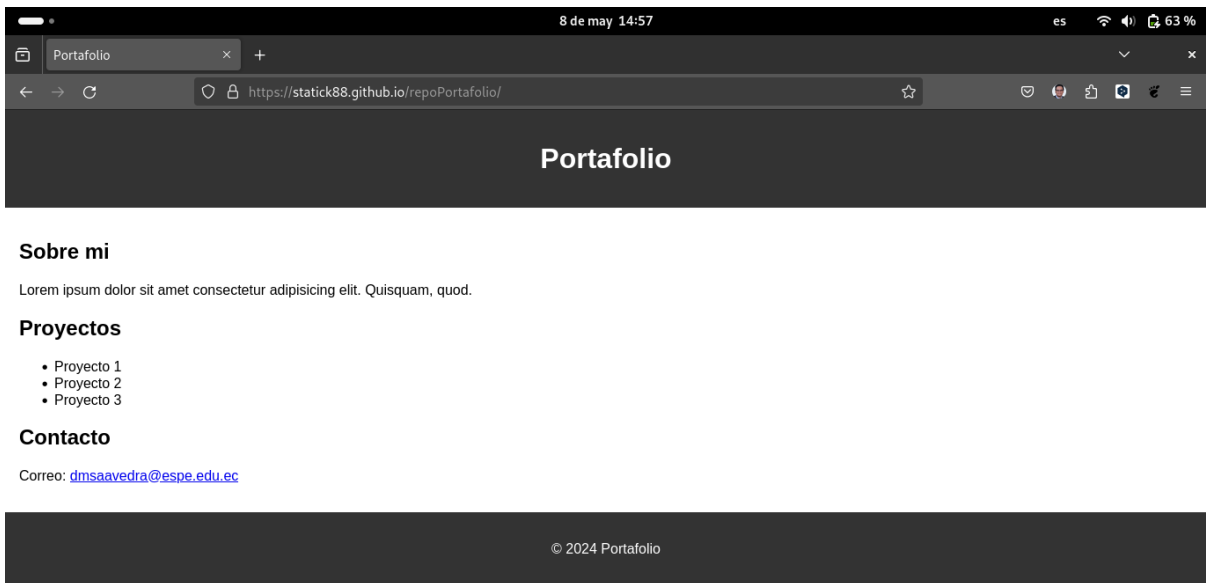
Si no se actualiza y sale algo como esto, no te preocupes, solo espera unos minutos y vuelve a intentar.



9. Esperamos a que se actualice la página y verificamos que la página se haya desplegado correctamente.



10. Ahora podemos acceder a nuestra página web desde la url generada.

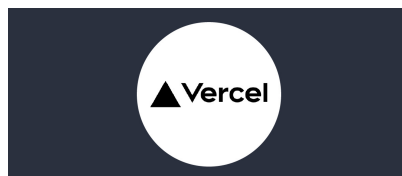


¡Listo! Hemos desplegado nuestra página web en Github Pages.

💡 Tip

Cada vez que actualicemos nuestro repositorio, la página web se actualizará automáticamente. De no ser así basta con eliminar la rama **gh-pages** y volver a crearla.

2.3 Vercel

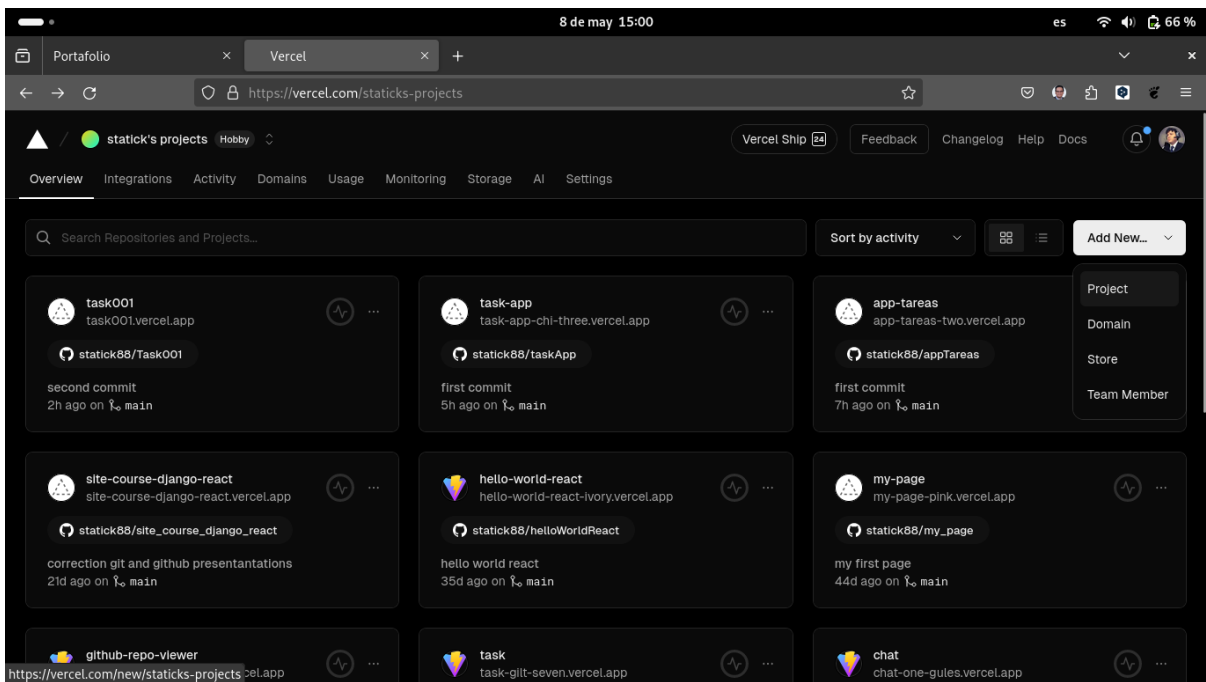


También es posible realizar el mismo proceso en Vercel, para ello necesitamos tener una cuenta en Vercel y un repositorio en Github.

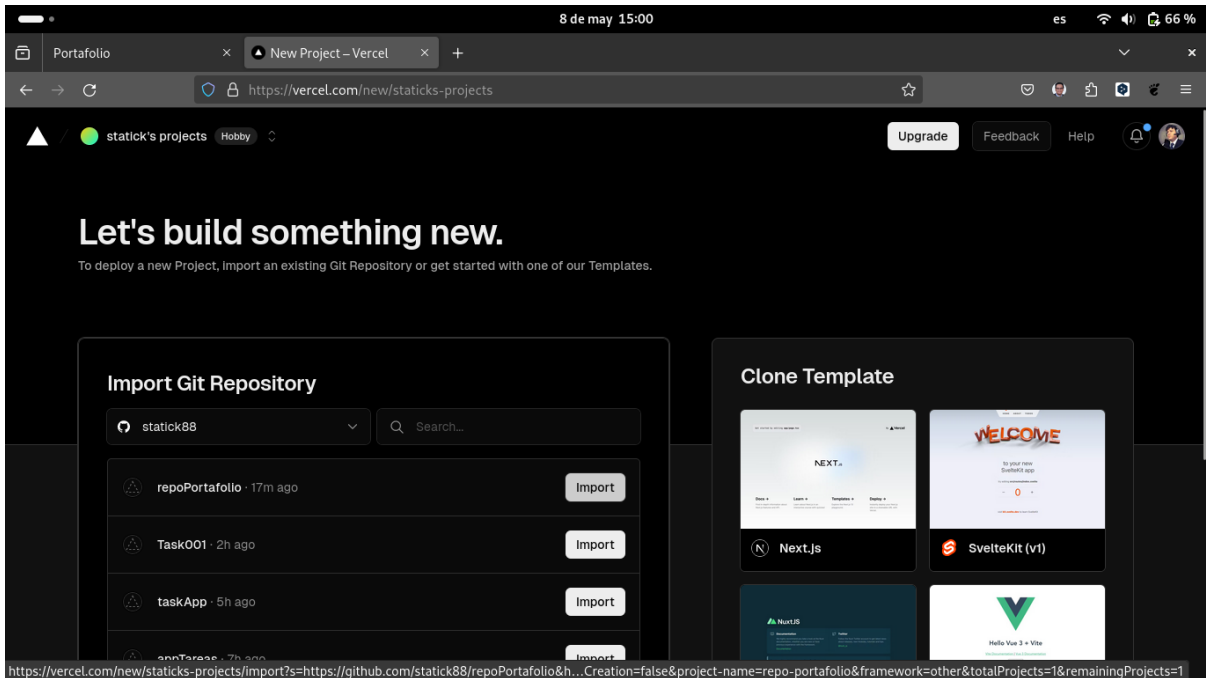
Podemos utilizar el repositorio que creamos en el tema anterior.

2.3.1 Crear un proyecto en Vercel

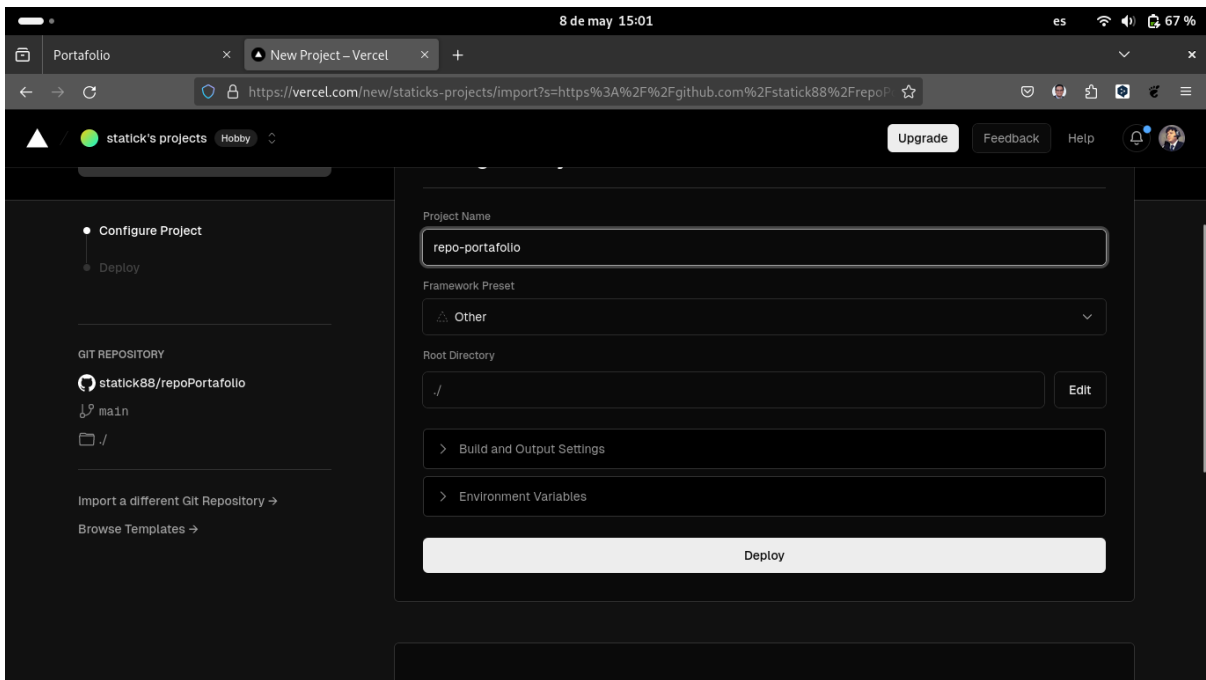
1. Ingresar a Vercel y crear un nuevo proyecto.



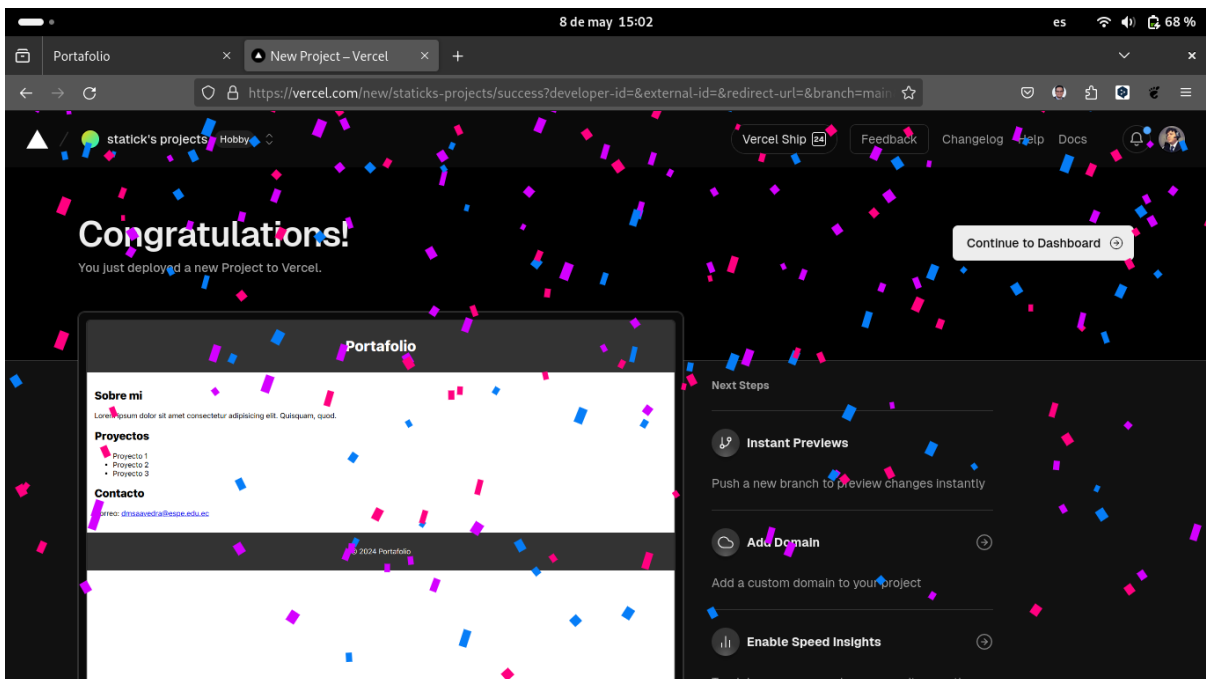
2. Seleccionar el repositorio de Github.



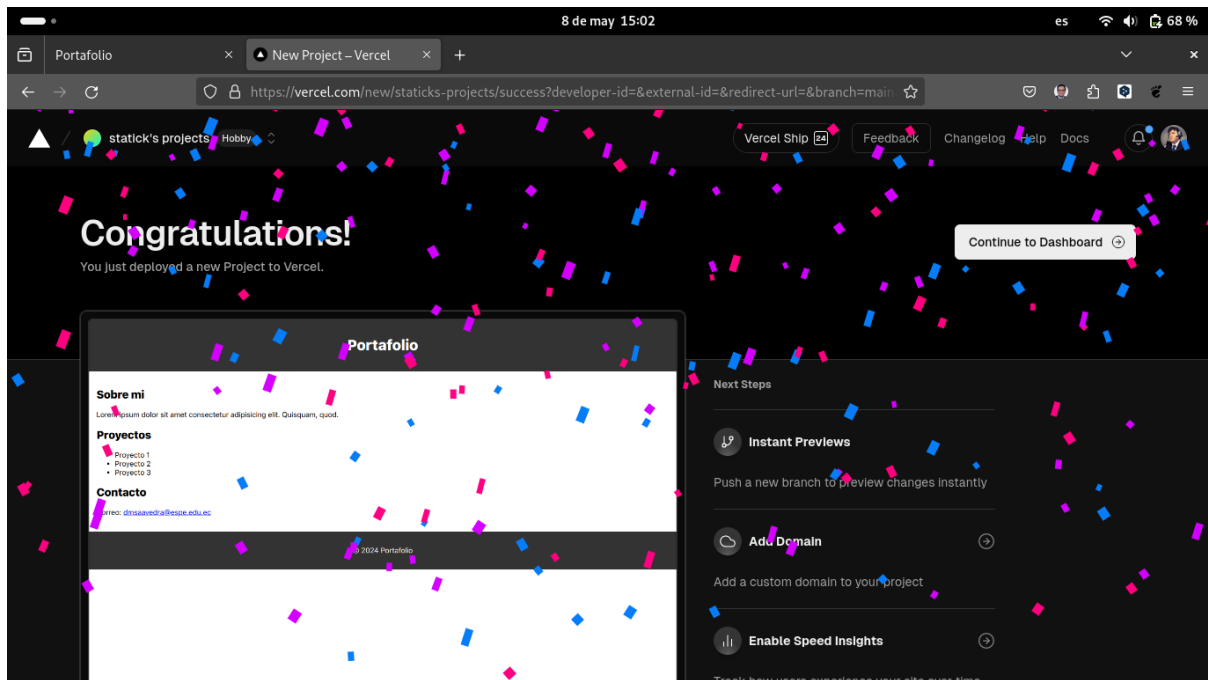
3. Seleccionamos el botón **Deploy**.



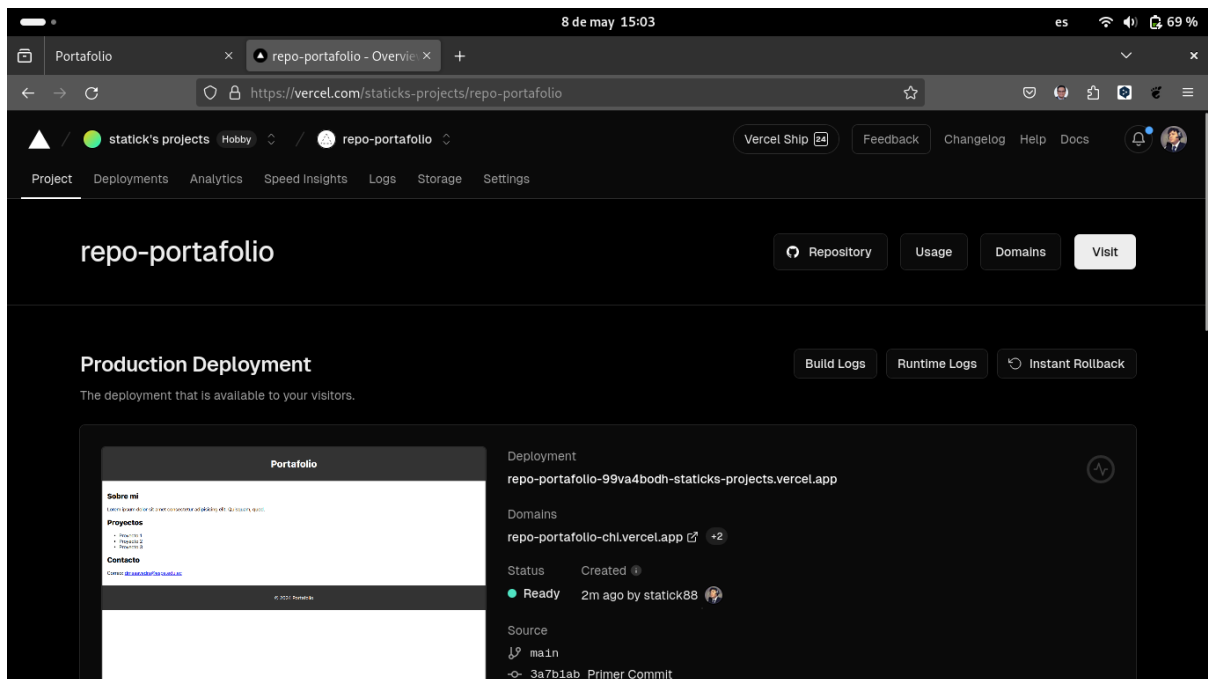
4. Esperamos a que se despliegue el proyecto.



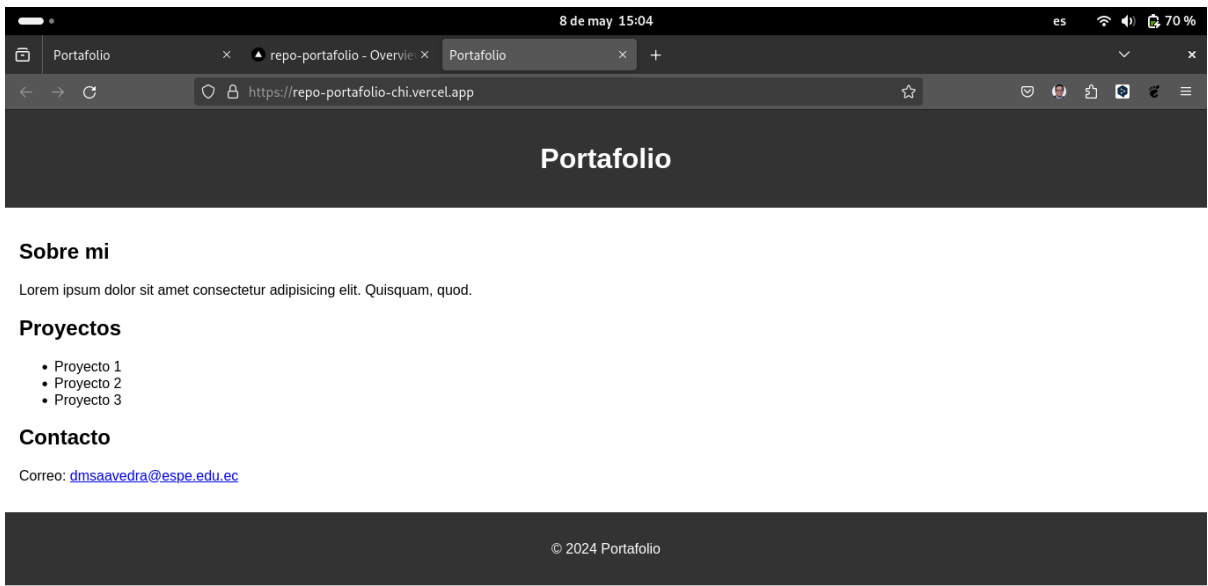
5. Una vez desplegado, podemos acceder la sección **Continue Dashboard**.



6. Accedemos al botón **Visit**



7. Visualizamos nuestro proyecto desplegado en Vercel.



¡Listo! Hemos desplegado nuestra página web en Vercel.

3 Creación de Aplicaciones Web

En la actualidad las aplicaciones web modernas son una parte esencial de la vida cotidiana, ya que permiten a los usuarios acceder a una amplia variedad de servicios y contenidos a través de Internet. Las aplicaciones web pueden ser tan simples como un sitio web estático o tan complejas como una aplicación web interactiva o una plataforma de comercio electrónico.

En este documento se explorarán los conceptos y tecnologías clave relacionados con la creación de aplicaciones web modernas, incluyendo HTML, CSS, JavaScript, Ajax, JSON, REST, GraphQL, WebSockets, Single Page Applications (SPAs), Progressive Web Applications (PWAs) y más.

4 HTML

4.1 ¿Qué es HTML?

HTML (HyperText Markup Language) es el lenguaje de marcado estándar utilizado para crear y diseñar páginas web. HTML se compone de elementos y etiquetas que se utilizan para estructurar y presentar el contenido de una página web, como texto, imágenes, enlaces, formularios, videos, etc.

4.2 ¿Cómo funciona HTML?

HTML se compone de elementos y etiquetas que se utilizan para definir la estructura y el contenido de una página web. Cada elemento HTML se define mediante una etiqueta de apertura y una etiqueta de cierre, y puede contener texto, imágenes, enlaces, formularios, videos, etc. Los elementos HTML se pueden anidar para crear una estructura jerárquica de la página web.

4.3 Ejemplo de HTML

```
<!DOCTYPE html>
<html>
<head>
  <title>My First Web Page</title>
</head>
<body>
  <h1>Hello, World!</h1>
  <p>This is my first web page.</p>
  
  <a href="https://www.example.com">Visit Example</a>
</body>
</html>
```

En este ejemplo, se muestra un documento HTML básico que contiene un encabezado, un párrafo, una imagen y un enlace. Cada elemento HTML se define mediante una etiqueta de apertura y una etiqueta de cierre, y puede contener atributos como `src`, `alt` y `href` para especificar la fuente, el texto alternativo y la URL del enlace.

4.4 Ventajas de HTML

- **Facilidad de uso:** HTML es un lenguaje de marcado sencillo y fácil de aprender que se utiliza para crear y diseñar páginas web.
- **Compatibilidad:** HTML es compatible con todos los navegadores modernos y es ampliamente utilizado en aplicaciones web y servicios web.
- **Flexibilidad:** HTML se puede combinar con CSS y JavaScript para crear páginas web interactivas y atractivas.
- **Estándar abierto:** HTML es un estándar abierto y estándar de facto para la creación de páginas web en Internet.

4.5 Desventajas de HTML

- **Limitaciones de diseño:** HTML tiene limitaciones en términos de diseño y presentación, lo que puede dificultar la creación de páginas web complejas y sofisticadas.
- **Compatibilidad con dispositivos:** HTML puede no ser compatible con todos los dispositivos y tamaños de pantalla, lo que puede afectar la experiencia del usuario en dispositivos móviles y tabletas.
- **SEO:** HTML puede ser menos accesible para los motores de búsqueda si no se optimiza correctamente para el SEO (Search Engine Optimization).
- **Seguridad:** HTML puede ser vulnerable a ataques de seguridad como Cross-Site Scripting (XSS) y Cross-Site Request Forgery (CSRF) si no se implementan correctamente las medidas de seguridad.

4.6 Ejemplo de uso de HTML

```
<!DOCTYPE html>
<html>
<head>
  <title>My First Web Page</title>
</head>
<body>
  <h1>Hello, World!</h1>
  <p>This is my first web page.</p>
  
  <a href="https://www.example.com">Visit Example</a>
</body>
</html>
```

En este ejemplo, se muestra un documento HTML básico que contiene un encabezado, un párrafo, una imagen y un enlace. Cada elemento HTML se define mediante una etiqueta de apertura y una etiqueta de cierre, y puede contener atributos como `src`, `alt` y `href` para especificar la fuente, el texto alternativo y la URL del enlace.

4.7 Conclusión

HTML es un lenguaje de marcado esencial para la creación y el diseño de páginas web en Internet. HTML se utiliza para estructurar y presentar el contenido de una página web, como texto, imágenes, enlaces, formularios, videos, etc. HTML es un estándar abierto y estándar de facto para la creación de páginas web y es compatible con todos los navegadores modernos. En resumen, HTML es una herramienta esencial para el desarrollo web y puede ayudar a crear páginas web atractivas, interactivas y accesibles.

5 CSS

5.1 ¿Qué es CSS?

CSS (Cascading Style Sheets) es un lenguaje de diseño utilizado para dar estilo y presentar páginas web creadas con HTML. CSS se utiliza para definir la apariencia y el diseño de una página web, como colores, fuentes, márgenes, bordes, alineación, etc.

5.2 ¿Cómo funciona CSS?

CSS se compone de reglas de estilo que se aplican a elementos HTML para definir su apariencia y diseño. Cada regla de estilo se compone de un selector, una propiedad y un valor, y se utiliza para dar estilo a elementos HTML específicos. Los estilos CSS se pueden aplicar en línea, en un archivo externo o en un bloque de estilo en la cabecera de la página.

5.3 Ejemplo de CSS

```
<!DOCTYPE html>
<html>
<head>
  <title>My First Web Page</title>
  <link rel="stylesheet" href="styles.css">
</head>
</head>
<body>
  <h1>Hello, World!</h1>
  <p>This is my first web page.</p>
  
  <a href="https://www.example.com">Visit Example</a>
</body>
</html>
```

A continuación se muestra un ejemplo de un archivo CSS externo llamado **styles.css** que contiene reglas de estilo para dar estilo a los elementos HTML de la página.

```
h1 {
  color: blue;
  font-size: 24px;
}

p {
  color: black;
  font-size: 16px;
}

img {
  width: 100%;
  height: auto;
}

a {
  color: green;
  text-decoration: none;
}
```

En este ejemplo, se muestra un documento HTML básico que contiene un encabezado, un párrafo, una imagen y un enlace con estilos CSS aplicados desde un archivo externo. Los estilos CSS se definen en un archivo separado llamado “styles.css” y se vinculan al documento HTML mediante la etiqueta **link** en la cabecera de la página.

5.4 Ventajas de CSS

- **Separación de contenido y diseño:** CSS permite separar el contenido de una página web del diseño y la presentación, lo que facilita la gestión y el mantenimiento del sitio.
- **Reutilización de estilos:** CSS permite reutilizar estilos en toda la página web mediante la definición de reglas de estilo que se aplican a múltiples elementos HTML.
- **Flexibilidad:** CSS proporciona una amplia variedad de propiedades y valores que se pueden utilizar para personalizar la apariencia y el diseño de una página web.
- **Compatibilidad:** CSS es compatible con todos los navegadores modernos y es ampliamente utilizado en aplicaciones web y servicios web.

5.5 Desventajas de CSS

- **Compatibilidad con navegadores:** CSS puede tener problemas de compatibilidad con navegadores más antiguos y versiones específicas de navegadores, lo que puede afectar la apariencia y el diseño de una página web.

- **Curva de aprendizaje:** CSS puede tener una curva de aprendizaje empinada para los desarrolladores que no están familiarizados con el diseño y la presentación de páginas web.
- **Rendimiento:** CSS puede afectar el rendimiento de una página web si se utilizan estilos complejos y pesados que requieren una gran cantidad de recursos del navegador.
- **Seguridad:** CSS puede ser vulnerable a ataques de seguridad como Cross-Site Scripting (XSS) si no se implementan correctamente las medidas de seguridad.

5.6 Ejemplo de uso de CSS

```
<!DOCTYPE html>
<html>
<head>
  <title>My First Web Page</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <h1>Hello, World!</h1>
  <p>This is my first web page.</p>
  
  <a href="https://www.example.com">Visit Example</a>
</body>
</html>
```

A continuación se muestra un ejemplo de un archivo CSS externo llamado **styles.css** que contiene reglas de estilo para dar estilo a los elementos HTML de la página.

```
h1 {
  color: blue;
  font-size: 24px;
}

p {
  color: black;
  font-size: 16px;
}

img {
  width: 100%;
  height: auto;
}

a {
  color: green;
}
```

```
text-decoration: none;  
}
```

En este ejemplo, se muestra un documento HTML básico que contiene un encabezado, un párrafo, una imagen y un enlace con estilos CSS aplicados desde un archivo externo. Los estilos CSS se definen en un archivo separado llamado “styles.css” y se vinculan al documento HTML mediante la etiqueta **link** en la cabecera de la página.

5.7 Conclusión

CSS es un lenguaje de diseño esencial para dar estilo y presentar páginas web creadas con HTML. CSS se utiliza para definir la apariencia y el diseño de una página web, como colores, fuentes, márgenes, bordes, alineación, etc. CSS proporciona una forma eficiente y flexible de personalizar la apariencia y el diseño de una página web y es compatible con todos los navegadores modernos. En resumen, CSS es una herramienta esencial para el desarrollo web y puede ayudar a crear páginas web atractivas, interactivas y accesibles.

6 JavaScript

6.1 ¿Qué es JavaScript?

JavaScript es un lenguaje de programación de alto nivel y de propósito general que se utiliza para crear aplicaciones web interactivas y dinámicas. JavaScript se utiliza para agregar interactividad, funcionalidad y comportamiento a las páginas web y se ejecuta en el navegador del cliente.

6.2 ¿Cómo funciona JavaScript?

JavaScript se compone de instrucciones y funciones que se utilizan para realizar operaciones en el lado del cliente, como manipular el DOM (Document Object Model), responder a eventos del usuario, realizar validaciones de formularios, etc. JavaScript se puede incrustar en una página web utilizando etiquetas **script** en la cabecera o el cuerpo de la página, o se puede cargar desde un archivo externo.

6.3 Ejemplo de JavaScript

```
<!DOCTYPE html>
<html>
<head>
  <title>My First Web Page</title>
  <script src="script.js"></script>
</head>
</head>
<body>
  <h1>Hello, World!</h1>
  <button onclick="greet()">Click Me</button>
</body>
</html>
```

A continuación se muestra un ejemplo de un archivo JavaScript externo llamado **script.js** que contiene una función **greet()** que muestra un mensaje de alerta cuando se llama.

```
function greet() {
  alert("Hello, World!");
}
```


En este ejemplo, se muestra un documento HTML básico que contiene un encabezado y un botón con un evento onclick que llama a una función JavaScript para mostrar un mensaje de alerta. La función `greet()` se define en una etiqueta `script` en la cabecera de la página y se llama cuando se hace clic en el botón.

6.4 Ventajas de JavaScript

- **Interactividad:** JavaScript permite agregar interactividad y funcionalidad a las páginas web, como eventos del usuario, animaciones, validaciones de formularios, etc.
- **Rendimiento:** JavaScript se ejecuta en el navegador del cliente y puede mejorar el rendimiento de la aplicación al realizar operaciones en el lado del cliente en lugar del servidor.
- **Compatibilidad:** JavaScript es compatible con todos los navegadores modernos y es ampliamente utilizado en aplicaciones web y servicios web.
- **Flexibilidad:** JavaScript es un lenguaje de programación versátil que se puede utilizar para crear aplicaciones web complejas y sofisticadas.

6.5 Desventajas de JavaScript

- **Seguridad:** JavaScript puede ser vulnerable a ataques de seguridad como Cross-Site Scripting (XSS) y Cross-Site Request Forgery (CSRF) si no se implementan correctamente las medidas de seguridad.
- **Rendimiento:** JavaScript puede afectar el rendimiento de una página web si se utilizan scripts complejos y pesados que requieren una gran cantidad de recursos del navegador.
- **Compatibilidad con dispositivos:** JavaScript puede no ser compatible con todos los dispositivos y tamaños de pantalla, lo que puede afectar la experiencia del usuario en dispositivos móviles y tabletas.
- **Curva de aprendizaje:** JavaScript puede tener una curva de aprendizaje empinada para los desarrolladores que no están familiarizados con la programación y el desarrollo web.

6.6 Ejemplo de uso de JavaScript

```
<!DOCTYPE html>
<html>
<head>
  <title>My First Web Page</title>
  <script src="script.js"></script>
```

```
</head>
<body>
  <h1>Hello, World!</h1>
  <button onclick="greet()">Click Me</button>
</body>
</html>
```

A continuación se muestra un ejemplo de un archivo JavaScript externo llamado **script.js** que contiene una función **greet()** que muestra un mensaje de alerta cuando se llama.

```
function greet() {
  alert("Hello, World!");
}
```

En este ejemplo, se muestra un documento HTML básico que contiene un encabezado y un botón con un evento onclick que llama a una función JavaScript para mostrar un mensaje de alerta. La función **greet()** se define en un archivo externo llamado **script.js** y se vincula al documento HTML mediante la etiqueta **script** en la cabecera de la página.

6.7 Conclusión

JavaScript es un lenguaje de programación esencial para crear aplicaciones web interactivas y dinámicas. JavaScript se utiliza para agregar interactividad, funcionalidad y comportamiento a las páginas web y se ejecuta en el navegador del cliente. JavaScript es compatible con todos los navegadores modernos y es ampliamente utilizado en aplicaciones web y servicios web. En resumen, JavaScript

7 Single Page Applications (SPAs)

7.1 ¿Qué es una Single Page Application (SPA)?

Una Single Page Application (SPA) es una aplicación web que se carga en una sola página y se actualiza dinámicamente a medida que el usuario interactúa con ella. Las SPAs utilizan tecnologías como HTML, CSS y JavaScript para crear una experiencia de usuario fluida y receptiva sin necesidad de recargar la página.

7.2 ¿Cómo funciona una SPA?

Una SPA se compone de una sola página HTML que contiene todo el contenido y la funcionalidad de la aplicación. La página se carga una sola vez y se actualiza dinámicamente a medida que el usuario interactúa con ella. Las SPAs utilizan JavaScript para cargar y mostrar contenido de forma asíncrona, lo que mejora el rendimiento y la usabilidad de la aplicación.

7.3 Ejemplo de SPA

```
<!DOCTYPE html>
<html>
<head>
  <title>My First Single Page Application</title>
  <link rel="stylesheet" href="styles.css">
  <script src="script.js"></script>
</head>
</head>
<body>
  <h1>Hello, World!</h1>
  <button onclick="showContent()">Show Content</button>
  <div id="content">
    <p>This is the content of the single page application.</p>
  </div>
</body>
</html>
```

A continuación se muestra un ejemplo de un archivo CSS externo llamado **styles.css** que contiene estilos para dar estilo a la SPA.

```
#content {  
  display: none;  
}
```

A continuación se muestra un ejemplo de un archivo JavaScript externo llamado **script.js** que contiene una función **showContent()** que muestra el contenido de la SPA cuando se llama.

```
function showContent() {  
  var content = document.getElementById("content");  
  content.style.display = "block";  
}
```

En este ejemplo, se muestra una Single Page Application (SPA) básica creada con HTML, CSS y JavaScript. La aplicación contiene un botón que llama a una función **showContent()** para mostrar el contenido de la SPA cuando se hace clic en él.

7.4 Ventajas de SPAs

- **Interactividad:** Las SPAs permiten a los usuarios interactuar con la aplicación web de forma fluida y receptiva sin recargar la página.
- **Rendimiento:** Las SPAs cargan el contenido de forma asíncrona y actualizan dinámicamente la página, lo que mejora el rendimiento y la usabilidad de la aplicación.
- **Usabilidad:** Las SPAs proporcionan una experiencia de usuario más atractiva y agradable al ofrecer una interfaz de usuario más dinámica y receptiva.
- **Compatibilidad:** Las SPAs funcionan en todos los navegadores modernos y son compatibles con la mayoría de los lenguajes de programación del lado del servidor.

7.5 Desventajas de SPAs

- **SEO:** Las SPAs pueden afectar el SEO (Search Engine Optimization) al hacer que el contenido de la página sea menos accesible para los motores de búsqueda.
- **Seguridad:** Las SPAs pueden ser vulnerables a ataques de seguridad como Cross-Site Scripting (XSS) y Cross-Site Request Forgery (CSRF) si no se implementan correctamente las medidas de seguridad.
- **Compatibilidad con dispositivos:** Las SPAs pueden no ser compatibles con todos los dispositivos y tamaños de pantalla, lo que puede afectar la experiencia del usuario en dispositivos móviles y tabletas.
- **Curva de aprendizaje:** Las SPAs pueden tener una curva de aprendizaje empinada para los desarrolladores que no están familiarizados con el desarrollo web y las tecnologías de front-end.

7.6 Ejemplo de uso de SPAs con React

```
<!DOCTYPE html>
<html>
<head>
  <title>My First Single Page Application with React</title>
  <script src="https://unpkg.com/react@17/umd/react.development.js"></script>
  <script src="https://unpkg.com/react-dom@17/umd/react-dom.development.js"></script>
  <script src="https://unpkg.com/babel-standalone@6/babel.min.js"></script>
</head>
<body>
  <div id="root"></div>
  <script src="script.js"></script>
</body>
</html>
```

A continuación se muestra un ejemplo de un archivo JavaScript externo llamado **script.js** que contiene una Single Page Application (SPA) creada con React.

```
class App extends
React.Component {
  render() {
    return (
      <div>
        <h1>Hello, World!</h1>
        <p>This is a Single Page Application created with React.</p>
      </div>
    );
  }
}

ReactDOM.render(<App />, document.getElementById('root'));
```

En este ejemplo, se muestra una Single Page Application (SPA) básica creada con React, una biblioteca de JavaScript para crear interfaces de usuario interactivas y dinámicas. La aplicación contiene un componente **App** que muestra un encabezado y un párrafo en la página.

7.7 Conclusión

Las Single Page Applications (SPAs) son una forma moderna y eficiente de crear aplicaciones web interactivas y dinámicas. Las SPAs se cargan en una sola página y se actualizan dinámicamente a medida que el usuario interactúa con ellas, lo que mejora el rendimiento y la usabilidad de la aplicación. Las SPAs utilizan tecnologías como HTML, CSS y JavaScript para crear una experiencia de usuario fluida y receptiva sin necesidad

de recargar la página. En resumen, las SPAs son una herramienta esencial para el desarrollo web moderno y pueden ayudar a crear aplicaciones web más rápidas, interactivas y atractivas.

7.8 Ajax

7.9 ¿Qué es Ajax?

Ajax es una técnica de desarrollo web para crear aplicaciones interactivas o RIA (Rich Internet Applications). Con Ajax, las aplicaciones web pueden enviar y recuperar datos del servidor en segundo plano sin interferir con la visualización y el comportamiento de la página. Los datos se pueden recuperar utilizando el formato XML, JSON o HTML.

7.10 ¿Cómo funciona Ajax?

Ajax funciona enviando una solicitud HTTP al servidor cuando se necesita recuperar datos. La solicitud se realiza en segundo plano, por lo que la página web no se recarga. Una vez que se recibe la respuesta del servidor, los datos se procesan y se actualiza la página web.

7.11 ¿Por qué usar Ajax?

- **Interactividad:** Permite a los usuarios interactuar con la aplicación web sin recargar la página.
- **Rendimiento:** Permite cargar datos de forma asíncrona, lo que mejora el rendimiento de la aplicación.
- **Usabilidad:** Mejora la experiencia del usuario al proporcionar una interfaz de usuario más dinámica y receptiva.
- **Compatibilidad:** Funciona en todos los navegadores modernos y es compatible con la mayoría de los lenguajes de programación del lado del servidor.

7.12 Ejemplo de uso de Ajax

```
<!DOCTYPE html>
<html>
<head>
  <title>Ajax Example</title>
  <script href="script.js"></script>
</head>
<body>
  <h2>Using the XMLHttpRequest Object</h2>
  <button type="button" onclick="loadDoc()">Get Data</button>
```

```
<p id="demo"></p>
</body>
</html>
```

A continuación se muestra un ejemplo de un archivo JavaScript externo llamado **script.js** que contiene una función **loadDoc()** que utiliza el objeto XMLHttpRequest para realizar una solicitud GET al servidor y recuperar los datos del archivo "ajax_info.txt".

```
function loadDoc() {
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
            document.getElementById("demo").innerHTML = this.responseText;
        }
    };
    xhttp.open("GET", "ajax_info.txt", true);
    xhttp.send();
}
```

En este ejemplo, se muestra cómo utilizar el objeto XMLHttpRequest para realizar una solicitud GET al servidor y recuperar los datos del archivo "ajax_info.txt". Una vez que se recibe la respuesta del servidor, los datos se muestran en el elemento **p** con el id "demo".

7.13 Ventajas de Ajax

- **Interactividad:** Permite a los usuarios interactuar con la aplicación web sin recargar la página.
- **Rendimiento:** Permite cargar datos de forma asíncrona, lo que mejora el rendimiento de la aplicación.
- **Usabilidad:** Mejora la experiencia del usuario al proporcionar una interfaz de usuario más dinámica y receptiva.
- **Compatibilidad:** Funciona en todos los navegadores modernos y es compatible con la mayoría de los lenguajes de programación del lado del servidor.

7.14 Desventajas de Ajax

- **Seguridad:** Puede ser vulnerable a ataques de seguridad como Cross-Site Scripting (XSS) y Cross-Site Request Forgery (CSRF).
- **SEO:** Puede afectar el SEO (Search Engine Optimization) al hacer que el contenido de la página sea menos accesible para los motores de búsqueda.
- **Complejidad:** Puede ser más complejo de implementar y depurar que las aplicaciones web tradicionales.

- **Compatibilidad:** Puede haber problemas de compatibilidad con navegadores más antiguos y versiones específicas de navegadores.

7.15 Ejemplo de uso de Ajax con jQuery

```
<!DOCTYPE html>
<html>
<head>
  <title>Ajax Example</title>
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <script href="script.js"></script>
</head>
<body>
  <h2>Using jQuery for Ajax</h2>
  <button id="btnGetData">Get Data</button>
  <p id="demo"></p>
</body>
</html>
```

A continuación se muestra un ejemplo de un archivo JavaScript externo llamado **script.js** que contiene una función **loadDoc()** que utiliza jQuery para realizar una solicitud GET al servidor y recuperar los datos del archivo “ajax_info.txt”.

```
$("#btnGetData").click(function() {
  $.get("ajax_info.txt", function(data) {
    $("#demo").html(data);
  });
});
```

En este ejemplo, se muestra cómo utilizar jQuery para realizar una solicitud GET al servidor y recuperar los datos del archivo “ajax_info.txt”. Una vez que se recibe la respuesta del servidor, los datos se muestran en el elemento **p** con el id “demo”.

7.16 Ejemplo de uso de Ajax con Fetch API

```
<!DOCTYPE html>
<html>
<head>
  <title>Ajax Example</title>
  <script href="script.js"></script>
</head>
<body>
  <h2>Using Fetch API for Ajax</h2>
```



```

    <button type="button" onclick="loadDoc()">Get Data</button>
    <p id="demo"></p>
</body>
</html>

```

A continuación se muestra un ejemplo de un archivo JavaScript externo llamado **script.js** que contiene una función **loadDoc()** que utiliza la Fetch API para realizar una solicitud GET al servidor y recuperar los datos del archivo “ajax_info.txt”.

```

function loadDoc() {
    fetch("ajax_info.txt")
        .then(response => response.text())
        .then(data => {
            document.getElementById("demo").innerHTML = data;
        });
}

```

En este ejemplo, se muestra cómo utilizar la Fetch API para realizar una solicitud GET al servidor y recuperar los datos del archivo “ajax_info.txt”. Una vez que se recibe la respuesta del servidor, los datos se muestran en el elemento **p** con el id “demo”.

7.17 Ejemplo de uso de Ajax con XMLHttpRequest y Promesas

```

<!DOCTYPE html>
<html>
<head>
    <title>Ajax Example</title>
    <script href="script.js"></script>
</head>
<body>
    <h2>Using XMLHttpRequest and Promises for Ajax</h2>
    <button type="button" onclick="loadDoc()">Get Data</button>
    <p id="demo"></p>
</body>
</html>

```

A continuación se muestra un ejemplo de un archivo JavaScript externo llamado **script.js** que contiene una función **loadDoc()** que utiliza el objeto XMLHttpRequest y las Promesas para realizar una solicitud GET al servidor y recuperar los datos del archivo “ajax_info.txt”.

```

function loadDoc() {
    return new Promise((resolve, reject) => {
        var xhttp = new XMLHttpRequest();

```

```

    xmlhttp.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
            resolve(this.responseText);
        }
    };
    xmlhttp.open("GET", "ajax_info.txt", true);
    xmlhttp.send();
}).then(data => {
    document.getElementById("demo").innerHTML = data;
});
}

```

En este ejemplo, se muestra cómo utilizar el objeto XMLHttpRequest y las Promesas para realizar una solicitud GET al servidor y recuperar los datos del archivo “ajax_info.txt”. Una vez que se recibe la respuesta del servidor, los datos se muestran en el elemento **p** con el id “demo”.

7.18 Ejemplo de uso de Ajax con Axios

```

<!DOCTYPE html>
<html>
<head>
    <title>Ajax Example</title>
    <script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>
    <script href="script.js">
    </script>
</head>
<body>
    <h2>Using Axios for Ajax</h2>
    <button type="button" onclick="loadDoc()">Get Data</button>
    <p id="demo"></p>
</body>
</html>

```

A continuación se muestra un ejemplo de un archivo JavaScript externo llamado **script.js** que contiene una función **loadDoc()** que utiliza Axios para realizar una solicitud GET al servidor y recuperar los datos del archivo “ajax_info.txt”.

```

function loadDoc() {
    axios.get("ajax_info.txt")
        .then(response => {
            document.getElementById("demo").innerHTML = response.data;
        });
}

```

En este ejemplo, se muestra cómo utilizar Axios para realizar una solicitud GET al servidor y recuperar los datos del archivo “ajax_info.txt”. Una vez que se recibe la respuesta del servidor, los datos se muestran en el elemento **p** con el id “demo”.

7.19 Conclusión

Ajax es una técnica poderosa y versátil que permite a los desarrolladores web crear aplicaciones interactivas y dinámicas. Con Ajax, es posible cargar datos de forma asíncrona, lo que mejora el rendimiento y la usabilidad de la aplicación. Además, Ajax es compatible con la mayoría de los lenguajes de programación del lado del servidor y funciona en todos los navegadores modernos. En resumen, Ajax es una herramienta esencial para el desarrollo web moderno y puede ayudar a crear aplicaciones web más rápidas, interactivas y atractivas.

8 JSON

8.1 ¿Qué es JSON?

JSON (JavaScript Object Notation) es un formato de intercambio de datos ligero y fácil de leer que se utiliza para transmitir datos entre un servidor y un cliente. JSON se basa en la sintaxis de JavaScript y es independiente del lenguaje, lo que significa que se puede utilizar con cualquier lenguaje de programación que admita la serialización y la deserialización de datos.

8.2 ¿Cómo funciona JSON?

JSON se compone de pares clave-valor que se organizan en objetos y matrices. Los objetos se delimitan con llaves `{}` y contienen pares clave-valor separados por comas. Las matrices se delimitan con corchetes `[]` y contienen valores separados por comas. Los valores pueden ser cadenas de texto, números, booleanos, objetos, matrices o nulos.

8.3 Ejemplo de JSON

```
{
  "name": "John Doe",
  "age": 30,
  "isStudent": false,
  "address": {
    "street": "123 Main St",
    "city": "Anytown",
    "state": "CA"
  },
  "hobbies": ["reading", "traveling", "photography"]
}
```

En este ejemplo, se muestra un objeto JSON que contiene información sobre una persona, incluido su nombre, edad, si es estudiante, dirección, hobbies, etc.

8.4 Ventajas de JSON

- **Ligero:** JSON es un formato de intercambio de datos ligero y fácil de leer.
- **Fácil de usar:** JSON se basa en la sintaxis de JavaScript y es fácil de entender y utilizar.
- **Independiente del lenguaje:** JSON es independiente del lenguaje, lo que significa que se puede utilizar con cualquier lenguaje de programación que admita la serialización y la deserialización de datos.
- **Interoperabilidad:** JSON es compatible con la mayoría de los lenguajes de programación y es ampliamente utilizado en aplicaciones web y servicios web.

8.5 Desventajas de JSON

- **Limitaciones de tipos de datos:** JSON no admite todos los tipos de datos, como fechas, binarios, etc.
- **Seguridad:** JSON puede ser vulnerable a ataques de seguridad como Cross-Site Scripting (XSS) si no se valida correctamente.
- **Rendimiento:** JSON puede ser menos eficiente en términos de rendimiento y tamaño de datos en comparación con otros formatos como XML.
- **Legibilidad:** JSON puede ser difícil de leer y entender para los humanos cuando se trabaja con datos complejos o anidados.

8.6 Ejemplo de uso de JSON en JavaScript

```
// Parse JSON string to JavaScript object
var jsonString = '{"name": "John Doe", "age": 30}';
var jsonObject = JSON.parse(jsonString);
console.log(jsonObject);

// Convert JavaScript object to JSON string
var jsonObject = {name: "John Doe", age: 30};
var jsonString = JSON.stringify(jsonObject);
console.log(jsonString);
```

En este ejemplo, se muestra cómo analizar una cadena JSON en un objeto JavaScript utilizando el método **JSON.parse()** y cómo convertir un objeto JavaScript en una cadena JSON utilizando el método **JSON.stringify()**.

8.7 Conclusión

JSON es un formato de intercambio de datos ligero y fácil de leer que se utiliza para transmitir datos entre un servidor y un cliente. JSON se basa en la sintaxis de JavaScript y es independiente del lenguaje, lo que significa que se puede utilizar con cualquier lenguaje de programación que admita la serialización y la deserialización de datos. JSON es una herramienta esencial para el desarrollo de aplicaciones web modernas y puede ayudar a mejorar la interoperabilidad y el rendimiento de la aplicación.

9 REST

9.1 ¿Qué es REST?

REST (Representational State Transfer) es un estilo arquitectónico para diseñar servicios web que se basa en los principios de la web y utiliza los métodos HTTP para realizar operaciones CRUD (Create, Read, Update, Delete) en recursos. REST se centra en la simplicidad, la escalabilidad, la interoperabilidad y el rendimiento.

9.2 Principios de REST

- **Recursos:** Los recursos son entidades que se pueden identificar mediante un URI (Uniform Resource Identifier) y se pueden manipular utilizando los métodos HTTP.
- **URI:** Cada recurso se identifica mediante un URI único que se utiliza para acceder y manipular el recurso.
- **Métodos HTTP:** Los métodos HTTP se utilizan para realizar operaciones CRUD en los recursos. Los métodos más comunes son GET (obtener), POST (crear), PUT (actualizar) y DELETE (eliminar).
- **Representaciones:** Los recursos se representan en diferentes formatos como JSON, XML, HTML, etc., y se pueden intercambiar entre el cliente y el servidor.
- **Estado de la aplicación:** La aplicación web se basa en el estado del servidor y no en el estado del cliente. Cada solicitud HTTP contiene toda la información necesaria para procesar la solicitud.

9.3 Ejemplo de REST

```
GET /api/users
GET /api/users/1
POST /api/users
PUT /api/users/1
DELETE /api/users/1
```

En este ejemplo, se muestran los métodos HTTP comunes utilizados en REST para realizar operaciones CRUD en los recursos de usuarios. El método GET se utiliza para recuperar una lista de usuarios o un usuario específico, el método POST se utiliza para crear un nuevo usuario, el método PUT se utiliza para actualizar un usuario existente y el método DELETE se utiliza para eliminar un usuario.

9.4 Ventajas de REST

- **Simplicidad:** REST se basa en los principios de la web y utiliza los métodos HTTP estándar para realizar operaciones CRUD en los recursos.
- **Escalabilidad:** REST es altamente escalable y puede manejar un gran número de solicitudes simultáneas.
- **Interoperabilidad:** REST es independiente del lenguaje y se puede utilizar con cualquier lenguaje de programación que admita las operaciones CRUD y los métodos HTTP.
- **Rendimiento:** REST es eficiente en términos de rendimiento y utiliza la caché para mejorar la velocidad de respuesta de la aplicación.

9.5 Desventajas de REST

- **Complejidad:** REST puede ser más complejo de implementar y mantener en comparación con otros estilos arquitectónicos como SOAP.
- **Seguridad:** REST puede ser vulnerable a ataques de seguridad como Cross-Site Scripting (XSS) y Cross-Site Request Forgery (CSRF) si no se implementan correctamente las medidas de seguridad.
- **Legibilidad:** REST puede ser difícil de leer y entender para los humanos cuando se trabaja con recursos y URIs complejos.

9.6 Ejemplo de uso de REST en JavaScript

```
// GET request
fetch("https://api.example.com/users")
  .then(response => response.json())
  .then(data => console.log(data));

// POST request
fetch("https://api.example.com/users", {
  method: "POST",
  headers: {
    "Content-Type": "application/json"
  },
  body: JSON.stringify({name: "John Doe", age: 30})
})
  .then(response => response.json())
  .then(data => console.log(data));
```


En este ejemplo, se muestra cómo realizar una solicitud GET y POST utilizando la Fetch API en JavaScript para interactuar con un servicio web RESTful que proporciona una API de usuarios.

9.7 Ejemplo del uso de una API REST con Thunder Client

```
GET https://api.example.com/users
```

```
POST https://api.example.com/users
```

```
{  
  "name": "John Doe",  
  "age": 30  
}
```

```
PUT https://api.example.com/users/1
```

```
{  
  "name": "Jane Doe",  
  "age": 25  
}
```

```
DELETE https://api.example.com/users/1
```

En este ejemplo, se muestra cómo utilizar Thunder Client, una extensión de Visual Studio Code, para realizar solicitudes GET, POST, PUT y DELETE a un servicio web RESTful que proporciona una API de usuarios.

9.8 Conclusión

10 GraphQL

10.1 ¿Qué es GraphQL?

GraphQL es un lenguaje de consulta y manipulación de datos desarrollado por Facebook en 2012 y lanzado como código abierto en 2015. GraphQL permite a los clientes solicitar solo los datos que necesitan y proporciona una forma más eficiente y flexible de interactuar con los servicios web.

10.2 Principios de GraphQL

- **Consulta única:** En GraphQL, se realiza una sola consulta para recuperar los datos necesarios en lugar de realizar múltiples solicitudes para diferentes recursos.
- **Tipado fuerte:** GraphQL utiliza un sistema de tipos fuertemente tipado para definir la estructura de los datos y garantizar la integridad de los datos.
- **Flexibilidad:** GraphQL permite a los clientes solicitar solo los campos necesarios y anidar las consultas para recuperar datos relacionados en una sola solicitud.
- **Documentación automática:** GraphQL proporciona una documentación automática de la API basada en el esquema de tipos definido, lo que facilita a los desarrolladores comprender y utilizar la API.

10.3 Ejemplo de GraphQL

```
query {  
  user(id: 1) {  
    id  
    name  
    email  
    posts {  
      id  
      title  
      content  
    }  
  }  
}
```

En este ejemplo, se muestra una consulta GraphQL para recuperar los datos de un usuario con el ID 1, incluido su nombre, correo electrónico y publicaciones relacionadas.

10.4 Ventajas de GraphQL

- **Consulta única:** GraphQL permite a los clientes solicitar solo los datos necesarios en una sola consulta, lo que reduce la cantidad de datos transferidos y mejora el rendimiento de la aplicación.
- **Tipado fuerte:** GraphQL utiliza un sistema de tipos fuertemente tipado para definir la estructura de los datos y garantizar la integridad de los datos.
- **Flexibilidad:** GraphQL permite a los clientes solicitar solo los campos necesarios y anidar las consultas para recuperar datos relacionados en una sola solicitud.
- **Documentación automática:** GraphQL proporciona una documentación automática de la API basada en el esquema de tipos definido, lo que facilita a los desarrolladores comprender y utilizar la API.

10.5 Desventajas de GraphQL

- **Curva de aprendizaje:** GraphQL puede tener una curva de aprendizaje empinada para los desarrolladores que no están familiarizados con el lenguaje de consulta y manipulación de datos.
- **Complejidad:** GraphQL puede ser más complejo de implementar y mantener en comparación con REST debido a la flexibilidad y la capacidad de anidar consultas.
- **Seguridad:** GraphQL puede ser vulnerable a ataques de seguridad como la inyección de consultas si no se implementan correctamente las medidas de seguridad.

10.6 Ejemplo de uso de GraphQL en JavaScript

```
import { ApolloClient, InMemoryCache, gql } from '@apollo/client';

const client = new ApolloClient({
  uri: 'https://api.example.com/graphql',
  cache: new InMemoryCache()
});

client
  .query({
    query: gql`
      query {
        user(id: 1) {
          id
          name
          email
          posts {
            id
```

```

        title
        content
      }
    }
  }
}
})
.then(result => console.log(result));

```

En este ejemplo, se muestra cómo realizar una consulta GraphQL utilizando Apollo Client en JavaScript para interactuar con un servicio web GraphQL que proporciona una API de usuarios y publicaciones.

10.7 Ejemplo de uso de GraphQL en React

```

import { useQuery, gql } from '@apollo/client';

const GET_USER = gql`
  query {
    user(id: 1) {
      id
      name
      email
      posts {
        id
        title
        content
      }
    }
  }
`;

function User() {
  const { loading, error, data } = useQuery(GET_USER);

  if (loading) return <p>Loading...</p>;
  if (error) return <p>Error :(</p>;

  return (
    <div>
      <p>{data.user.name}</p>
      <p>{data.user.email}</p>
      {data.user.posts.map(post => (
        <div key={post.id}>
          <h3>{post.title}</h3>

```

```
        <p>{post.content}</p>
      </div>
    )})
  </div>
);
}

export default User;
```

En este ejemplo, se muestra cómo utilizar la función **useQuery** de Apollo Client en React para realizar una consulta GraphQL y mostrar los datos de un usuario y sus publicaciones en un componente de React.

10.8 Conclusión

GraphQL es un lenguaje de consulta y manipulación de datos eficiente y flexible que permite a los clientes solicitar solo los datos necesarios en una sola consulta. GraphQL es una herramienta poderosa y versátil que puede ayudar a mejorar el rendimiento, la flexibilidad y la eficiencia de las aplicaciones web y los servicios web. GraphQL es una excelente alternativa a REST y puede ser una opción ideal para desarrollar aplicaciones web modernas y escalables.

11 WebSockets

11.1 ¿Qué son los WebSockets?

Los WebSockets son una tecnología de comunicación bidireccional y en tiempo real que permite a los clientes y servidores establecer una conexión persistente y enviar mensajes de forma asíncrona. Los WebSockets se basan en el protocolo TCP y proporcionan una forma eficiente y escalable de enviar datos en tiempo real entre el cliente y el servidor.

11.2 ¿Cómo funcionan los WebSockets?

Los WebSockets se inician mediante un proceso de “apretón de manos” entre el cliente y el servidor para establecer una conexión persistente. Una vez establecida la conexión, los clientes y servidores pueden enviar y recibir mensajes de forma asíncrona sin tener que recargar la página. Los WebSockets utilizan un protocolo binario ligero y eficiente para enviar y recibir datos en tiempo real.

11.3 Ejemplo de uso de WebSockets

```
// Client-side code
const socket = new WebSocket("wss://api.example.com");

socket.onopen = function() {
  console.log("WebSocket connection established");
};

socket.onmessage = function(event) {
  console.log("Message received: " + event.data);
};

socket.onclose = function() {
  console.log("WebSocket connection closed");
};

socket.send("Hello, server!");

// Server-side code
```

```

const WebSocket = require('ws');
const wss = new WebSocket.Server({ port: 8080 });

wss.on('connection', function connection(ws) {
  console.log("WebSocket connection established");

  ws.on('message', function incoming(message) {
    console.log("Message received: " + message);
    ws.send("Hello, client!");
  });

  ws.on('close', function close() {
    console.log("WebSocket connection closed");
  });
});

```

En este ejemplo, se muestra cómo establecer una conexión WebSocket entre el cliente y el servidor utilizando la API WebSocket en JavaScript. Una vez establecida la conexión, el cliente y el servidor pueden enviar y recibir mensajes de forma asíncrona en tiempo real.

11.4 Ventajas de WebSockets

- **Comunicación bidireccional:** Los WebSockets permiten una comunicación bidireccional y en tiempo real entre el cliente y el servidor.
- **Conexión persistente:** Los WebSockets establecen una conexión persistente que permite a los clientes y servidores enviar y recibir mensajes de forma asíncrona sin tener que recargar la página.
- **Eficiencia:** Los WebSockets utilizan un protocolo binario ligero y eficiente para enviar y recibir datos en tiempo real.
- **Escalabilidad:** Los WebSockets son altamente escalables y pueden manejar un gran número de conexiones simultáneas.

11.5 Desventajas de WebSockets

- **Seguridad:** Los WebSockets pueden ser vulnerables a ataques de seguridad como Cross-Site Scripting (XSS) y Cross-Site Request Forgery (CSRF) si no se implementan correctamente las medidas de seguridad.
- **Compatibilidad:** Los WebSockets pueden no ser compatibles con todos los navegadores y versiones de navegadores, lo que puede limitar su uso en ciertos entornos.
- **Complejidad:** Los WebSockets pueden ser más complejos de implementar y mantener en comparación con otras tecnologías de comunicación como AJAX y REST.

- **Rendimiento:** Los WebSockets pueden tener un mayor consumo de recursos en comparación con otras tecnologías de comunicación debido a la conexión persistente y la transferencia de datos en tiempo real.

11.6 Ejemplo de uso de WebSockets con Socket.IO

```
// Client-side code
const socket = io("https://api.example.com");

socket.on("connect", function() {
  console.log("WebSocket connection established");
});

socket.on("message", function(data) {
  console.log("Message received: " + data);
});

socket.emit("message", "Hello, server!");

// Server-side code

const io = require('socket.io')(8080);

io.on('connection', function(socket) {
  console.log("WebSocket connection established");

  socket.on('message', function(data) {
    console.log("Message received: " + data);
    socket.emit("message", "Hello, client!");
  });

  socket.on('disconnect', function() {
    console.log("WebSocket connection closed");
  });
});
```

En este ejemplo, se muestra cómo establecer una conexión WebSocket utilizando Socket.IO en JavaScript. Socket.IO es una biblioteca que facilita la comunicación en tiempo real entre el cliente y el servidor utilizando WebSockets y otros métodos de transporte compatibles.

11.7 Conclusión

Los WebSockets son una tecnología poderosa y versátil que permite una comunicación bidireccional y en tiempo real entre el cliente y el servidor. Los WebSockets son altamente

escalables, eficientes y eficaces para enviar y recibir datos en tiempo real. Los WebSockets son una excelente opción para aplicaciones web y servicios web que requieren una comunicación en tiempo real y una conexión persistente entre el cliente y el servidor.

12 Progressive Web Application (PWA)

12.1 ¿Qué es una Progressive Web Application (PWA)?

Una Progressive Web Application (PWA) es una aplicación web que utiliza tecnologías web modernas para proporcionar una experiencia de usuario similar a la de una aplicación nativa en dispositivos móviles y de escritorio. Las PWAs se caracterizan por ser rápidas, fiables y seguras, y pueden funcionar sin conexión a Internet utilizando la caché y otras tecnologías web.

12.2 Características de una PWA

- **Carga rápida:** Las PWAs se cargan rápidamente en el navegador y proporcionan una experiencia de usuario fluida y receptiva.
- **Funcionamiento sin conexión:** Las PWAs pueden funcionar sin conexión a Internet utilizando la caché y otras tecnologías web para almacenar datos localmente en el dispositivo.
- **Interactividad:** Las PWAs permiten a los usuarios interactuar con la aplicación web de forma similar a una aplicación nativa utilizando gestos táctiles, notificaciones push y otras funcionalidades avanzadas.
- **Seguridad:** Las PWAs utilizan HTTPS para garantizar la seguridad de los datos y proteger la privacidad de los usuarios.

12.3 Ventajas de una PWA

- **Experiencia de usuario mejorada:** Las PWAs proporcionan una experiencia de usuario similar a la de una aplicación nativa en dispositivos móviles y de escritorio.
- **Carga rápida:** Las PWAs se cargan rápidamente en el navegador y proporcionan una experiencia de usuario fluida y receptiva.
- **Funcionamiento sin conexión:** Las PWAs pueden funcionar sin conexión a Internet utilizando la caché y otras tecnologías web para almacenar datos localmente en el dispositivo.
- **Interactividad:** Las PWAs permiten a los usuarios interactuar con la aplicación web de forma similar a una aplicación nativa utilizando gestos táctiles, notificaciones push y otras funcionalidades avanzadas.

12.4 Desventajas de una PWA

- **Compatibilidad:** Las PWAs pueden no ser compatibles con todos los navegadores y versiones de navegadores, lo que puede limitar su uso en ciertos entornos.
- **SEO:** Las PWAs pueden ser menos accesibles para los motores de búsqueda debido a la carga dinámica de contenido y la falta de etiquetas meta y URL únicas para cada página.
- **Seguridad:** Las PWAs pueden ser vulnerables a ataques de seguridad como Cross-Site Scripting (XSS) y Cross-Site Request Forgery (CSRF) si no se implementan correctamente las medidas de seguridad.
- **Rendimiento:** Las PWAs pueden tener un mayor consumo de recursos en comparación con las aplicaciones web tradicionales debido a la carga dinámica de contenido y la transferencia de datos en tiempo real.

12.5 Ejemplo de una PWA con React

```
import React from 'react';
import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';
import Home from './components/Home';
import About from './components/About';

function App() {
  return (
    <Router>
      <Switch>
        <Route exact path="/" component={Home} />
        <Route path="/about" component={About} />
      </Switch>
    </Router>
  );
}

export default App;
```

En este ejemplo, se muestra cómo crear una PWA utilizando React y React Router. React es una biblioteca de JavaScript para construir interfaces de usuario y React Router es una biblioteca para manejar la navegación en una PWA.

12.6 Ejemplo de una PWA con Angular

```

import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { HomeComponent } from './home.component';

const routes: Routes = [
  { path: '', component: HomeComponent },
  { path: 'about', component: AboutComponent }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})

export class AppRoutingModule { }

```

En este ejemplo, se muestra cómo crear una PWA utilizando Angular y Angular Router. Angular es un framework de JavaScript para construir aplicaciones web y Angular Router es una biblioteca para manejar la navegación en una PWA.

12.7 Conclusión

Las Progressive Web Applications (PWAs) son aplicaciones web modernas que utilizan tecnologías web modernas para proporcionar una experiencia de usuario similar a la de una aplicación nativa en dispositivos móviles y de escritorio. Las PWAs se caracterizan por ser rápidas, fiables y seguras, y pueden funcionar sin conexión a Internet utilizando la caché y otras tecnologías web. Las PWAs son una excelente opción para desarrollar aplicaciones web modernas y escalables que requieren una carga rápida, una interactividad mejorada y un rendimiento optimizado.

13 Conclusiones

En resumen, las tecnologías web modernas como Ajax, JSON, REST, GraphQL, Web-Sockets, Single Page Applications (SPAs) y Progressive Web Applications (PWAs) han revolucionado la forma en que se desarrollan y se utilizan las aplicaciones web. Estas tecnologías permiten a los desarrolladores web crear aplicaciones interactivas, dinámicas y receptivas que proporcionan una experiencia de usuario mejorada y una mayor eficiencia en el rendimiento. Con el uso de estas tecnologías, es posible crear aplicaciones web modernas y escalables que se adaptan a las necesidades y expectativas de los usuarios en la era digital actual.

Part II

Laboratorios

14 Laboratorio Ajax

En este laboratorio vamos a aprender a hacer peticiones AJAX con JavaScript puro. Para ello vamos a hacer una aplicación que nos permita buscar pokemon y ver sus detalles.

14.1 Requisitos

Para este laboratorio vamos a utilizar tecnologías como html, css y javascript. Además, vamos a necesitar un navegador web y un editor de código, se recomienda leer acerca de peticiones con ajax antes de empezar este laboratorio.

14.2 Instrucciones

1. Creamos un archivo **index.html** y agregamos el siguiente código:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Pokemon</title>
</head>
<body>
  <h1>Pokemon</h1>
  <input type="text" id="search" placeholder="Buscar pokemon">
  <button id="search-button">Buscar</button>
  <div id="pokemon"></div>
  <script src="app.js"></script>
</body>
</html>
```

En esta sección hemos creado la estructura básica de nuestro proyecto. Hemos agregado un título, un input para buscar pokemon, un botón para buscar pokemon, un div para mostrar la información del pokemon y un script para nuestro archivo **app.js**.

2. Creamos un archivo **app.js** y agregamos el siguiente código:

```

document.getElementById('search-button').addEventListener('click', () => {
  const search = document.getElementById('search').value;
  fetch(`https://pokeapi.co/api/v2/pokemon/${search}`)
    .then(response => response.json())
    .then(data => {
      const pokemon = document.getElementById('pokemon');
      pokemon.innerHTML = `
        <h2>${data.name}</h2>
        
        <p>Altura: ${data.height}</p>
        <p>Peso: ${data.weight}</p>
      `;
    })
    .catch(error => {
      console.error(error);
    });
});

```

En esta sección hemos agregado un evento al botón de buscar pokemon. Cuando se haga click en el botón, se va a hacer una petición a la API de [PokeAPI](#). Si la petición es exitosa, se va a mostrar la información del pokemon en el div **pokemon**. Si la petición falla, se va a mostrar un mensaje de error en la consola.

Para más información sobre la API de [PokeAPI](#), se puede consultar la [documentación](#).

5. Mediante el plugin de Live Server, abrimos el archivo **index.html** en el navegador y probamos la aplicación.
6. Vamos a mejorar la presentación de nuestro proyecto agregando estilos CSS. Creamos un archivo **styles.css** y agregamos el siguiente código:

```

body {
  font-family: Arial, sans-serif;
  margin: 0;
  padding: 0;
  display: flex;
  flex-direction: column;
  align-items: center;
}

h1 {
  margin: 1rem 0;
}

input {
  padding: 0.5rem;
  margin: 0.5rem 0;
}

```



```

button {
  padding: 0.5rem;
  margin: 0.5rem 0;
  background-color: #007bff;
  color: white;
  border: none;
  cursor: pointer;
}

#pokemon {
  display: flex;
  flex-direction: column;
  align-items: center;
  margin: 1rem 0;
}

img {
  width: 100px;
  height: 100px;
  margin: 0.5rem 0;
}

p {
  margin: 0.5rem 0;
}

```

En esta sección hemos agregado estilos CSS para mejorar la presentación de nuestro proyecto, como la fuente, el color de fondo, el color del texto, el tamaño de la letra, el espaciado, el color del botón, el cursor, el tamaño de la imagen, el margen de la imagen, el margen del párrafo, etc.

7. Agregamos el archivo **styles.css** al archivo **index.html**:

```

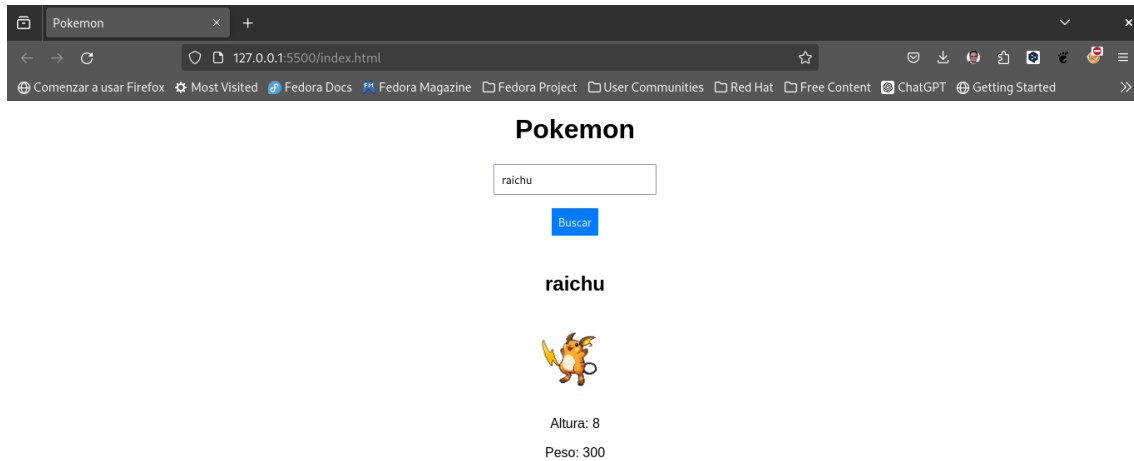
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Pokemon</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <h1>Pokemon</h1>
  <input type="text" id="search" placeholder="Buscar pokemon">
  <button id="search-button">Buscar</button>
  <div id="pokemon"></div>
  <script src="app.js"></script>

```

```
</body>  
</html>
```

En esta sección hemos agregado el archivo **styles.css** al archivo **index.html**.

8. Mediante el plugin de Live Server, abrimos el archivo **index.html** en el navegador y probamos la aplicación.



15 Deploy

Se sugiere hacer el deploy de la aplicación en Github Pages, Vercel, Netlify, etc.

16 Reto

Ahora es necesario mejorar la aplicación. Se sugiere agregar las siguientes funcionalidades:

1. Agregar un mensaje de error si el pokemon no existe.
2. Agregar un mensaje de error si la petición falla.
3. Agregar un spinner mientras se hace la petición.
4. Agregar un botón para limpiar el div **pokemon**.
5. Agregar un botón para buscar un pokemon aleatorio.
6. Agregar un botón para buscar un pokemon por id.
7. Agregar un botón para buscar un pokemon por nombre.
8. Agregar un botón para buscar un pokemon por tipo.
9. Agregar un botón para buscar un pokemon por habilidad.

16.1 Conclusión

Felicidades, has completado el laboratorio de Ajax. Has aprendido a hacer peticiones AJAX con JavaScript puro. Ahora puedes hacer aplicaciones web que hagan peticiones a APIs de terceros.

17 ¿Dónde utilizamos Ajax?

Utilizamos Ajax en aplicaciones web que necesitan hacer peticiones a APIs de terceros. En este laboratorio hemos hecho una aplicación que busca pokemon en la API de [PokeAPI](#).

Cuando hacemos una petición AJAX, podemos obtener información de la API de terceros y mostrarla en nuestra aplicación web. Por ejemplo, podemos obtener información de un pokemon y mostrarla en nuestra aplicación web.

18 Laboratorio de Railway



Figure 18.1: Railway

18.1 Objetivo

El objetivo de este laboratorio es implementar una aplicación sencilla de flask que permita mostrar un mensaje “**Hello World from Railway whit Flask!**” en la ruta raíz de la aplicación.

18.2 Instrucciones

1. Crear una cuenta en [Railway](#).
2. Crear un proyecto en Flask.

18.3 Desarrollo

1. Iniciamos con la creación de la aplicación Flask.

Para ello vamos a crear un directorio llamado flask, vamos a seguir las buenas prácticas y vamos a crear un entorno virtual para instalar las dependencias de la aplicación.

```
python3 -m venv venv
source venv/bin/activate
```

2. Instalamos las dependencias de la aplicación.

```
pip install flask gunicorn
```

3. Creamos el archivo **main.py** con el siguiente contenido.

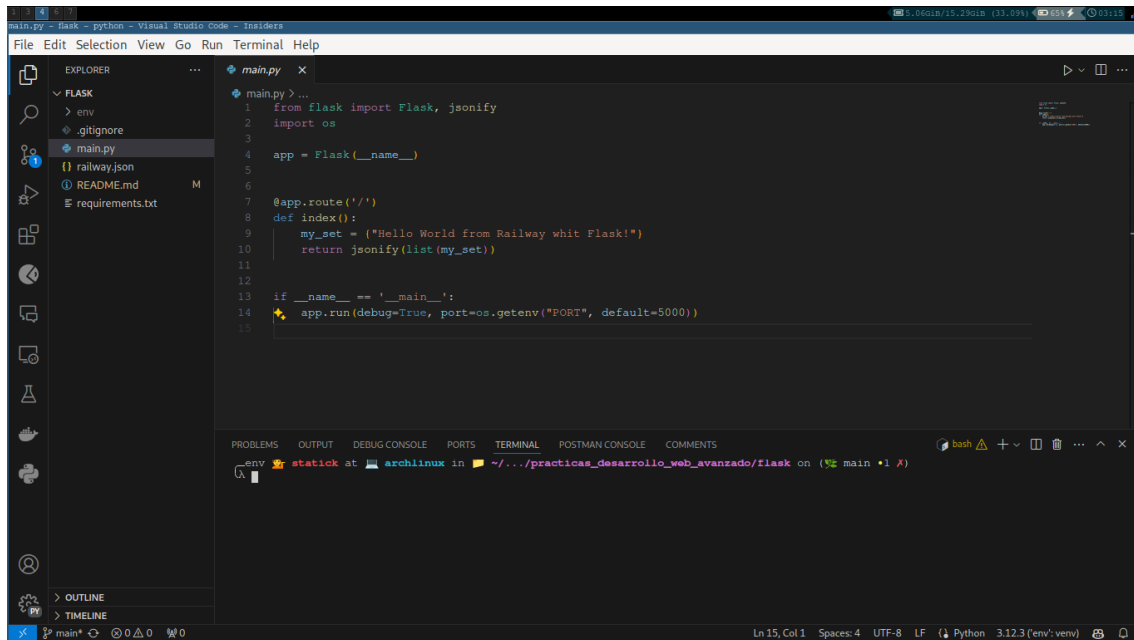


Figure 18.2: Aplicación Flask

```
from flask import Flask, jsonify
import os

app = Flask(__name__)

@app.route('/')
def index():
    my_set = {"Hello World from Railway whit Flask!"}
    return jsonify(list(my_set))

if __name__ == '__main__':
    app.run(debug=True, port=os.getenv("PORT", default=5000))
```

4. Creamos el archivo **railway.json** con el siguiente contenido.

```
{
  "$schema": "https://railway.app/railway.schema.json",
  "build": {
    "builder": "NIXPACKS"
  },
  "deploy": {
    "startCommand": "gunicorn main:app",
```

```
    "restartPolicyType": "ON_FAILURE",
    "restartPolicyMaxRetries": 10
  }
}
```

5. Creamos el archivo **.gitignore**, para ello vamos a utilizar el servicio de [gitignore.io](https://www.gitignore.io).

Puede copiar el archivo **.gitignore** de la siguiente url <https://www.toptal.com/developers/gitignore/api/git,python,flask>

O utilizar el siguiente comando.

```
curl -o .gitignore https://www.toptal.com/developers/gitignore/api/git,python,flask
```

6. Creamos el archivo Readme.md con el siguiente contenido.

```
---
title: Flask
description: Un popular framework minimalista para servidores en Python
tags:
  - python
  - flask
---

# Ejemplo de Python Flask

Esta es una aplicación de [Flask] (https://flask.palletsprojects.com/en/1.1.x/) que sirve

## Características

- Python
- Flask

## Cómo usar

- Instala los requisitos de Python `pip install -r requirements.txt`
- Inicia el servidor para desarrollo `python3 main.py`
```

Tip

Se recomienda crear el archivo **requirements.txt** con el siguiente contenido.

```
click==7.1.2
Flask==1.1.2
gunicorn==20.0.4
itsdangerous==1.1.0
Jinja2==2.11.3
jsonify==0.5
MarkupSafe==1.1.1
```



```
setuptools==70.0.0
Werkzeug==1.0.1
```

7. Inicializamos el repositorio de git.

```
git init
```

8. Realizamos el primer commit.

```
git add .
git commit -m "Initial commit"
```

9. Creamos un repositorio en GitHub.

10. Agregamos el repositorio remoto.

```
git remote add origin <url>
```

11. Realizamos el primer push.

```
git push -u origin master
```

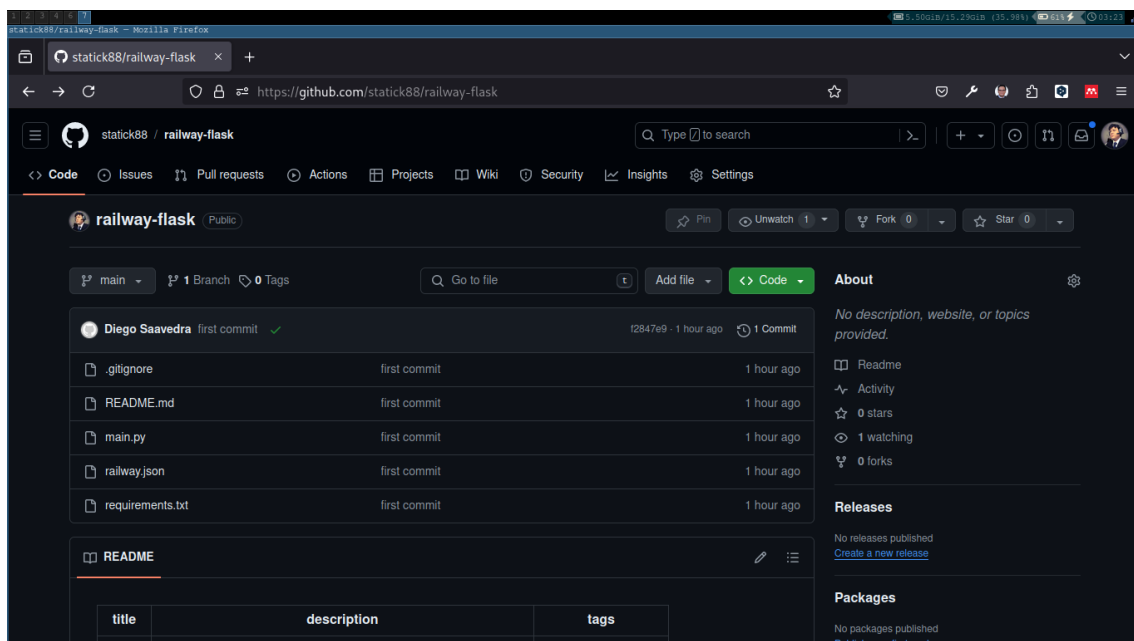


Figure 18.3: Repositorio en GitHub

12. Creamos una nueva aplicación en Railway.

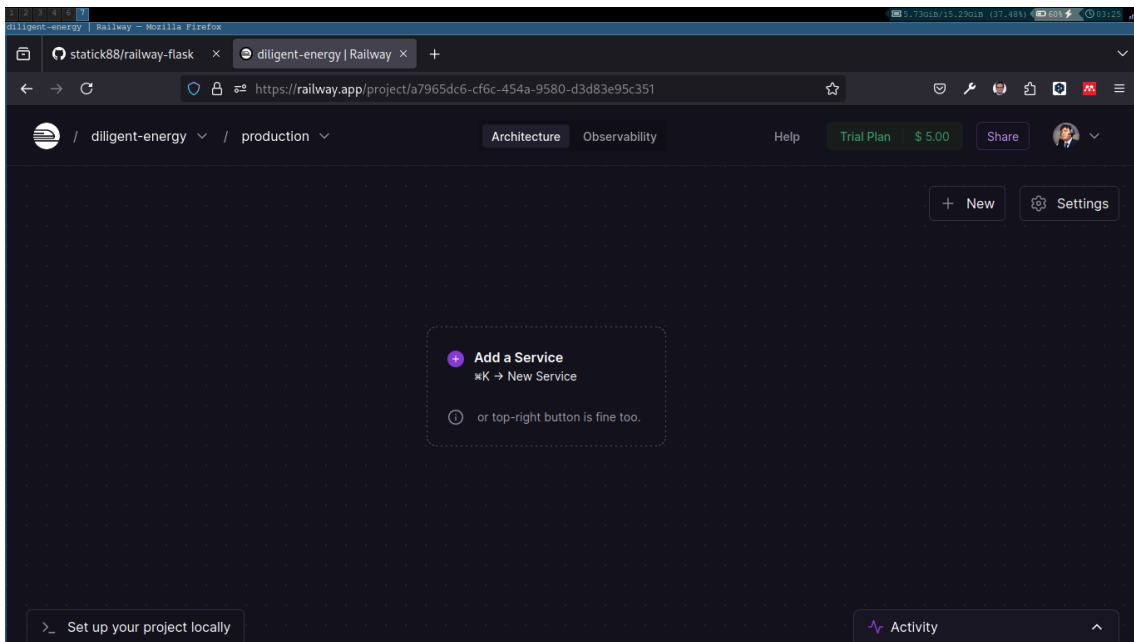


Figure 18.4: Aplicación en Railway

13. Vinculamos el repositorio de GitHub con Railway.

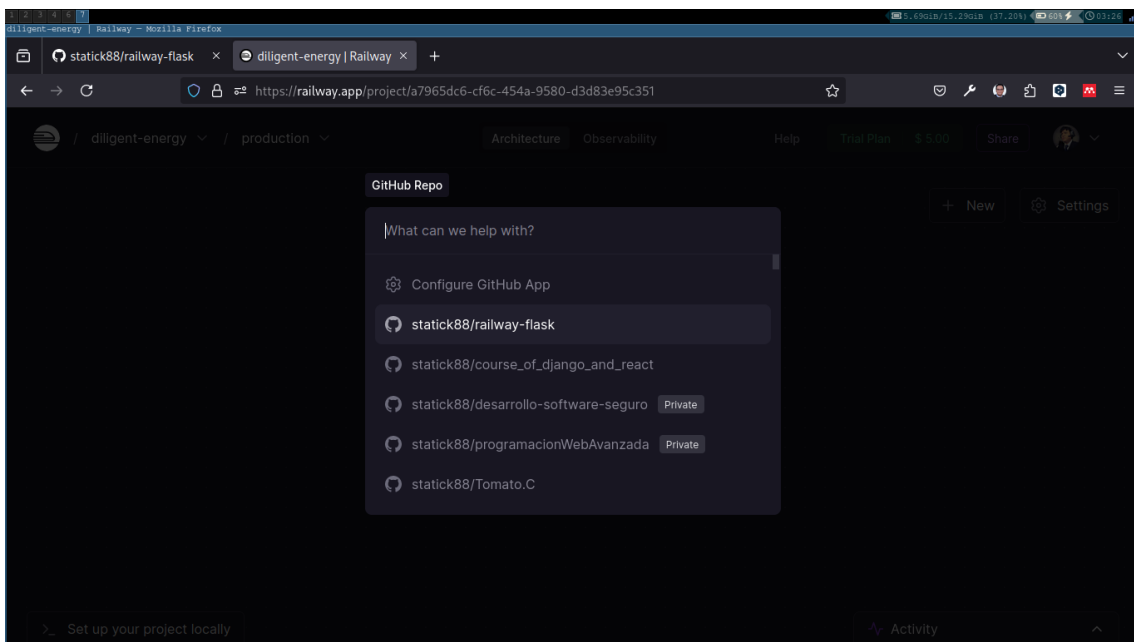


Figure 18.5: Vinculación de repositorio

14. Desplegamos la aplicación.

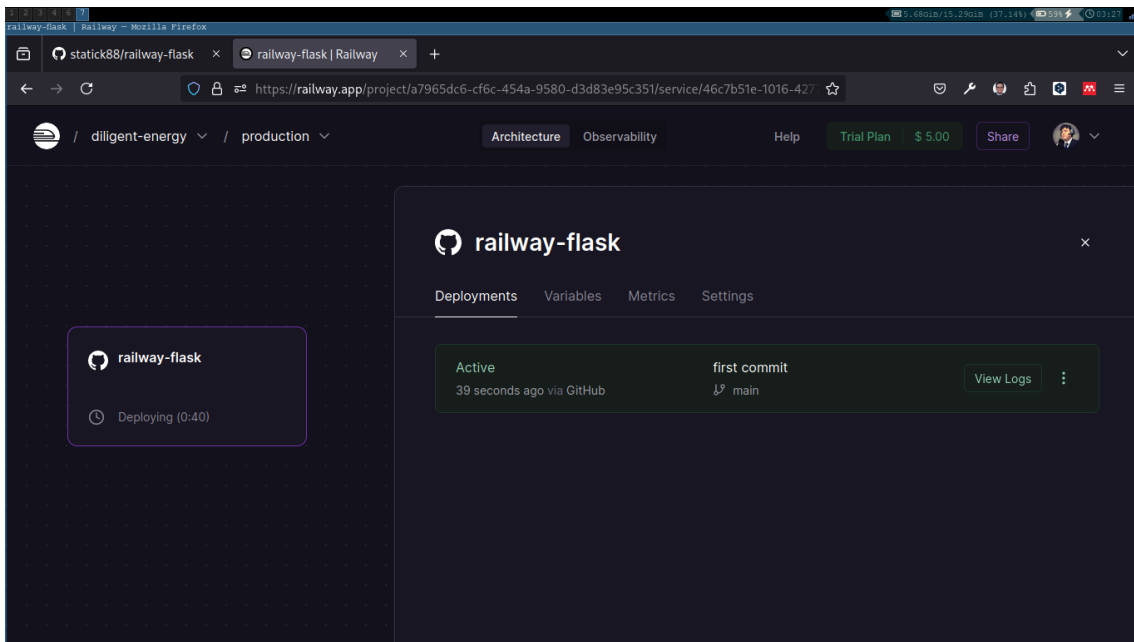


Figure 18.6: Despliegue de la aplicación

15. Verificamos que la aplicación se despliegue correctamente.

💡 Tip

En la sección Settings creamos una url para poder visualizar la aplicación.

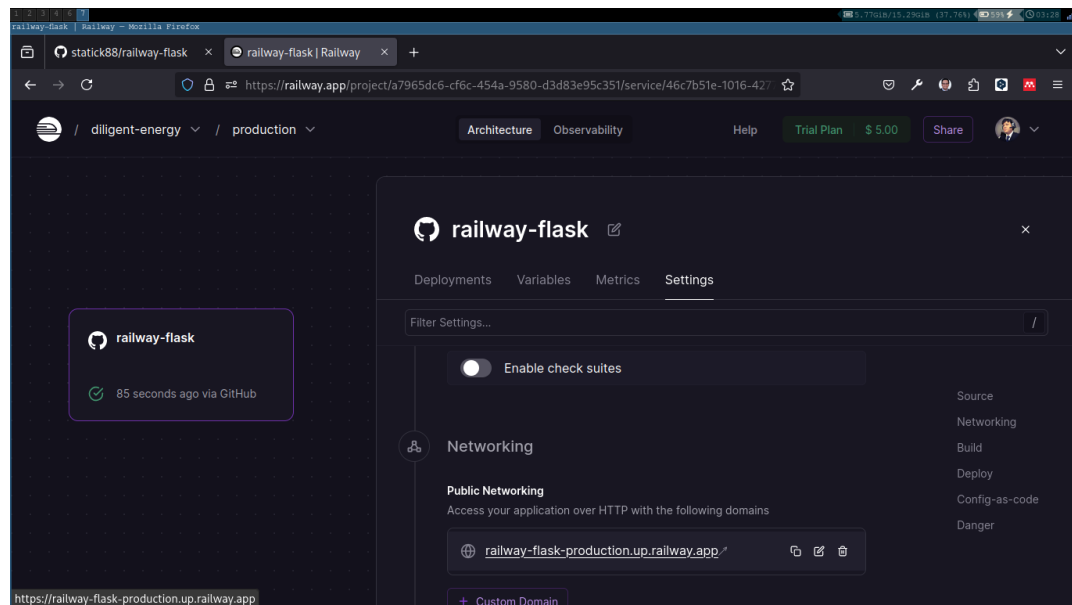


Figure 18.7: URL de la aplicación

16. Verificamos que la aplicación muestre el mensaje **“Hello World from Railway whit Flask!”** en la ruta raíz de la aplicación.

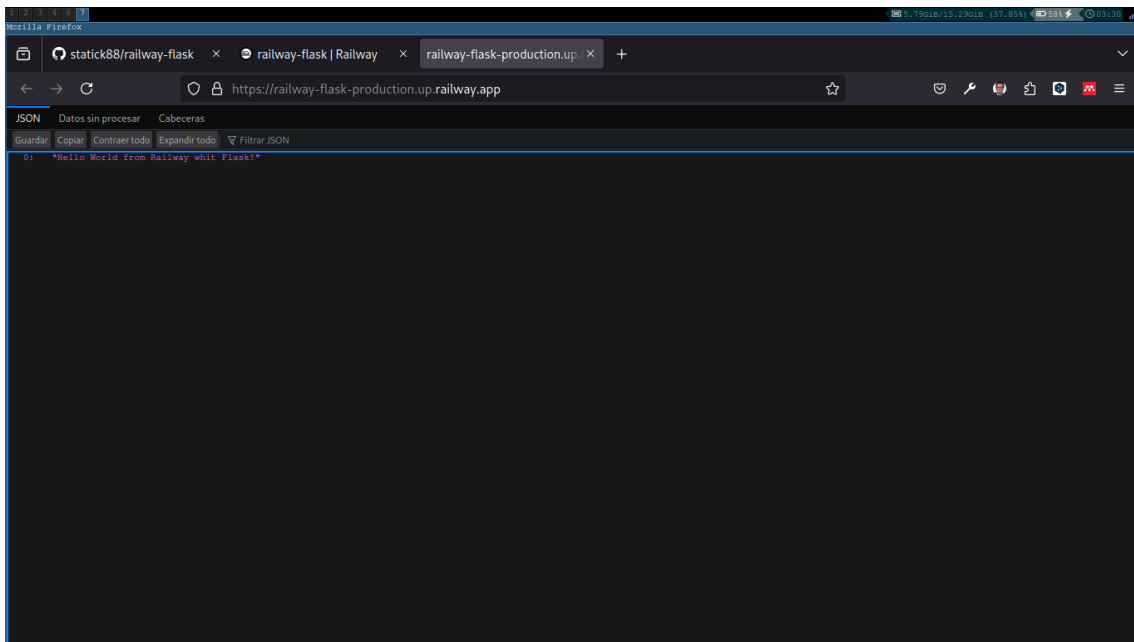


Figure 18.8: Mensaje de la aplicación

Con ello hemos terminado el laboratorio.

18.4 Conclusiones

Hemos aprendido a crear una aplicación en Flask y a desplegarla en Railway.

19 Ejercicio

1. Modificar el mensaje que se muestra en la ruta raíz de la aplicación “**Hello World my name is: your name**”.
2. Desplegar la aplicación en Railway.
3. Verificar que la aplicación se despliegue correctamente.

Ver código

```
from flask import Flask, jsonify
import os

app = Flask(__name__)

@app.route('/')
def index():
    my_set = {"Hello World my name is: Diego Saavedra"}
    return jsonify(list(my_set))

if __name__ == '__main__':
    app.run(debug=True, port=os.getenv("PORT", default=5000))
```

Realizamos los cambios en el repositorio de GitHub y desplegamos la aplicación en Railway.

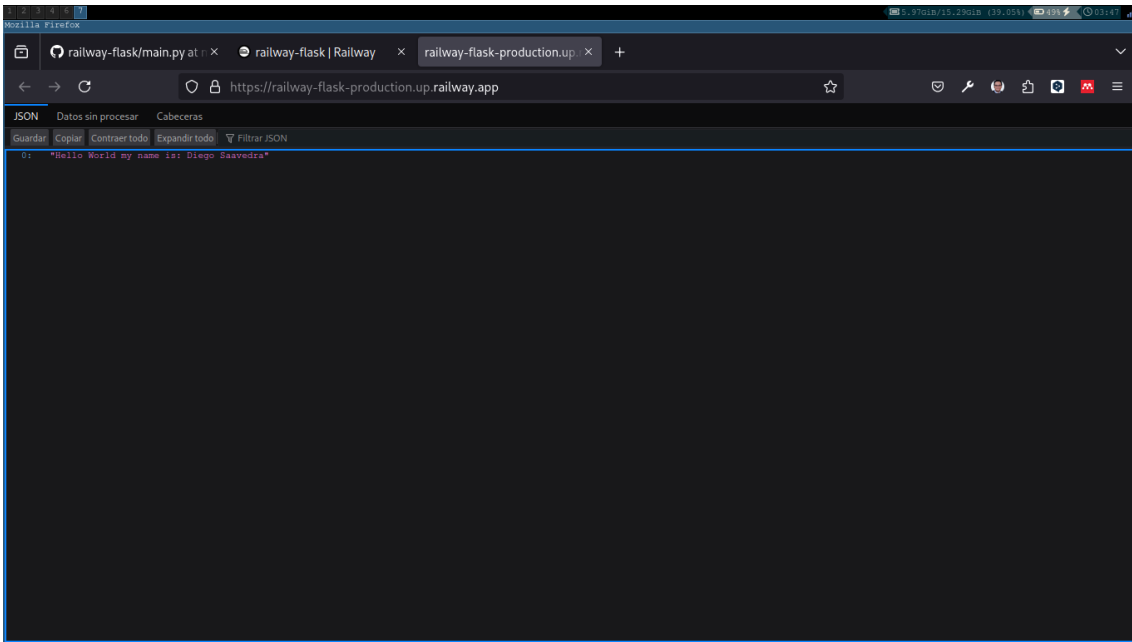


Figure 19.1: Mensaje modificado

20 Recursos

- [Flask](#)
- [Railway](#)
- [gitignore.io](#)
- [GitHub](#)
- [Repositorio en Railway](#)

21 Laboratorio Jinga 2



Figure 21.1: Django

21.1 Objetivo

El objetivo de este laboratorio es construir una aplicación web que permita la creación de un inventario de productos utilizando django y jinja2.

21.2 Requisitos

Para este laboratorio necesitarás tener instalado lo siguiente:

- Python 3.12
- Django 4.2

21.3 Desarrollo

Para este laboratorio necesitarás crear una aplicación web que permita la creación de un inventario de productos. Para ello, deberás seguir los siguientes pasos:

1. Crea un proyecto de django llamado **jinga2_lab2**.
2. Creación de un entorno virtual para el proyecto, lo puedes hacer con el siguiente comando:


```
python3 -m venv venv
```

3. Activa el entorno virtual con el siguiente comando:

```
source venv/bin/activate
```

En entornos Windows, el comando anterior se cambia por el siguiente:

```
venv\Scripts\activate
```

4. Instala django en tu entorno virtual con el siguiente comando:

```
pip install django==4.2
```

Creamos un proyecto llamado **inventario** con el siguiente comando:

```
django-admin startproject inventario .
```

5. Crea una aplicación llamada **productos** con el siguiente comando:

```
python manage.py startapp productos
```

6. Agregamos la aplicación **productos** al archivo **settings.py** del proyecto:

```
INSTALLED_APPS = [  
    ...  
    'productos',  
]
```

7. Crea un modelo llamado **Producto** en la aplicación **productos** con los siguientes campos:

- Nombre del producto
- Precio del producto
- Cantidad del producto

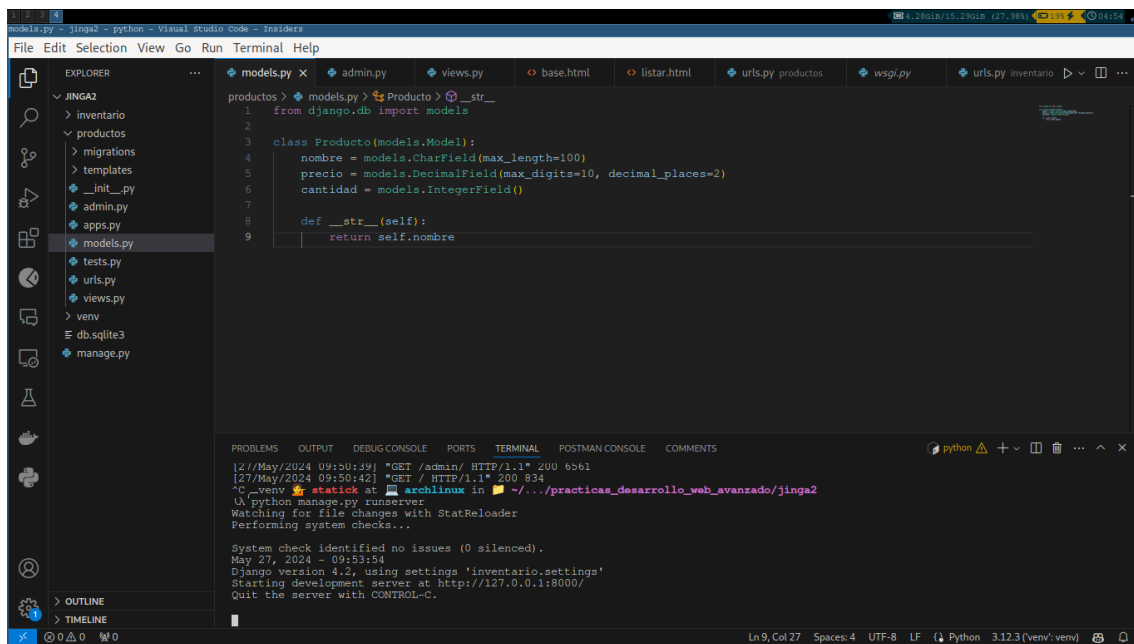


Figure 21.2: Producto

```
from django.db import models

class Producto(models.Model):
    nombre = models.CharField(max_length=100)
    precio = models.DecimalField(max_digits=10, decimal_places=2)
    cantidad = models.IntegerField()

    def __str__(self):
        return self.nombre
```

8. Registramos el modelo **Producto** en el archivo **admin.py** de la aplicación **productos**:

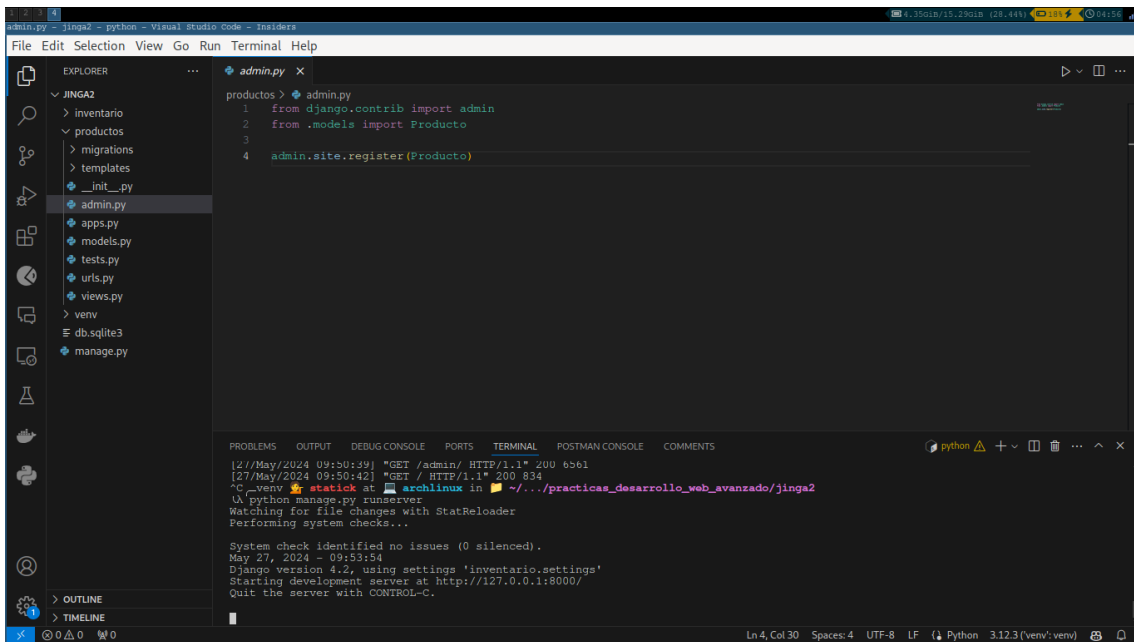


Figure 21.3: Admin

```

from django.contrib import admin
from .models import Producto

admin.site.register(Producto)

```

9. Crea una vista llamada `listar_productos` en la aplicación **productos** que muestre todos los productos registrados en el sistema:

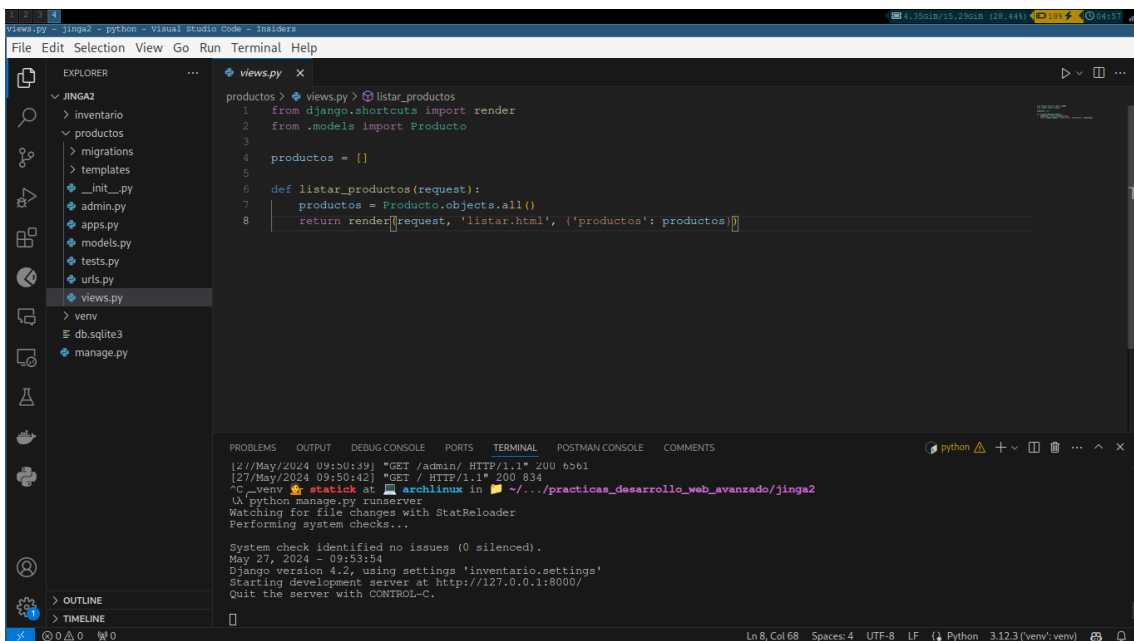


Figure 21.4: Listar

```

from django.shortcuts import render
from .models import Producto

productos = []

def listar_productos(request):
    productos = Producto.objects.all()
    return render(request, 'listar.html', {'productos': productos})

```

💡 Tip

Para poder utilizar el sistema de herencia de plantillas denominado Jinja2, es necesario seguir las buenas prácticas para ello vamos a crear un archivo **base.html** en la carpeta **templates** de la aplicación **productos**.

10. Crea el archivo **base.html** en la carpeta **templates** de la aplicación **productos** con el siguiente contenido:

```

products > templates > base.html > html > body
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>
7     {% block title %}
8     Inventario
9     {% endblock %}
10  </title>
11  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet" int
12  </head>
13  <body>
14    <div class="container">
15      {% block content %}
16      {% endblock %}
17    </div>
18    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-
19  </body>
20 </html>

```

Figure 21.5: Base

```

<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>
    {% block title %}
    Inventario

```

```

        {% endblock %}
    </title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css"
</head>
<body>
    <div class="container">
        {% block content %}
        {% endblock %}
    </div>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"
</body>
</html>

```

11. Crea una plantilla llamada **listar.html** en la carpeta **productos/templates** con el siguiente contenido:

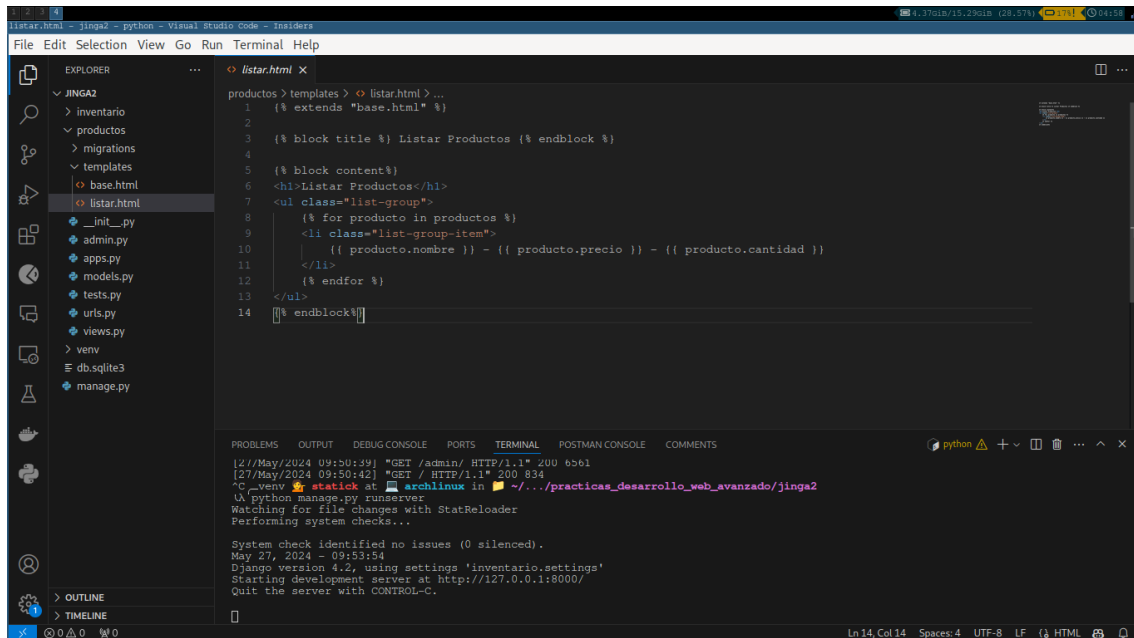


Figure 21.6: Listar

```

{% extends "base.html" %}

{% block title %} Listar Productos {% endblock %}

{% block content%}
<h1>Listar Productos</h1>
<ul class="list-group">
    {% for producto in productos %}
    <li class="list-group-item">
        {{ producto.nombre }} - {{ producto.precio }} - {{ producto.cantidad }}
    </li>

```

```
{% endfor %}  
</ul>  
{% endblock%}
```

12. Crea una URL llamada `listar_productos` en el archivo `urls.py` de la aplicación `productos`:

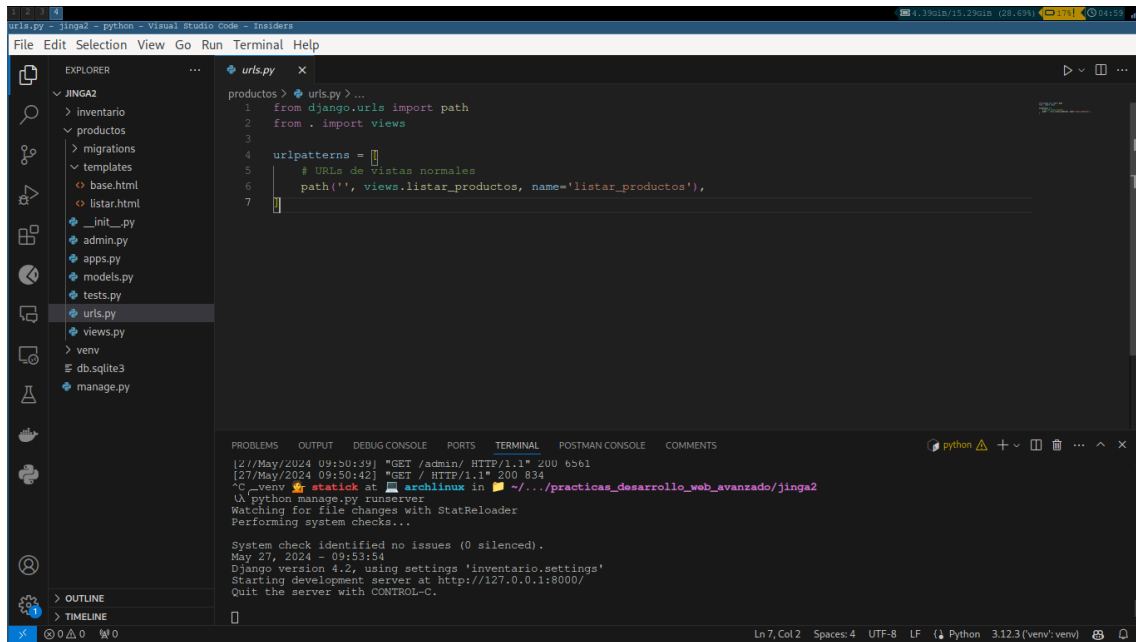


Figure 21.7: URL

```
from django.urls import path  
from . import views  
  
urlpatterns = [  
    # URLs de vistas normales  
    path('', views.listar_productos, name='listar_productos'),  
]
```

13. Agrega la URL de la aplicación `productos` al archivo `urls.py` del proyecto:

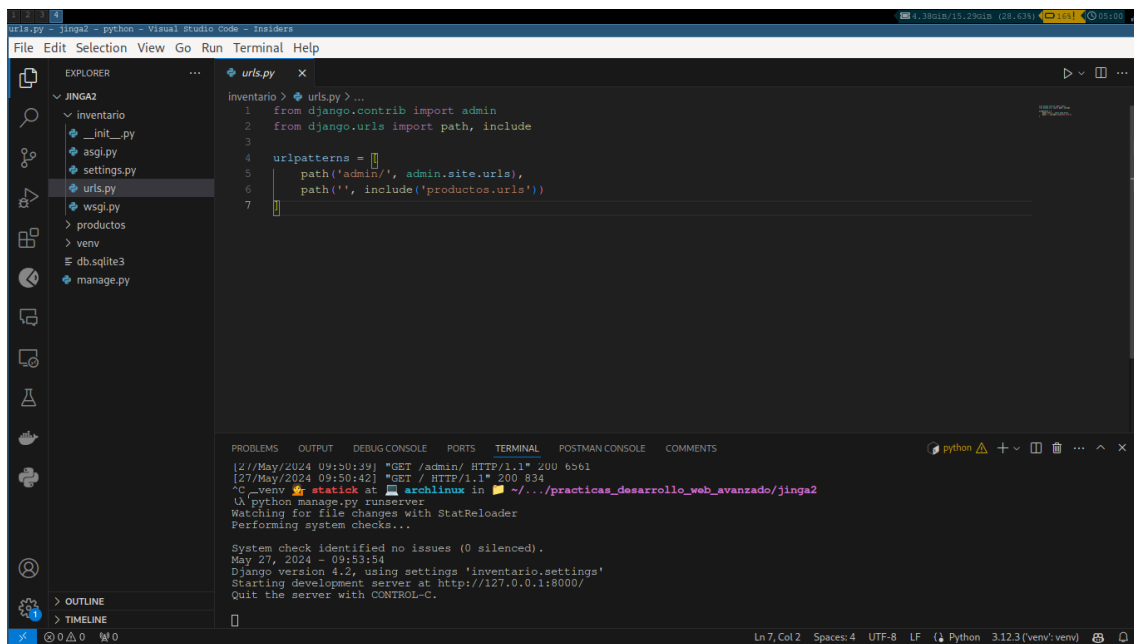


Figure 21.8: URL

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('productos.urls'))
]
```

14. Realicemos las migraciones de la aplicación **productos** con el siguiente comando:

```
python manage.py makemigrations productos
python manage.py migrate
```

15. Crea un superusuario con el siguiente comando:

```
python manage.py createsuperuser
```

16. Ejecuta el servidor con el siguiente comando:

```
python manage.py runserver
```

17. Abre tu navegador y accede a la dirección `http://localhost:8000/admin/` e inicia sesión con el superusuario que creaste anteriormente.

18. Registra algunos productos en el sistema.

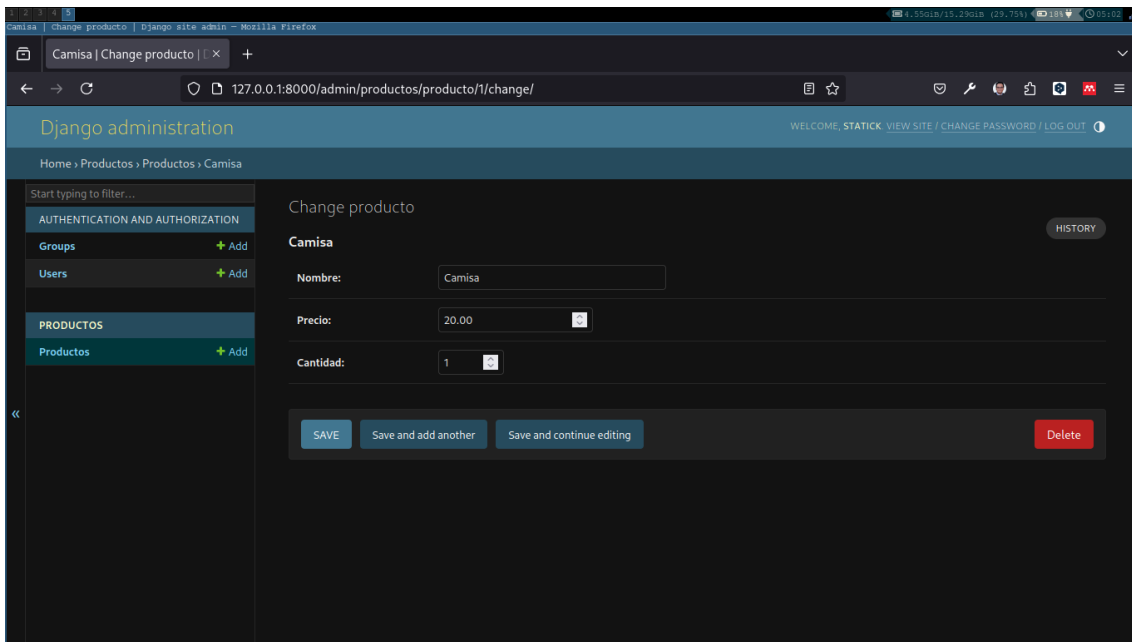


Figure 21.9: Productos

19. Accede a la dirección <http://localhost:8000/> y verifica que los productos registrados se muestren en la página.

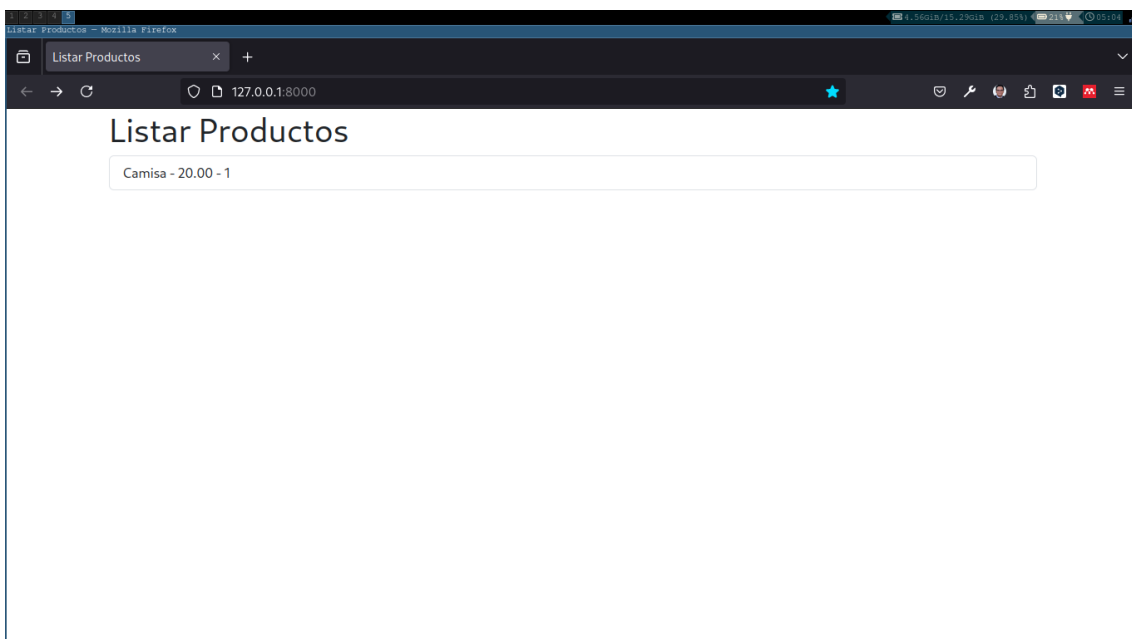


Figure 21.10: Listar

20. Inicializamos el repositorio de git con el siguiente comando:

```
git init
```

Se sugiere crear el archivo `.gitignore` para evitar subir archivos innecesarios al repositorio.


```
curl https://www.toptal.com/developers/gitignore/api/django > .gitignore
```

21. Realiza un commit con los cambios realizados:

```
git add .  
git commit -m "Initial commit"
```

22. Crea un repositorio en GitHub y sube los cambios realizados:

```
git remote add origin <url-repositorio>  
git branch -M main  
git push -u origin main
```

22 Resultado

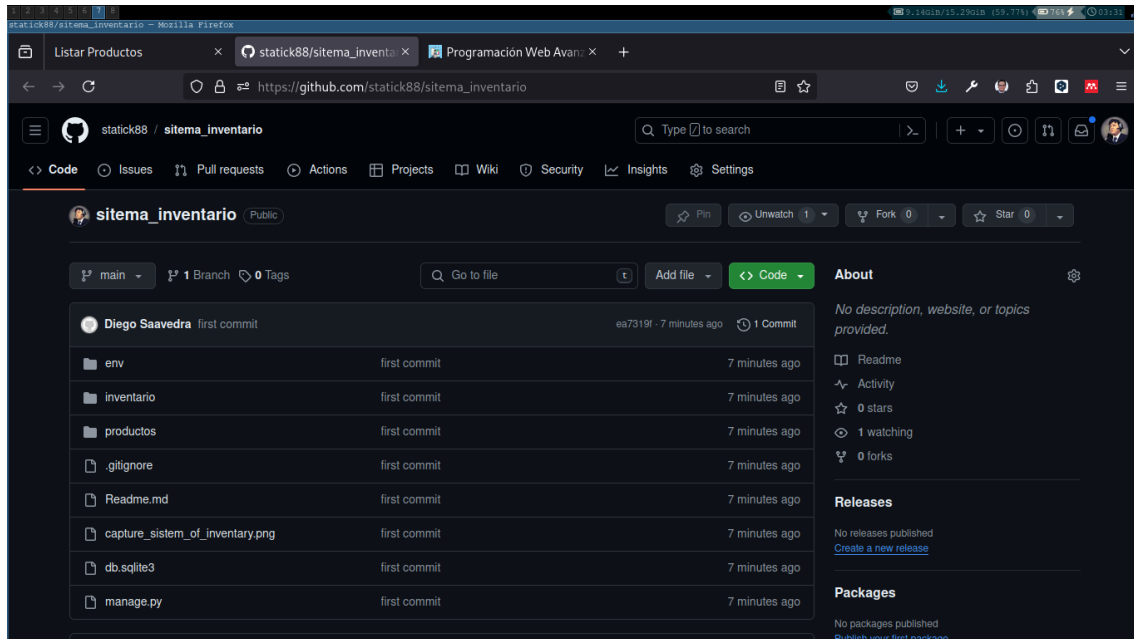


Figure 22.1: Listar

Al finalizar este laboratorio, deberás haber creado una aplicación web que permita la creación de un inventario de productos utilizando django y jinja2.

23 Ejercicios

1. Crear las vistas de crear, editar y eliminar productos en la aplicación **productos**.
2. Crear las plantillas de crear, editar y eliminar productos en la carpeta **productos/templates**.
3. Crear las URLs de crear, editar y eliminar productos en el archivo **urls.py** de la aplicación **productos**.
4. Crear los enlaces de crear, editar y eliminar productos en la plantilla **listar.html**.
5. Realizar las migraciones de la aplicación **productos**.
6. Crear un superusuario.
7. Registrar algunos productos en el sistema.
8. Verificar que los productos registrados se muestren en la página.
9. Inicializar el repositorio de git.
10. Realizar un commit con los cambios realizados.
11. Crear un repositorio en GitHub y subir los cambios realizados.

24 Conclusiones

En este laboratorio, aprendimos a crear una aplicación web que permita la creación de un inventario de productos utilizando django y jinja2. Para ello, creamos un modelo llamado **Producto** en la aplicación **productos** con los campos **nombre**, **precio** y **cantidad**. Luego, creamos una vista llamada **listar_productos** que muestra todos los productos registrados en el sistema. Finalmente, creamos una plantilla llamada **listar.html** que muestra los productos en una lista.

25 Referencias

- [Django](#)
- [Jinja2](#)
- [Python](#)
- [Git](#)
- [Repositorio en GitHub](#)

26 Laboratorio de Ruby on Rails



Figure 26.1: Ruby on Rails

En este laboratorio aprenderás a crear una aplicación web simple utilizando Ruby on Rails. Rails es un framework de desarrollo web escrito en el lenguaje de programación Ruby que sigue el patrón de diseño Modelo-Vista-Controlador (MVC). Aprenderás a generar un modelo, un controlador, rutas y vistas, y a realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en una base de datos.

26.1 ¿Qué es Ruby on Rails?

Ruby on Rails es un framework de desarrollo web escrito en el lenguaje de programación Ruby. Fue creado por David Heinemeier Hansson y lanzado en 2004. Rails es un framework de código abierto que sigue el patrón de diseño Modelo-Vista-Controlador (MVC) y se basa en la filosofía de la convención sobre configuración.

Rails es conocido por su simplicidad y facilidad de uso, lo que lo convierte en una excelente opción para desarrollar aplicaciones web de manera rápida y eficiente. Rails proporciona una serie de herramientas y bibliotecas que facilitan el desarrollo de aplicaciones web, como la generación automática de código, la integración con bases de datos y la gestión de sesiones y cookies.

26.2 Instalación de Ruby on Rails

Para instalar Ruby on Rails en tu sistema, primero necesitas tener Ruby instalado. Puedes instalar Ruby siguiendo las instrucciones en el sitio web oficial de Ruby: <https://www.ruby-lang.org/en/documentation/installation/>

Una vez que tengas Ruby instalado, puedes instalar Rails utilizando el siguiente comando en tu terminal:

```
gem install rails
```

Este comando instalará la última versión de Rails en tu sistema. Una vez que la instalación haya finalizado, puedes verificar que Rails se haya instalado correctamente ejecutando el siguiente comando:

```
rails --version
```

Este comando debería mostrar la versión de Rails que has instalado en tu sistema.

27 Creación de una nueva aplicación Rails

27.1 Parte 1: Configuración del Proyecto

27.1.1 Creación del Proyecto:

Crea un nuevo proyecto de Rails llamado `inventory_app`:

```
rails new inventory_app
```

Esto generará la estructura básica de un proyecto de Rails.

27.1.2 Navega a la Carpeta del Proyecto:

Entra en el directorio del proyecto recién creado:

```
cd inventory_app
```

27.1.3 Inicia el Servidor:

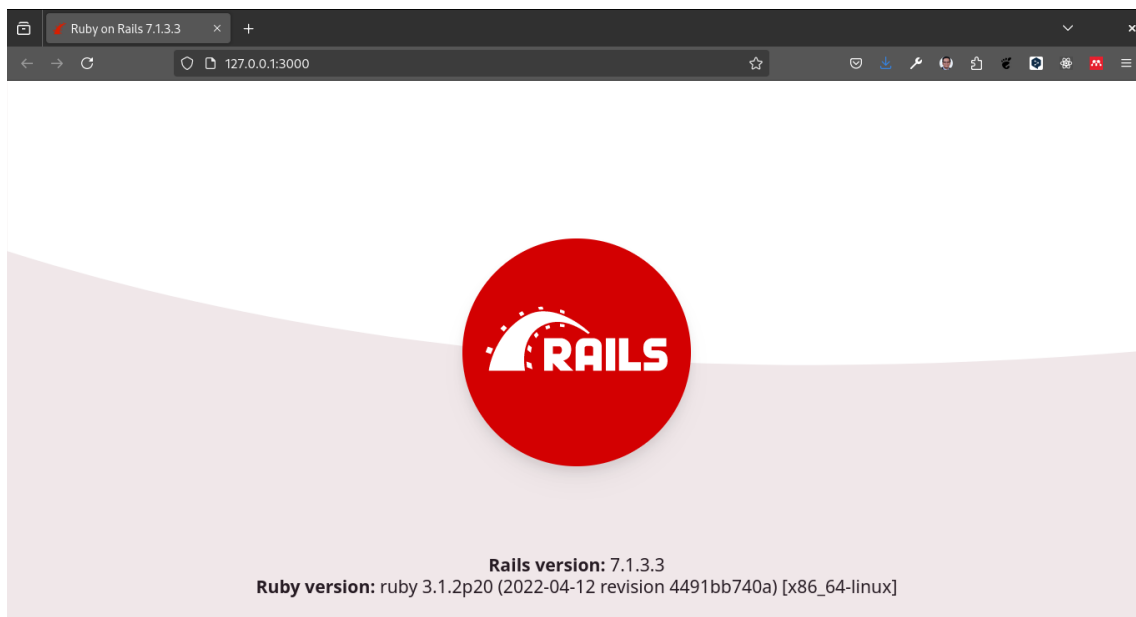


Figure 27.1: Servidor de Rails corriendo

Para asegurarte de que todo está funcionando correctamente, inicia el servidor:

```
rails server
```

Luego, abre tu navegador y ve a <http://localhost:3000>. Deberías ver la página de bienvenida de Rails.

27.2 Parte 2: Generación del Modelo

27.2.1 Conceptos Básicos

En Rails, un modelo representa una tabla en la base de datos. Vamos a crear un modelo llamado Item para manejar nuestro inventario.

27.2.2 Generación del Modelo:

Genera el modelo Item con atributos name, description, y quantity:

```
rails generate model Item name:string description:text quantity:integer
```

Este comando creará un archivo de migración para la tabla items y el modelo Item.

27.2.3 Ejecutar la Migración:

Aplica la migración para crear la tabla items en la base de datos:

```
rails db:migrate
```

27.3 Parte 3: Generación del Controlador

27.3.1 Conceptos Básicos

Un controlador en Rails recibe las solicitudes del usuario, interactúa con el modelo y la base de datos, y luego renderiza una vista.

27.3.2 Generación del Controlador:

Genera el controlador Items con las acciones necesarias (index, show, new, create, edit, update, destroy):

```
rails generate controller Items index show new edit
```

Este comando generará un controlador con las acciones especificadas y las vistas correspondientes.

27.4 Parte 4: Rutas

27.4.1 Conceptos Básicos

Las rutas en Rails configuran las URL de la aplicación y especifican qué controlador y acción deben manejar una solicitud determinada.

27.4.2 Configurar las Rutas:

Abre el archivo `config/routes.rb` y configura las rutas para los items:

```
Rails.application.routes.draw do
  resources :items
  root 'items#index'
end
```

27.5 Parte 5: Vistas

27.5.1 Conceptos Básicos

Las vistas en Rails son plantillas que muestran la información al usuario. Utilizan HTML con código Ruby embebido (ERB).

27.5.2 Crear las Vistas:

Abre el archivo `app/views/items/index.html.erb` y agrega el siguiente código:

```
<h1>Inventory</h1>

<%= link_to 'New Item', new_item_path %>

<ul>
  <% @items.each do |item| %>
    <li>
      <%= link_to item.name, item_path(item) %>
      <%= link_to 'Edit', edit_item_path(item) %>
      <%= link_to 'Destroy', item, method: :delete, data: { confirm: 'Are you sure?' } %>
    </li>
  <% end %>
</ul>
```

27.5.3 Controlador y Acción index:

Abre el archivo `app/controllers/items_controller.rb` y agrega la acción `index`:

```
class ItemsController < ApplicationController
  def index
    @items = Item.all
  end
end
```

27.6 Parte 6: CRUD - Create

27.6.1 Conceptos Básicos

Para crear un nuevo ítem, necesitamos una vista para el formulario y una acción en el controlador para manejar la creación.

27.6.2 Formulario para Crear un Nuevo Ítem:

Abre el archivo `app/views/items/new.html.erb` y agrega el siguiente código:

```
<h1>New Item</h1>

<%= form_with model: @item, local: true do |form| %>
  <div>
    <%= form.label :name %>
    <%= form.text_field :name %>
  </div>
  <div>
    <%= form.label :description %>
    <%= form.text_area :description %>
  </div>
  <div>
    <%= form.label :quantity %>
    <%= form.number_field :quantity %>
  </div>
  <div>
    <%= form.submit %>
  </div>
<% end %>
```

27.6.3 Acción `new` y `create` en el Controlador:

Abre `app/controllers/items_controller.rb` y agrega las acciones `new` y `create`:

```

class ItemsController < ApplicationController
  def index
    @items = Item.all
  end

  def show
    @item = Item.find(params[:id])
  end

  def new
    @item = Item.new
  end

  def create
    @item = Item.new(item_params)
    if @item.save
      redirect_to @item
    else
      render :new
    end
  end

  private

  def item_params
    params.require(:item).permit(:name, :description, :quantity)
  end
end

```

27.7 Parte 7: CRUD - Read

27.7.1 Conceptos Básicos

Para leer los detalles de un ítem, necesitamos una vista show.

27.7.2 Vista show:

Abre `app/views/items/show.html.erb` y agrega el siguiente código:

```

<h1><%= @item.name %></h1>
<p><%= @item.description %></p>
<p>Quantity: <%= @item.quantity %></p>

<%= link_to 'Edit', edit_item_path(@item) %> |
<%= link_to 'Back', items_path %>

```

27.8 Parte 8: CRUD - Update

27.8.1 Conceptos Básicos

Para actualizar un ítem, necesitamos una vista para el formulario de edición y una acción en el controlador para manejar la actualización.

27.8.2 Formulario para Editar un Ítem:

Abre `app/views/items/edit.html.erb` y agrega el siguiente código:

```
<h1>Edit Item</h1>

<%= form_with model: @item, local: true do |form| %>
<div>
<%= form.label :name %>
<%= form.text_field :name %>
</div>
<div>
<%= form.label :description %>
<%= form.text_area :description %>
</div>
<div>
<%= form.label :quantity %>
<%= form.number_field :quantity %>
</div>
<div>
<%= form.submit %>
</div>
<% end %>
```

27.8.3 Acción edit y update en el Controlador:

Abre `app/controllers/items_controller.rb` y agrega las acciones edit y update:

```
class ItemsController < ApplicationController
  def index
    @items = Item.all
  end

  def show
    @item = Item.find(params[:id])
  end

  def new
    @item = Item.new
  end
end
```

```

end

def create
  @item = Item.new(item_params)
  if @item.save
    redirect_to @item
  else
    render :new
  end
end

def edit
  @item = Item.find(params[:id])
end

def update
  @item = Item.find(params[:id])
  if @item.update(item_params)
    redirect_to @item
  else
    render :edit
  end
end

private

def item_params
  params.require(:item).permit(:name, :description, :quantity)
end
end

```

27.8.4 Parte 9: CRUD - Delete

27.8.5 Conceptos Básicos

Para eliminar un ítem, necesitamos una acción en el controlador que maneje la eliminación.

27.8.6 Acción destroy en el Controlador:

Abre `app/controllers/items_controller.rb` y agrega la acción `destroy`:

```

class ItemsController < ApplicationController
  def index
    @items = Item.all
  end
end

```

```

def show
  @item = Item.find(params[:id])
end

def new
  @item = Item.new
end

def create
  @item = Item.new(item_params)
  if @item.save
    redirect_to @item
  else
    render :new
  end
end

def edit
  @item = Item.find(params[:id])
end

def update
  @item = Item.find(params[:id])
  if @item.update(item_params)
    redirect_to @item
  else
    render :edit
  end
end

def destroy
  @item = Item.find(params[:id])
  @item.destroy
  redirect_to items_path
end

private

def item_params
  params.require(:item).permit(:name, :description, :quantity)
end
end

```

27.9 Parte 10: Pruebas y Verificación

27.9.1 Conceptos Básicos

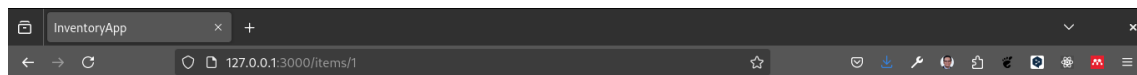
Es importante probar que nuestra aplicación funciona correctamente y que todas las operaciones CRUD están funcionando como se espera.

27.9.2 Verificar la Funcionalidad:

Inicia el servidor de Rails nuevamente (si no está ya en ejecución):

```
rails server
```

27.10 Pruebas Manuales:



Camisa

this is a camisa

Quantity: 3

[Edit](#) | [Back](#)

Abre tu navegador y navega a <http://localhost:3000/items>. Deberías ver la lista de ítems.

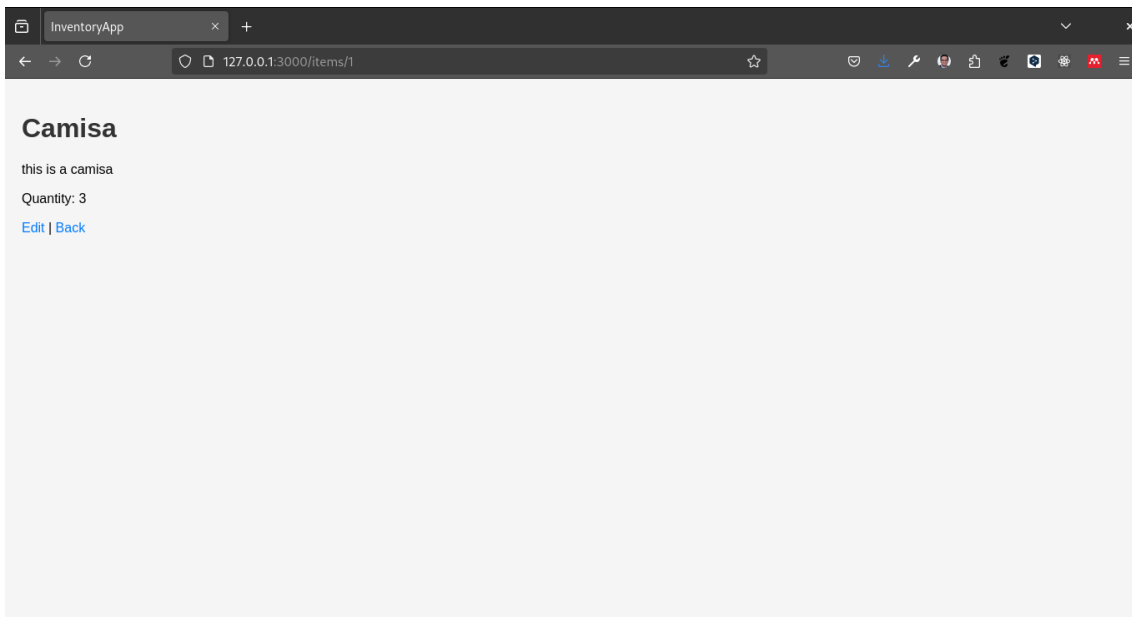
Crear un Nuevo Ítem: Haz clic en “New Item” y completa el formulario para crear un nuevo ítem.

Ver un Ítem: Haz clic en el nombre de un ítem para ver sus detalles.

Editar un Ítem: Haz clic en “Edit” junto a un ítem para editarlo.

Eliminar un Ítem: Haz clic en “Destroy” junto a un ítem para eliminarlo.

27.11 Parte 11: Estilo y Mejoras



27.11.1 Conceptos Básicos

Podemos mejorar la apariencia y la usabilidad de nuestra aplicación añadiendo algo de estilo con CSS.

27.11.2 Agregar Estilo:

Abre el archivo `app/assets/stylesheets/application.css` y agrega algunas reglas CSS básicas para mejorar el diseño:

```
body {  
  font-family: Arial, sans-serif;  
  margin: 0;  
  padding: 20px;  
  background-color: #f5f5f5;  
}  
  
h1 {  
  color: #333;  
}  
  
ul {  
  list-style: none;  
  padding: 0;  
}
```

```
li {
  padding: 10px;
  background: #fff;
  margin-bottom: 10px;
  border-radius: 5px;
  box-shadow: 0 1px 3px rgba(0,0,0,0.1);
}

a {
  color: #007bff;
  text-decoration: none;
}

a:hover {
  text-decoration: underline;
}

form {
  background: #fff;
  padding: 20px;
  border-radius: 5px;
  box-shadow: 0 1px 3px rgba(0,0,0,0.1);
}

div {
  margin-bottom: 10px;
}
```

Puede analizar el código de esta aplicación de ejemplo para obtener una idea de cómo implementar la funcionalidad requerida en el siguiente enlace: [Código de la Aplicación de Inventario](#).

28 Actividad

Crema una aplicación web simple utilizando Ruby on Rails para gestionar un inventario. La aplicación debe permitir al usuario realizar las siguientes operaciones:

- Ver la lista de ítems en el inventario.
- Crear un nuevo ítem en el inventario.
- Ver los detalles de un ítem en el inventario.
- Editar un ítem en el inventario.
- Eliminar un ítem del inventario.

Utiliza los conceptos y pasos descritos anteriormente para crear la aplicación. Asegúrate de probar la funcionalidad de la aplicación y de mejorar el diseño con un poco de estilo CSS.

Ver Solución

```
# app/controllers/items_controller.rb

class ItemsController < ApplicationController
  def index
    @items = Item.all
  end

  def show
    @item = Item.find(params[:id])
  end

  def new
    @item = Item.new
  end

  def create
    @item = Item.new(item_params)
    if @item.save
      redirect_to @item
    else
      render :new
    end
  end

  def edit
    @item = Item.find(params[:id])
  end
end
```

```

end

def update
  @item = Item.find(params[:id])
  if @item.update(item_params)
    redirect_to @item
  else
    render :edit
  end
end

def destroy
  @item = Item.find(params[:id])
  @item.destroy
  redirect_to items_path
end

private

def item_params
  params.require(:item).permit(:name, :description, :quantity)
end
end

```

```

# app/views/items/index.html.erb

<h1>Inventory</h1>

<%= link_to 'New Item', new_item_path %>

<ul>
  <% @items.each do |item| %>
    <li>
      <%= link_to item.name, item_path(item) %>
      <%= link_to 'Edit', edit_item_path(item) %>
      <%= link_to 'Destroy', item_path(item), data: { turbo_method: :delete } %>
    </li>
  <% end %>
</ul>

```

```

# app/views/items/new.html.erb

<h1>New Item</h1>

<%= form_with model: @item, local: true do |form| %>
  <div>
    <%= form.label :name %>
    <%= form.text_field :name %>

```

```

</div>
<div>
  <%= form.label :description %>
  <%= form.text_area :description %>
</div>
<div>
  <%= form.label :quantity %>
  <%= form.number_field :quantity %>
</div>
<div>
  <%= form.submit %>
</div>
<% end %>

```

```

# app/views/items/show.html.erb

<h1><%= @item.name %></h1>

<p><%= @item.description %></p>

<p>Quantity: <%= @item.quantity %></p>

<%= link_to 'Edit', edit_item_path(@item) %> |

<%= link_to 'Back', items_path %>

```

```

# app/views/items/edit.html.erb

<h1>Edit Item</h1>

<%= form_with model: @item, local: true do |form| %>
  <div>
    <%= form.label :name %>
    <%= form.text_field :name %>
  </div>
  <div>
    <%= form.label :description %>
    <%= form.text_area :description %>
  </div>
  <div>
    <%= form.label :quantity %>
    <%= form.number_field :quantity %>
  </div>
  <div>
    <%= form.submit %>
  </div>
<% end %>

```

```
# app/assets/stylesheets/application.css

body {
  font-family: Arial, sans-serif;
  margin: 0;
  padding: 20px;
  background-color: #f5f5f5;
}

h1 {
  color: #333;
}

ul {
  list-style: none;
  padding: 0;
}

li {
  padding: 10px;
  background: #fff;
  margin-bottom: 10px;
  border-radius: 5px;
  box-shadow: 0 1px 3px rgba(0,0,0,0.1);
}

a {
  color: #007bff;
  text-decoration: none;
}

a:hover {
  text-decoration: underline;
}

form {
  background: #fff;
  padding: 20px;
  border-radius: 5px;
  box-shadow: 0 1px 3px rgba(0,0,0,0.1);
}

div {
  margin-bottom: 10px;
}
```

29 Conclusión

En esta introducción a Ruby on Rails, hemos cubierto los conceptos básicos de Rails y hemos creado una aplicación web simple para gestionar un inventario. Rails es un framework poderoso y flexible que facilita el desarrollo de aplicaciones web. Con Rails, puedes crear aplicaciones web de manera rápida y eficiente, siguiendo las mejores prácticas de desarrollo web.

30 Laboratorio Firebase Cloud Functions REST API CRUD

30.1 Descripción

En este laboratorio vamos a aprender a utilizar las Cloud Functions de Firebase para crear una REST API que nos permita realizar operaciones CRUD sobre una colección de documentos en Firestore.

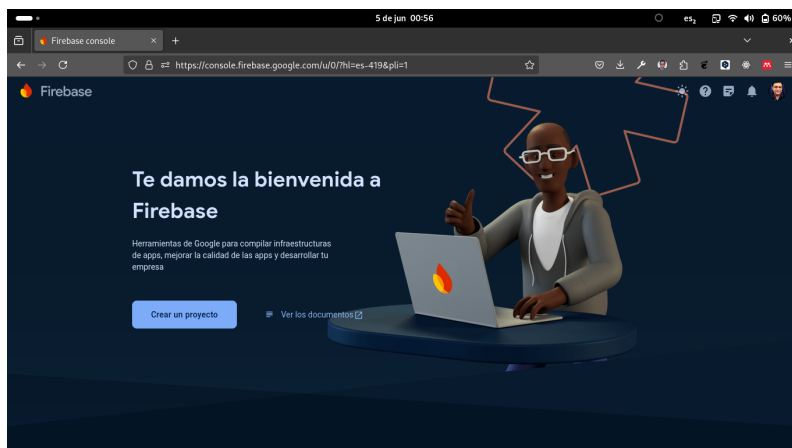
30.2 Requisitos

- Cuenta en Firebase
- Cuenta en GitHub
- Node.js
- npm

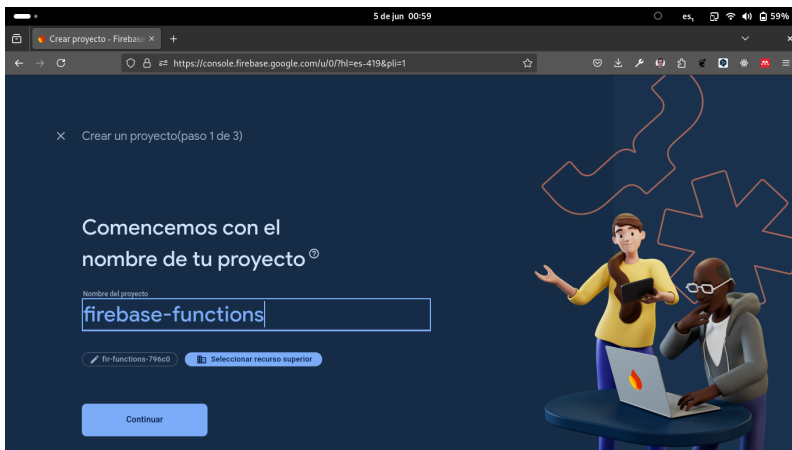
30.3 Desarrollo

30.3.1 Paso 1: Crear un proyecto en Firebase

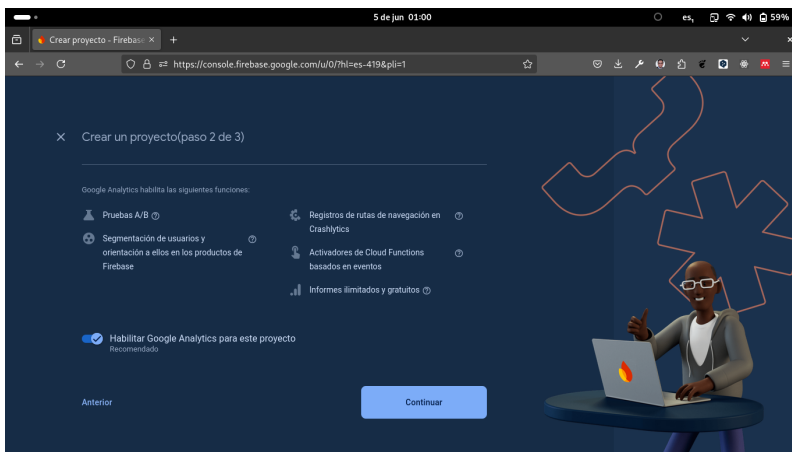
1. Accede a la [Consola de Firebase](#).



2. Haz clic en el botón **Crear un proyecto**.



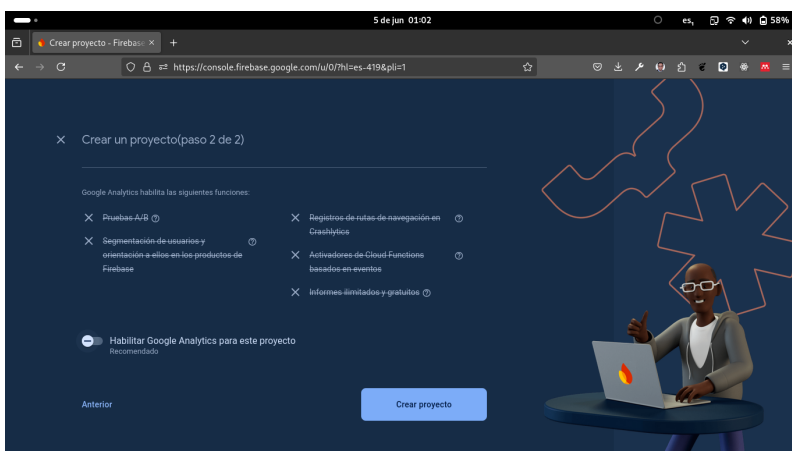
3. Ingresa el nombre de tu proyecto y haz clic en el botón **Continuar**.



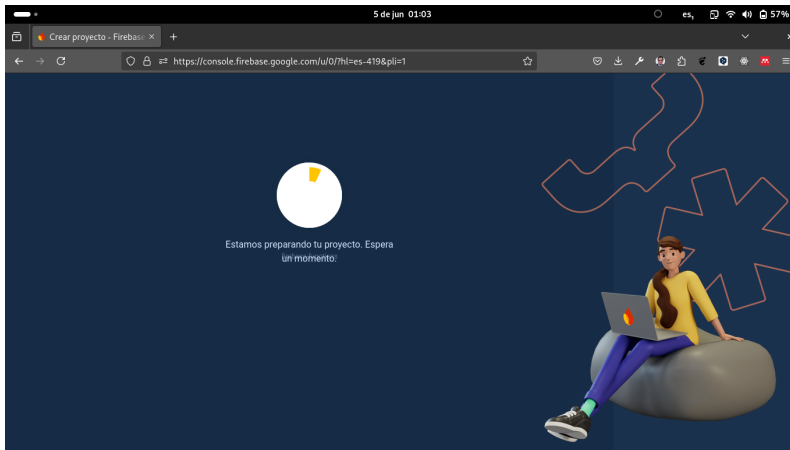
Tip

En este proyecto no es necesario activar Google Analytics.

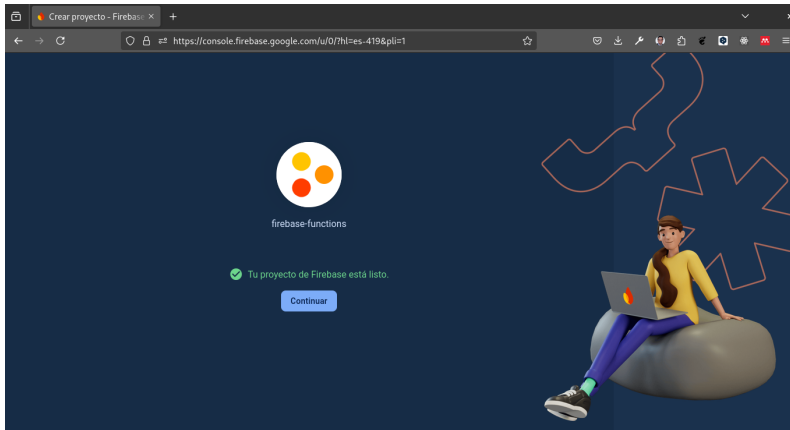
4. Aceptamos las condiciones y hacemos clic en el botón **Crear proyecto**.



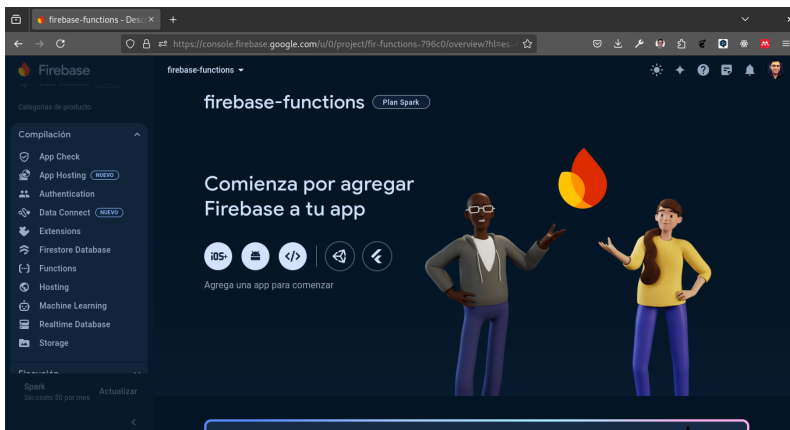
Esperamos un momento hasta que se cree el proyecto.



Finalmente debe aparecernos de la siguiente manera.



Con estos pasos podemos ver al lado izquierdo algunas de las opciones que nos ofrece Firebase.



Dentro de las opciones más destacadas tenemos:

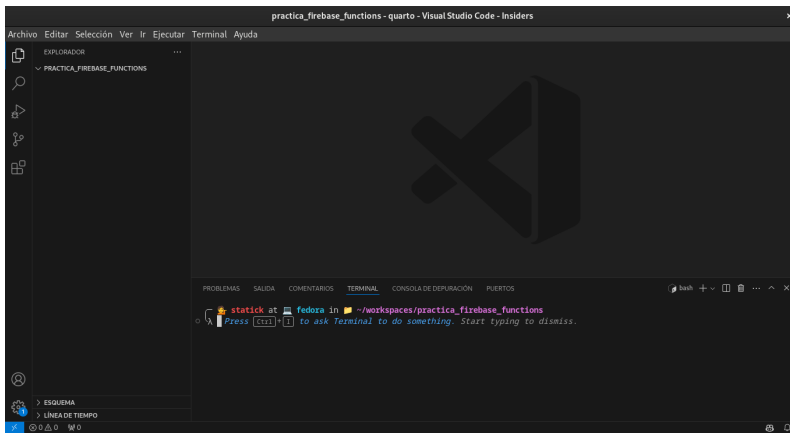
- **Authentication:** Nos permite autenticar a los usuarios de nuestra aplicación.
- **Database:** Nos permite almacenar datos en tiempo real.
- **Firestore:** Nos permite almacenar datos en la nube.
- **Storage:** Nos permite almacenar archivos en la nube.
- **Functions:** Nos permite ejecutar código en la nube.

- **Hosting:** Nos permite alojar nuestra aplicación web.

De las opciones antes mencionadas utilizaremos Firestore y Functions.

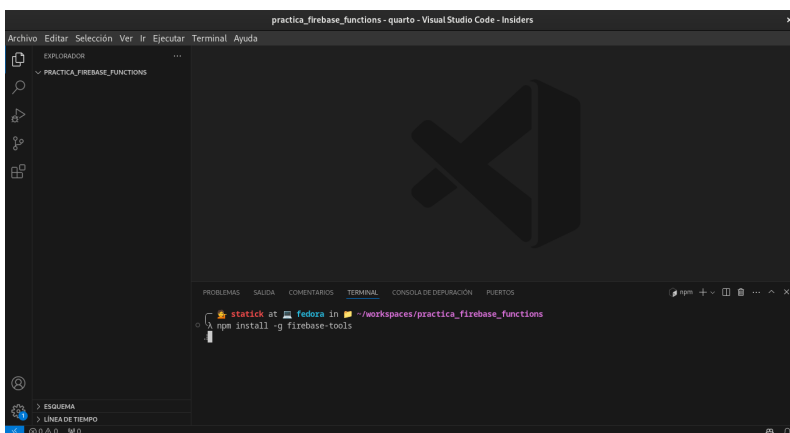
30.3.2 Paso 2: Inicializar un proyecto de Node.js con Firebase

1. Creamos un nuevo directorio en nuestro sistema operativo, para este ejemplo lo llamaremos **practica_firebase_functions**.
2. Ingresamos al directorio que acabamos de crear con Visual Studio Code.
3. Abrimos una terminal en Visual Studio Code.



4. Necesitamos instalar Firebase CLI, para ello ejecutamos el siguiente comando:

```
npm install -g firebase-tools
```



El comando anterior instala Firebase CLI de forma global en nuestro sistema operativo.

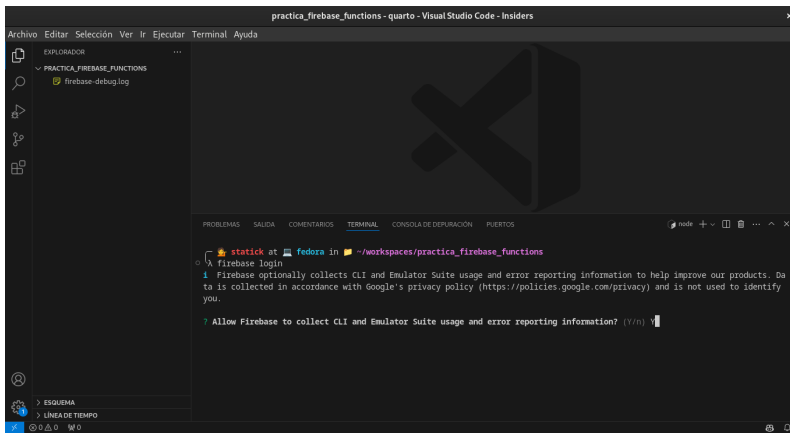
4. Ahora nos conectamos Firebase con nuestro proyecto de Node.js ejecutando el siguiente comando:

firebase login

Lo primero que nos pedirá es una serie de preguntas para asegurarse de que estamos conectando Firebase con el proyecto correcto.

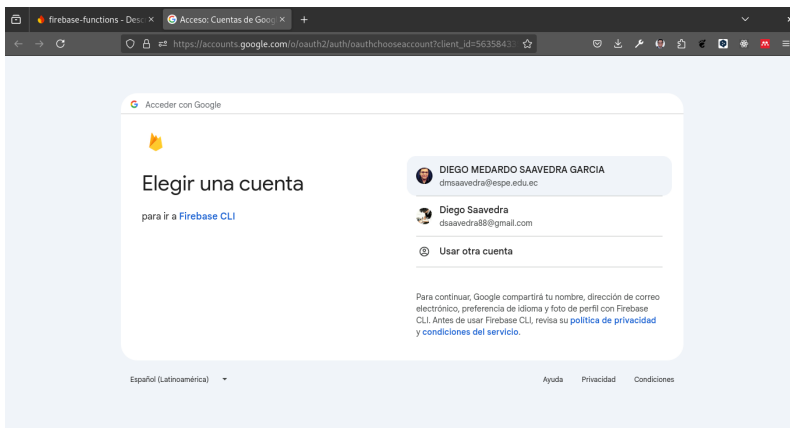
? Allow Firebase to collect CLI and Emulator Suite usage and error reporting information? (Y/n)

En el caso de que no queramos compartir esta información con Firebase, podemos responder con la letra **n**, pero se sugiere responder con la letra **Y**.

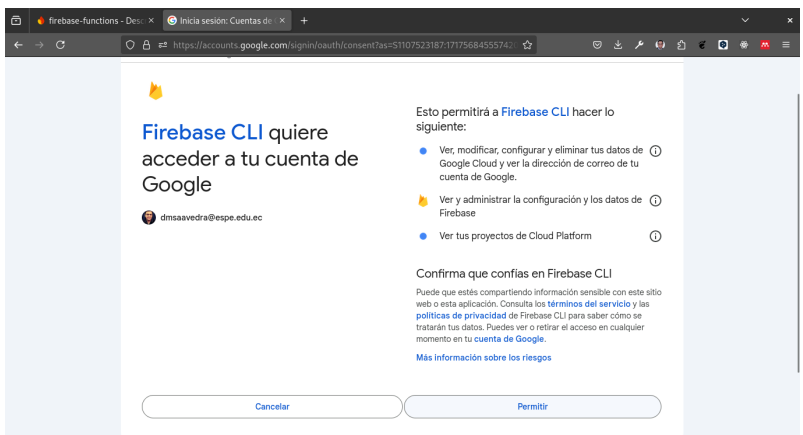
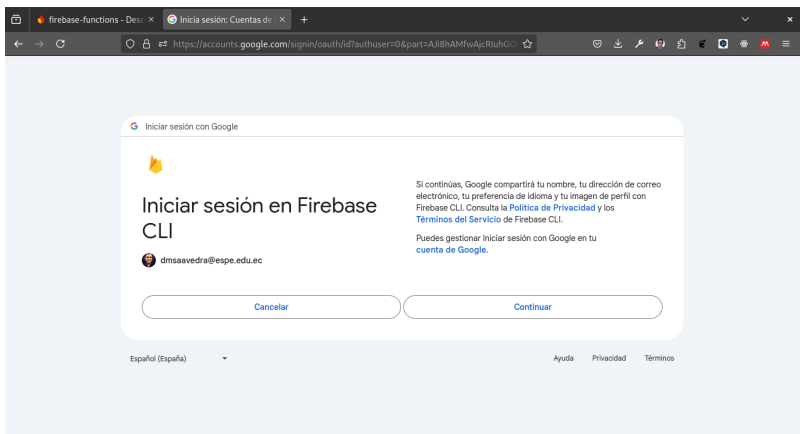


```
static@st: ~ - ssh - fedora in ~ - /workspaces/practica_firebase_functions
└─$ firebase login
1 Firebase optionally collects CLI and Emulator Suite usage and error reporting information to help improve our products. Data is collected in accordance with Google's privacy policy (https://policies.google.com/privacy) and is not used to identify you.
? Allow Firebase to collect CLI and Emulator Suite usage and error reporting information? (Y/n) Y
```

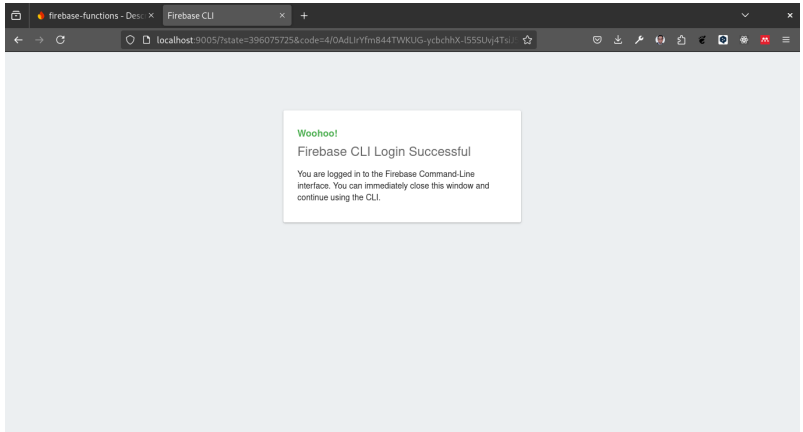
Esto nos pedirá que mediante nuestro navegador iniciemos sesión con nuestra cuenta de Google, se sugiere utilizar la cuenta institucional, sin embargo, se puede utilizar cualquier cuenta de Google.

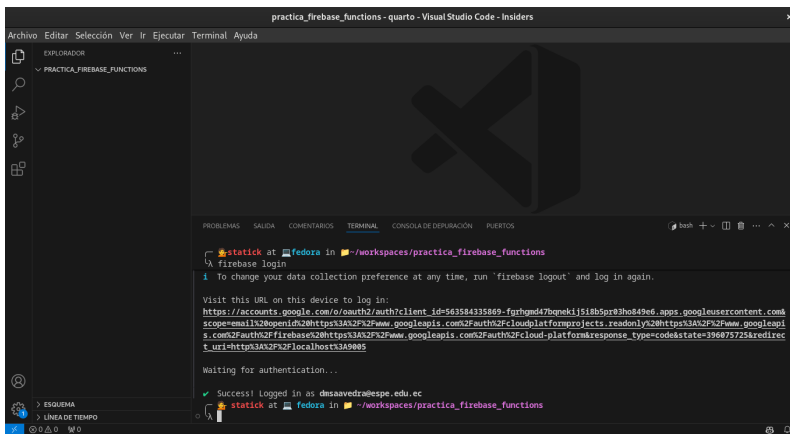


Concedemos los permisos necesarios para que Firebase pueda acceder a nuestra cuenta de Google.



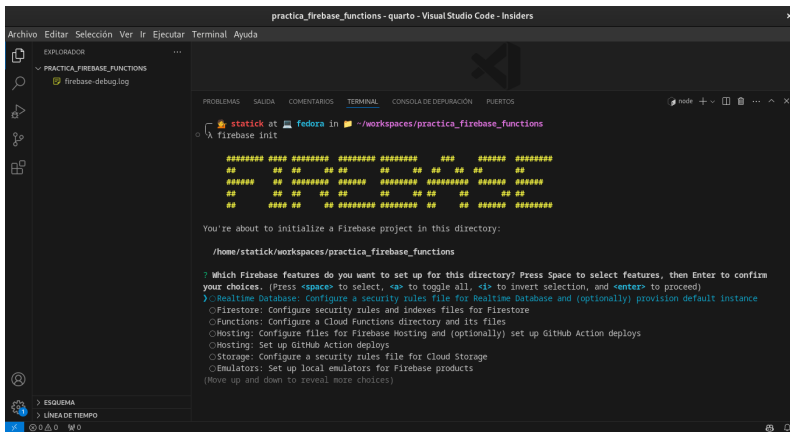
Finalmente nos aparecerá un mensaje en la terminal indicando que hemos iniciado sesión correctamente.





5. Inicializamos un proyecto de Firebase ejecutando el siguiente comando:

```
firebase init
```

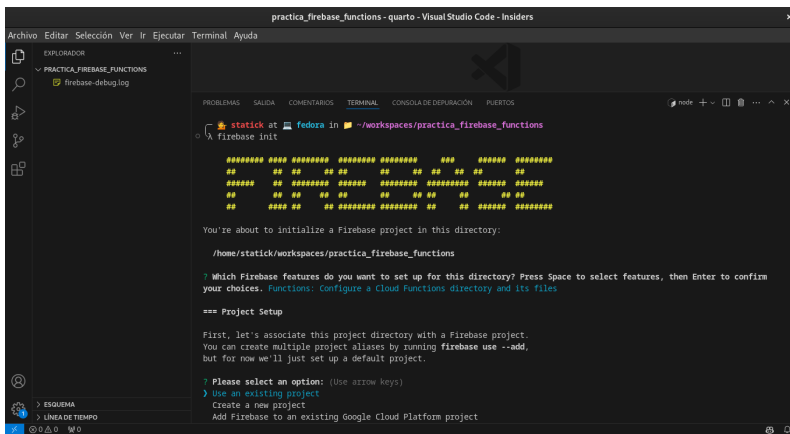


Esto nos mostrará una serie de opciones que debemos seleccionar:

```
? Which Firebase features do you want to set up for this directory? Press Space to select
your choices. (Press <space> to select, <a> to toggle all, <i> to invert selection, and <
  Realtime Database: Configure a security rules file for Realtime Database and (optionall
  Firestore: Configure security rules and indexes files for Firestore
  Functions: Configure a Cloud Functions directory and its files
  Hosting: Configure files for Firebase Hosting and (optionally) set up GitHub Action dep
  Hosting: Set up GitHub Action deploys
  Storage: Configure a security rules file for Cloud Storage
  Emulators: Set up local emulators for Firebase products
(Move up and down to reveal more choices)
```

Para este laboratorio seleccionamos la opción **Functions**, lo hacemos con la tecla de espacio y luego presionamos la tecla **Enter**.

Esto nos llevará a la siguiente pantalla.



=== Project Setup

First, let's associate this project directory with a Firebase project. You can create multiple project aliases by running `firebase use --add`, but for now we'll just set up a default project.

? Please select an option: (Use arrow keys)

Use an existing project

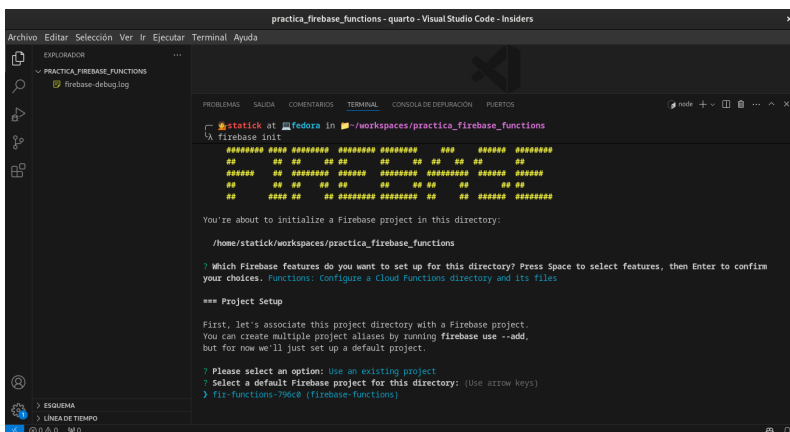
Create a new project

Add Firebase to an existing Google Cloud Platform project

Don't set up a default project

Seleccionamos la opción **Use an existing project** y presionamos la tecla **Enter**.

Dentro de las opciones nos aparecerá los proyectos que tenemos en Firebase, seleccionamos el proyecto que creamos en el paso 1 y presionamos la tecla **Enter**.

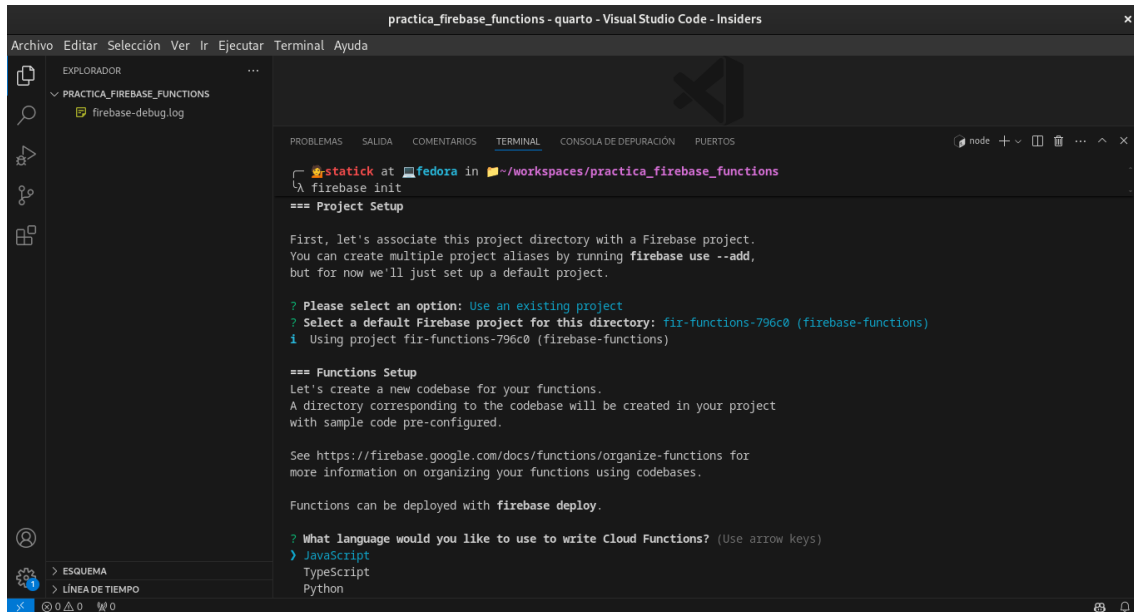


=== Project Setup

First, let's associate this project directory with a Firebase project. You can create multiple project aliases by running `firebase use --add`, but for now we'll just set up a default project.

```
? Please select an option: Use an existing project
? Select a default Firebase project for this directory: (Use arrow keys)
fir-functions-796c0 (firebase-functions)
```

Ahora nos sugerirá el lenguaje de programación que vamos a utilizar.



```
practica_firebase_functions - quarto - Visual Studio Code - Insiders
Archivo Editar Selección Ver Ir Ejecutar Terminal Ayuda
EXPLORADOR
PRACTICA_FIREBASE_FUNCTIONS
  firebase-debug.log
PROBLEMAS SALIDA COMENTARIOS TERMINAL CONSOLA DE DEPURACIÓN PUERTOS
statomic at fedora in ~/workspaces/practica_firebase_functions
λ firebase init
=== Project Setup

First, let's associate this project directory with a Firebase project.
You can create multiple project aliases by running firebase use --add,
but for now we'll just set up a default project.

? Please select an option: Use an existing project
? Select a default Firebase project for this directory: fir-functions-796c0 (firebase-functions)
i Using project fir-functions-796c0 (firebase-functions)

=== Functions Setup

Let's create a new codebase for your functions.
A directory corresponding to the codebase will be created in your project
with sample code pre-configured.

See https://firebase.google.com/docs/functions/organize-functions for
more information on organizing your functions using codebases.

Functions can be deployed with firebase deploy.

? What language would you like to use to write Cloud Functions? (Use arrow keys)
> JavaScript
  TypeScript
  Python
```

=== Functions Setup

Let's create a new codebase for your functions.

A directory corresponding to the codebase will be created in your project with sample code pre-configured.

See <https://firebase.google.com/docs/functions/organize-functions> for more information on organizing your functions using codebases.

Functions can be deployed with `firebase deploy`.

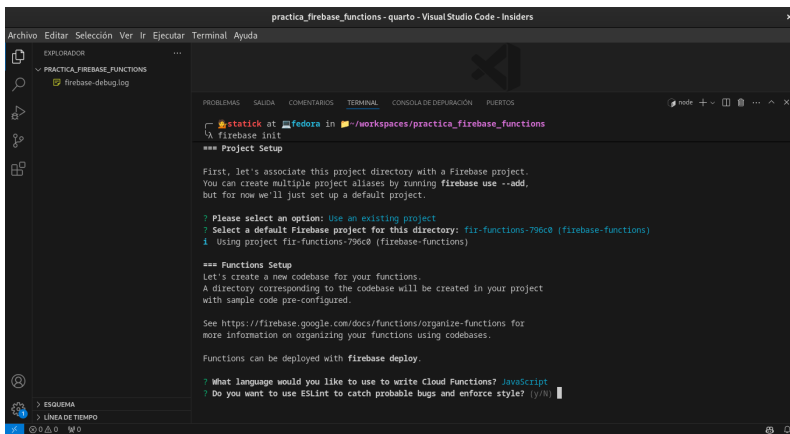
```
? What language would you like to use to write Cloud Functions? (Use arrow keys)
JavaScript
TypeScript
Python
```

Para este laboratorio seleccionamos la opción **JavaScript** y presionamos la tecla **Enter**.

Finalmente nos sugiere instalar ESLint, es un linter que nos ayuda a mantener un código limpio y ordenado.

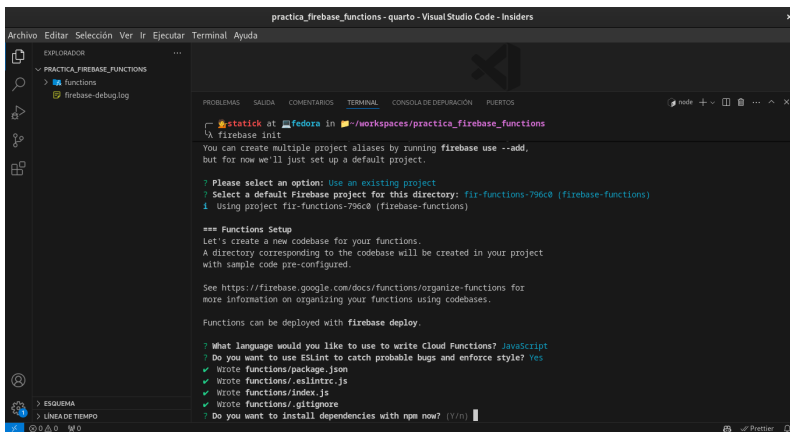
💡 Tip

Si lo activa, posiblemente tenga inconvenientes en la etapa de Deploy, para esta práctica no es requerido, sin embargo en un entorno de producción es recomendable.



? Do you want to use ESLint to catch probable bugs and enforce style? (y/N)

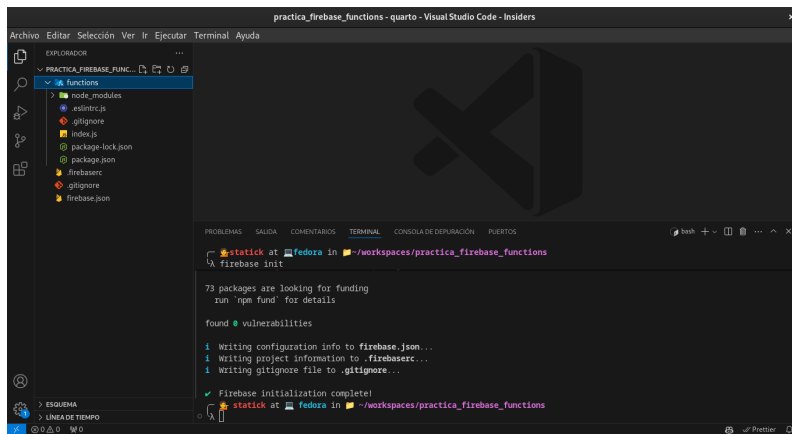
Como ultimo paso nos sugiere instalar las dependencias necesarias para el proyecto.



```
Wrote functions/package.json
Wrote functions/.eslintrc.js
Wrote functions/index.js
Wrote functions/.gitignore
? Do you want to install dependencies with npm now? (Y/n)
```

Para este laboratorio seleccionamos la opción **Y** y presionamos la tecla **Enter**.

Con todos estos pasos hemos creado una estructura de proyecto de Firebase con Cloud Functions.



30.3.3 Crear nuestro CRUD en nuestro proyecto de Firebase

1. Dentro de la carpeta **functions** nos dirigimos al archivo **index.js**.

Para probar que todo está funcionando correctamente, vamos a modificar el archivo **index.js** con el siguiente código:

```
const functions = require('firebase-functions');
const express = require('express');
const app = express();

app.get('/hello-world', (req, res) => {
  res.send('Hello World');
});

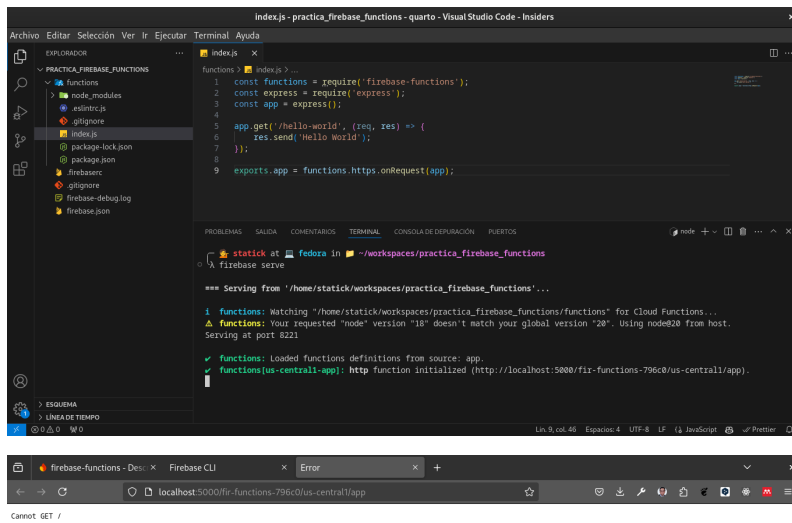
exports.app = functions.https.onRequest(app);
```

En el código anterior estamos creando una función que nos permite acceder a la ruta **/hello-world** y nos devuelve un mensaje **Hello World**.

2. Para probar que todo está funcionando correctamente, ejecutamos el siguiente comando:

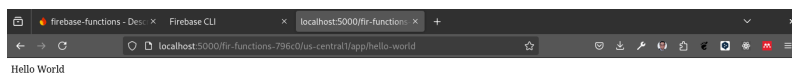
```
firebase serve
```

Este comando nos permitirá ejecutar nuestra aplicación en un servidor local en el puerto **5000**.



Para que todo funcione correctamente debemos acceder a la siguiente URL en nuestro navegador <http://localhost:5000/fir-functions-796c0/us-central1/app/hello-world>

Con ello podemos ver el mensaje **Hello World**.



Ahora lo que queremos no es un hello world, sino un CRUD, para ello vamos a crear una colección de documentos en Firestore.

Para ello el siguiente paso será autenticar nuestra función con Firebase.

30.3.4 Paso 3: Autenticar nuestra función con Firebase

1. Dentro de la carpeta **functions** nos dirigimos al archivo **index.js**.
2. Agregamos el siguiente código al inicio del archivo **index.js**.

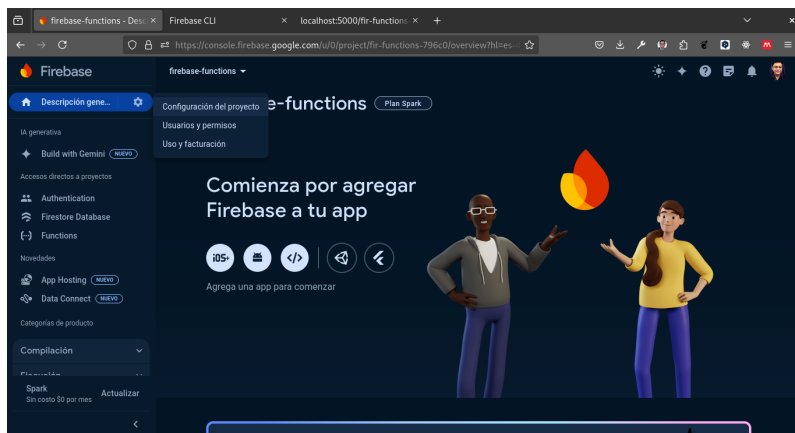
```
const functions = require('firebase-functions');
const admin = require('firebase-admin');
const express = require('express');

const app = express();
const admin.initializeApp({
  credential: admin.credential.applicationDefault(),
  databaseURL: 'https://fir-functions-796c0.firebaseio.com'
});

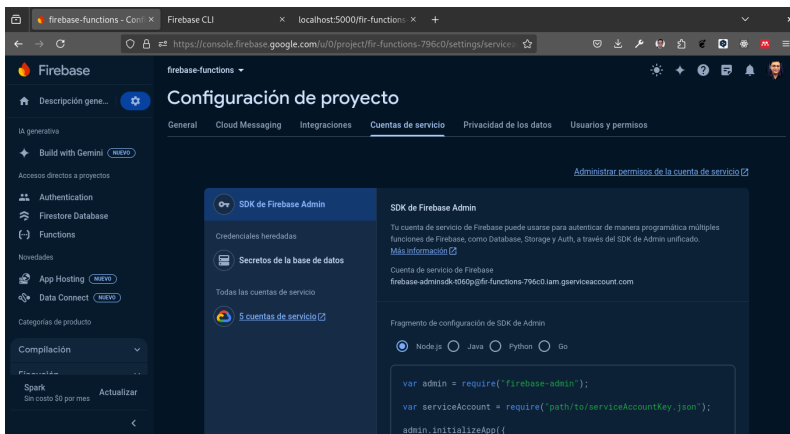
app.get('/hello-world', (req, res) => {
  res.send('Hello World');
});

exports.app = functions.https.onRequest(app);
```

Este código lo obtenemos desde la configuración de nuestro proyecto en Firebase.



En esta sección nos vamos a dirigir a Cuentas de Servicio.



Como podemos observar en la imagen anterior, nos aparece un mensaje que nos indica que no tenemos ninguna cuenta de servicio, por lo que debemos crear una.

Tip

Actualmente Firebase tiene soporte para Node.js, Java, Python y Go.

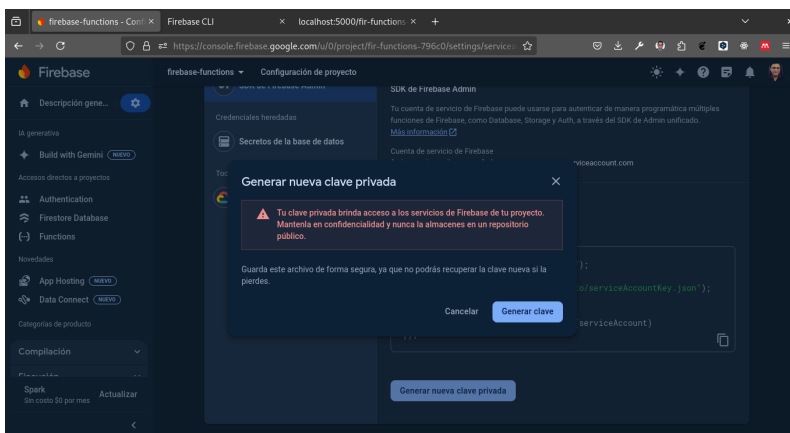
En nuestro caso vamos a elegir Node.js.

```
var admin = require("firebase-admin");

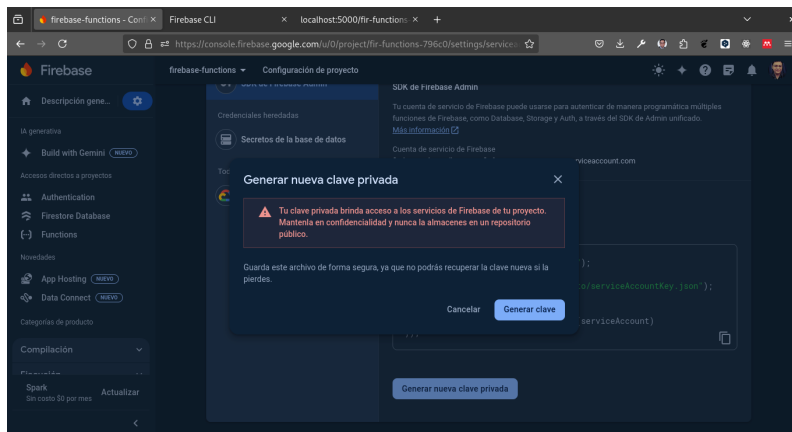
var serviceAccount = require("path/to/serviceAccountKey.json");

admin.initializeApp({
  credential: admin.credential.cert(serviceAccount)
});
```

Como se puede observar en el código anterior en la sección de **serviceAccountKey.json** debemos colocar la ruta de nuestro archivo de credenciales, este archivo lo obtenemos en el botón **Generar nueva clave privada**.



Al hacer clic en el botón **Generar nueva clave privada** se descargará un archivo **.json** que contiene las credenciales de nuestra cuenta de servicio.



Lo descargamos y lo guardamos en la carpeta **functions**, podemos renombrarlo para facilitar su uso.

Si todo salió bien finalmente debemos tener un código como el siguiente:

```
const functions = require('firebase-functions');
const admin = require("firebase-admin");
const express = require('express');
const bodyParser = require('body-parser');
const serviceAccount = require("./permisos.json");

const app = express();

admin.initializeApp({
  credential: admin.credential.cert(serviceAccount)
});

const db = admin.firestore();

app.use(bodyParser.json());

app.get('/hello-world', (req, res) => {
  res.send('Hello World');
});

app.post('/api/products', async (req, res) => {
  try {
    if (!req.body.id || !req.body.name) {
      return res.status(400).send({error: 'ID and Name are required'});
    }

    await db.collection('products')
      .doc(req.body.id)
      .set({name: req.body.name});

    return res.status(204).send();
  }
});
```

```
    } catch (error) {
      console.error("Error adding document: ", error);
      return res.status(500).send({error: 'Internal Server Error'});
    }
  });

exports.app = functions.https.onRequest(app);
```

Vamos a analizar un poco el código anterior:

- **admin.initializeApp**: Inicializa la aplicación de Firebase con las credenciales de nuestra cuenta de servicio.
- **db**: Es una instancia de Firestore que nos permite interactuar con la base de datos.
- **app.use(bodyParser.json())**: Nos permite recibir datos en formato JSON.
- **app.post('/api/products')**: Nos permite crear un documento en Firestore.
- **req.body.id**: Es el identificador del documento.
- **req.body.name**: Es el nombre del documento.
- **db.collection('products').doc(req.body.id).set({name: req.body.name})**: Crea un documento en la colección **products** con el identificador **req.body.id** y el nombre **req.body.name**.
- **res.status(204).send()**: Nos devuelve un código de estado **204** que indica que la operación fue exitosa.
- **res.status(500).send({error: 'Internal Server Error'})**: Nos devuelve un código de estado **500** que indica que hubo un error en el servidor.

Tip

En este caso estamos utilizando el método **POST** para crear un documento en Firestore, sin embargo, podemos utilizar los métodos **GET**, **PUT** y **DELETE** para realizar operaciones de lectura, actualización y eliminación respectivamente.

Tip

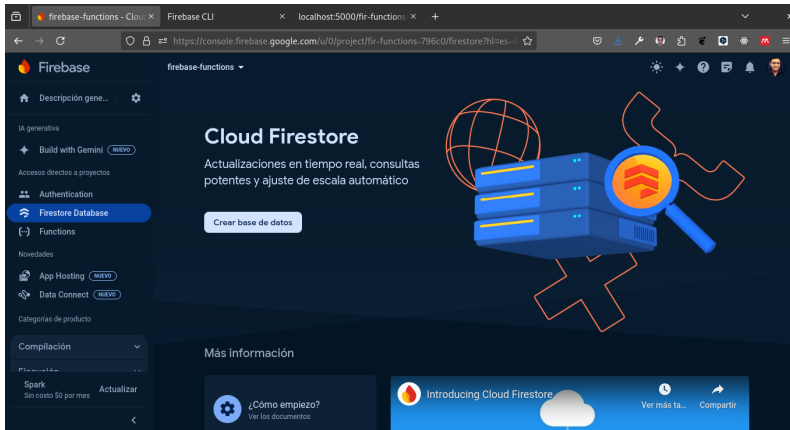
Para poder probarlo podemos utilizar un cliente como Thunder Client, Postman, Insomnia, etc.

Warning

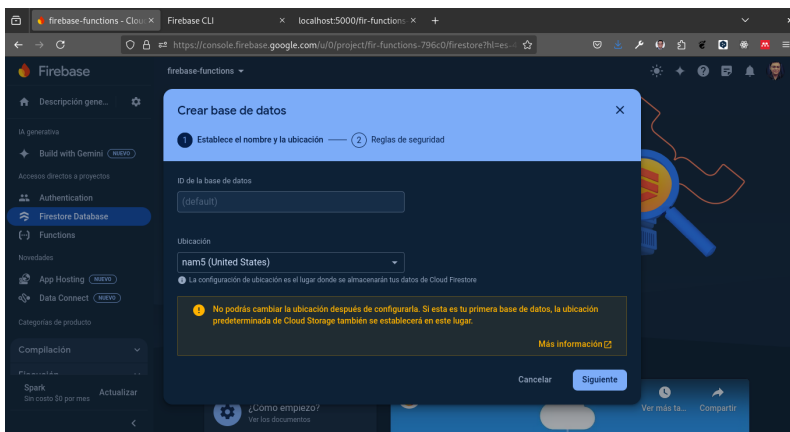
El paso anterior no nos funcionará mientras no tengamos una colección en Firestore.

30.3.5 Paso 4: Crear un documento en Firestore

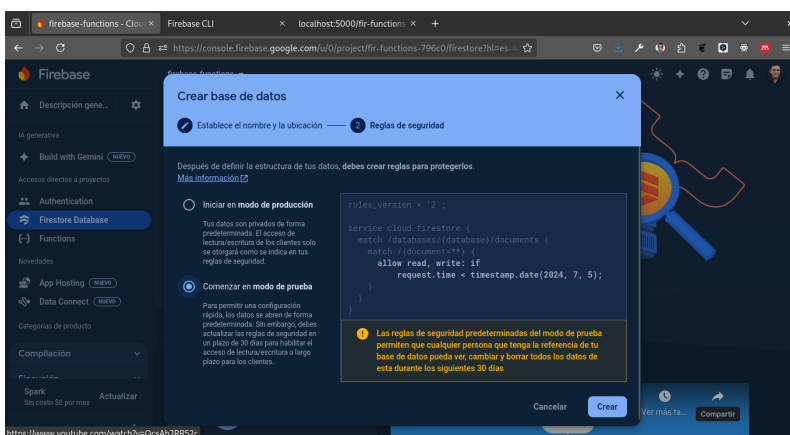
1. Para probar que todo está funcionando correctamente, debemos crear un documento en Firestore,
2. Dentro de la consola de Firebase, nos dirigimos a la opción **Firestore**.

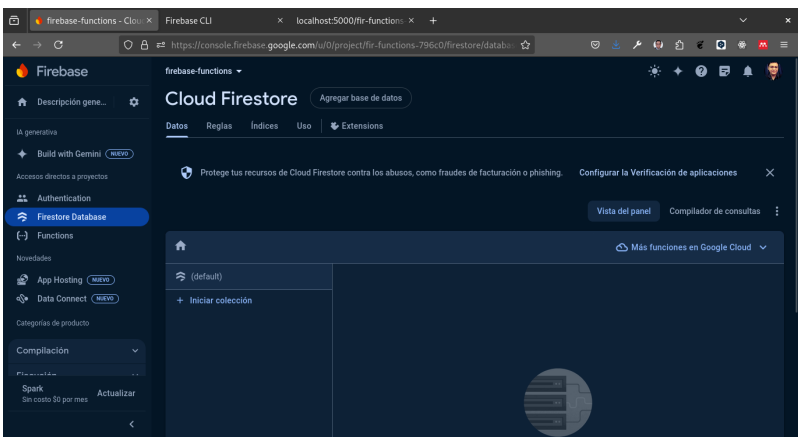
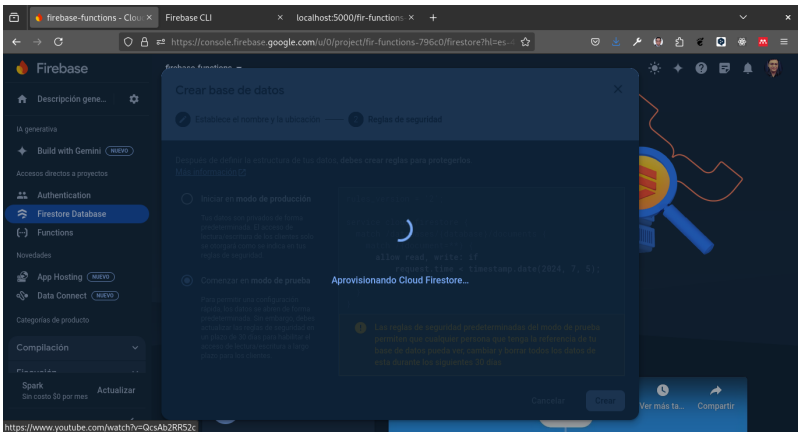


3. Hacemos clic en el botón **Crear base de datos**.

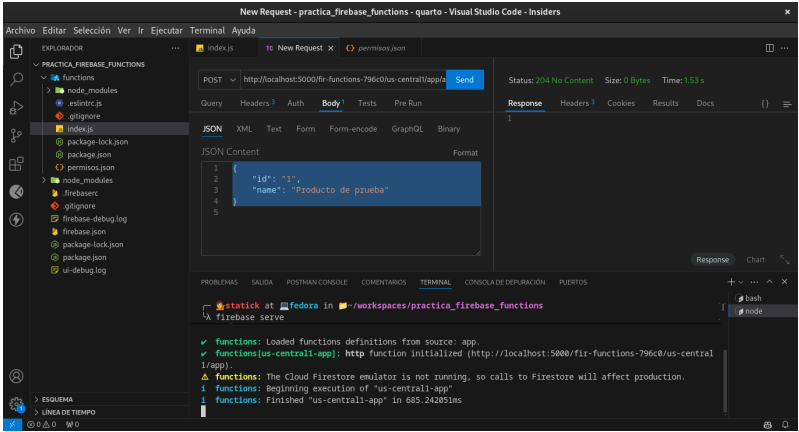


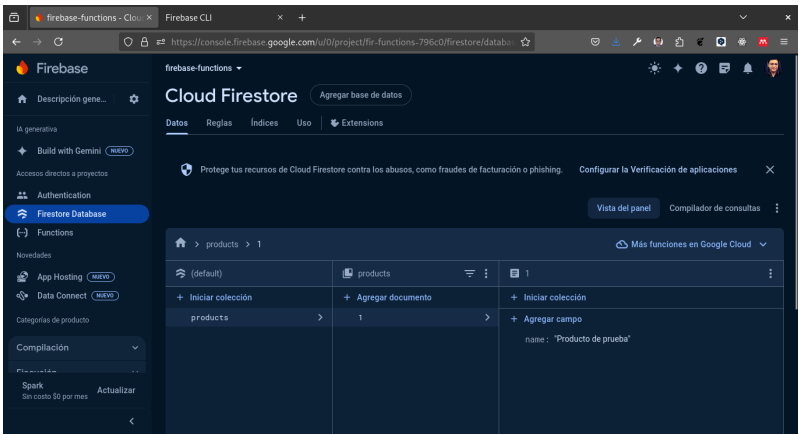
4. Seleccionamos la opción **Iniciar en modo de prueba** y hacemos clic en el botón **Siguiente**.



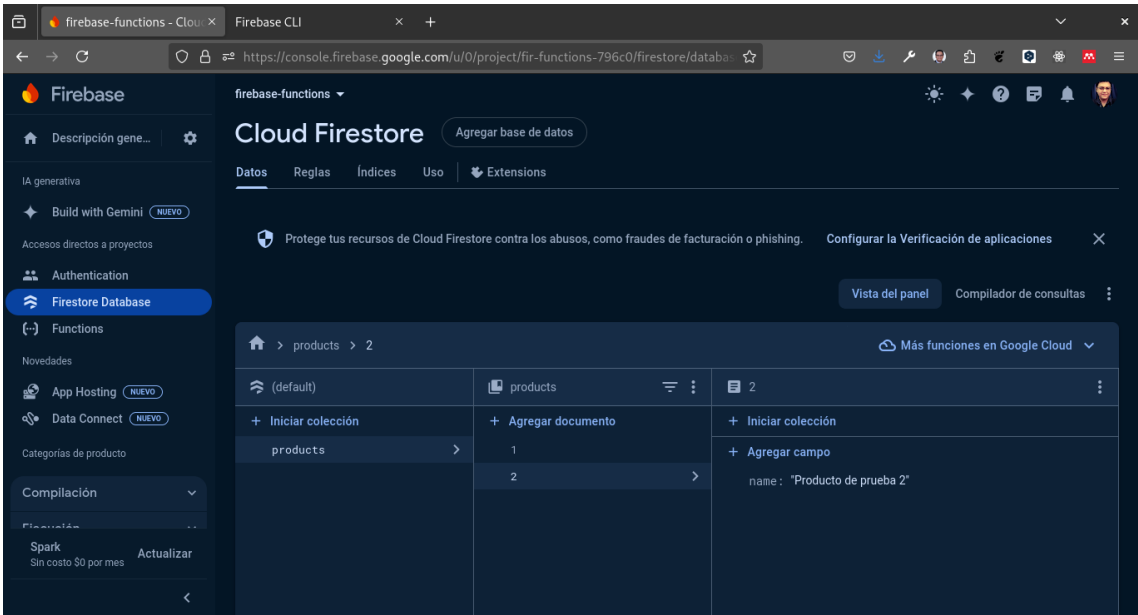
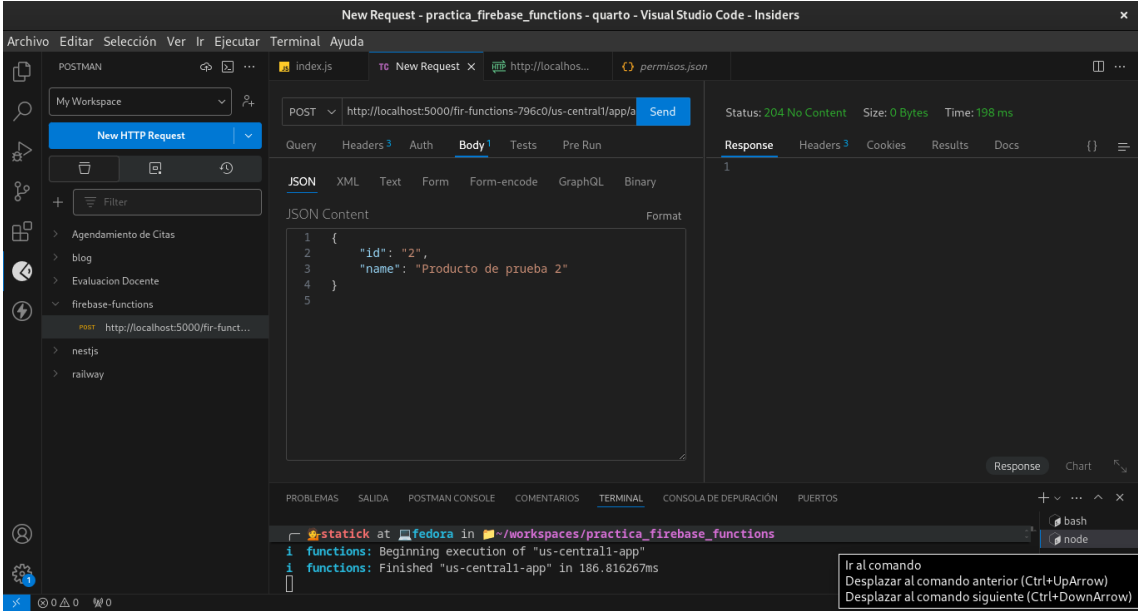


Si todo salio bien debemos tener una salida como la siguiente.





Hagamos una segunda prueba



Ahora vamos a crear el CRUD completo.

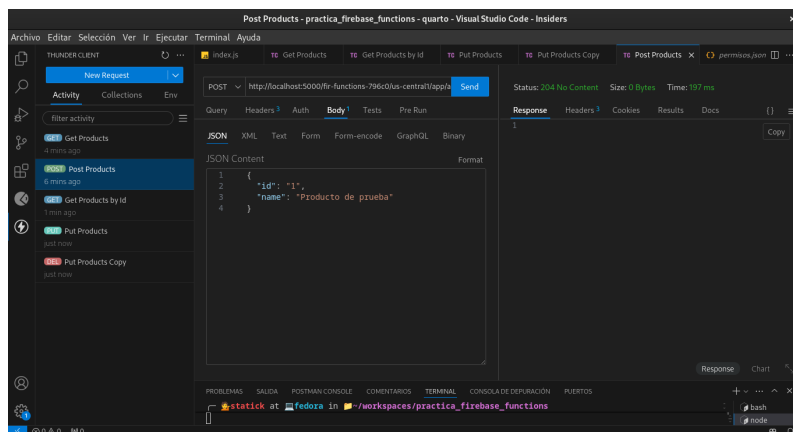
Empezamos por el método **POST**.

```
// Create a new product
app.post('/api/products', async (req, res) => {
  try {
    const { id, name } = req.body;
    if (!id || !name) {
      return res.status(400).send({ error: 'ID and Name are required' });
    }

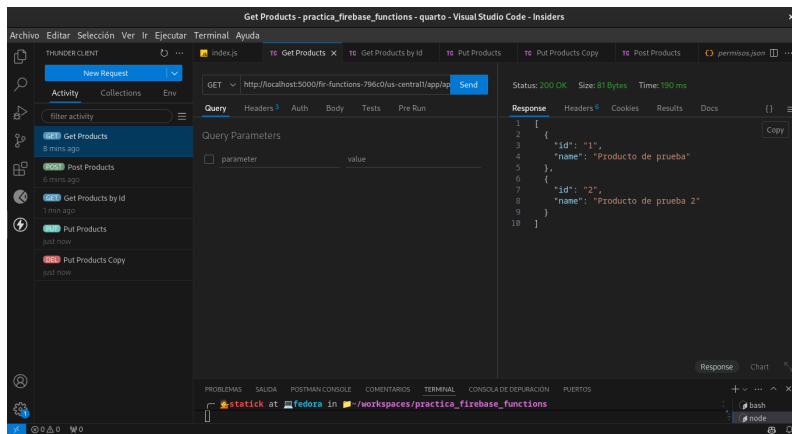
    await db.collection('products').doc(id).set({ name });
    return res.status(204).send();
  } catch (error) {
    console.error("Error adding document: ", error);
    return res.status(500).send({ error: 'Internal Server Error' });
  }
});
```

En el código anterior estamos creando un nuevo producto en la colección **products**. Para ello necesitamos enviar un objeto JSON con los campos **id** y **name**. Ejemplo

```
{
  "id": "1",
  "name": "Product 1"
}
```



Si todo sale bien, nos devolverá un código de estado **204**.



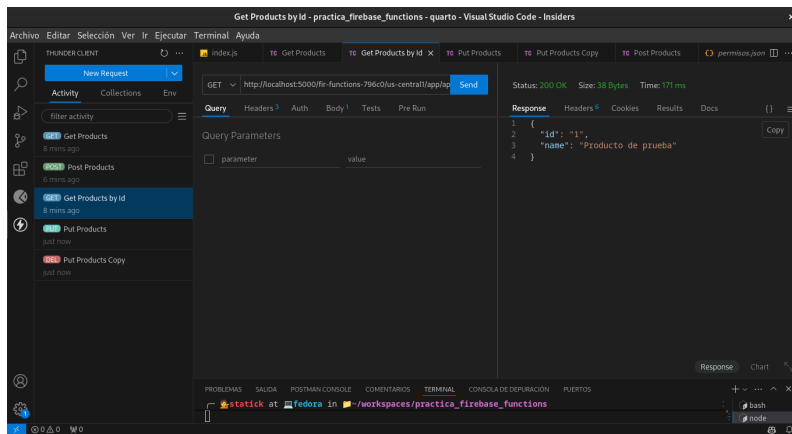
Continuamos con el método **GET**.

```
// Get all products
app.get('/api/products', async (req, res) => {
  try {
    const querySnapshot = await db.collection('products').get();
    const products = querySnapshot.docs.map(doc => ({ id: doc.id, ...doc.data() }));
    return res.status(200).send(products);
  } catch (error) {
    console.error("Error getting documents: ", error);
    return res.status(500).send({ error: 'Internal Server Error' });
  }
});

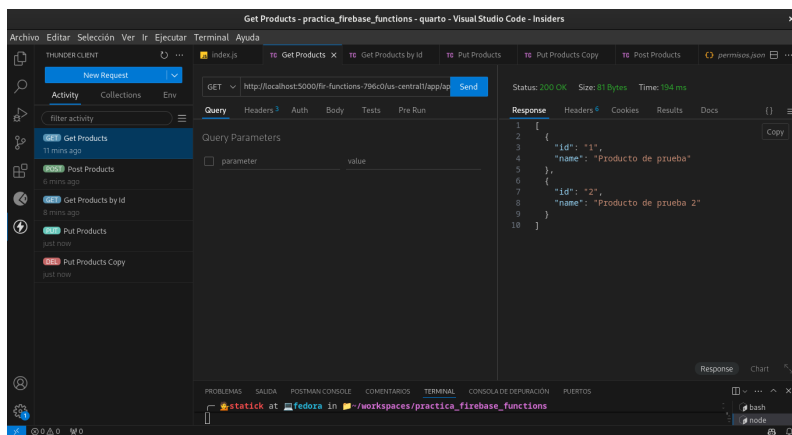
// Get a product by ID
app.get('/api/products/:id', async (req, res) => {
  try {
    const doc = await db.collection('products').doc(req.params.id).get();
    if (!doc.exists) {
      return res.status(404).send({ error: 'Product not found' });
    }
    return res.status(200).send({ id: doc.id, ...doc.data() });
  } catch (error) {
    console.error("Error getting document: ", error);
    return res.status(500).send({ error: 'Internal Server Error' });
  }
});
```

En el código anterior estamos obteniendo todos los productos de la colección **products** y un producto en específico por su **id**.

Para obtener todos los productos debemos acceder a la siguiente URL <http://localhost:5000/fir-functions-796c0/us-central1/app/api/products>



Para obtener un producto en específico debemos acceder a la siguiente URL <http://localhost:5000/fir-functions-796c0/us-central1/app/api/products/1>



Continuamos con el método **PUT**.

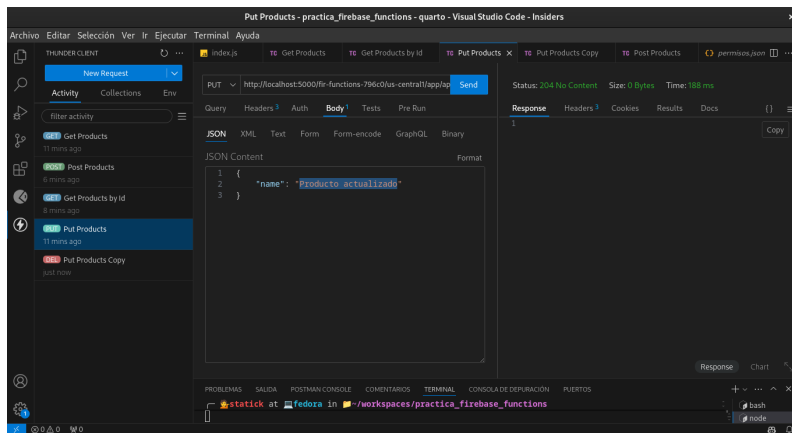
```
// Update a product by ID
app.put('/api/products/:id', async (req, res) => {
  try {
    const { name } = req.body;
    if (!name) {
      return res.status(400).send({ error: 'Name is required' });
    }

    await db.collection('products').doc(req.params.id).update({ name });
    return res.status(204).send();
  } catch (error) {
    console.error("Error updating document: ", error);
    return res.status(500).send({ error: 'Internal Server Error' });
  }
});
```

En el código anterior estamos actualizando un producto en la colección **products** por su **id**.

Para actualizar un producto debemos enviar un objeto JSON con el campo **name**. Ejemplo

```
{  
  "name": "Producto actualizado"  
}
```



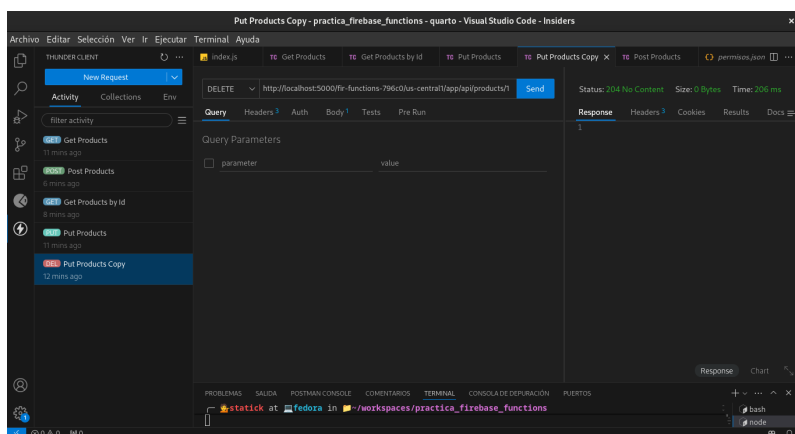
Si todo sale bien, nos devolverá un código de estado **204**.

Finalmente continuamos con el método **DELETE**.

```
// Delete a product by ID  
app.delete('/api/products/:id', async (req, res) => {  
  try {  
    await db.collection('products').doc(req.params.id).delete();  
    return res.status(204).send();  
  } catch (error) {  
    console.error("Error deleting document: ", error);  
    return res.status(500).send({ error: 'Internal Server Error' });  
  }  
});
```

En el código anterior estamos eliminando un producto en la colección **products** por su **id**.

Para eliminar un producto debemos acceder a la siguiente URL <http://localhost:5000/fir-functions-796c0/us-central1/app/api/products/1> con el método **DELETE**.



Si todo sale bien, nos devolverá un código de estado **204**.

Con estos pasos hemos creado una REST API que nos permite realizar operaciones CRUD sobre una colección de documentos en Firestore.

30.3.6 Paso 5: Desplegar nuestra aplicación en Firebase

Antes de desplegar nuestra aplicación en Firebase, debemos realizar algunos cambios en nuestro proyecto.

1. Dentro de la carpeta **functions** encontraremos el archivo `.gitignore`, debemos agregar una línea que nos permita eliminar nuestro archivo de credenciales.

```
...
permissions.json
```

1. Para desplegar nuestra aplicación en Firebase, ejecutamos el siguiente comando:

```
firebase deploy
```

Este comando nos permitirá desplegar nuestra aplicación en Firebase.

Warning

Toma en cuenta que este último paso necesita de adquirir un plan de pago en Firebase. Por lo tanto no será necesario realizar este paso.

Podemos subir nuestro proyecto a GitHub para que los demás puedan ver nuestro código.

Eso es todo, hemos creado una REST API que nos permite realizar operaciones CRUD sobre una colección de documentos en Firestore.

30.4 Conclusiones

- Firebase Cloud Functions nos permite ejecutar código en la nube.
- Firebase Firestore nos permite almacenar datos en la nube.
- Podemos utilizar Firebase Cloud Functions y Firebase Firestore para crear una REST API que nos permita realizar operaciones CRUD sobre una colección de documentos en Firestore.

30.5 Referencias

- [Firebase](#)
- [Firebase CLI](#)
- [Firebase Firestore](#)
- [Firebase Cloud Functions](#)
- [Firebase Authentication](#)
- [Firebase Hosting](#)
- [Firebase Storage](#)
- [Firebase Realtime Database](#)
- [Firebase Emulators](#)
- [Firebase GitHub](#)
- [Firebase Pricing](#)
- [Código del Proyecto](#)

31 Laboratorio de Simple Object Access Protocol (SOAP)

31.1 Objetivo

Vamos a crear un cliente en Spring Boot que va a permitir consumir un servicio web creado en SOAP en la llamado Calculator, podemos encontrar las especificaciones en el siguiente enlace <http://www.dneonline.com/calculator.asmx>.

31.2 Materiales

Para este laboratorio necesitas tener instalado en tu computador:

- Spring Initializr
- Open JDK 17
- Apache Maven 3.9.7
- Spring Boot 2.6.3
- IntelliJ IDEA
- Git

31.3 Conceptos básicos.

31.3.1 SOAP

SOAP (Simple Object Access Protocol) es un protocolo de comunicación que permite la comunicación entre aplicaciones de software. Está basado en XML, define una estructura de mensajes que pueden ser intercambiados entre aplicaciones.

31.3.2 WSDL

WSDL (Web Services Description Language) es un lenguaje basado en XML que se utiliza para describir servicios web. Un archivo WSDL describe los métodos que un servicio web expone, los parámetros que recibe y los tipos de datos que retorna.

31.3.3 Spring Boot

Spring Boot es un framework de Java que facilita la creación de aplicaciones web. Proporciona un conjunto de herramientas y librerías que permiten crear aplicaciones web de forma rápida y sencilla.

31.3.4 Maven

Maven es una herramienta de gestión de proyectos que se utiliza para compilar, empaquetar y desplegar aplicaciones Java. Permite gestionar las dependencias de un proyecto y automatizar tareas de construcción.

31.3.5 JAXB

JAXB (Java Architecture for XML Binding) es una tecnología de Java que se utiliza para convertir objetos de Java a XML y viceversa. Permite mapear objetos de Java a documentos XML y viceversa.

31.3.6 Marshalling

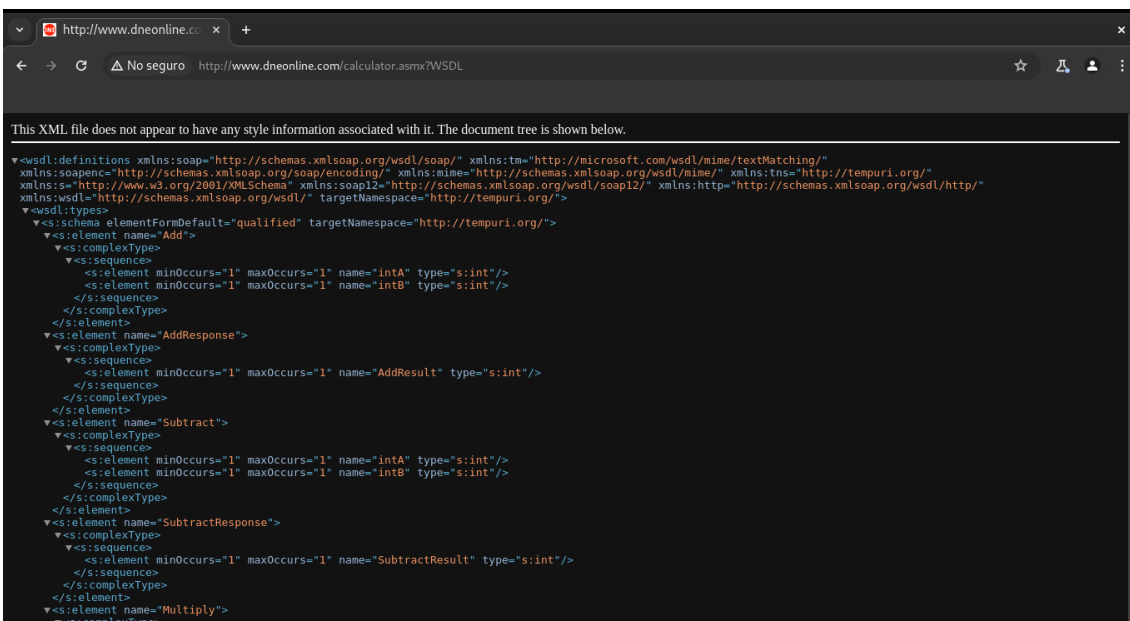
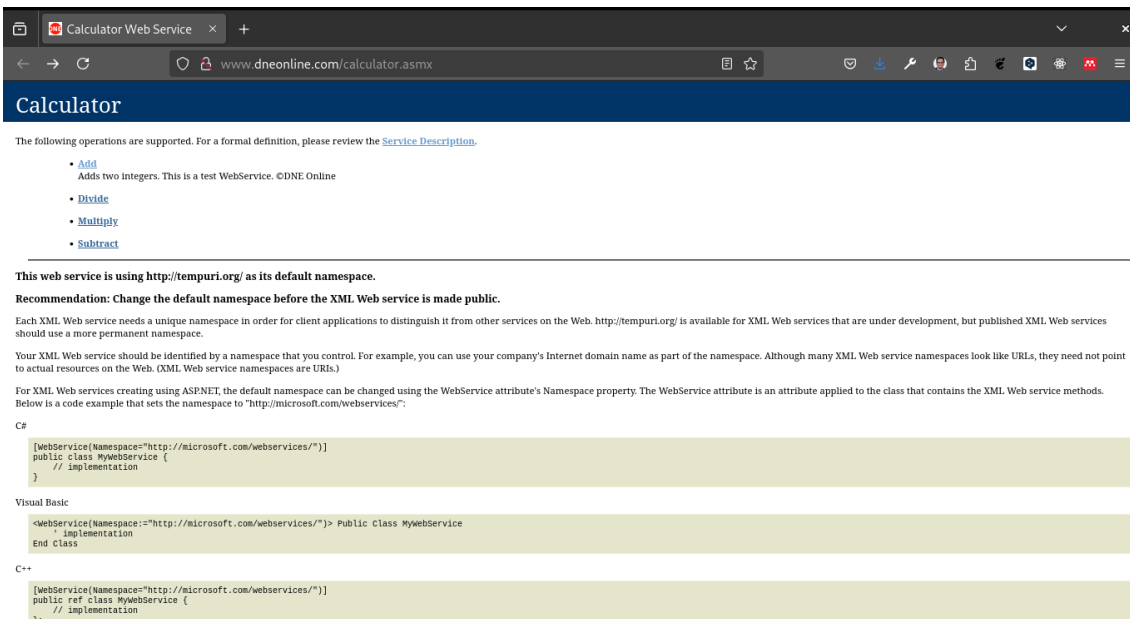
Marshalling es el proceso de convertir un objeto de Java a un documento XML. JAXB se encarga de convertir los objetos de Java a XML.

31.3.7 Unmarshalling

Unmarshalling es el proceso de convertir un documento XML a un objeto de Java. JAXB se encarga de convertir los documentos XML a objetos de Java.

31.4 Introducción

En este laboratorio vamos a crear un cliente en Spring Boot que va a permitir consumir un servicio web creado en SOAP. El servicio web que vamos a consumir se llama Calculator y podemos encontrar las especificaciones en el siguiente enlace <http://www.dneonline.com/calculator.asmx>.



El servicio web Calculator tiene los siguientes métodos:

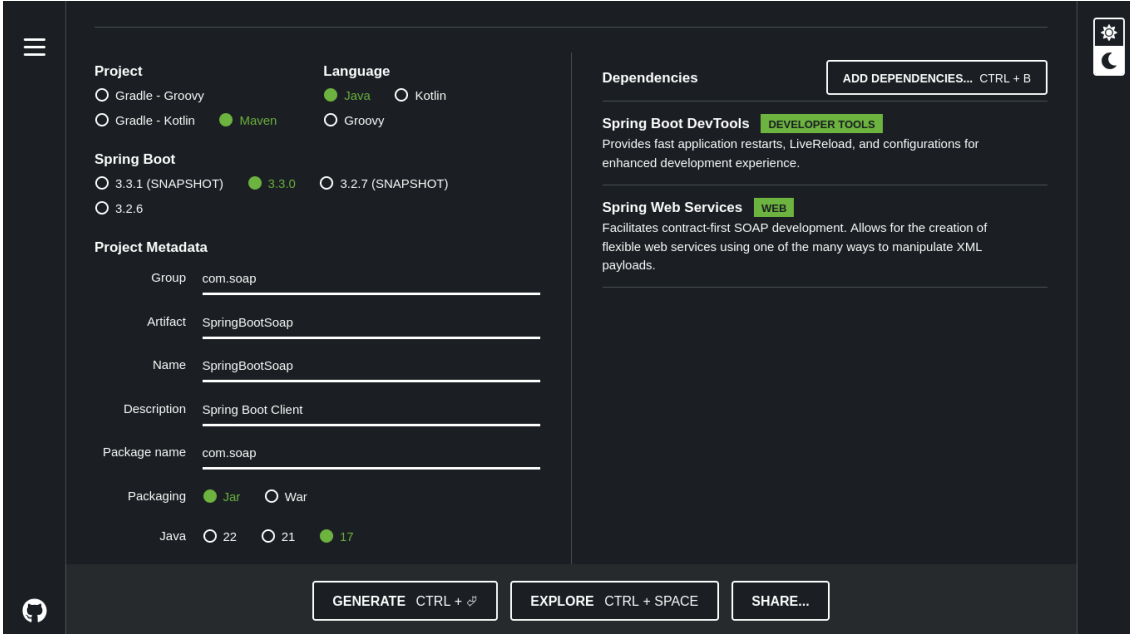
- Add: Suma dos números enteros y retorna el resultado.
- Subtract: Resta dos números enteros y retorna el resultado.
- Multiply: Multiplica dos números enteros y retorna el resultado.
- Divide: Divide dos números enteros y retorna el resultado.

Vamos a crear un cliente en Spring Boot que va a permitir consumir estos métodos del servicio web Calculator.

31.5 Instrucciones

Vamos a empezar creando nuestro proyecto con Spring Initializr, para ello vamos a seguir los siguientes pasos:

1. Abre Spring Initializr en tu navegador web <https://start.spring.io/>.
2. Llena los campos del formulario con la siguiente información:



The screenshot shows the Spring Initializr web form with the following configuration:

- Project:** Gradle - Groovy, Gradle - Kotlin, Maven, Groovy
- Language:** Java, Kotlin
- Spring Boot:** 3.3.1 (SNAPSHOT), 3.3.0, 3.2.7 (SNAPSHOT), 3.2.6
- Project Metadata:**
 - Group: com.soap
 - Artifact: SpringBootTest
 - Name: SpringBootTest
 - Description: Spring Boot Client
 - Package name: com.soap
 - Packaging: Jar, War
 - Java: 22, 21, 17
- Dependencies:** Includes Spring Boot DevTools (DEVELOPER TOOLS) and Spring Web Services (WEB).

Buttons at the bottom: GENERATE CTRL + G, EXPLORE CTRL + SPACE, SHARE...

Project: Maven

Language: Java

Spring Boot: 3.3.0

Group: com.soap

Artifact: SpringBootTest

Name: SpringBootTest

Description: Spring Boot Soap Client

Package name: com.soap

Packaging: Jar

Java: 17

3. Haz clic en el botón Generate para descargar el proyecto.
4. Descomprime el archivo descargado y ábrelo en IntelliJ IDEA.

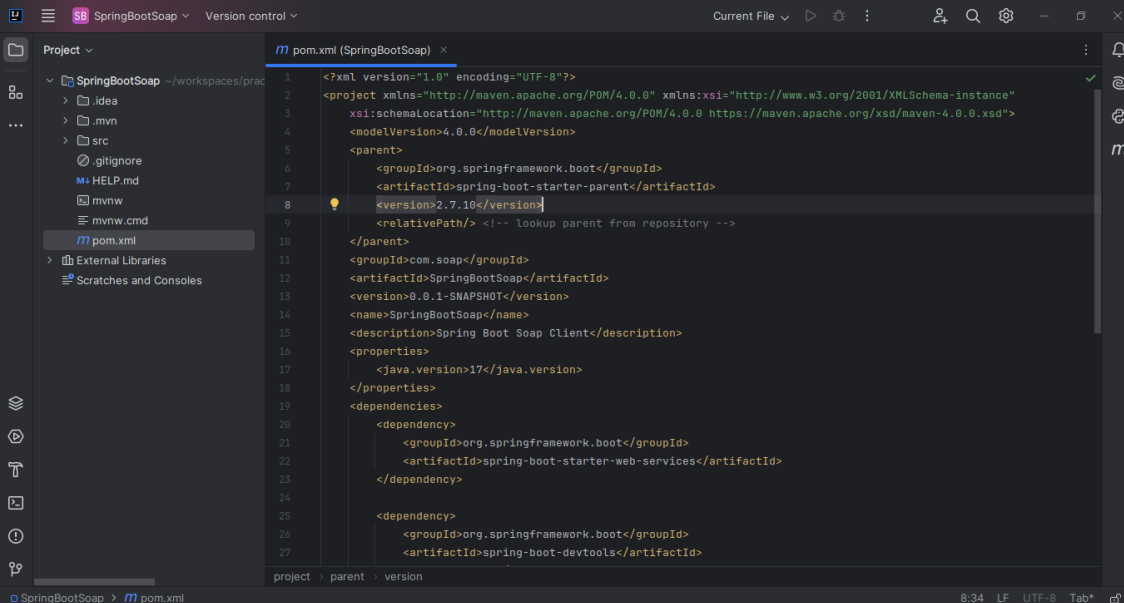
31.5.1 Crear un cliente SOAP

Vamos a crear un cliente SOAP que consuma el servicio web Calculator. Para ello vamos a seguir los siguientes pasos:

1. Abrimos el proyecto descomprimido en IntelliJ IDEA.

Warning

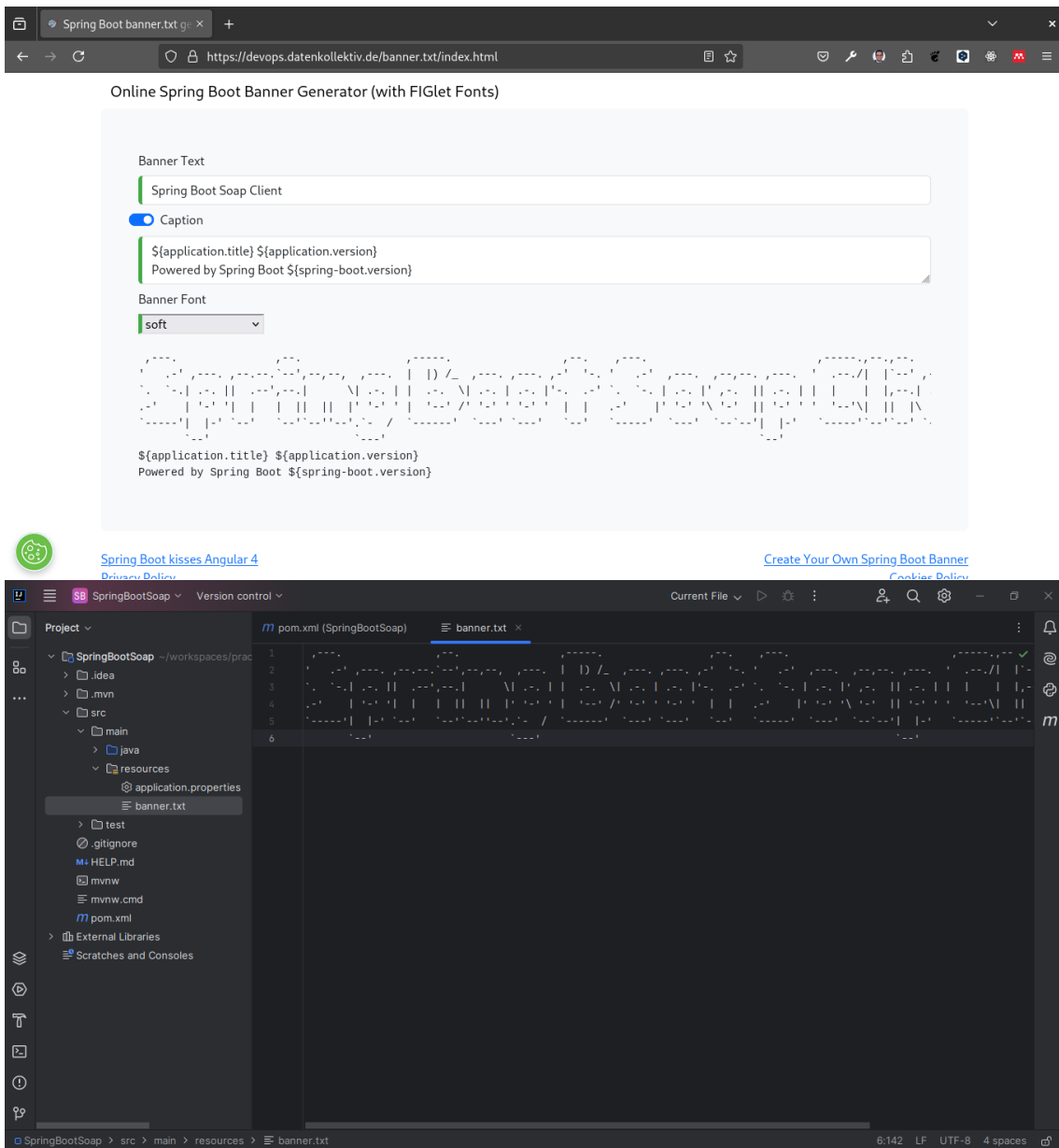
El proyecto que vamos a consumir desde el cliente está en la versión de Spring Boot 2.7.10, actualmente no nos permite Spring Initializr crear un proyecto con esta versión, así que realizamos este cambio en el archivo **pom.xml** del proyecto que acabamos de crear.



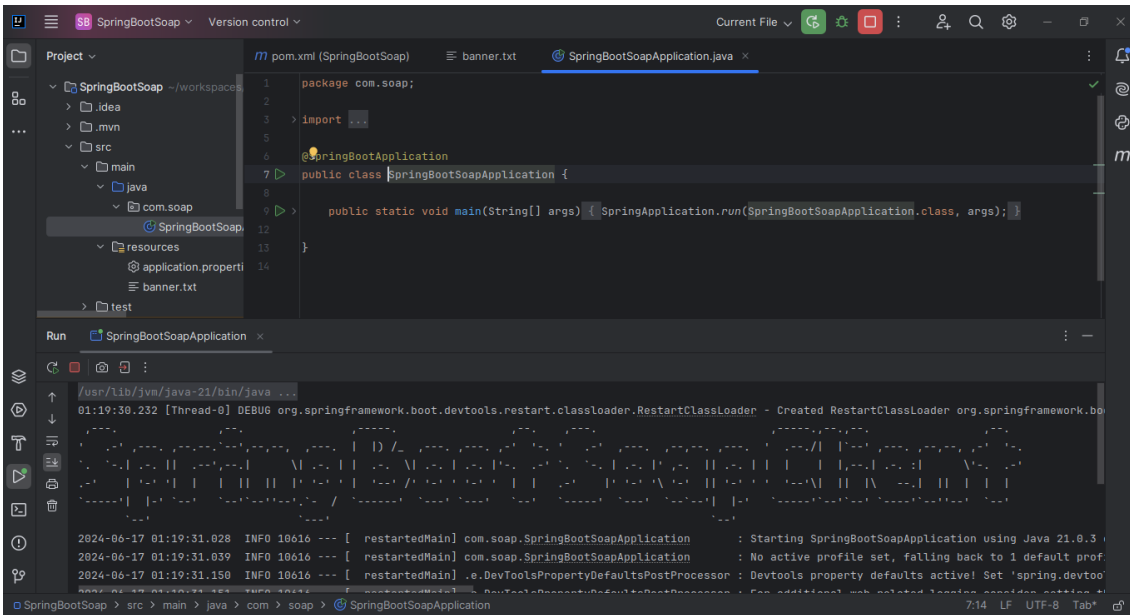
```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
4   <modelVersion>4.0.0</modelVersion>
5   <parent>
6     <groupId>org.springframework.boot</groupId>
7     <artifactId>spring-boot-starter-parent</artifactId>
8     <version>2.7.10</version>
9     <relativePath/> <!-- Lookup parent from repository -->
10  </parent>
11  <groupId>com.soap</groupId>
12  <artifactId>SpringBootSoap</artifactId>
13  <version>0.0.1-SNAPSHOT</version>
14  <name>SpringBootSoap</name>
15  <description>Spring Boot Soap Client</description>
16  <properties>
17    <java.version>17</java.version>
18  </properties>
19  <dependencies>
20    <dependency>
21      <groupId>org.springframework.boot</groupId>
22      <artifactId>spring-boot-starter-web-services</artifactId>
23    </dependency>
24
25    <dependency>
26      <groupId>org.springframework.boot</groupId>
27      <artifactId>spring-boot-devtools</artifactId>
```

Tip

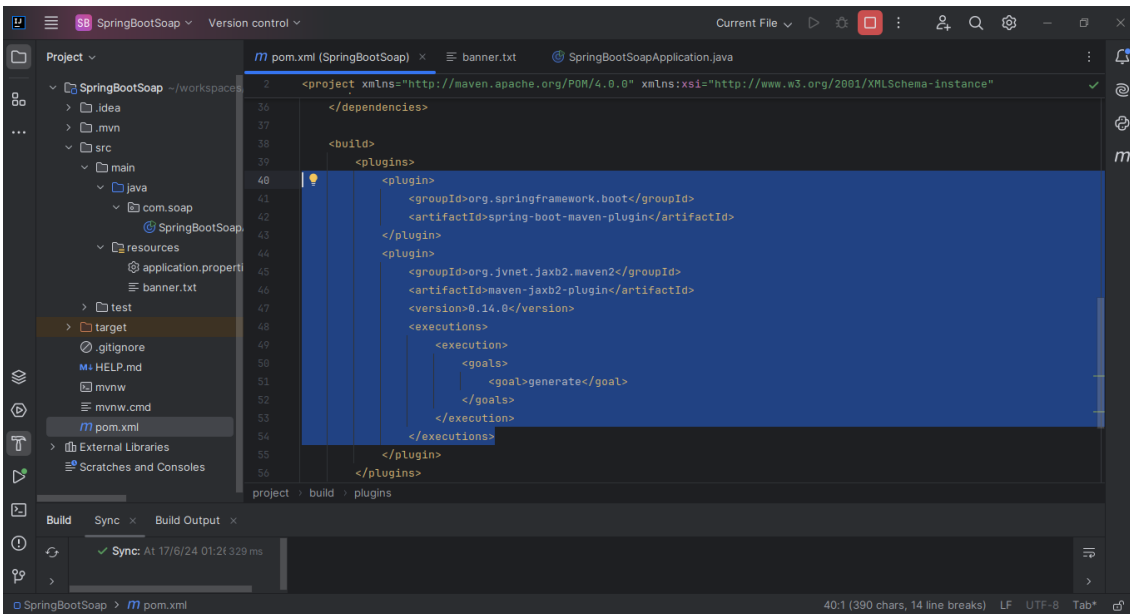
Si deseas puedes crear un banner personalizado para tu aplicación, para ello puedes seguir los pasos en el siguiente enlace <https://devops.datenkollektiv.de/banner.txt/index.html> escribes el nombre de tu aplicación y copias el banner generado en el archivo **banner.txt** que se encuentra en la carpeta **resources**.



2. Corremos nuestra aplicación para comprobar que todo está funcionando correctamente.



3. Ahora vamos a configurar nuestro archivo pom.xml para agregar un plugin que nos permita generar las clases necesarias para consumir el servicio web Calculator.



Si todo salio bien, comparemos si nuestro archivo pom.xml quedo de la siguiente manera:

```

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/m
<modelVersion>4.0.0</modelVersion>
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
      
```

```

    <version>2.7.10</version>
    <relativePath/> <!-- lookup parent from repository -->
</parent>
<groupId>com.soap</groupId>
<artifactId>SpringBootSoap</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>SpringBootSoap</name>
<description>Spring Boot Soap Client</description>
<properties>
    <java.version>17</java.version>
</properties>
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web-services</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
        <optional>>true</optional>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.glassfish.jaxb</groupId>
        <artifactId>jaxb-runtime</artifactId>
    </dependency>
</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
        <plugin>
            <groupId>org.jvnet.jaxb2.maven2</groupId>
            <artifactId>maven-jaxb2-plugin</artifactId>
            <version>0.14.0</version>
            <executions>
                <execution>
                    <goals>
                        <goal>generate</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>

```

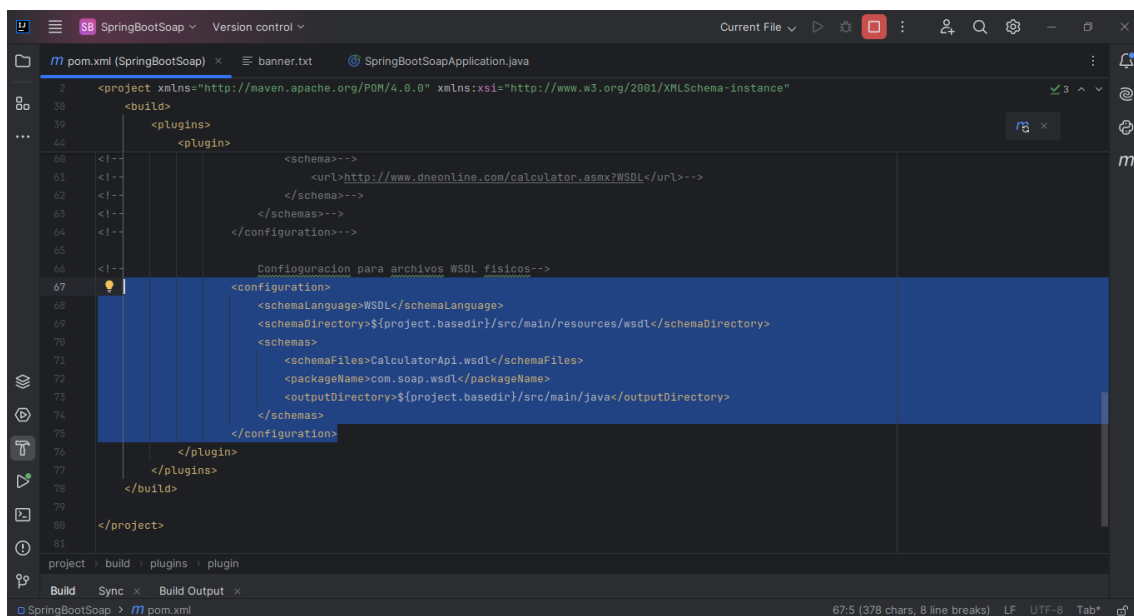


```

        </goals>
    </execution>
</executions>
<configuration>
    <schemaLanguage>WSDL</schemaLanguage>
    <generateDirectory>${project.basedir}/src/main/java</generateDirector
    <generatePackage>com.soap.wsdl</generatePackage>
    <schemas>
        <schema>
            <url>http://www.dneonline.com/calculator.asmx?WSDL</url>
        </schema>
    </schemas>
</configuration>
</plugin>
</plugins>
</build>
</project>

```

En algunas ocasiones posiblemente nos entreguen en físico el archivo **calculator.wsdl** que contiene la descripción del servicio web, en este caso debemos agregarlo en la carpeta **resources** del proyecto, podemos crear una carpeta llamada **wsdl** y agregar el archivo **calculator.wsdl**. Luego debemos modificar el archivo **pom.xml** para que tome este archivo y genere las clases necesarias.



La sección de configuración cambiaría de la siguiente manera:

```

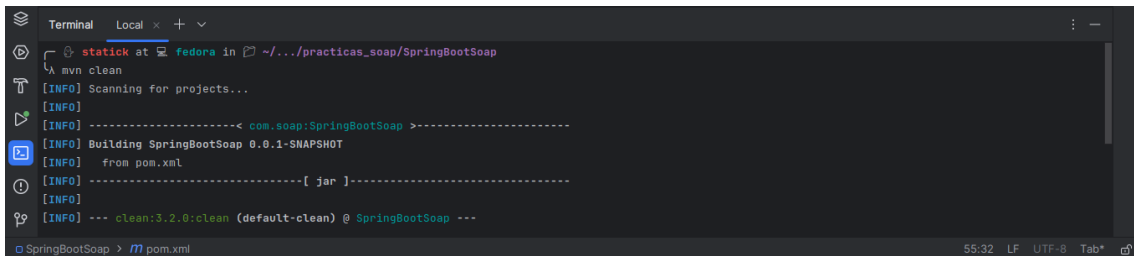
<configuration>
  <schemaLanguage>WSDL</schemaLanguage>
  <schemaDirectory>${project.basedir}/src/main/resources/wsdl</schemaDirectory>

```

```
<schemas>
  <schemaFiles>CalculatorApi.wsdl</schemaFiles>
  <packageName>com.soap.wsdl</packageName>
  <outputDirectory>${project.basedir}/src/main/java</outputDirectory>
</schemas>
</configuration>
```

4. Vamos a limpiar nuestro proyecto con el siguiente comando:

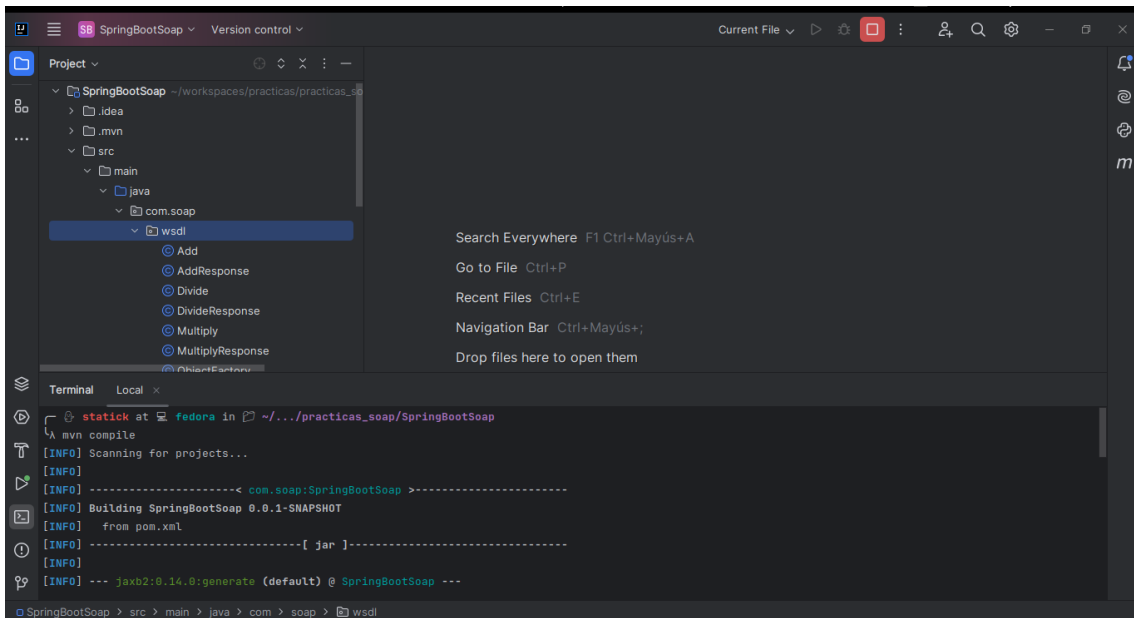
```
mvn clean
```



A terminal window showing the execution of the 'mvn clean' command. The output includes: [INFO] Scanning for projects..., [INFO] Building SpringBootSoap 0.0.1-SNAPSHOT from pom.xml, [INFO] [jar], and [INFO] --- clean:3.2.0:clean (default-clean) @ SpringBootSoap ---

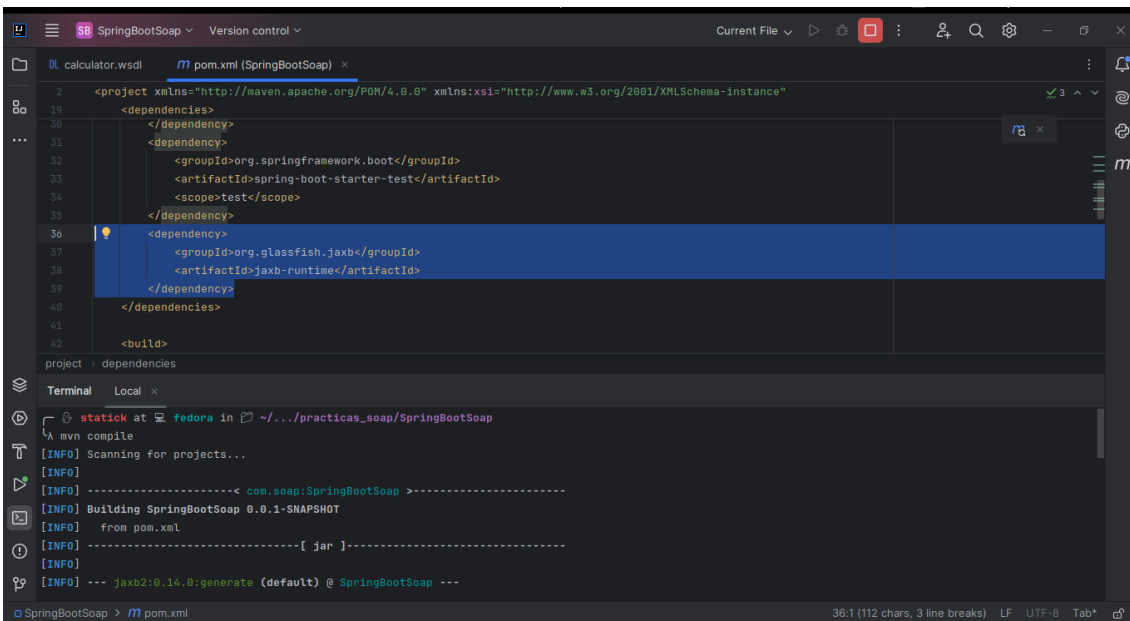
5. Ahora vamos a compilar nuestro proyecto con el siguiente comando:

```
mvn compile
```

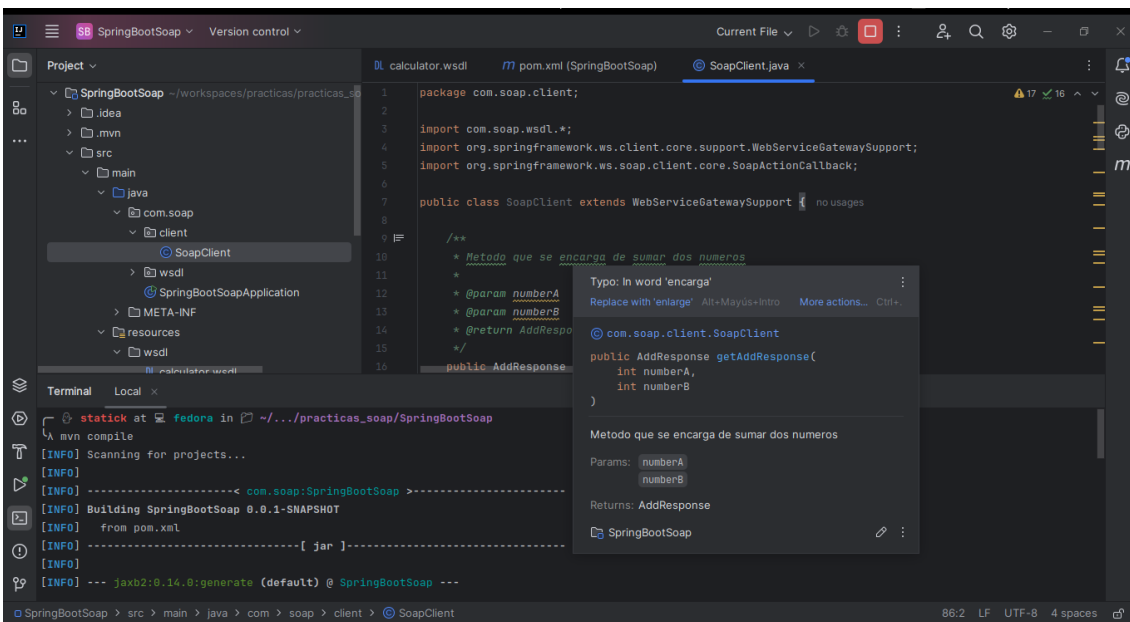


The screenshot shows an IDE with the project structure on the left and a terminal window at the bottom. The project structure includes: SpringBootSoap -> .idea, .mvn, src -> main -> java -> com.soap -> wsdl. The terminal shows the execution of 'mvn compile' with output: [INFO] Scanning for projects..., [INFO] Building SpringBootSoap 0.0.1-SNAPSHOT from pom.xml, [INFO] [jar], and [INFO] --- jaxb2:0.14.0:generate (default) @ SpringBootSoap ---

6. Si todo salió bien, podemos ver que se generaron las clases necesarias para consumir el servicio web Calculator.



7. Ahora vamos a crear un paquete llamado **com.soap.client** y dentro de este paquete vamos a crear una clase llamada **SoapClient**.



En esta clase vamos a agregar el código necesario para consumir el servicio web Calculador.

Empezamos con el método AddResponse que se encarga de sumar dos números.

```
package com.soap.client;

import com.soap.wsdl.*;
import org.springframework.ws.client.core.support.WebServiceGatewaySupport;
import org.springframework.ws.soap.client.core.SoapActionCallback;
```

```

public class SoapClient extends WebServiceGatewaySupport {

    /**
     * Metodo que se encarga de sumar dos numeros
     *
     * @param numberA
     * @param numberB
     * @return AddResponse
     */
    public AddResponse getAddResponse(int numberA, int numberB) { ①

        Add addRequest = new Add(); ②
        addRequest.setIntA(numberA); ③
        addRequest.setIntB(numberB); ④

        SoapActionCallback soapActionCallback = new SoapActionCallback("http://tempuri.org/");

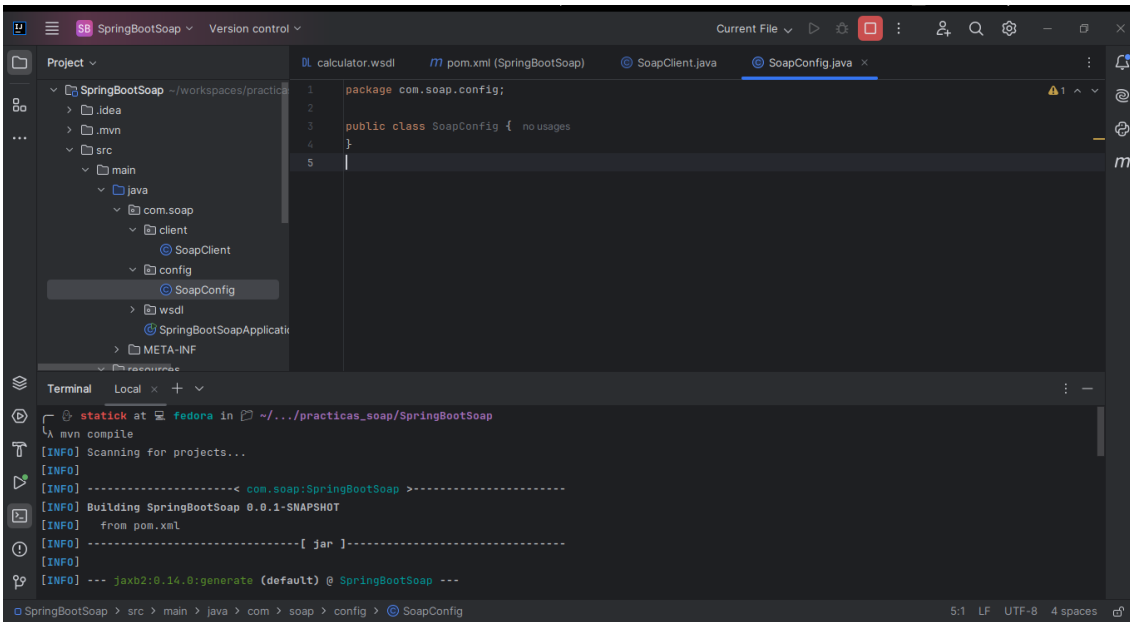
        AddResponse addResponse = (AddResponse) getWebServiceTemplate().marshalSendAndReceive(addRequest, soapActionCallback);

        return addResponse; ⑦
    }
}

```

- ① Creamos el método **getAddResponse** que recibe dos números enteros y retorna un objeto de tipo **AddResponse**.
- ② Creamos un objeto de tipo **Add** que representa la petición de sumar dos números.
- ③ Seteamos el primer número en la petición.
- ④ Seteamos el segundo número en la petición.
- ⑤ Creamos un objeto de tipo **SoapActionCallback** que representa la acción SOAP que se va a realizar.
- ⑥ Realizamos la petición al servicio web y obtenemos la respuesta.
- ⑦ Retornamos la respuesta.

Ahora vamos a crear el paquete **com.soap.config** y dentro de este paquete vamos a crear una clase llamada **SoapConfig**.



Dentro de esta clase vamos a agregar la configuración necesaria para consumir el servicio web Calculator.

```
package com.soap.config;

import com.soap.client.SoapClient;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.oxm.jaxb.Jaxb2Marshaller;

@Configuration
public class SoapConfig {

    @Bean
    public Jaxb2Marshaller marshaller(){
        Jaxb2Marshaller marshaller = new Jaxb2Marshaller();
        marshaller.setContextPath("com.soap.wsdl");
        return marshaller;
    }

    @Bean
    public SoapClient getSoapClient(Jaxb2Marshaller marshaller){
        SoapClient soapClient = new SoapClient();
        soapClient.setDefaultUri("http://www.dneonline.com/calculator.asmx");
        soapClient.setMarshaller(marshaller);
        soapClient.setUnmarshaller(marshaller);

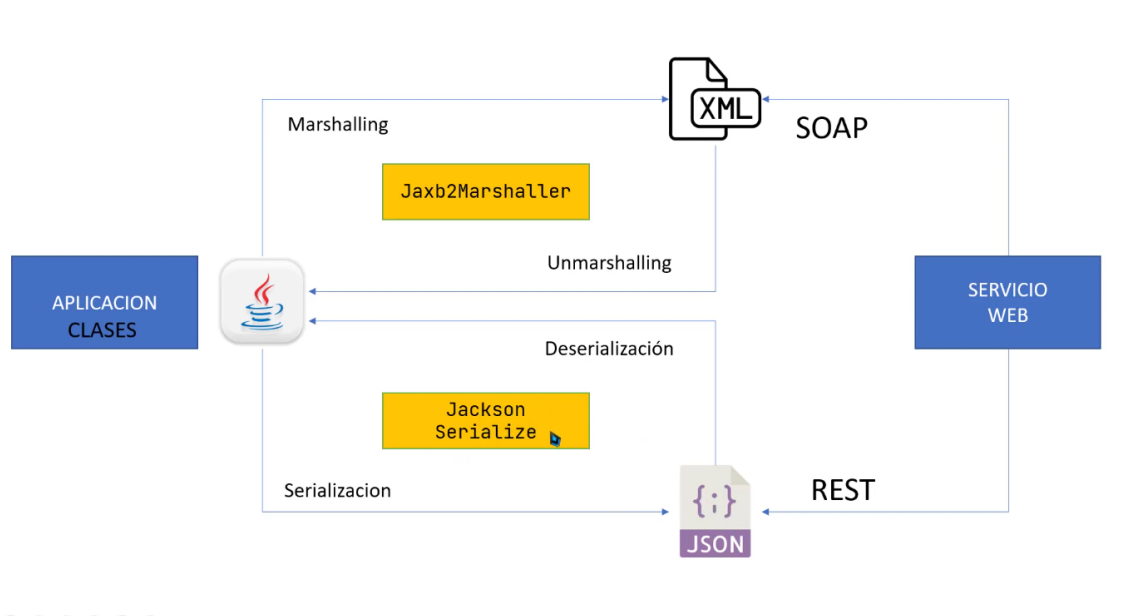
        return soapClient;
    }
}
```

- ① Importamos las clases necesarias.
- ② Importamos la clase **Jaxb2Marshaller** que se encarga de convertir los objetos de Java a XML y viceversa.
- ③ Anotamos la clase con (**Configuration?**) para indicar que es una clase de configuración.
- ④ Creamos un bean de tipo **Jaxb2Marshaller**.
- ⑤ Creamos un objeto de tipo **Jaxb2Marshaller**.
- ⑥ Seteamos el contexto del paquete donde se encuentran las clases generadas a partir del archivo WSDL.
- ⑦ Retornamos el objeto **Jaxb2Marshaller**.
- ⑧ Creamos un bean de tipo **SoapClient**.
- ⑨ Creamos un método que recibe un objeto de tipo **Jaxb2Marshaller** y retorna un objeto de tipo **SoapClient**.
- ⑩ Creamos un objeto de tipo **SoapClient**.
- ⑪ Seteamos la URL del servicio web.
- ⑫ Seteamos el marshaller en el objeto **SoapClient**.
- ⑬ Seteamos el unmarshaller en el objeto **SoapClient**.
- ⑭ Retornamos el objeto **SoapClient**.

En esta clase creamos un bean de tipo **Jaxb2Marshaller** que se encarga de convertir los objetos de Java a XML y viceversa.

También creamos un bean de tipo **SoapClient** que se encarga de consumir el servicio web Calculator.

31.5.2 ¿Qué es Unmarshalling y Marshalling?



Marshalling es un concepto que se utiliza en la serialización de objetos, es decir, convertir un objeto en un formato que se pueda almacenar o transmitir. Unmarshaller es el proceso

inverso, es decir, convertir un objeto en un formato que se pueda utilizar en la aplicación. Recordemos que SOAP trabaja con archivos XML.

También es importante conocer que cuando trabajamos con servicios REST quien utiliza los archivos JSON, en este caso se utiliza el concepto de Deserialización y Serialización.

Puedes encontrar más información en el siguiente enlace <https://www.baeldung.com/jaxb>.

Para probar nuestra aplicación y verificar que todo está resultando bien vamos al archivo **SpringBootTestApplication** y agregamos el siguiente código:

```
package com.soap;

import com.soap.client.SoapClient;
import com.soap.wsdl.AddResponse;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

@SpringBootApplication
public class SpringBootTestApplication {

    private static final Logger LOGGER = LoggerFactory.getLogger(SpringBootTestApplication.class);

    public static void main(String[] args) {
        SpringApplication.run(SpringBootTestApplication.class, args);
    }

    @Bean
    CommandLineRunner init(SoapClient soapClient){
        return args -> {

            AddResponse addResponse = soapClient.getAddResponse(2, 2);

            LOGGER.info("El resultado de la suma de los numeros {} y {} es: {}", 2, 2, addResponse);
        };
    }
}
```

- ① Creamos un objeto de tipo **Logger** para imprimir mensajes en la consola.
- ② Creamos un bean de tipo **CommandLineRunner**.
- ③ Creamos un método que recibe un objeto de tipo **SoapClient** y retorna un objeto de tipo **CommandLineRunner**.
- ④ Creamos un objeto de tipo **CommandLineRunner**.
- ⑤ Realizamos una petición al servicio web para sumar dos números.

⑥ Imprimimos el resultado de la suma en la consola.

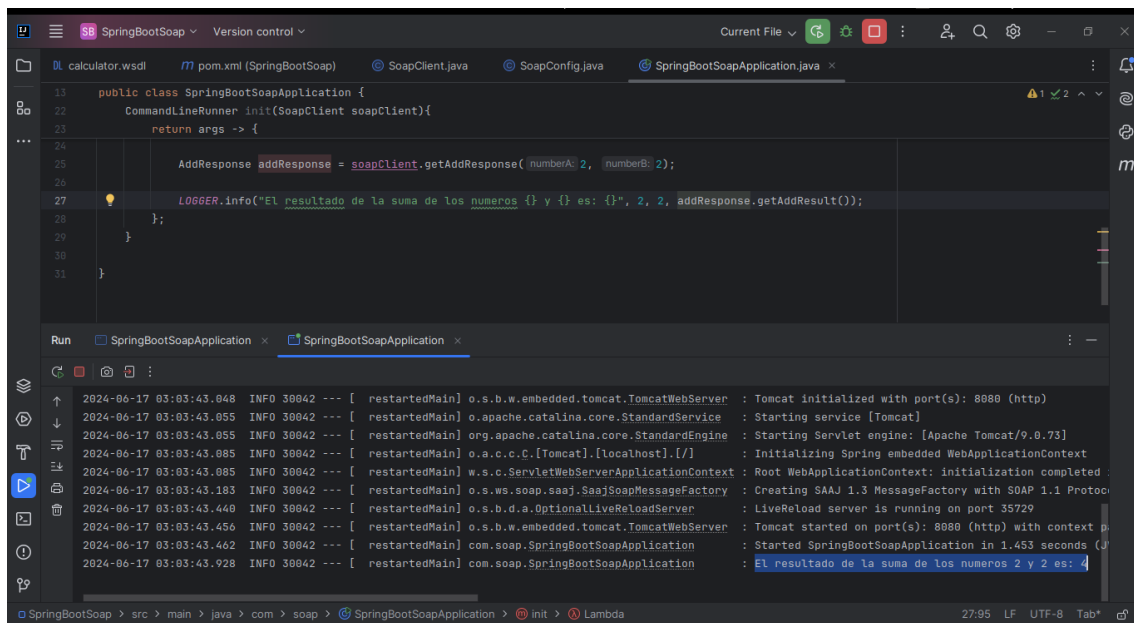
En nuestra aplicación principal **SpringBootSoapApplication** agregamos un bean de tipo **CommandLineRunner** que se encarga de ejecutar el método **init** al iniciar la aplicación.

Este método se encarga de consumir el servicio web Calculator y sumar dos números.

31.5.3 Ejecutar la aplicación

Ahora vamos a ejecutar nuestra aplicación para comprobar que todo está funcionando correctamente.

1. Corremos nuestra aplicación.



The screenshot shows an IDE with the following code in `SpringBootSoapApplication.java`:

```
11 public class SpringBootSoapApplication {
22     CommandLineRunner init(SoapClient soapClient){
23         return args -> {
24
25             AddResponse addResponse = soapClient.getAddResponse(numberA: 2, numberB: 2);
26
27             LOGGER.info("El resultado de la suma de los numeros {} y {} es: {}", 2, 2, addResponse.getAddResult());
28         };
29     }
30 }
31 }
```

The Run console shows the following output:

```
2024-06-17 03:03:43.048 INFO 30842 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2024-06-17 03:03:43.055 INFO 30842 --- [ restartedMain] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2024-06-17 03:03:43.055 INFO 30842 --- [ restartedMain] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.73]
2024-06-17 03:03:43.085 INFO 30842 --- [ restartedMain] o.a.c.c.c.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2024-06-17 03:03:43.085 INFO 30842 --- [ restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed
2024-06-17 03:03:43.183 INFO 30842 --- [ restartedMain] o.s.ws.soap.saa1.Saa1SoapMessageFactory : Creating SAAJ 1.3 MessageFactory with SOAP 1.1 Protocol
2024-06-17 03:03:43.440 INFO 30842 --- [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is running on port 35729
2024-06-17 03:03:43.456 INFO 30842 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path
2024-06-17 03:03:43.462 INFO 30842 --- [ restartedMain] com.soap.SpringBootSoapApplication : Started SpringBootSoapApplication in 1.453 seconds
2024-06-17 03:03:43.928 INFO 30842 --- [ restartedMain] com.soap.SpringBootSoapApplication : El resultado de la suma de los numeros 2 y 2 es: 4
```

2. Verificamos que el resultado de la suma sea correcto.

En la línea de comandos debemos ver el siguiente mensaje:

```
El resultado de la suma de los numeros 2 y 2 es: 4
```

Para desarrollar los demás métodos es necesario comentar las líneas que acabamos de ingresar en el archivo **SpringBootSoapApplication** y agregar el siguiente código:

```
package com.soap;

import com.soap.client.SoapClient;
import com.soap.wsdl.AddResponse;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.boot.CommandLineRunner;
```



```

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

@SpringBootApplication
public class SpringBootSoapApplication {

    // private static final Logger LOGGER = LoggerFactory.getLogger(SpringBootSoapApplication.class);

    public static void main(String[] args) {
        SpringApplication.run(SpringBootSoapApplication.class, args);
    }

    // @Bean
    // CommandLineRunner init(SoapClient soapClient){
    //     return args -> {
    //         //
    //         AddResponse addResponse = soapClient.getAddResponse(2, 2);
    //         //
    //         LOGGER.info("El resultado de la suma de los numeros {} y {} es: {}", 2, 2, addResponse);
    //     };
    // }
}

```

Ahora vamos a el metodo para restar dos números, para ello vamos a modificar la clase **SoapClient**.

```

/**
 * Metodo que encarga de restar dos numero
 * @param numberA
 * @param numberB
 * @return SubtractResponse
 */
public SubtractResponse getSubtractResponse(int numberA, int numberB) {

    Subtract subtractRequest = new Subtract();           ①
    subtractRequest.setIntA(numberA);                     ②
    subtractRequest.setIntB(numberB);                     ③

    SoapActionCallback soapActionCallback = new SoapActionCallback("http://tempuri.org/");

    SubtractResponse subtractResponse = (SubtractResponse) getWebServiceTemplate().marshalSendAndReceive(soapActionCallback, subtractRequest);

    return subtractResponse;                             ⑥
}

```

① Creamos un objeto de tipo **Subtract** que representa la petición de restar dos números.

- ② Seteamos el primer número en la petición.
- ③ Seteamos el segundo número en la petición.
- ④ Creamos un objeto de tipo **SoapActionCallback** que representa la acción SOAP que se va a realizar.
- ⑤ Realizamos la petición al servicio web y obtenemos la respuesta.
- ⑥ Retornamos la respuesta.

31.6 Crear el Controlador

Vamos a crear un controlador que se encargue de recibir las peticiones y llamar a los métodos correspondientes en el cliente SOAP.

1. Creamos un paquete llamado **com.soap.controller** y dentro de este paquete vamos a crear una clase llamada **SoapController**.

```

16 public class SoapController {
17
18     @Autowired 2 usages
19     private SoapClient soapClient;
20
21     @PostMapping("/sumar") no usages
22     public ResponseEntity<?> add(@RequestParam int numberA, @RequestParam int numberB) {
23
24         AddResponse addResponse = soapClient.getAddResponse(numberA, numberB);
25
26         Map<String, Integer> response = new HashMap<>();
27         response.put("resultado", addResponse.getAddResult());
28         return ResponseEntity.ok().body(response);
29     }
30
31     @PostMapping("/restar") no usages
32     public ResponseEntity<?> subtract(@RequestParam int numberA, @RequestParam int numberB) {
33         SubtractResponse subtractResponse = soapClient.getSubtractResponse(numberA, numberB);
34
35         Map<String, Integer> response = new HashMap<>();
36         response.put("resultado", subtractResponse.getSubtractResult());
37         return ResponseEntity.ok().body(response);
38     }
39 }

```

En esta clase vamos a agregar el código necesario para crear un controlador que se encargue de recibir las peticiones y llamar a los métodos correspondientes en el cliente SOAP.

```

package com.soap.controller;

import com.soap.client.SoapClient;
import com.soap.wsdl.AddResponse;
import com.soap.wsdl.SubtractResponse;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

import java.util.HashMap;

```

```

import java.util.Map;

@RestController
public class SoapController {

    @Autowired
    private SoapClient soapClient;

    @PostMapping("/sumar")
    public ResponseEntity<?> add(@RequestParam int numberA, @RequestParam int numberB) {

        AddResponse addResponse = soapClient.getAddResponse(numberA, numberB);

        Map<String, Integer> response = new HashMap<>();
        response.put("resultado", addResponse.getAddResult());
        return ResponseEntity.ok().body(response);

    }

    @PostMapping("/restar")
    public ResponseEntity<?> subtract(@RequestParam int numberA, @RequestParam int numberB) {

        SubtractResponse subtractResponse = soapClient.getSubtractResponse(numberA, numberB);

        Map<String, Integer> response = new HashMap<>();
        response.put("resultado", subtractResponse.getSubtractResult());
        return ResponseEntity.ok().body(response);

    }
}

```

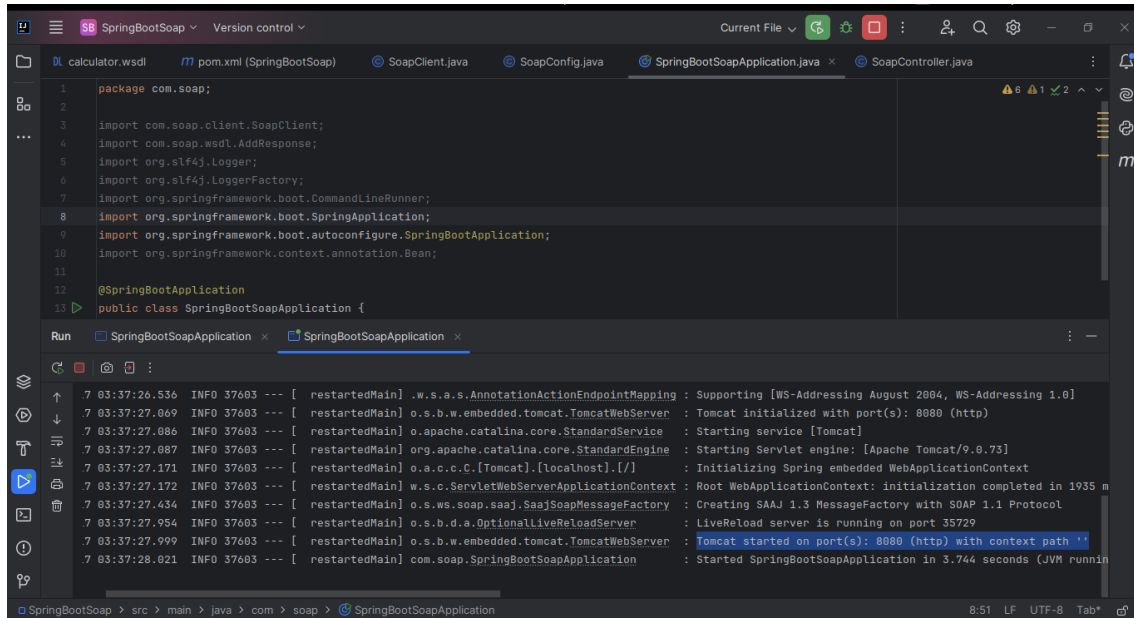
- ① Creamos un endpoint que recibe dos números y retorna la suma de los mismos.
- ② Creamos un método que recibe dos números y retorna un objeto de tipo **ResponseEntity**.
- ③ Realizamos una petición al servicio web para sumar dos números.
- ④ Creamos un objeto de tipo **Map** para almacenar la respuesta.
- ⑤ Agregamos el resultado de la suma al objeto **Map**.
- ⑥ Retornamos un objeto de tipo **ResponseEntity** con el resultado de la suma.
- ⑦ Creamos un endpoint que recibe dos números y retorna la resta de los mismos.
- ⑧ Creamos un método que recibe dos números y retorna un objeto de tipo **ResponseEntity**.
- ⑨ Realizamos una petición al servicio web para restar dos números.
- ⑩ Creamos un objeto de tipo **Map** para almacenar la respuesta.
- ⑪ Agregamos el resultado de la resta al objeto **Map**.
- ⑫ Retornamos un objeto de tipo **ResponseEntity** con el resultado de la resta.

En esta clase creamos un controlador que se encarga de recibir las peticiones y llamar a los métodos correspondientes en el cliente SOAP.

31.7 Ejecutar la aplicación

Ahora vamos a ejecutar nuestra aplicación para comprobar que todo está funcionando correctamente.

1. Corremos nuestra aplicación.



```
1 package com.soap;
2
3 import com.soap.client.SoapClient;
4 import com.soap.wsdl.AddResponse;
5 import org.slf4j.Logger;
6 import org.slf4j.LoggerFactory;
7 import org.springframework.boot.CommandLineRunner;
8 import org.springframework.boot.SpringApplication;
9 import org.springframework.boot.autoconfigure.SpringBootApplication;
10 import org.springframework.context.annotation.Bean;
11
12 @SpringBootApplication
13 public class SpringBootSoapApplication {
14
15     @Bean
16     CommandLineRunner soapClient() {
17         return new SoapClient();
18     }
19
20     public static void main(String[] args) {
21         SpringApplication.run(SpringBootSoapApplication.class, args);
22     }
23 }
```

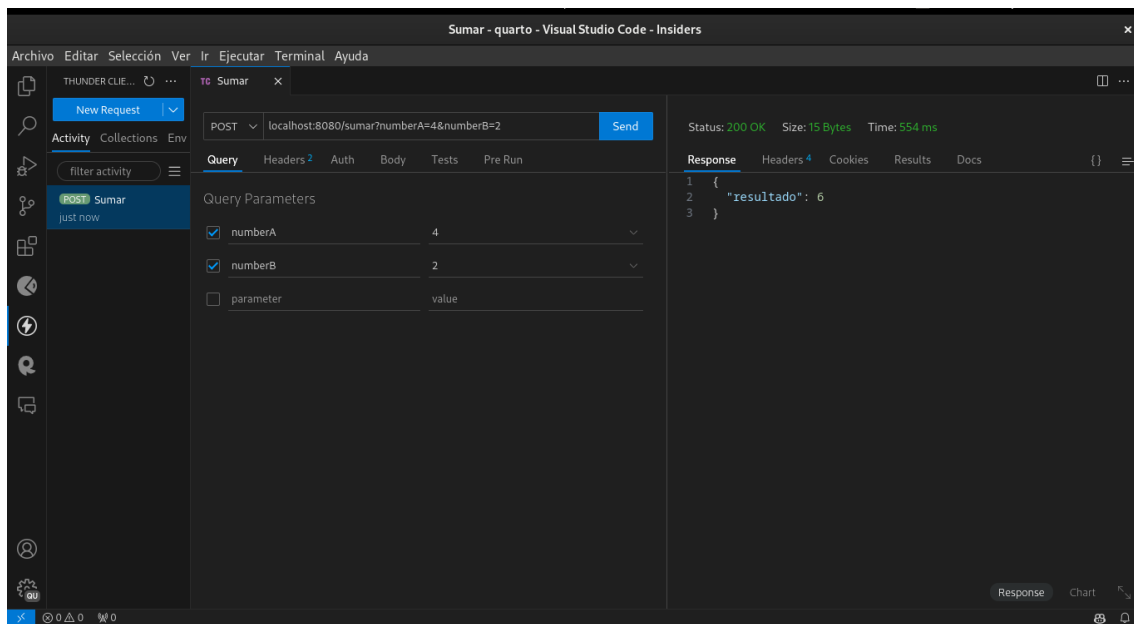
```
7 03:37:26.536 INFO 37603 --- [ restartedMain] .w.s.a.s.AnnotationActionEndpointMapping : Supporting [WS-Addressing August 2004, WS-Addressing 1.0]
7 03:37:27.069 INFO 37603 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
7 03:37:27.086 INFO 37603 --- [ restartedMain] o.apache.catalina.core.StandardService : Starting service [Tomcat]
7 03:37:27.087 INFO 37603 --- [ restartedMain] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.73]
7 03:37:27.171 INFO 37603 --- [ restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
7 03:37:27.172 INFO 37603 --- [ restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1935 ms
7 03:37:27.434 INFO 37603 --- [ restartedMain] o.s.ws.soap.saaJ.SaaJSoapMessageFactory : Creating SAAJ 1.3 MessageFactory with SOAP 1.1 Protocol
7 03:37:27.954 INFO 37603 --- [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is running on port 35729
7 03:37:27.999 INFO 37603 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
7 03:37:28.021 INFO 37603 --- [ restartedMain] com.soap.SpringBootSoapApplication : Started SpringBootSoapApplication in 3.744 seconds (JVM running for 10.047s)
```

Podemos ver que nuestra aplicación se está ejecutando correctamente en el puerto 8080.

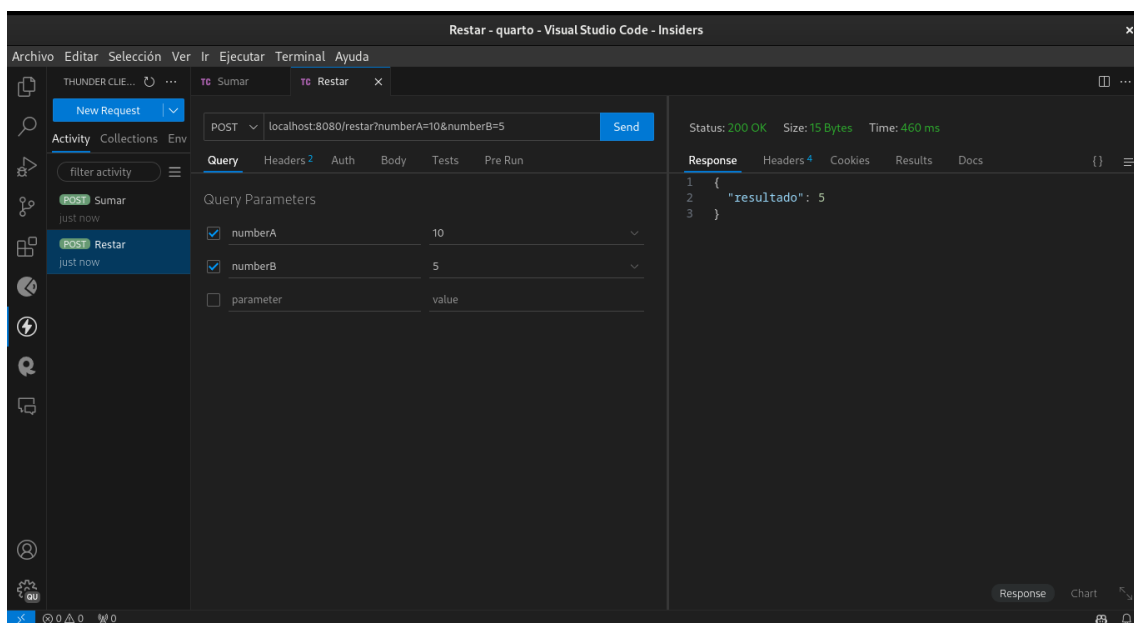
Para probar los endpoints que creamos vamos a utilizar Thunder Client, una extensión de Visual Studio Code que nos permite hacer peticiones HTTP.

2. Abrimos Visual Studio Code y buscamos la extensión Thunder Client.

31.7.1 Probar el endpoint de suma



31.7.2 Probar el endpoint de resta



Como podemos observar, nuestra aplicación está funcionando correctamente y podemos consumir el servicio web Calculator.

31.8 Reto

1. Crea un método en el cliente SOAP que se encargue de multiplicar dos números.

2. Crea un endpoint en el controlador que reciba dos números y retorne el resultado de la multiplicación.
3. Ejecuta la aplicación y prueba el endpoint de multiplicación.

Ver solución

```

/**
 * Metodo que encarga de multiplicar dos numero
 * @param numberA
 * @param numberB
 * @return MultiplyResponse
 */
public MultiplyResponse getMultiplyResponse(int numberA, int numberB) { ①

    Multiply multiplyRequest = new Multiply(); ②
    multiplyRequest.setIntA(numberA); ③
    multiplyRequest.setIntB(numberB); ④

    SoapActionCallback soapActionCallback = new SoapActionCallback("http://tempuri.or

    return (MultiplyResponse) getWebServiceTemplate().marshalSendAndReceive("http://w
}
}

```

- ① Creamos el método **getMultiplyResponse** que recibe dos números enteros y retorna un objeto de tipo **MultiplyResponse**.
- ② Creamos un objeto de tipo **Multiply** que representa la petición de multiplicar dos números.
- ③ Seteamos el primer número en la petición.
- ④ Seteamos el segundo número en la petición.
- ⑤ Creamos un objeto de tipo **SoapActionCallback** que representa la acción SOAP que se va a realizar.
- ⑥ Realizamos la petición al servicio web y obtenemos la respuesta.

```

@PostMapping("/multiplicar") ①
public ResponseEntity<?> multiply(@RequestParam int numberA, @RequestParam int numberB) {
    MultiplyResponse multiplyResponse = soapClient.getMultiplyResponse(numberA, numberB);

    Map<String, Integer> response = new HashMap<>(); ④
    response.put("resultado", multiplyResponse.getMultiplyResult()); ⑤
    return ResponseEntity.ok().body(response); ⑥
}
}

```

- ① Creamos un endpoint que recibe dos números y retorna el resultado de la multiplicación.
- ② Creamos un método que recibe dos números y retorna un objeto de tipo **ResponseEntity**.
- ③ Realizamos una petición al servicio web para multiplicar dos números.

- ④ Creamos un objeto de tipo **Map** para almacenar la respuesta.
- ⑤ Agregamos el resultado de la multiplicación al objeto **Map**.
- ⑥ Retornamos un objeto de tipo **ResponseEntity** con el resultado de la multiplicación.

31.9 Referencias

- <https://www.baeldung.com/spring-soap-web-service>
- <https://www.youtube.com/watch?v=GLynmR08HxU>
- https://github.com/statick88/calculator_soap

...

32 Laboratorio de API Rest con Spring Boot.

32.1 Objetivo

El objetivo de este laboratorio es crear una API Rest con Spring Boot que permita realizar operaciones CRUD sobre una entidad de dominio, para ello utilizaremos Spring Data JPA y Lombok, levantaremos un contenedor de Docker con MySQL y utilizaremos el cliente Thunder Client para probar los servicios.

32.2 Requisitos

- JDK 17
- Maven 3.3.0
- IDE (IntelliJ IDEA, Eclipse, NetBeans, etc.)

32.3 Dependencias

- Spring Web
- Spring Data JPA
- Mysql Driver
- Lombok
- DevTools

32.4 Conceptos a aprender

32.4.1 API Rest

Una API Rest es una interfaz de programación de aplicaciones que utiliza el protocolo HTTP para realizar operaciones CRUD sobre recursos.

32.4.2 API Restful

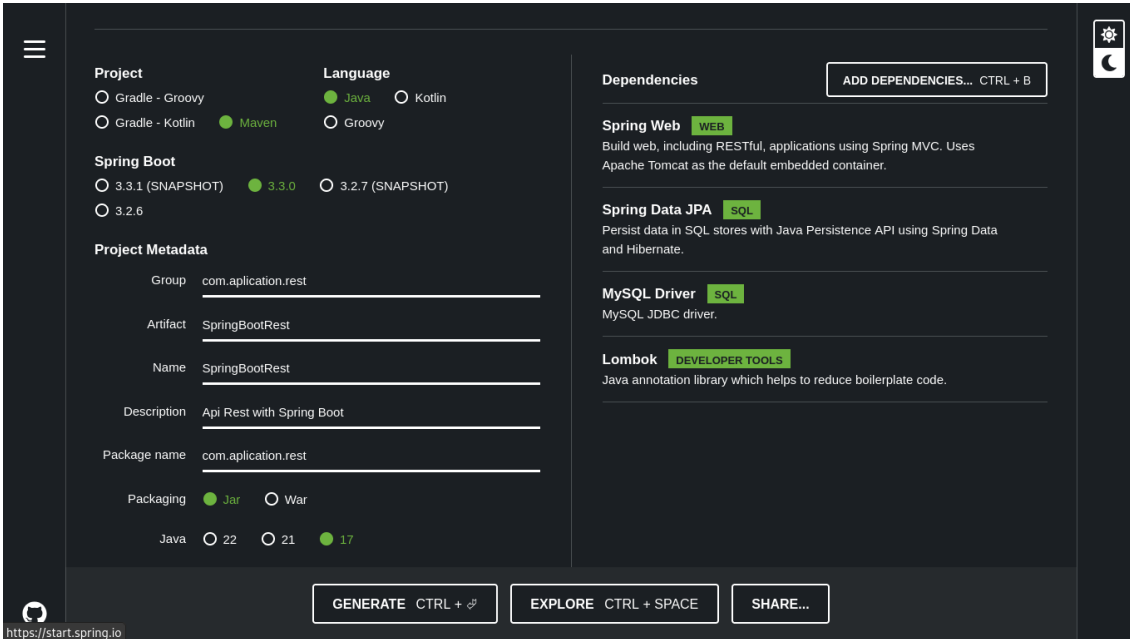
Una API Restful es una API Rest que sigue los principios de la arquitectura REST.

32.4.3 Rest

REST (Representational State Transfer) es un estilo de arquitectura de software que define un conjunto de restricciones para el diseño de servicios web.

32.5 Crear un proyecto Spring Boot

Para crear un proyecto Spring Boot, se puede utilizar el [Spring Initializr](#).



The screenshot shows the Spring Initializr web application interface. It is divided into several sections:

- Project:** Radio buttons for `Gradle - Groovy`, `Gradle - Kotlin`, and `Maven` (selected).
- Language:** Radio buttons for `Java` (selected), `Kotlin`, and `Groovy`.
- Spring Boot:** Radio buttons for `3.3.1 (SNAPSHOT)`, `3.3.0` (selected), and `3.2.7 (SNAPSHOT)`.
- Project Metadata:** Text input fields for `Group` (com.application.rest), `Artifact` (SpringBootRest), `Name` (SpringBootRest), `Description` (Api Rest with Spring Boot), and `Package name` (com.application.rest).
- Packaging:** Radio buttons for `Jar` (selected) and `War`.
- Java:** Radio buttons for `22`, `21`, and `17` (selected).
- Dependencies:** A list of selected dependencies with their categories: `Spring Web` (WEB), `Spring Data JPA` (SQL), `MySQL Driver` (SQL), and `Lombok` (DEVELOPER TOOLS). A button `ADD DEPENDENCIES... CTRL + B` is present.

At the bottom, there are buttons for `GENERATE CTRL + G`, `EXPLORE CTRL + SPACE`, and `SHARE...`. The URL `https://start.spring.io` is visible in the bottom left corner.

1. Ingresar a [Spring Initializr](#).
2. Completar los campos del formulario.
 - Project: Maven Project
 - Language: Java
 - Spring Boot: 3.1.1
 - Group: com.application.rest
 - Artifact: SpringBootRest
 - Name: SpringBootRest
 - Description: Api Rest with Spring Boot
 - Package name: com.application.rest
 - Packaging: Jar
 - Java: 17
 - Dependencies: Spring Web, Spring Data JPA, Mysql Driver, Lombok, DevTools

3. Hacer clic en el botón Generate.

Puedes utilizar el siguiente [link](#) para generar el proyecto.

4. Descomprimir el archivo zip descargado.
5. Importar el proyecto en el IDE.

32.6 Configurar la conexión a la base de datos

Para configurar la conexión a la base de datos, se debe modificar el archivo **application.properties**.

1. Abrir el archivo **application.properties**.

```
# Configuración de la Base de Datos
spring.datasource.url=jdbc:mysql://localhost:3306/rest_api_db
spring.datasource.username=root
spring.datasource.password=150919

## Configuración de Hibernate
spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect
spring.jpa.hibernate.ddl-auto=create-drop
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
```

33 Configuración de la Base de Datos

Para esta base de datos tenemos 2 opciones:

1. Crear un contenedor de Docker con MySQL.
2. Crear una base de datos en MySQL de forma nativa

En este laboratorio vamos a utilizar la opción 1.

33.0.1 Crear un contenedor de Docker con MySQL

1. Para crear el contenedor vamos a utilizar `docker-compose.yml`

```
version: '3.8'

services:
  db:
    container_name: mysql-db
    image: mysql:8
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: 150919
      MYSQL_DATABASE: rest_api_db
    ports:
      - "3306:3306"
    volumes:
      - ./mysql-data:/var/lib/mysql
```

2. Vamos a probar si el contenedor se levanta correctamente.

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

Un resumen de lo que sucede en el comando anterior:

Inicialización de MySQL: El servidor MySQL se inicializa y se crean los archivos de la base de datos.

InnoDB: El motor InnoDB se inicializa satisfactoriamente.

Usuario root sin contraseña: Se crea el usuario `root@localhost` con una contraseña vacía. Esto se indica como una advertencia debido al uso de `-initialize-insecure` en tu configuración.

Inicio del servidor MySQL: El servidor MySQL se inicia y está listo para aceptar conexiones en el puerto 3306.

Advertencias sobre zonas horarias: Hay advertencias sobre la carga de archivos de zonas horarias que no se pueden cargar, pero no afectan la funcionalidad básica del servidor MySQL.

Creación de base de datos: Se crea la base de datos `rest_api_db`.

Detención del servidor temporal: Se detiene el servidor temporal que se inició inicialmente para realizar la inicialización.

MySQL listo para iniciar: El servidor MySQL está completamente iniciado y listo para aceptar conexiones.

3. Verificar si el contenedor se encuentra en ejecución.

```
docker ps
```

```
static at fedora in ~
└─$ docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS                               NAMES
6af0a7406b44  mysql:8  "docker-entrypoint.s..." 4 minutes ago  Up 4 minutes  0.0.0.0:3306->3306/tcp, :::3306->3306/tcp, 33060/tcp  mysql-db
static at fedora in ~
```

4. Conectarse al contenedor de MySQL.

```
docker exec -it mysql-db mysql -u root -p
```

 Tip

El password es **150919**

5. Verificar si la base de datos `rest_api_db` fue creada.

```
show databases;
```

```
statick at fedora in ~
└─λ docker exec -it mysql-db mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 10
Server version: 8.4.0 MySQL Community Server - GPL

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+-----+
| Database                |
+-----+
| information_schema      |
| mysql                   |
| performance_schema     |
| rest_api_db             |
| sys                     |
+-----+
5 rows in set (0.01 sec)

mysql> █
```

6. Salir del contenedor de MySQL.

```
exit
```

33.1 Crear Entidades

1. Crear una nuevo paquete llamado **entities**.
2. Crear una nueva clase llamada **Maker**.

```
package com.aplicacion.rest.entities;

import jakarta.persistence.*;
import lombok.*;

import java.util.ArrayList;
import java.util.List;

@Getter
@Setter
@Builder
```

```

@Entity
@Table(name = "fabricante")
public class Maker {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "nombre")
    private String name;

    @OneToMany(mappedBy = "meker", cascade = CascadeType.ALL, fetch = FetchType.LAZY, orphanRemoval = true)
    private List<Product> productList = new ArrayList<>();
}

```

3. Crear una nueva clase llamada **Product**.

```

import jakarta.persistence.*;
import lombok.*;

import java.math.BigDecimal;

@Getter
@Setter
@Builder
@Entity
@Table(name = "producto")
public class Product {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "nombre")
    private String name;

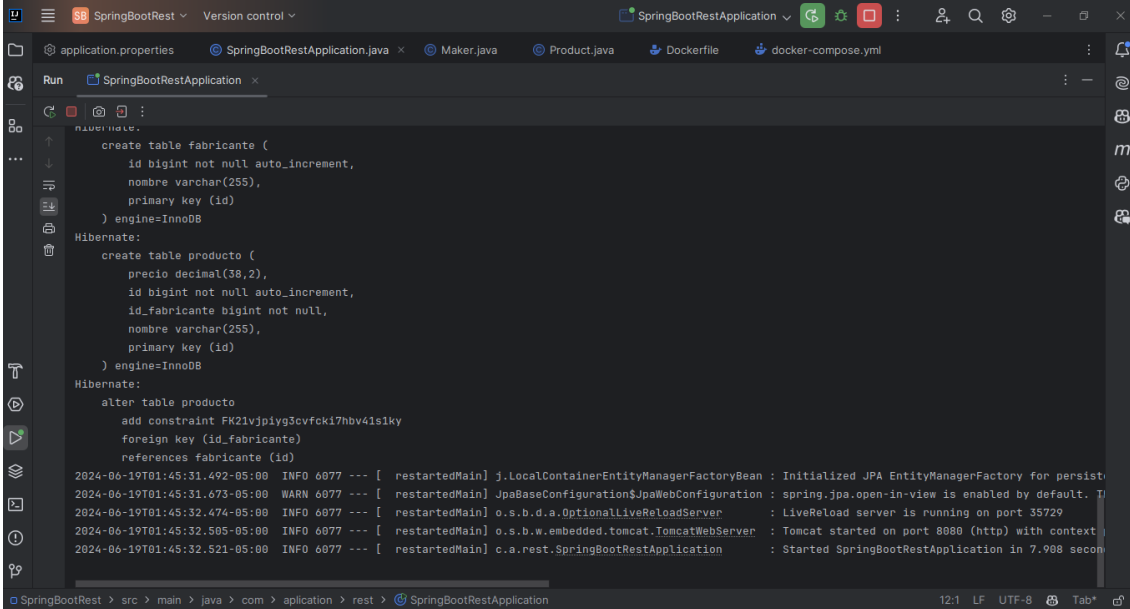
    @Column(name = "precio")
    private BigDecimal price;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "id_fabricante", nullable = false)
    private Maker meker;
}

```

33.2 Probar el Servidor

1. Ejecutar el proyecto y verificamos si el servidor se encuentra en ejecución.



```
create table fabricante (
  id bigint not null auto_increment,
  nombre varchar(255),
  primary key (id)
) engine=InnoDB
Hibernate:
create table producto (
  precio decimal(38,2),
  id bigint not null auto_increment,
  id_fabricante bigint not null,
  nombre varchar(255),
  primary key (id)
) engine=InnoDB
Hibernate:
alter table producto
add constraint FK21vjpiyg3cvfck17hbv41s1ky
foreign key (id_fabricante)
references fabricante (id)
2024-06-19T01:45:31.492-05:00 INFO 6077 --- [ restartedMain] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persist
2024-06-19T01:45:31.673-05:00 WARN 6077 --- [ restartedMain] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. T
2024-06-19T01:45:32.474-05:00 INFO 6077 --- [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is running on port 35729
2024-06-19T01:45:32.505-05:00 INFO 6077 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context
2024-06-19T01:45:32.521-05:00 INFO 6077 --- [ restartedMain] c.a.rest.SpringBootRestApplication : Started SpringBootRestApplication in 7.988 secon
```

2. Verificar si la base de datos `rest_api_db` fue creada y las tablas **fabricante** y **producto** fueron creadas.

```
show databases;
use rest_api_db;
show tables;
```

```

mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql |
| performance_schema |
| rest_api_db |
| sys |
+-----+
5 rows in set (0.00 sec)

mysql> use rest_api_db;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+-----+
| Tables_in_rest_api_db |
+-----+
| fabricante |
| producto |
+-----+
2 rows in set (0.00 sec)

mysql> █

```

3. Vamos a agregar datos en nuestras tablas.

Para ello vamos a crear un archivo **import.sql** en la carpeta **resources**.

```

INSERT INTO fabricante (nombre) VALUES ('Apple');
INSERT INTO fabricante (nombre) VALUES ('Samsung');
INSERT INTO fabricante (nombre) VALUES ('Xiaomi');

INSERT INTO producto (nombre, precio, id_fabricante) VALUES ('iPhone 13', 1000, 1);
INSERT INTO producto (nombre, precio, id_fabricante) VALUES ('Galaxy S21', 800, 2);
INSERT INTO producto (nombre, precio, id_fabricante) VALUES ('Redmi Note 10', 300, 3);

```

4. Ejecutar el proyecto y verificamos si los datos fueron insertados en las tablas.


```
1 INSERT INTO fabricante (nombre) VALUES ('Apple');
2 INSERT INTO fabricante (nombre) VALUES ('Samsung');
3 INSERT INTO fabricante (nombre) VALUES ('Xiaomi');
4
5 INSERT INTO producto (nombre, precio, id_fabricante) VALUES ('iPhone 13', 1000, 1);
6 INSERT INTO producto (nombre, precio, id_fabricante) VALUES ('Galaxy S21', 800, 2);
7 INSERT INTO producto (nombre, precio, id_fabricante) VALUES ('Redmi Note 10', 300, 3);
```

```
hibernate:
alter table producto
add constraint FK21vjpiyg3cvfcki7hbv41s1ky
foreign key (id_fabricante)
references fabricante (id)
Hibernate: INSERT INTO fabricante (nombre) VALUES ('Apple')
Hibernate: INSERT INTO fabricante (nombre) VALUES ('Samsung')
Hibernate: INSERT INTO fabricante (nombre) VALUES ('Xiaomi')
Hibernate: INSERT INTO producto (nombre, precio, id_fabricante) VALUES ('iPhone 13', 1000, 1)
Hibernate: INSERT INTO producto (nombre, precio, id_fabricante) VALUES ('Galaxy S21', 800, 2)
Hibernate: INSERT INTO producto (nombre, precio, id_fabricante) VALUES ('Redmi Note 10', 300, 3)
2024-06-19T01:59:26.640-05:00 INFO 10992 --- [ restartedMain] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persist
2024-06-19T01:59:26.764-05:00 WARN 10992 --- [ restartedMain] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default.
2024-06-19T01:59:27.449-05:00 INFO 10992 --- [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is running on port 35729
2024-06-19T01:59:27.488-05:00 INFO 10992 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context
2024-06-19T01:59:27.510-05:00 INFO 10992 --- [ restartedMain] c.a.rest.SpringBootRestApplication : Started SpringBootRestApplication in 8.429 seconds
```

5. Visualizamos los datos en la base de datos.

```
SELECT * FROM fabricante;
SELECT * FROM producto;
```

```
mysql> SELECT * FROM fabricante;
+----+-----+
| id | nombre |
+----+-----+
|  1 | Apple  |
|  2 | Samsung|
|  3 | Xiaomi |
+----+-----+
3 rows in set (0.00 sec)

mysql> SELECT * FROM producto;
+-----+-----+-----+-----+
| precio | id | id_fabricante | nombre |
+-----+-----+-----+-----+
| 1000.00 | 1 | 1 | iPhone 13 |
|  800.00 | 2 | 2 | Galaxy S21 |
|  300.00 | 3 | 3 | Redmi Note 10 |
+-----+-----+-----+-----+
3 rows in set (0.00 sec)

mysql> █
```

33.3 Creación del API Rest

Tip

En las entidades de nuestro proyecto es necesario agregar una anotación (**JsonBackReference?**) en la entidad **Maker** y **Product** para evitar un error de referencia cíclica.

```
@JoinColumn(name = "id_fabricante", nullable = false)
@JsonBackReference
private Maker maker;
```

①

① En la entidad **Maker** se agrega la anotación **@JsonBackReference**.

```

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "id_fabricante", nullable = false)
    @JsonBackReference
    private Maker maker;

```

①

① En la entidad **Product** se agrega la anotación **@JsonBackReference**.

Ahora vamos a crear un API Rest que permita realizar operaciones CRUD sobre las entidades **Maker** y **Product**.

1. Creamos un paquete llamado **respository**.
2. Creamos una interfaz llamada **MakerRepository**.

```

package com.application.rest.repository;

import com.application.rest.entities.Maker;
import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

import java.util.Optional;

@Repository
public interface MakerRepository extends CrudRepository<Maker, Long>{
}

```

En el caso de la interfaz **MakerRepository** se extiende de **CrudRepository** que es una interfaz de Spring Data JPA que proporciona métodos CRUD para la entidad **Maker**.

3. Creamos una interfaz llamada **ProductRepository**.

```

package com.application.rest.repository;

import com.application.rest.entities.Product;
import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

import java.math.BigDecimal;
import java.util.List;

@Repository
public interface ProductRepository extends CrudRepository<Product, Long> {
    List<Product> findByPriceBetween(BigDecimal minPrice, BigDecimal maxPrice);
}

```

En el caso de la interfaz **ProductRepository** se extiende de **CrudRepository** que es una interfaz de Spring Data JPA que proporciona métodos CRUD para la entidad **Product**.

4. Creamos un paquete llamado **persistence**
5. Creamos una interaz llamada **IMakerDAO**.

```
package com.application.rest.persistence;

import com.application.rest.entities.Maker;

import java.util.List;
import java.util.Optional;

public interface IMakerDAO {

    List<Maker> findAll();

    Optional<Maker> findById(Long id);

    void save(Maker maker);

    void deleteById(Long id);
}
```

En el código anterior se definen los métodos que se van a utilizar para realizar operaciones CRUD sobre la entidad **Maker**.

6. Creamos una interaz llamada **IProductDAO**.

```
package com.application.rest.persistence;

import com.application.rest.entities.Product;

import java.math.BigDecimal;
import java.util.List;
import java.util.Optional;

public interface IProductDAO {

    List<Product> findAll();

    Optional<Product> findById(Long id);

    List<Product> findByPriceInRange(BigDecimal minPrice, BigDecimal maxPrice);

    void save(Product product);

    void deleteById(Long id);
}
```

En el código anterior se definen los métodos que se van a utilizar para realizar operaciones CRUD sobre la entidad **Product**.

7. Creamos un paquete llamado **impl** dentro del paquete **persistence**.
8. Creamos una clase llamada **MakerDAOImpl**.

```
package com.aplication.rest.persistence.impl;

import com.aplication.rest.entities.Maker;
import com.aplication.rest.persistence.IMakerDAO;
import com.aplication.rest.repository.MakerRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import java.util.List;
import java.util.Optional;

@Component
public class MakerDAOImpl implements IMakerDAO {

    @Autowired
    private MakerRepository makerRepository;

    @Override
    public List<Maker> findAll() {
        return (List<Maker>) makerRepository.findAll();
    }

    @Override
    public Optional<Maker> findById(Long id) {
        return makerRepository.findById(id);
    }

    @Override
    public void save(Maker maker) {
        makerRepository.save(maker);
    }

    @Override
    public void deleteById(Long id) {
        makerRepository.deleteById(id);
    }
}
```

En el código anterior se implementan los métodos definidos en la interfaz **IMakerDAO**.

9. Creamos una clase llamada **ProductDAOImpl**.

```

package com.aplicacion.rest.persistence.impl;

import com.aplicacion.rest.entities.Product;
import com.aplicacion.rest.persistence.IProductDAO;
import com.aplicacion.rest.repository.ProductRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import java.math.BigDecimal;
import java.util.List;
import java.util.Optional;

@Component
public class ProductDAOImpl implements IProductDAO {

    @Autowired
    private ProductRepository productRepository;

    @Override
    public List<Product> findAll() {
        return (List<Product>)productRepository.findAll();
    }

    @Override
    public Optional<Product> findById(Long id) {
        return productRepository.findById(id);
    }

    @Override
    public List<Product> findByPriceInRange(BigDecimal minPrice, BigDecimal maxPrice) {
        return productRepository.findByPriceBetween(minPrice, maxPrice);
    }

    @Override
    public void save(Product product) {
        productRepository.save(product);
    }

    @Override
    public void deleteById(Long id) {
        productRepository.deleteById(id);
    }
}

```

En el código anterior se implementan los métodos definidos en la interfaz **IProduct-DAO**.

10. Creamos un paquete llamado **service**.

11. Creamos una interfaz llamada **IMakerService**.

```
package com.aplication.rest.service;

import com.aplication.rest.entities.Maker;

import java.util.List;
import java.util.Optional;

public interface IMakerService {

    List<Maker> findAll();

    Optional<Maker> findById(Long id);

    void save(Maker maker);

    void deleteById(Long id);
}
```

El código anterior define los métodos que se van a utilizar para realizar operaciones CRUD sobre la entidad **Maker**.

12. Creamos una interfaz llamada **IProductService**.

```
package com.aplication.rest.service;

import com.aplication.rest.entities.Product;

import java.math.BigDecimal;
import java.util.List;
import java.util.Optional;

public interface IProductService {

    List<Product> findAll();

    Optional<Product> findById(Long id);

    List<Product> findByPriceinRange(BigDecimal minPrice, BigDecimal maxPrice);

    void save(Product product);

    void deleteById(Long id);
}
```

En el código anterior se definen los métodos que se van a utilizar para realizar operaciones CRUD sobre la entidad **Product**.

13. Creamos un paquete llamado **impl** dentro del paquete **service**.
14. Creamos una clase llamada **MakerServiceImpl**.

```
package com.aplication.rest.service.impl;

import com.aplication.rest.entities.Maker;
import com.aplication.rest.persistence.IMakerDAO;
import com.aplication.rest.service.IMakerService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;

@Service
public class MakerServiceImpl implements IMakerService {

    @Autowired
    private IMakerDAO makerDAO;

    @Override
    public List<Maker> findAll() {
        return makerDAO.findAll();
    }

    @Override
    public Optional<Maker> findById(Long id) {
        return makerDAO.findById(id);
    }

    @Override
    public void save(Maker maker) {
        makerDAO.save(maker);
    }

    @Override
    public void deleteById(Long id) {
        makerDAO.deleteById(id);
    }
}
```

En el código anterior se implementan los métodos definidos en la interfaz **IMakerService**.

15. Creamos una clase llamada **ProductServiceImpl**.


```

package com.aplicacion.rest.service.impl;

import com.aplicacion.rest.entities.Product;
import com.aplicacion.rest.persistence.IProductDAO;
import com.aplicacion.rest.service.IProductService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.math.BigDecimal;
import java.util.List;
import java.util.Optional;

@Service
public class ProductServiceImpl implements IProductService {

    @Autowired
    private IProductDAO productDAO;

    @Override
    public List<Product> findAll() {
        return productDAO.findAll();
    }

    @Override
    public Optional<Product> findById(Long id) {
        return productDAO.findById(id);
    }

    @Override
    public List<Product> findByPriceInRange(BigDecimal minPrice, BigDecimal maxPrice) {
        return productDAO.findByPriceInRange(minPrice, maxPrice);
    }

    @Override
    public void save(Product product) {
        productDAO.save(product);
    }

    @Override
    public void deleteById(Long id) {
        productDAO.deleteById(id);
    }
}

```

En el código anterior se implementan los métodos definidos en la interfaz **IProductService**.

16. Creamos un paquete llamado **controllers**.

17. Creamos el paquete llamado **dto** dentro del paquete **controllers**.

18. Creamos una clase llamada **MakerDTO**.

```
package com.aplication.rest.controllers.dto;

import com.aplication.rest.entities.Product;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.ArrayList;
import java.util.List;

@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class MakerDTO {

    private Long id;
    private String name;
    private List<Product> productList = new ArrayList<>();
}
```

En el código anterior se define una clase **MakerDTO** que se va a utilizar para mapear los datos de la entidad **Maker**.

19. Creamos una clase llamada **ProductDTO**.

```
package com.aplication.rest.controllers.dto;

import com.aplication.rest.entities.Maker;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.math.BigDecimal;

@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class ProductDTO {

    private Long id;
    private String name;
    private BigDecimal price;
}
```

```
private Maker maker;
}
```

En el código anterior se define una clase **ProductDTO** que se va a utilizar para mapear los datos de la entidad **Product**.

20. Creamos una clase llamada **MakerController**.

```
package com.aplication.rest.controllers;

import com.aplication.rest.controllers.dto.MakerDTO;
import com.aplication.rest.entities.Maker;
import com.aplication.rest.service.IMakerService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.net.URI;
import java.net.URISyntaxException;
import java.util.List;
import java.util.Optional;

@RestController
@RequestMapping("/api/maker")
public class MakerController {

    @Autowired
    private IMakerService makerService;

    @GetMapping("/find/{id}")
    public ResponseEntity<?> findById(@PathVariable Long id) {
        Optional<Maker> makerOptional = makerService.findById(id);

        if (makerOptional.isPresent()) {
            Maker maker = makerOptional.get();

            MakerDTO makerDTO = MakerDTO.builder()
                .id(maker.getId())
                .name(maker.getName())
                .productList(maker.getProductList())
                .build();

            return ResponseEntity.ok(makerDTO);
        }

        return ResponseEntity.notFound().build();
    }
}
```

```

@GetMapping("/findAll")
public ResponseEntity<?> findAll() {
    List<MakerDTO> makerList = makerService.findAll()
        .stream()
        .map(maker -> MakerDTO.builder()
            .id(maker.getId())
            .name(maker.getName())
            .productList(maker.getProductList())
            .build())
        .toList();
    return ResponseEntity.ok(makerList);
}

@PostMapping("/save")
public ResponseEntity<?> save(@RequestBody MakerDTO makerDTO) throws URISyntaxException {

    if(makerDTO.getName().isBlank()){
        return ResponseEntity.badRequest().build();
    }

    makerService.save(Maker.builder()
        .name(makerDTO.getName())
        .build());

    return ResponseEntity.created(new URI("/api/maker/save")).build();
}

@PutMapping("/update/{id}")
public ResponseEntity<?> update(@PathVariable Long id, @RequestBody MakerDTO makerDTO) {

    Optional<Maker> makerOptional = makerService.findById(id);

    if (makerOptional.isPresent()) {
        Maker maker = makerOptional.get();
        maker.setName(makerDTO.getName());
        makerService.save(maker);
        return ResponseEntity.ok("Registro Actualizado");
    }
    return ResponseEntity.notFound().build();
}

@DeleteMapping("/delete/{id}")
public ResponseEntity<?> delete(@PathVariable Long id) {

    if (id != null) {
        makerService.deleteById(id);
        return ResponseEntity.ok("Registro Eliminado");
    }
}

```

```

        return ResponseEntity.notFound().build();
    }
}

```

En el código anterior se define un controlador llamado **MakerController** que permite realizar operaciones CRUD sobre la entidad **Maker**.

21. Creamos una clase llamada **ProductController**.

```

package com.aplication.rest.controllers;

import com.aplication.rest.controllers.dto.ProductDTO;
import com.aplication.rest.entities.Product;
import com.aplication.rest.service.IProductService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.net.URI;
import java.net.URISyntaxException;
import java.util.List;
import java.util.Optional;

@RestController
@RequestMapping("/api/product")
public class ProductController {

    @Autowired
    private IProductService productService;

    @GetMapping("/find/{id}")
    public ResponseEntity<?> findById(@PathVariable Long id) {
        Optional<Product> productOptional = productService.findById(id);
        if (productOptional.isPresent()){
            Product product = productOptional.get();
            ProductDTO productDTO = ProductDTO.builder()
                .id(product.getId())
                .name(product.getName())
                .price(product.getPrice())
                .maker(product.getMaker())
                .build();
            return ResponseEntity.ok(productDTO);
        } else {
            return ResponseEntity.badRequest().build();
        }
    }

    @GetMapping("/findAll")

```

```

public ResponseEntity<?> findAll() {
    List<ProductDTO> productList = productService.findAll()
        .stream()
        .map(product -> ProductDTO.builder()
            .id(product.getId())
            .name(product.getName())
            .price(product.getPrice())
            .maker(product.getMaker())
            .build()
        ).toList();
    return ResponseEntity.ok(productList);
}

@PostMapping("/save")
public ResponseEntity<?> save(ProductDTO productDTO) throws URISyntaxException {
    if (productDTO.getName().isBlank() || productDTO.getPrice()==null || productDTO.g
        return ResponseEntity.badRequest().build();
    }

    Product product = Product.builder()
        .name(productDTO.getName())
        .price(productDTO.getPrice())
        .maker(productDTO.getMaker())
        .build();
    productService.save(product);

    return ResponseEntity.created(new URI("/api/product/save")).build();
}

@PutMapping("/update/{id}")
public ResponseEntity<?> update(@PathVariable Long id, @RequestBody ProductDTO produc

    Optional<Product> productOptional = productService.findById(id);

    if (productOptional.isPresent()){
        Product product = productOptional.get();
        product.setName(productDTO.getName());
        product.setPrice(productDTO.getPrice());
        product.setMaker(productDTO.getMaker());
        productService.save(product);
        return ResponseEntity.ok("Registro Actualizado");
    }

    return ResponseEntity.notFound().build();
}

>DeleteMapping("/delete/{id}")
public ResponseEntity<?> deleteById(@PathVariable Long id) {

```

```

    if(id != null){
        productService.deleteById(id);
        return ResponseEntity.ok("Registro Eliminado");
    }
    return ResponseEntity.badRequest().build();
}
}
}

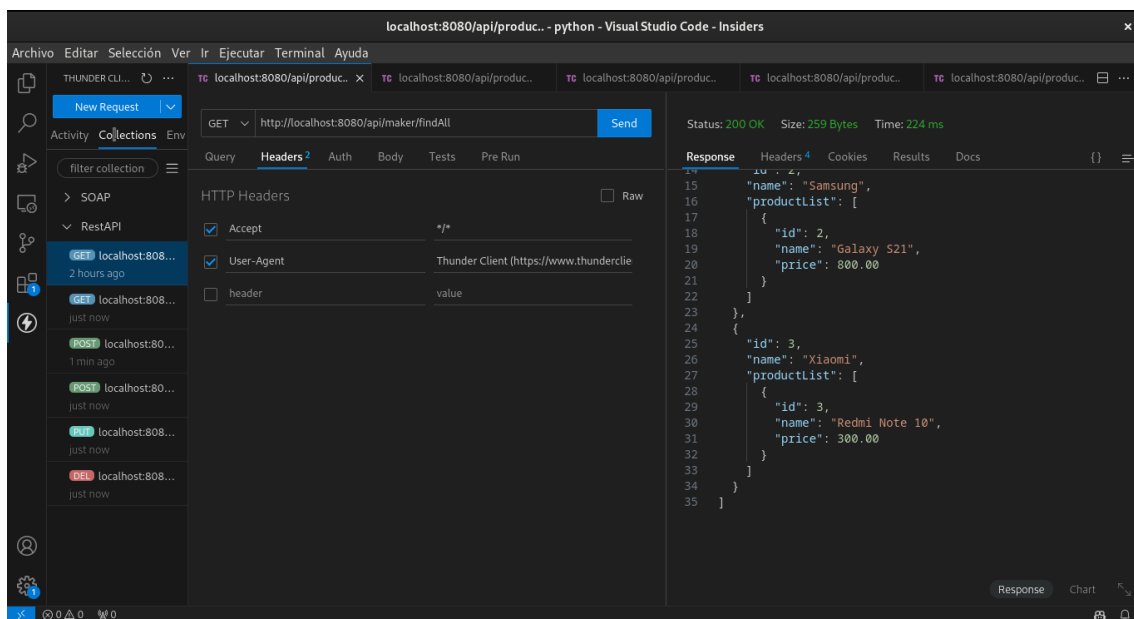
```

En el código anterior se define un controlador llamado **ProductController** que permite realizar operaciones CRUD sobre la entidad **Product**.

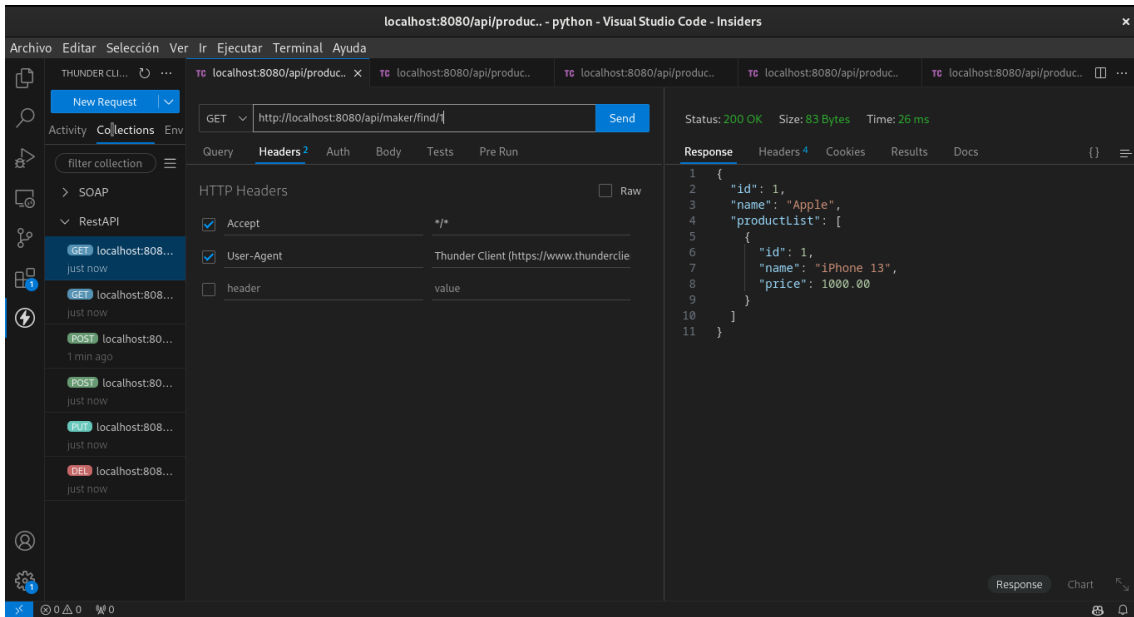
33.4 Probar el API Rest

1. Ejecutar el proyecto y verificamos si el servidor se encuentra en ejecución.

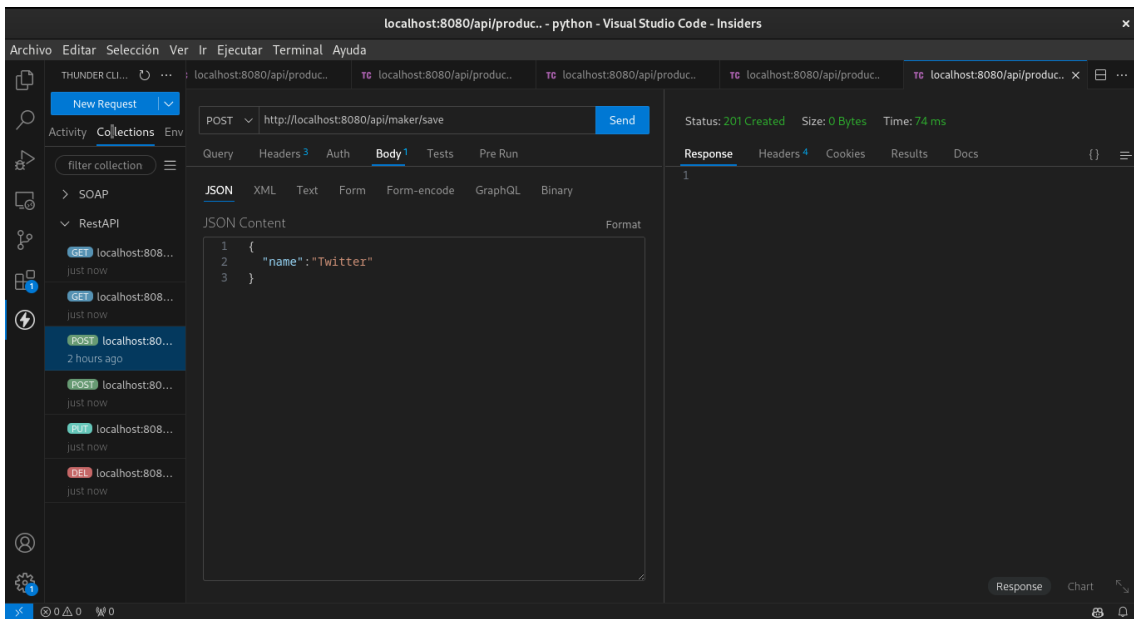
Metodo **findAll** de Maker



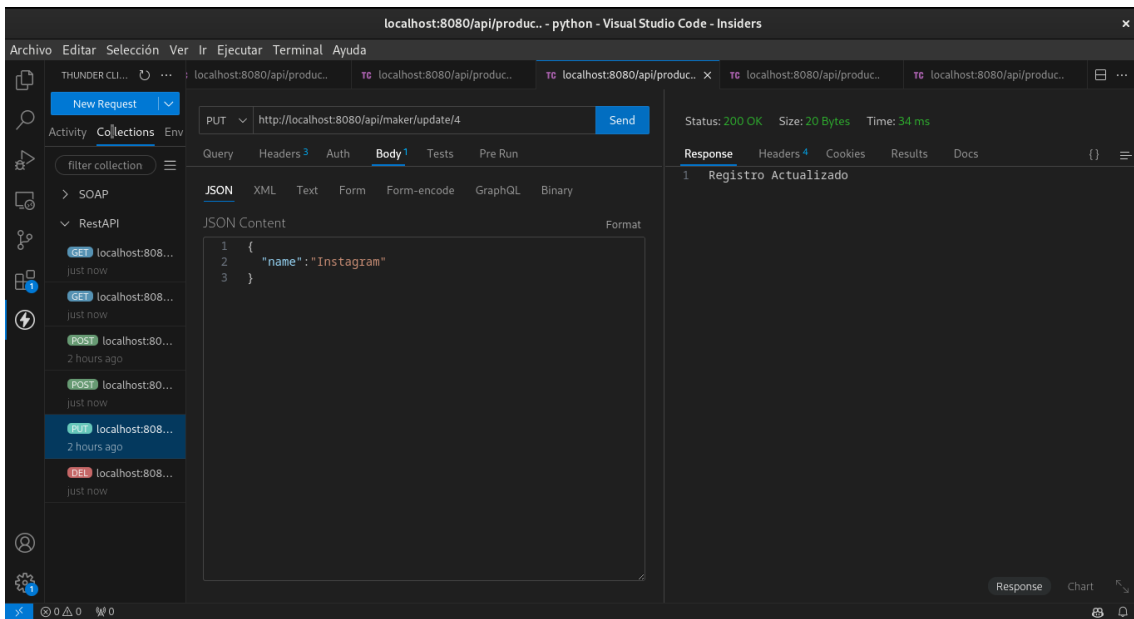
Metodo **find/{id}** de Maker



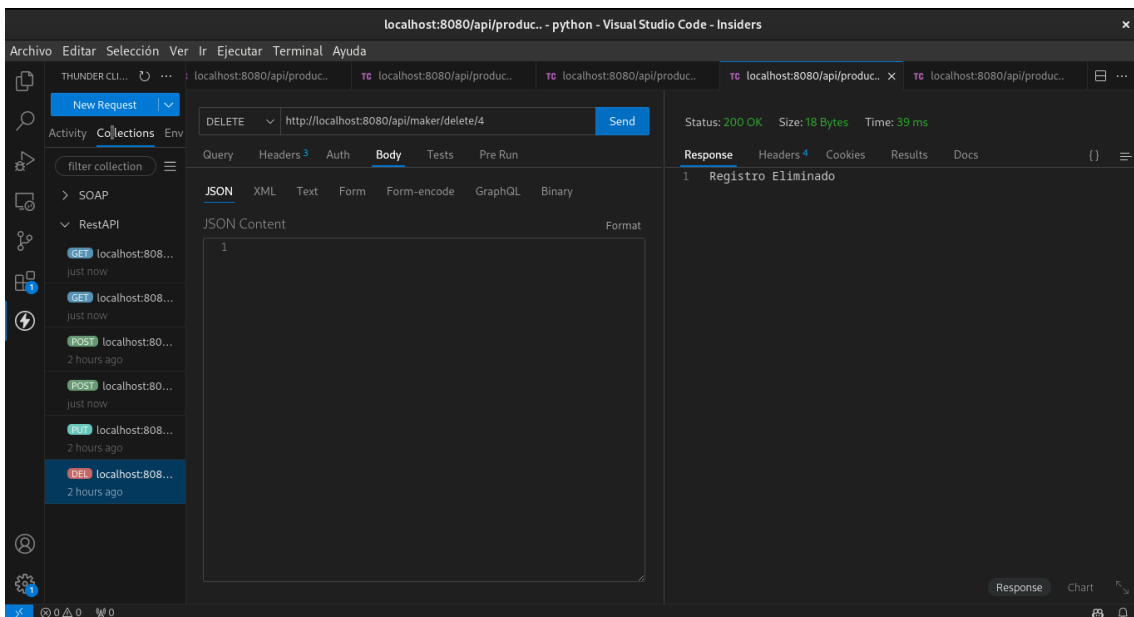
Metodo save de Maker



Metodo update/{id} de Maker



Metodo delete/{id} de Maker



34 Reto

Warning

Existen algunos errores en el código del producto, en particular en el archivo **ProductController** y en el **ProductDTO**. Por favor corrígelos para cumplir con el Reto.

1. Implementar los métodos **findAll** y **find/{id}** de la entidad **Product**.
2. Implementar los métodos **save**, **update/{id}** y **delete/{id}**** de la entidad **Product**.
3. Probar los métodos implementados.

35 Conclusiones

- En este laboratorio se ha creado una API Rest con Spring Boot que permite realizar operaciones CRUD sobre las entidades **Maker** y **Product**.
- Se ha utilizado Spring Data JPA para realizar operaciones CRUD sobre las entidades.
- Se ha utilizado Lombok para reducir la cantidad de código boilerplate.
- Se ha utilizado Docker para crear un contenedor con MySQL.
- Se ha utilizado el cliente Thunder Client para probar los servicios.

36 Referencias

- [Spring Boot](#)
- [Spring Data JPA](#)
- [Spring Initializr](#)
- [Docker](#)
- [MySQL](#)
- [Lombok](#)
- [Código del Proyecto](#)

37 Laboratorio de Dev Containers

37.1 Objetivo

El objetivo de este laboratorio es comprender la importancia de los Dev Containers en el Desarrollo de Software

37.2 Requerimientos

Para este laboratorio se requiere tener instalado el SDK de .NET Core 6. Además, se recomienda utilizar **Visual Studio Code** como editor de código. Utilizaremos el entorno de **Docker Containers** para ejecutar la aplicación.

En este laboratorio utilizaremos el lenguaje de programación C# y Docker Containers para ejecutar la aplicación en un contenedor de Docker.

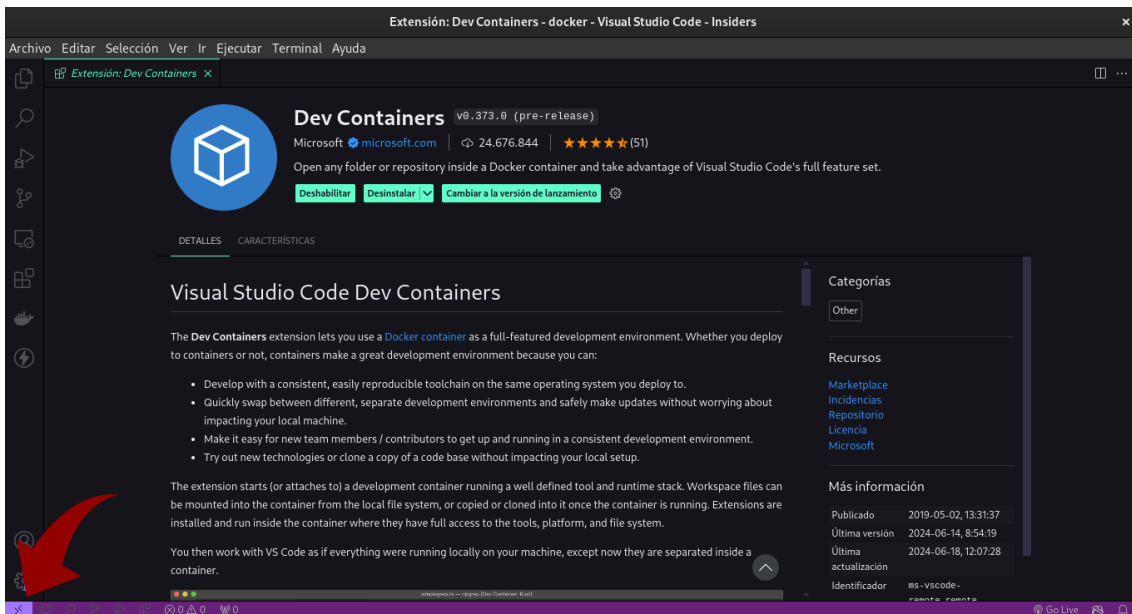
¿Qué son Docker Containers?

Son entornos de ejecución ligeros y portátiles que permiten empaquetar una aplicación y sus dependencias en un contenedor aislado. Docker Containers se utilizan para ejecutar aplicaciones en entornos de desarrollo, pruebas y producción de forma consistente y confiable.

37.3 Procedimiento

Empezamos instalando la extensión de **Docker Containers** en Visual Studio Code. Esta extensión nos permite crear, ejecutar y depurar contenedores de Docker directamente desde el editor de código.

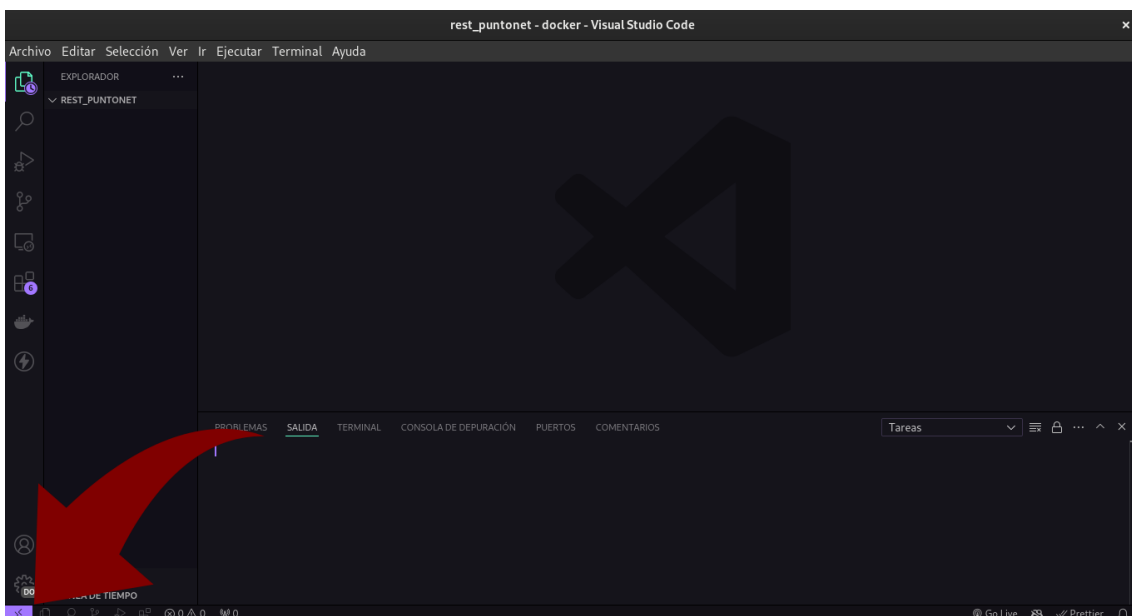
37.3.1 Instalar la extensión de Docker Containers



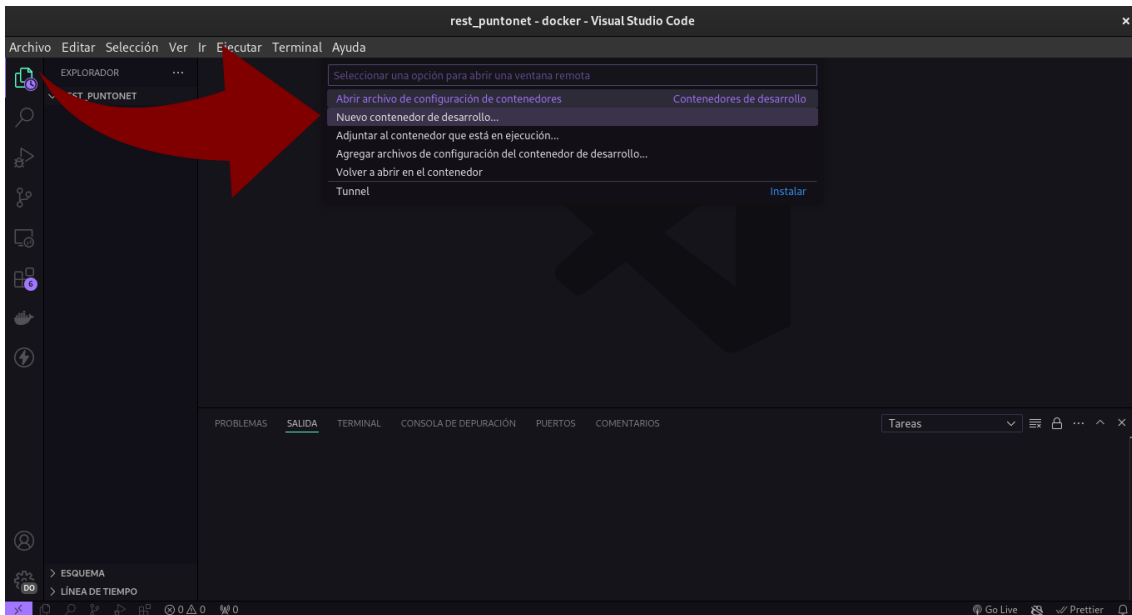
1. Abre Visual Studio Code.
2. Haz clic en la pestaña **Extensions** en la barra lateral izquierda.
3. Busca **Docker Containers** en el campo de búsqueda.
4. Haz clic en **Install** para instalar la extensión.

37.3.2 Crear un Docker Container

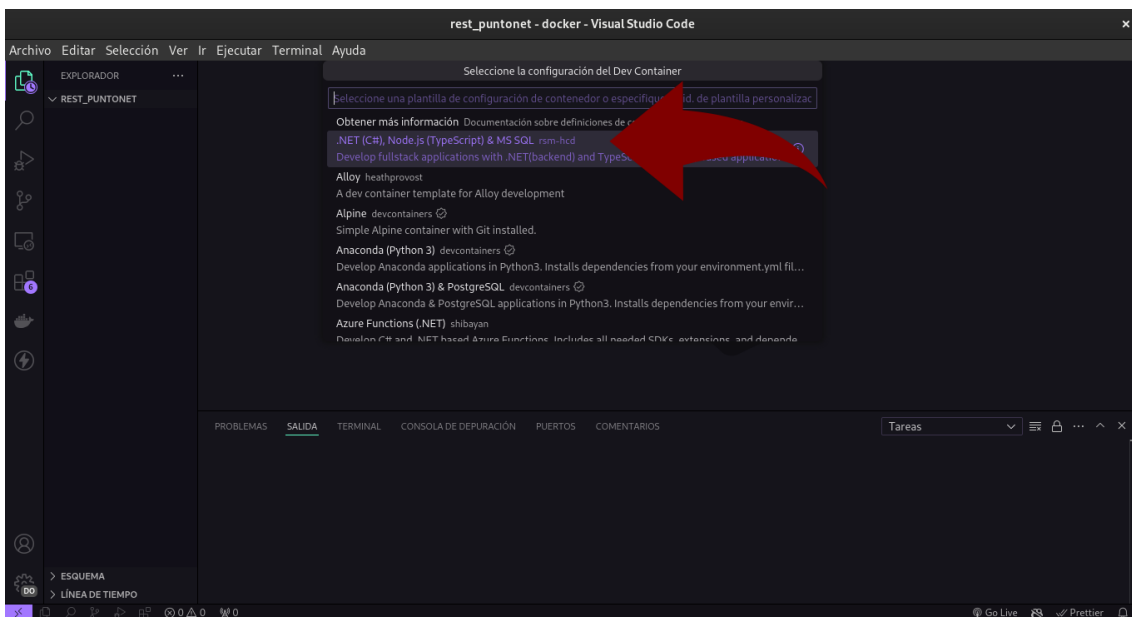
1. Para crear un Docker Container es necesario dar clic en la parte inferior izquierda de **Visual Studio Code**.



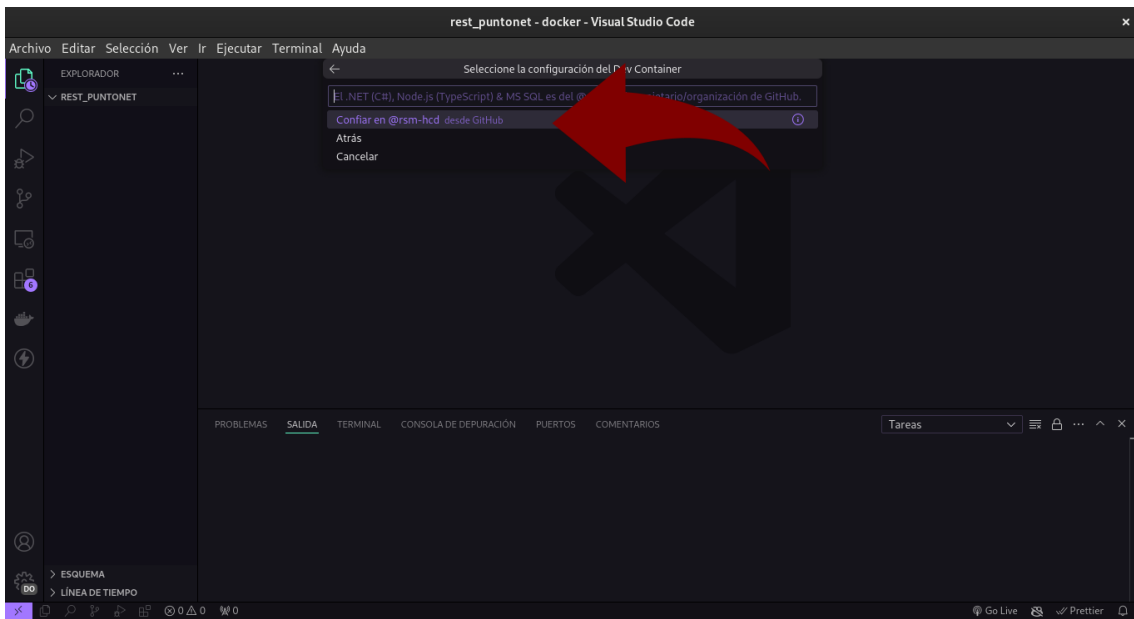
2. Selecciona la opción **Add Development Container Configuration Files**.



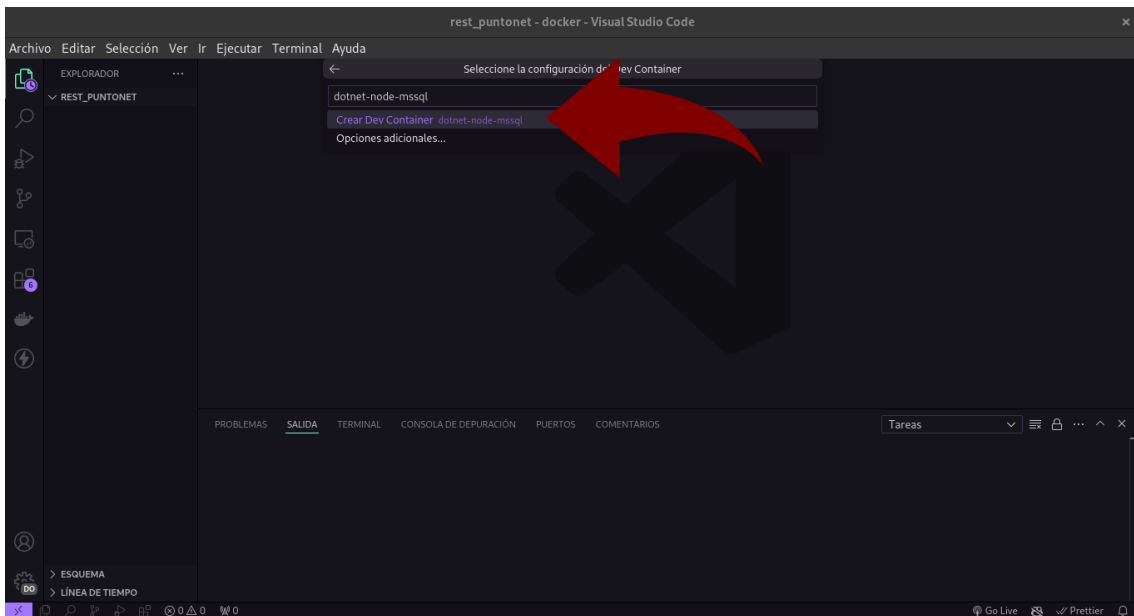
3. Selecciona la opción **.NET 6**.



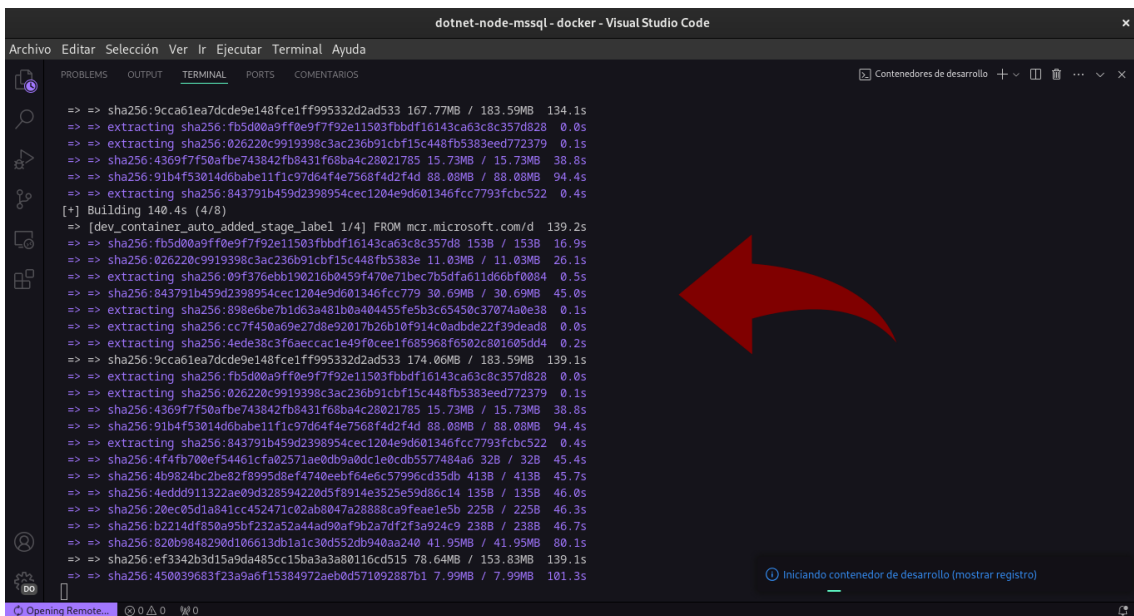
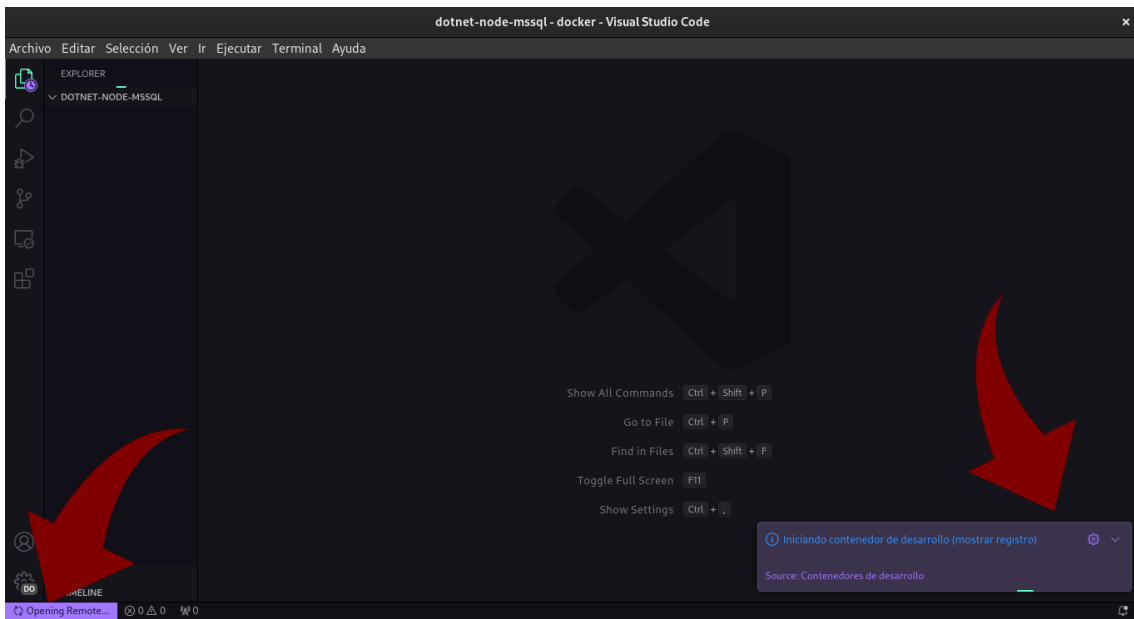
4. Tienes la opción de seleccionar entre las opciones existentes, regresar al paso anterior o cancelar



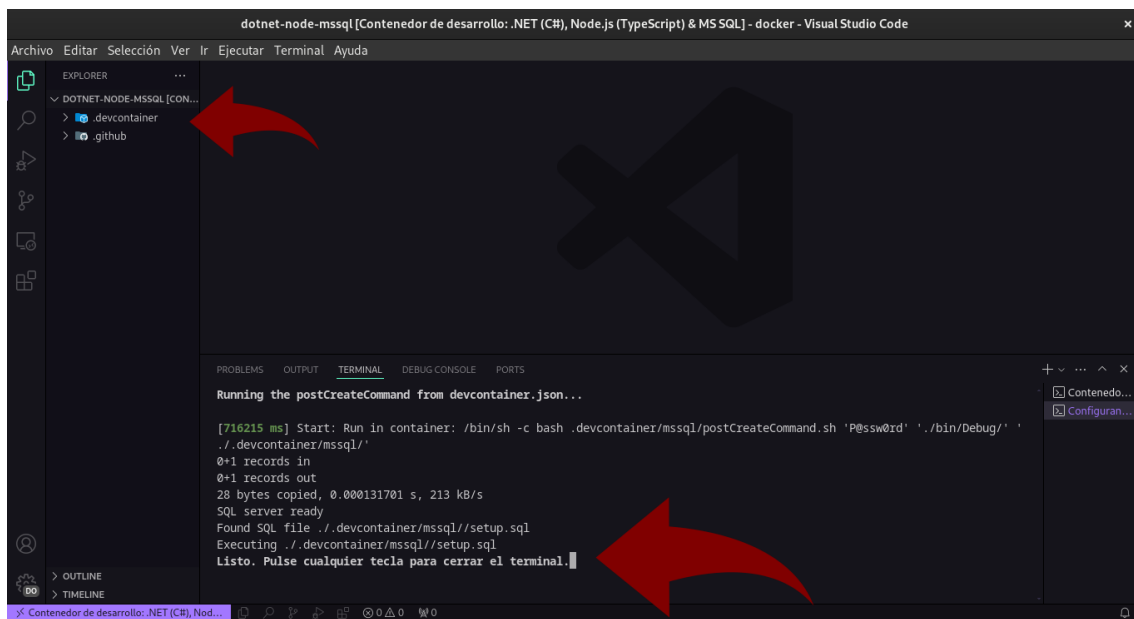
5. Finalmente nos aparece la opción de **Crear Dev Container**



6. Esperar a que se instalen las dependencias.

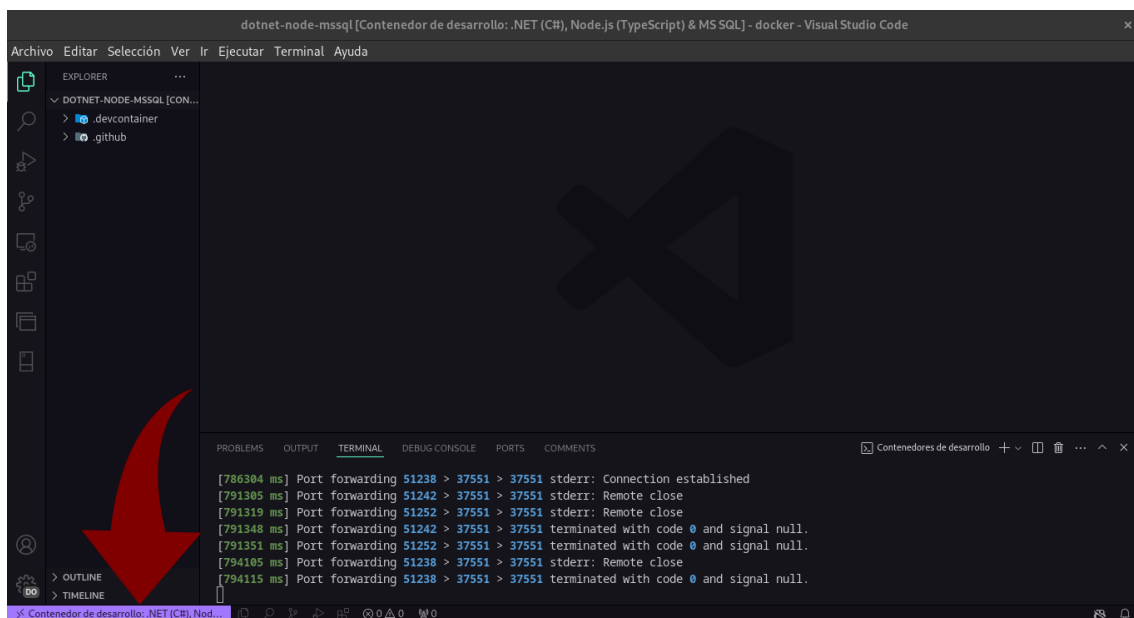


7. Cuando ha finalizado la creación de Dev Container debe aparecer un mensaje como el siguiente.



Observa que se han creado 2 directorios nuevos en tu proyecto, **.devcontainer** y **.github**. Por ahora no entraremos en detalle sobre estos directorios, pero es importante que sepas que son necesarios para configurar el entorno de desarrollo en un Dev Container.

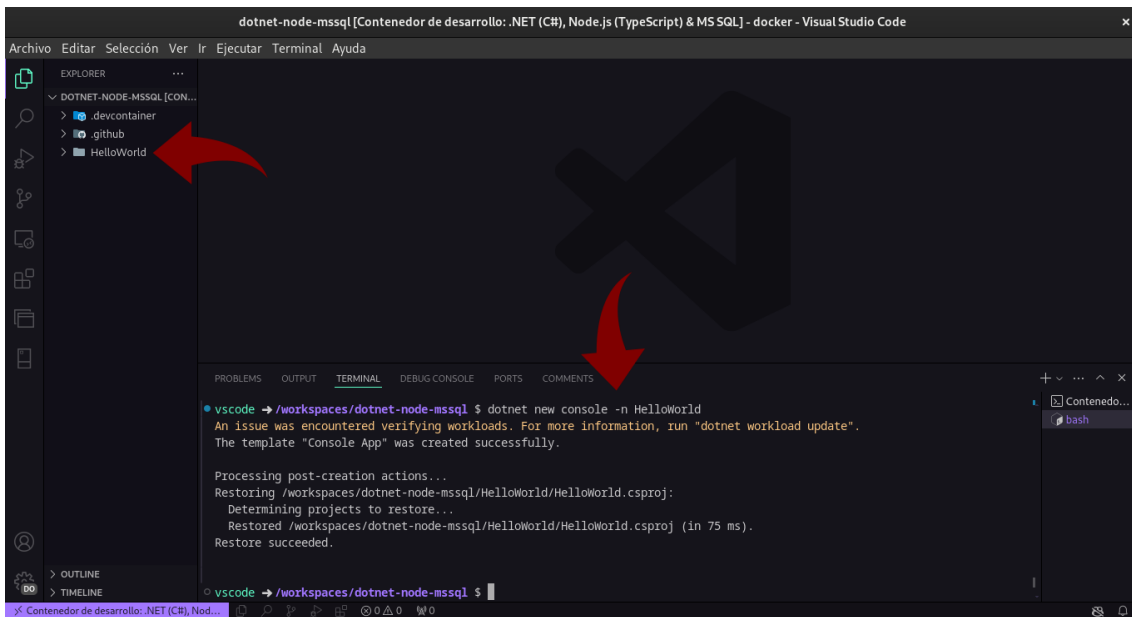
8. Ahora nos aparece un Dev Container de .Net que está listo para ser utilizado en nuestro proyecto.



37.3.3 Probar el Dev Container

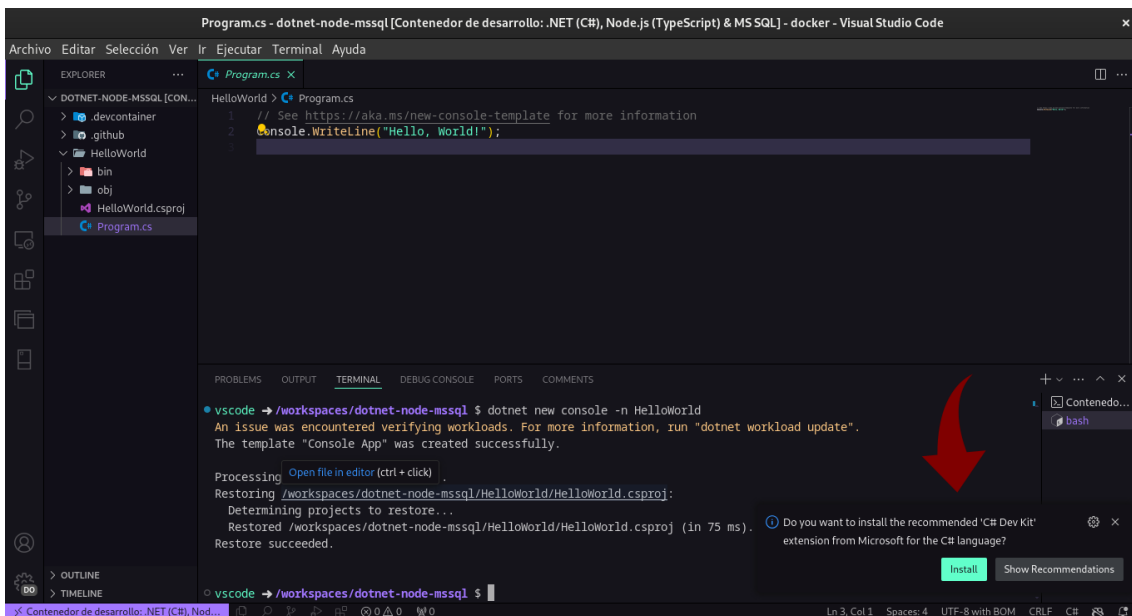
1. Para probar el Dev Container, creamos un proyecto de .NET Core abriendo una terminal en Visual Studio Code y ejecutando el siguiente comando:

```
dotnet new console -n HelloWorld
```

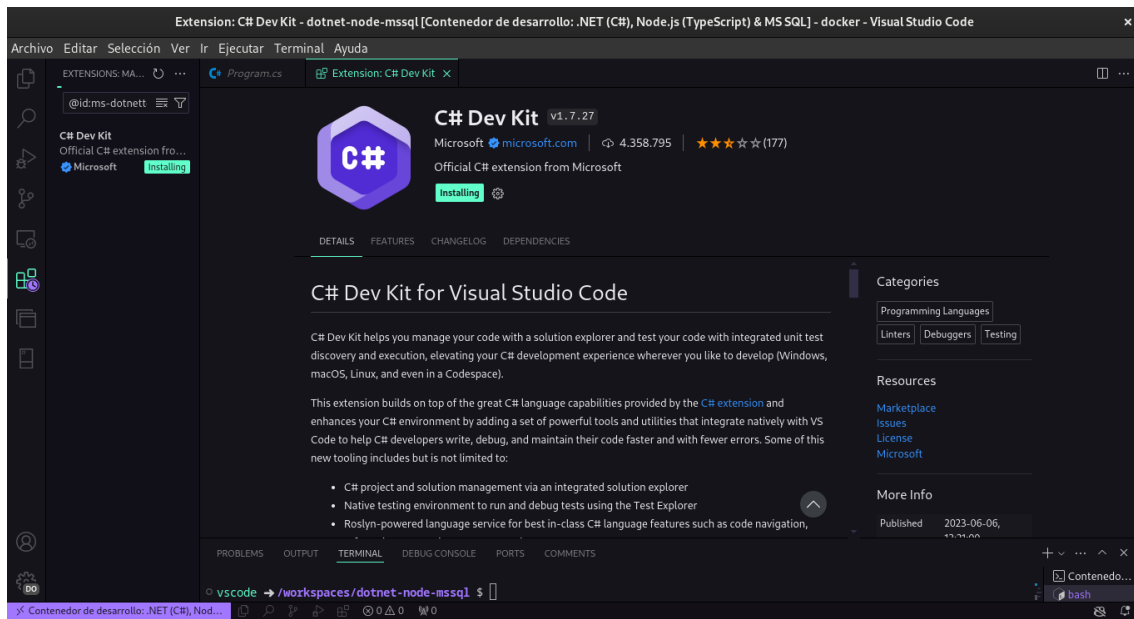


2. Abre el archivo **Program.cs** para verificar si tiene el siguiente código:

```
// See https://aka.ms/new-console-template for more information  
Console.WriteLine("Hello, World!");
```



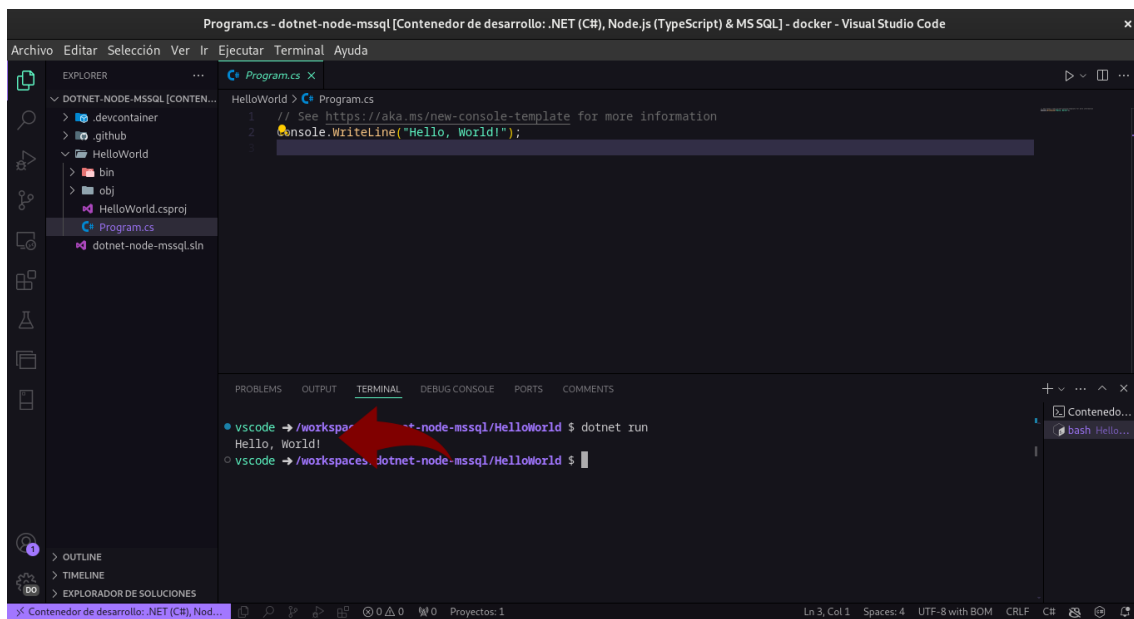
Se sugiere instalar la extensión de “C# Dev Kit”, toma en cuenta que esta extensión es opcional. Y se instalará en el Dev Container



3. Ahora probemos nuestro código ejecutando el siguiente comando en la terminal:

```
dotnet run
```

4. Deberías ver el mensaje **Hello, World!** en la terminal.



37.4 Reto

Creando un Dev Container que te permita ejecutar un Hola Mundo en cualquier tecnología que conozcas.

37.5 Conclusiones

En este laboratorio hemos aprendido acerca del uso de Dev Containers para la ejecución de Proyectos de Desarrollo de Software.

38 Laboratorio: Implementación de POO en TypeScript

38.1 Objetivo

Implementar conceptos de Programación Orientada a Objetos (POO) en una aplicación web utilizando TypeScript. Requisitos Previos

- Node.js y npm instalados

38.2 Paso 1: Instalación de TypeScript

- Inicializar un proyecto Node.js:

```
mkdir poo-typescript
cd poo-typescript
npm init -y
```

Con los comandos anteriores se crea un nuevo proyecto Node.js con un archivo package.json predeterminado.

Instalar TypeScript:

```
npm install typescript --save-dev
```

Con el comando anterior, se instala TypeScript como una dependencia de desarrollo en el proyecto.

Crear el archivo de configuración tsconfig.json:

```
npx tsc --init
```

Este comando crea un archivo tsconfig.json con la configuración predeterminada para TypeScript.

38.3 Estructura del Proyecto:

```
/poo-typescript
  src
    encapsulamiento.ts
    herencia.ts
    polimorfismo.ts
    main.ts
    tsconfig.json
```

En la estructura del proyecto, src contendrá los archivos TypeScript con ejemplos de encapsulamiento, herencia y polimorfismo, y main.ts será el archivo principal para ejecutar los ejemplos.

38.4 Paso 2: Definición de Clases y Encapsulamiento

- Crear una clase Persona en src/encapsulamiento.ts:
- Agregar métodos para obtener y establecer propiedades privadas en Persona:

```
// src/encapsulamiento.ts

class Persona {
  private nombre: string;
  private edad: number;

  constructor(nombre: string, edad: number) {
    this.nombre = nombre;
    this.edad = edad;
  }

  public getNombre(): string {
    return this.nombre;
  }

  public getEdad(): number {
    return this.edad;
  }

  public setNombre(nombre: string): void {
    this.nombre = nombre;
  }

  public setEdad(edad: number): void {
    this.edad = edad;
  }
}
```

```
export { Persona };
```

En el código anterior se define una clase Persona con propiedades privadas nombre y edad, y métodos para obtener y establecer los valores de estas propiedades.

38.5 Paso 3: Herencia

- Crear una clase Estudiante que herede de Persona en src/herencia.ts:

```
// src/herencia.ts

import { Persona } from './encapsulamiento';

class Estudiante extends Persona {
  private matricula: string;

  constructor(nombre: string, edad: number, matricula: string) {
    super(nombre, edad);
    this.matricula = matricula;
  }

  public getMatricula(): string {
    return this.matricula;
  }

  public setMatricula(matricula: string): void {
    this.matricula = matricula;
  }
}

export { Estudiante };
```

En el código anterior se define una clase Estudiante que hereda de Persona y agrega una propiedad matricula con métodos para obtener y establecer su valor.

38.6 Paso 4: Polimorfismo

- Crear una clase Profesor y una función para demostrar polimorfismo en src/polimorfismo.ts:


```

// src/polimorfismo.ts

import { Persona } from './encapsulamiento';

class Profesor extends Persona {
  private materia: string;

  constructor(nombre: string, edad: number, materia: string) {
    super(nombre, edad);
    this.materia = materia;
  }

  public getMateria(): string {
    return this.materia;
  }

  public setMateria(materia: string): void {
    this.materia = materia;
  }

  public presentar(): string {
    return `Soy el profesor ${this.getNombre()} y enseño ${this.materia}.`;
  }
}

class Estudiante extends Persona {
  private matricula: string;

  constructor(nombre: string, edad: number, matricula: string) {
    super(nombre, edad);
    this.matricula = matricula;
  }

  public getMatricula(): string {
    return this.matricula;
  }

  public setMatricula(matricula: string): void {
    this.matricula = matricula;
  }

  public presentar(): string {
    return `Soy el estudiante ${this.getNombre()} y mi matrícula es ${this.matricula}`;
  }
}

function presentarPersona(persona: Persona): string {
  if (persona instanceof Profesor) {

```

```

        return persona.presentar();
    } else if (persona instanceof Estudiante) {
        return persona.presentar();
    } else {
        return `Soy ${persona.getNombre()} y tengo ${persona.getEdad()} años.`;
    }
}

export { Profesor, Estudiante, presentarPersona };

```

En el código anterior se define una clase Profesor que hereda de Persona y agrega una propiedad materia con métodos para obtener y establecer su valor, y un método presentar para mostrar información específica del profesor. También se define una función presentarPersona que demuestra polimorfismo al presentar a una persona como profesor, estudiante o persona genérica.

38.7 Paso 5: Implementación en el Archivo Principal

- Crear el archivo principal src/main.ts para ejecutar ejemplos:

```

// src/main.ts

import { Persona } from './encapsulamiento';
import { Estudiante } from './herencia';
import { Profesor, presentarPersona } from './polimorfismo';

const persona = new Persona('Carlos', 30);
console.log(`Persona: ${persona.getNombre()}, ${persona.getEdad()} años`);

const estudiante = new Estudiante('Ana', 22, '2022001');
console.log(`Estudiante: ${estudiante.getNombre()}, ${estudiante.getEdad()} años, Matrícula: ${estudiante.getMatricula()}`);

const profesor = new Profesor('Luis', 40, 'Matemáticas');
console.log(`Profesor: ${profesor.getNombre()}, ${profesor.getEdad()} años, Materia: ${profesor.getMateria()}`);

console.log(presentarPersona(persona));
console.log(presentarPersona(estudiante));
console.log(presentarPersona(profesor));

```

En el código anterior se crean instancias de Persona, Estudiante y Profesor, y se muestra información sobre cada uno de ellos. También se llama a la función presentarPersona para mostrar información polimórfica sobre las personas.

38.8 Paso 6: Compilar y Ejecutar

- Compilar el proyecto:

```
npx tsc
```

Ejecutar el archivo JavaScript generado:

```
node src/main.js
```

39 Conclusiones

En este laboratorio, se ha implementado la Programación Orientada a Objetos (POO) en una aplicación web utilizando TypeScript. Se han definido clases con propiedades y métodos, se ha demostrado el encapsulamiento, la herencia y el polimorfismo, y se ha ejecutado un ejemplo para mostrar el funcionamiento de estos conceptos en TypeScript.

40 Reto

Extiende la aplicación con más clases y métodos que demuestren otros conceptos de POO, como abstracción, interfaces, métodos estáticos, etc.

41 Laboratorio: Creación de una Plataforma de Gestión de Cursos con Next.js y TypeScript

En este laboratorio, crearás una aplicación de gestión de cursos utilizando Next.js y TypeScript siguiendo el patrón MVC (Modelo-Vista-Controlador).

41.1 Objetivo

Desarrollar una plataforma para gestionar cursos, permitiendo a los usuarios inscribirse y ver los cursos disponibles. Requisitos Previos

- Node.js y npm instalados
- Conocimientos básicos de Next.js y TypeScript

41.2 Paso 1: Crear la Aplicación Next.js

- Crear una nueva aplicación Next.js:

```
npx create-next-app@latest gestion-cursos
cd gestion-cursos
```

Ejecutar la aplicación:

```
npm run dev
```

Abre tu navegador y navega a <http://localhost:3000> para ver la aplicación en funcionamiento.

41.3 Paso 2: Crear la Estructura del Proyecto

- Crear la estructura de directorios:

```

/gestion-cursos
  pages
    index.tsx
    courses.tsx
  models
    course.ts
  views
    courseView.tsx
  controllers
    courseController.ts
  ...

```

41.4 Paso 3: Modelo (models/course.ts)

- Definir la clase Course y gestionar los datos de los cursos:

```

// models/course.ts

class Course {
  constructor(
    public id: number,
    public name: string,
    public description: string,
    public instructor: string
  ) {}
}

class CourseModel {
  private courses: Course[] = [];

  addCourse(course: Course): void {
    this.courses.push(course);
  }

  getCourses(): Course[] {
    return this.courses;
  }
}

export { Course, CourseModel };

```

41.5 Paso 4: Vista (views/courseView.tsx)

- Crear una vista para mostrar los cursos:

```

// views/courseView.tsx

import React from 'react';
import { Course } from '../models/course';

interface CourseViewProps {
  courses: Course[];
}

const CourseView: React.FC<CourseViewProps> = ({ courses }) => {
  return (
    <div>
      {courses.map((course) => (
        <div key={course.id}>
          <h2>{course.name}</h2>
          <p>{course.description}</p>
          <p>Instructor: {course.instructor}</p>
        </div>
      ))}
    </div>
  );
};

export default CourseView;

```

41.6 Paso 5: Controlador (controllers/courseController.ts)

- Gestionar la lógica entre el modelo y la vista:

```

// controllers/courseController.ts

import { Course, CourseModel } from "../models/course";

class CourseController {
  private model: CourseModel;

  constructor() {
    this.model = new CourseModel();
  }

  initialize(): Course[] {
    this.model.addCourse(new Course(1, 'Curso de Next.js', 'Aprende a crear aplicaciones con Next.js'));
    this.model.addCourse(new Course(2, 'Curso de TypeScript', 'Domina TypeScript para el desarrollo de aplicaciones'));
    return this.model.getCourses();
  }
}

```



```
}  
  
export default CourseController;
```

41.7 Paso 6: Integración en Next.js (pages/courses.tsx)

- Integrar el modelo, vista y controlador en una página de Next.js:

```
// pages/courses.tsx  
  
import React, { useEffect, useState } from 'react';  
import CourseView from '../views/courseView';  
import CourseController from '../controllers/courseController';  
import { Course } from '../models/course';  
  
const CoursesPage: React.FC = () => {  
  const [courses, setCourses] = useState<Course[]>([]);  
  const controller = new CourseController();  
  
  useEffect(() => {  
    const coursesData = controller.initialize();  
    setCourses(coursesData);  
  }, []);  
  
  return (  
    <div>  
      <h1>Lista de Cursos</h1>  
      <CourseView courses={courses} />  
    </div>  
  );  
};  
  
export default CoursesPage;
```

41.8 Paso 7: Página de Inicio (pages/index.tsx)

- Crear una página de inicio simple que enlace a la página de cursos:

```
// pages/index.tsx  
  
import React from 'react';  
import Link from 'next/link';  
  
const HomePage: React.FC = () => {
```

```
return (  
  <div>  
    <h1>Bienvenido a la Plataforma de Cursos</h1>  
    <Link href="/courses">  
      Ver Cursos  
    </Link>  
  </div>  
);  
};  
  
export default HomePage;
```

42 Extra

42.1 Paso 8: Estilos CSS

- Agregar estilos CSS a la aplicación:

Recordemos que instalamos TailwindCSS en el proyecto, por lo que podemos utilizar sus clases para estilizar la aplicación.

```
/* styles/globals.css */

@tailwind base;
@tailwind components;
@tailwind utilities;

body {
  font-family: 'Arial', sans-serif;
  margin: 0;
  padding: 0;
}

.container {
  max-width: 800px;
  margin: 0 auto;
  padding: 20px;
}

h1 {
  color: #333;
}

a {
  color: #0070f3;
  text-decoration: none;
  cursor: pointer;
}

a:hover {
  text-decoration: underline;
}
```

Incluimos los estilos en el archivo `_app.tsx`:

```
import '../..//styles/globals.css';

import { ReactElement, ComponentType } from 'react';

function MyApp({ Component, pageProps }: { Component: ComponentType<any>, pageProps: any
  return <Component {...pageProps} />;
}

export default MyApp;
```

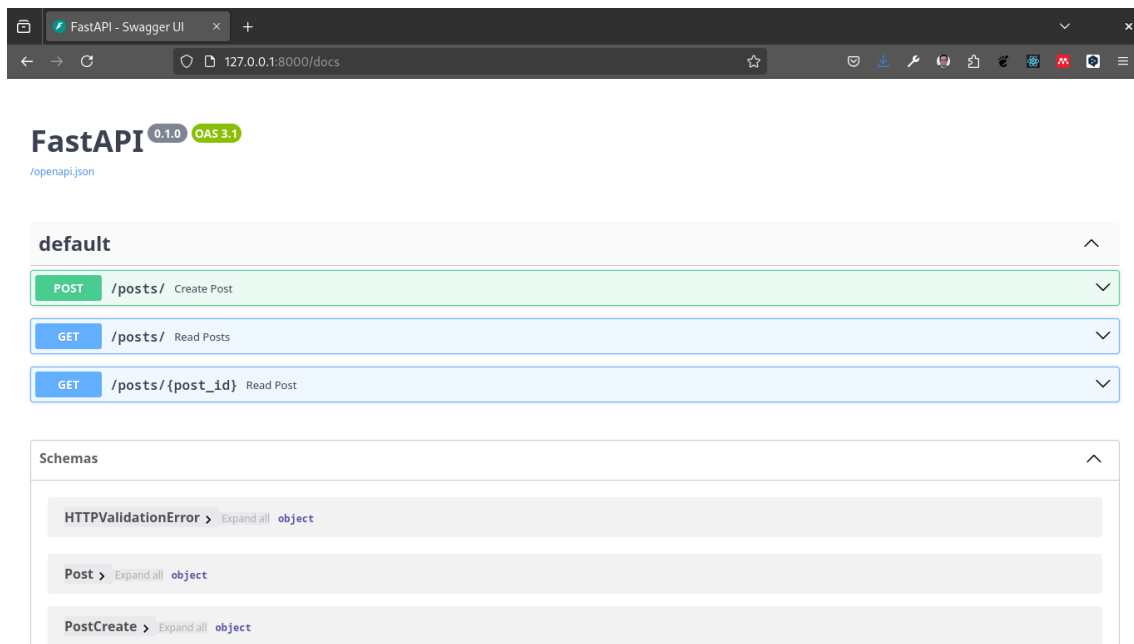
43 Conclusión

En este laboratorio, has aprendido a crear una plataforma de gestión de cursos utilizando Next.js y TypeScript siguiendo el patrón MVC. Has creado un modelo para gestionar los datos de los cursos, una vista para mostrar los cursos y un controlador para gestionar la lógica entre el modelo y la vista. Integraste estos elementos en una aplicación Next.js y creaste una página de inicio y una página de cursos. También has aprendido a agregar estilos CSS a la aplicación utilizando TailwindCSS. ¡Felicidades por completar este laboratorio!

44 Reto

- Agregar funcionalidades para inscribirse en un curso y ver los detalles de un curso.
- Crear una página de administración para agregar, editar y eliminar cursos.

45 Laboratorio: Creación de un CMS Monolítico con FastAPI



45.1 Objetivo:

Desarrollar un sistema de gestión de contenido (CMS) utilizando la arquitectura monolítica con FastAPI, donde todas las funcionalidades están integradas en una sola aplicación web.

45.2 Requisitos Previos:

1. Conocimientos básicos de Python.
2. Conocimientos básicos de FastAPI.
3. Python 3.7 o superior instalado en tu entorno.
4. Entorno virtual de Python configurado.

45.3 Pasos del Laboratorio:

1. Preparación del Entorno

- Instalar FastAPI y Uvicorn:
- Crea un entorno virtual y instala FastAPI junto con Uvicorn para el servidor ASGI.

```
python -m venv env
```

```
source venv/bin/activate # En Windows usa `venv\Scripts\activate`
```

```
pip install fastapi uvicorn[standard] sqlalchemy alembic
```

45.4 Configurar la Estructura del Proyecto:

Crea la estructura básica del proyecto:

```
cms_project/
  app/
    main.py
    models.py
    schemas.py
    crud.py
    database.py
  alembic.ini
  migrations/
```

2. Desarrollo del CMS

Configurar la Base de Datos:

En **app/database.py**, configura la conexión a la base de datos usando SQLAlchemy:

```
from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker

SQLALCHEMY_DATABASE_URL = "sqlite:///./test.db"
engine = create_engine(SQLALCHEMY_DATABASE_URL, connect_args={"check_same_thread": False})
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
Base = declarative_base()
```

Definir los Modelos:

En **app/models.py**, define los modelos de datos usando SQLAlchemy:

```
from sqlalchemy import Column, Integer, String, Text
from app.database import Base

class Post(Base):
    __tablename__ = "posts"
```



```
id = Column(Integer, primary_key=True, index=True)
title = Column(String, index=True)
content = Column(Text)
```

Definir los Esquemas:

En `app/schemas.py`, define los esquemas de Pydantic para la validación de datos:

```
from pydantic import BaseModel

class PostBase(BaseModel):
    title: str
    content: str

class PostCreate(PostBase):
    pass

class Post(PostBase):
    id: int

    class Config:
        orm_mode = True
```

Implementar las Operaciones CRUD:

En `app/crud.py`, implementa las operaciones CRUD para los posts:

```
from sqlalchemy.orm import Session
from . import models, schemas

def create_post(db: Session, post: schemas.PostCreate):
    db_post = models.Post(**post.dict())
    db.add(db_post)
    db.commit()
    db.refresh(db_post)
    return db_post

def get_post(db: Session, post_id: int):
    return db.query(models.Post).filter(models.Post.id == post_id).first()

def get_posts(db: Session, skip: int = 0, limit: int = 10):
    return db.query(models.Post).offset(skip).limit(limit).all()
```

45.5 Crear los Endpoints de la API:

En `app/main.py`, define los endpoints de la API usando FastAPI:

```

from typing import List
from fastapi import FastAPI, Depends
from sqlalchemy.orm import Session
from . import models, schemas, crud, database

app = FastAPI()

models.Base.metadata.create_all(bind=database.engine)

def get_db():
    db = database.SessionLocal()
    try:
        yield db
    finally:
        db.close()

@app.post("/posts/", response_model=schemas.Post)
def create_post(post: schemas.PostCreate, db: Session = Depends(get_db)):
    return crud.create_post(db=db, post=post)

@app.get("/posts/{post_id}", response_model=schemas.Post)
def read_post(post_id: int, db: Session = Depends(get_db)):
    db_post = crud.get_post(db, post_id)
    if db_post is None:
        raise HTTPException(status_code=404, detail="Post not found")
    return db_post

@app.get("/posts/", response_model=List[schemas.Post])
def read_posts(skip: int = 0, limit: int = 10, db: Session = Depends(get_db)):
    posts = crud.get_posts(db, skip=skip, limit=limit)
    return posts

```

Tip

Crear el directorio `__init__.py` en la carpeta `app` para que Python reconozca la carpeta como un paquete.

3. Despliegue y Pruebas

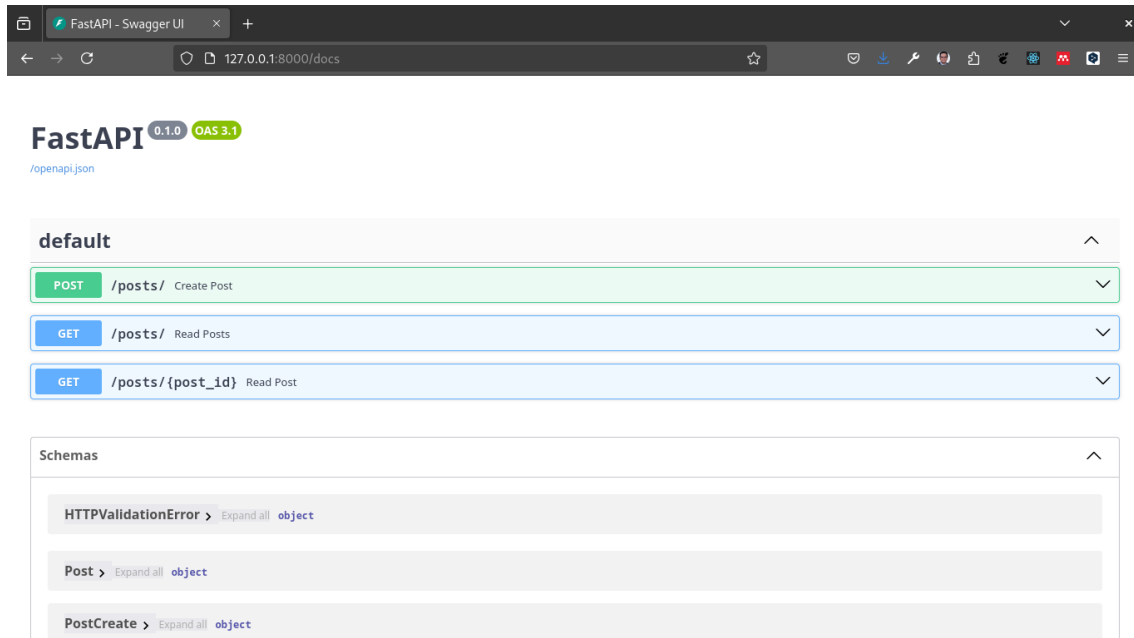
Ejecutar la Aplicación:

Usa Uvicorn para ejecutar la aplicación:

```
uvicorn app.main:app --reload
```

45.6 Pruebas:

- Accede a <http://127.0.0.1:8000/docs> para ver y probar la documentación interactiva de la API generada automáticamente por FastAPI.



- Prueba **crear**, **leer**, y **listar posts** a través de la interfaz de Swagger.

4. Evaluación

Desarrollo: Evalúa la facilidad de desarrollo y la organización del código.

Rendimiento: Mide el rendimiento de la aplicación al manejar diferentes volúmenes de datos.

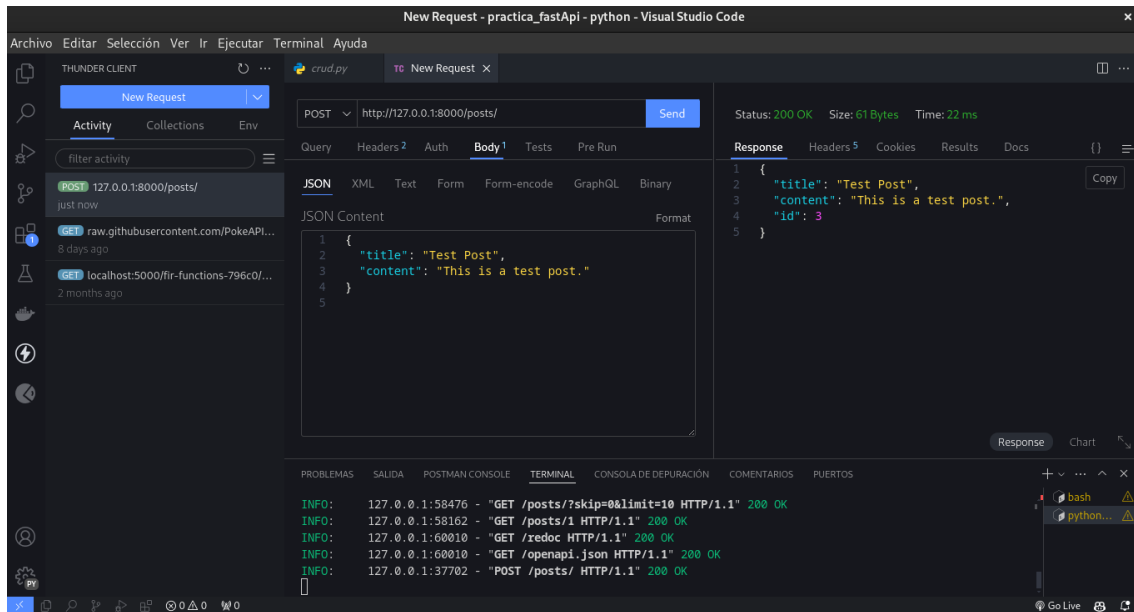
Mantenimiento: Considera la facilidad de mantenimiento y extensión de la aplicación a medida que crece.

Recursos:

- [FastAPI Documentation](#)
- [SQLAlchemy Documentation](#)
- [Pydantic Documentation](#)

Este laboratorio te permitirá familiarizarte con FastAPI y la arquitectura monolítica, entendiendo cómo construir una aplicación integral con todas sus funcionalidades en un solo proyecto.

45.7 Pruebas



1. Instalar Thunder Client

Si aún no lo has hecho, primero debes instalar Thunder Client. Es una extensión para Visual Studio Code (VSCode).

- Abre VSCode.
- Ve a la pestaña de Extensiones (Ctrl+Shift+X).
- Busca Thunder Client y haz clic en Instalar.

2. Configurar Thunder Client

Una vez que Thunder Client esté instalado, puedes comenzar a configurar y probar tus endpoints. Crear una Nueva Solicitud

- Abre Thunder Client en VSCode. Lo encontrarás en el panel lateral izquierdo como un ícono de rayo.
- Crea una nueva colección para organizar tus solicitudes. Puedes hacer esto desde el panel principal de Thunder Client haciendo clic en el botón + al lado de “Collections” y luego en “Create Collection”.

45.7.1 Crear una Solicitud en Thunder Client

- Selecciona la colección en la que deseas agregar la solicitud.
- Haz clic en el botón + New Request.
- Elige el método HTTP (GET, POST, PUT, DELETE, etc.) dependiendo del endpoint que deseas probar. Introduce la URL de tu endpoint en el campo correspondiente. Por ejemplo, <http://127.0.0.1:8000/posts/>.

45.7.2 Configuración de la Solicitud

Para una solicitud POST:

- Método: POST
- URL: <http://127.0.0.1:8000/posts/>
- Encabezados: Configura los encabezados si es necesario. Para JSON, agrega Content-Type: application/json.

45.7.3 Cuerpo:

Selecciona la pestaña Body.

Elige el tipo de cuerpo como Raw y elige JSON.

Introduce el JSON que deseas enviar. Por ejemplo:

```
{
  "title": "Test Post",
  "content": "This is a test post."
}
```

Haz clic en Send para enviar la solicitud y ver la respuesta.

Para una solicitud GET:

- **Método:** GET
- **URL:** <http://127.0.0.1:8000/posts/1>
- Haz clic en Send para enviar la solicitud y ver la respuesta.

45.7.4 Revisar la Respuesta

Una vez que envíes la solicitud, Thunder Client te mostrará la respuesta de la API, que incluye:

- Código de estado (por ejemplo, 200 OK).
- Encabezados de la respuesta.
- Cuerpo de la respuesta, que será el JSON u otro formato que tu API devuelve.

3. Organizar y Guardar Solicitudes

Puedes guardar tus solicitudes para uso futuro:

- Después de enviar una solicitud, haz clic en Save.
- Elige un nombre para la solicitud y, si lo deseas, una descripción.
- Guarda la solicitud en la colección que creaste anteriormente.

4. Ejecutar Pruebas en Secuencia

Puedes crear una serie de solicitudes dentro de una colección para probar diferentes aspectos de tu API en secuencia. Esto te permitirá automatizar un flujo de pruebas básico.

Thunder Client es una herramienta poderosa y fácil de usar para probar tus endpoints de FastAPI. Te permite realizar pruebas rápidas y ver los resultados de manera clara. Si necesitas ayuda con una configuración específica o con la interpretación de respuestas, no dudes en preguntar.

45.8 Reto

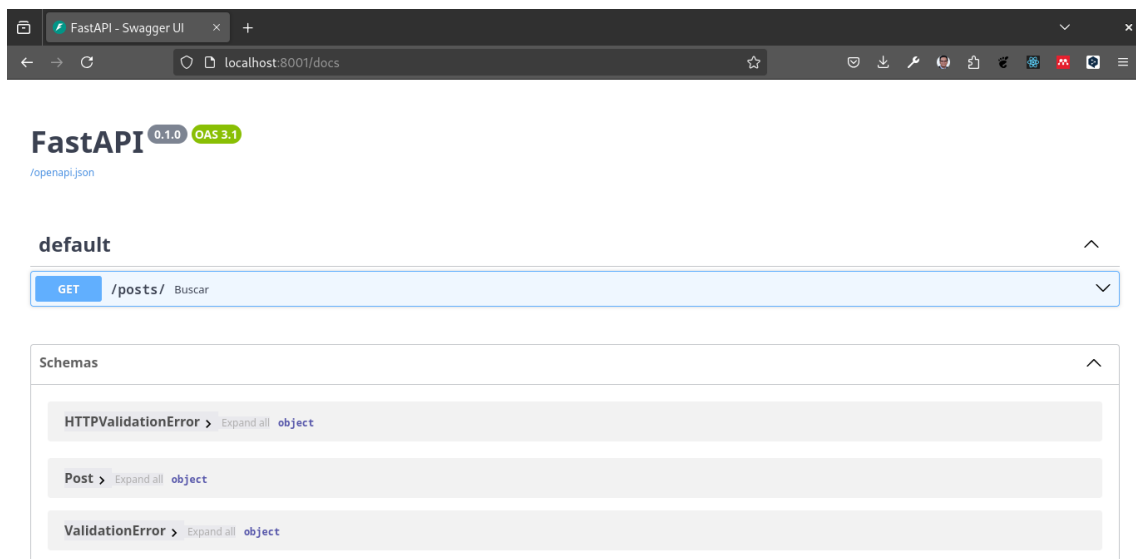
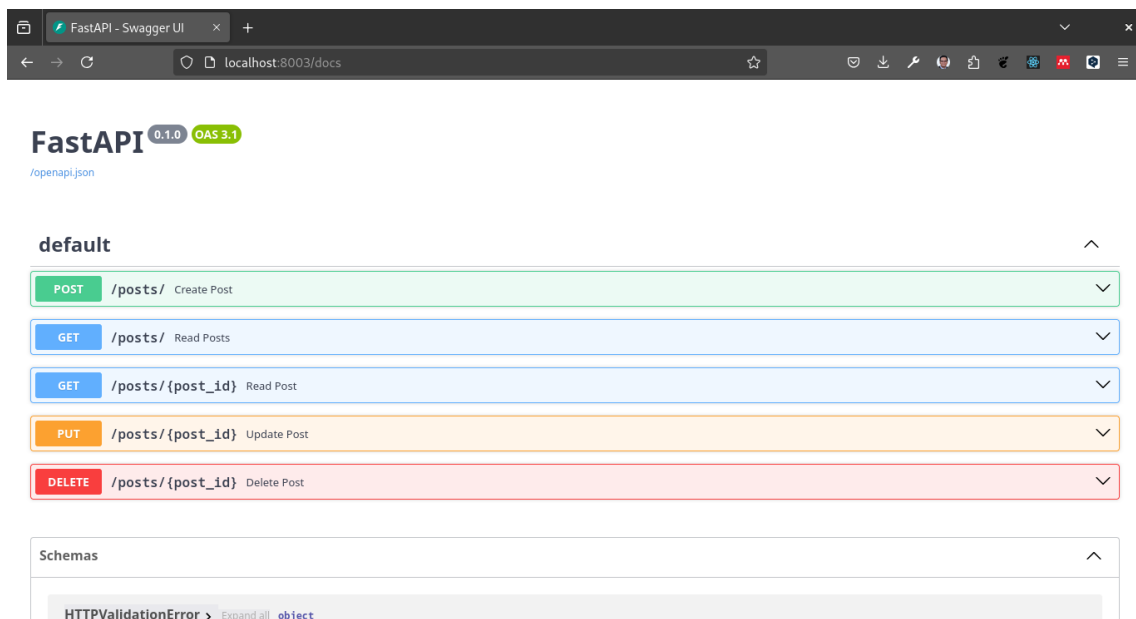
En este laboratorio has aprendido a crear un CMS monolítico con FastAPI. Ahora, te desafiamos a extender la funcionalidad de tu CMS agregando nuevas características, como:

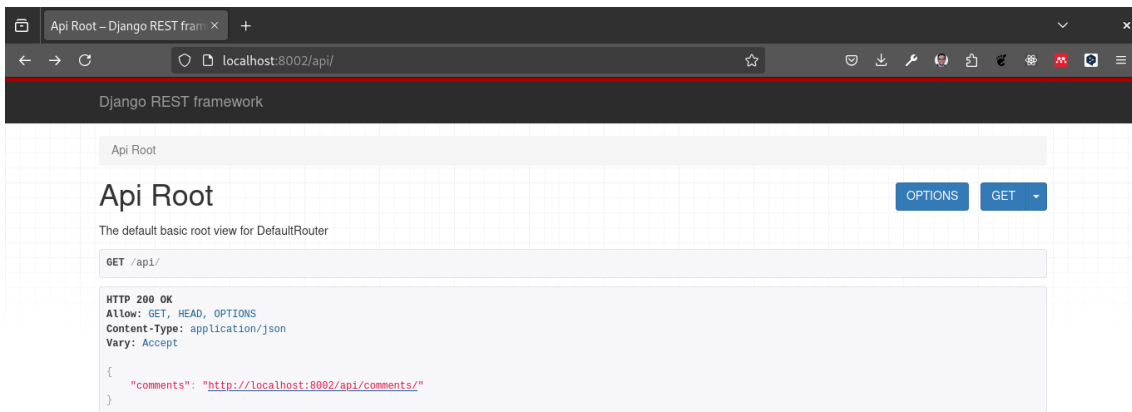
- **Autenticación:** Implementa un sistema de autenticación para permitir que los usuarios se registren y accedan a la aplicación.
- **Comentarios:** Agrega la capacidad de que los usuarios comenten en los posts.
- **Imágenes:** Permite a los usuarios subir imágenes para acompañar sus posts.
- **Búsqueda:** Implementa una función de búsqueda para que los usuarios puedan buscar posts por título o contenido.
- **Temas:** Permite a los usuarios clasificar los posts por temas o categorías.

Estos son solo algunos ejemplos de cómo puedes extender tu CMS.

¡Sé creativo y diviértete desarrollando nuevas funcionalidades!

46 Tutorial para la Creación de Microservicios con Docker y Docker Compose





Este tutorial te guiará a través de la creación de cuatro microservicios diferentes y cómo integrarlos usando Docker y Docker Compose.

46.1 Requisitos Previos

- Docker instalado en tu sistema.
- Docker Compose instalado en tu sistema.
- Conocimientos básicos de Python y Docker.

46.2 Estructura del Proyecto

Organizaremos nuestro proyecto con la siguiente estructura:

```
|-- autenticacion/
|   |-- Dockerfile
|   |-- requirements.txt
|   |-- (otros archivos de código y configuración)
|
|-- busqueda/
|   |-- fastAPI/
|   |   |-- Dockerfile
|   |   |-- requirements.txt
|   |   |-- (otros archivos de código y configuración)
|
|-- cms_project/
|   |-- Dockerfile
|   |-- requirements.txt
|   |-- (otros archivos de código y configuración)
```



```
| |-- db/  
|  
|-- gestion_post/  
| |-- Dockerfile  
| |-- requirements.txt  
| |-- (otros archivos de código y configuración)  
|  
|-- docker-compose.yml
```

46.3 Paso 1: Crear el Microservicio de Autenticación

Utilizar el siguiente repositorio de GitHub para clonar el código de ejemplo:

<https://github.com/pacoquirolga/AutenticacionMicroservicio>

Para clonar el repositorio, ejecuta el siguiente comando:

```
git clone https://github.com/pacoquirolga/AutenticacionMicroservicio.git
```

46.4 Paso 2: Crear el Microservicio de Búsqueda

Utilizar el siguiente repositorio de GitHub para clonar el código de ejemplo:

<https://github.com/AvilesDanie/Microservicio-Busqueda>

Para clonar el repositorio, ejecuta el siguiente comando:

```
git clone https://github.com/AvilesDanie/Microservicio-Busqueda.git
```

46.5 Paso 3: Microservicio de Comentarios

Utilizar el siguiente repositorio de GitHub para clonar el código de ejemplo:

https://github.com/esmora2/cms_project

Para clonar el repositorio, ejecuta el siguiente comando:

```
git clone https://github.com/esmora2/cms_project.git
```

Este microservicio contiene la opción de publicar post y comentarios.

46.6 Paso 4: Crear el Microservicio de Gestión de Post

Utilizar el siguiente repositorio de GitHub para clonar el código de ejemplo:

<https://github.com/LeonardoYaranga/-Microservicio-de-Gesti-n-de-Posts-G4>

Para clonar el repositorio, ejecuta el siguiente comando:

```
git clone https://github.com/LeonardoYaranga/-Microservicio-de-Gesti-n-de-Posts-G4.git
```

46.7 Paso 5: Crear el Archivo docker-compose.yml General

Crema un archivo docker-compose.yml en el directorio raíz del proyecto con el siguiente contenido:

```
services:
  # Servicio de Autenticación
  db_autenticacion:
    image: mysql:9.0.1
    container_name: AutenticacionDB
    environment:
      MYSQL_ROOT_PASSWORD: 12345
      MYSQL_DATABASE: AutenticacionDB
    ports:
      - "3306:3306"
    networks:
      - app-network

  autenticacion:
    build:
      context: ./autenticacion
    container_name: MicroservicioAutenticacion
    ports:
      - "8000:8000"
    networks:
      - app-network
    depends_on:
      - db_autenticacion

  # Servicio de Búsqueda
  busqueda:
    build:
      context: ./busqueda/fastAPI
    container_name: MicroservicioBusqueda
    ports:
      - "8001:8000"
    networks:
```

```

- app-network

# Servicio de CMS
db_cms:
  image: nouchka/sqlite3
  container_name: CMSDB
  volumes:
    - ./cms_project/db:/data
  environment:
    - SQLITE_DATABASE=mydatabase.db
  networks:
    - app-network

cms:
  build:
    context: ./cms_project
  container_name: MicroservicioCMS
  ports:
    - "8002:8000"
  networks:
    - app-network
  depends_on:
    - db_cms

# Servicio de Gestión de Post
gestion_post:
  build:
    context: ./gestion_post
  container_name: MicroservicioGestionPost
  ports:
    - "8003:8000"
  networks:
    - app-network
  environment:
    - DATABASE_URL=sqlite:///./test.db
  command: uvicorn app.main:app --host 0.0.0.0 --port 8000 --reload

networks:
  app-network:
    driver: bridge

```

46.8 Paso 6: Levantar los Microservicios

Navega al directorio raíz de tu proyecto y ejecuta el siguiente comando para construir y levantar todos los contenedores:

The screenshot shows a Visual Studio Code editor with a Docker Compose file named `docker-compose.yml` open. The file defines two services: `autenticacion` and `busqueda`. The `autenticacion` service is configured with a container name `MicroservicioAutenticacion`, a port mapping `8000:8000`, and depends on the `db_autenticacion` service. The `busqueda` service is a comment indicating it's a search service. The terminal window shows the output of the `docker-compose up --build -d` command, indicating that all services were successfully built and started.

```
services:
  autenticacion:
    build:
      container_name: MicroservicioAutenticacion
      ports:
        - "8000:8000"
      networks:
        - app-network
      depends_on:
        - db_autenticacion
# Servicio de Búsqueda
busqueda:
  build:
    context: ./busqueda/fastAPI
```

```
[busqueda] resolving provenance for metadata file 0.2s
[gestion_post] resolving provenance for metadata file 0.2s
[autenticacion] resolving provenance for metadata file 0.1s
[cms] resolving provenance for metadata file 0.0s
[+] Running 7/7
✔ Network lab_microservicios_app-network Created 0.3s
✔ Container CMSDB Started 1.1s
✔ Container AutenticacionDB Started 1.5s
✔ Container MicroservicioBusqueda Started 1.3s
✔ Container MicroservicioGestionPost Started 1.5s
✔ Container MicroservicioCMS Started 1.5s
✔ Container MicroservicioAutenticacion Started 1.9s
```

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

Esto levantará cada microservicio en su propio contenedor y los hará accesibles en los siguientes puertos:

- Búsqueda <http://localhost:8001/docs>
- CMS <http://localhost:8002/api/>
- Gestión de Post <http://localhost:8003/docs>

Ahora puedes acceder a cada microservicio en tu navegador y probar su funcionalidad.

46.9 Reto

Ajustar el archivo `docker-compose.yml` para que los microservicios funcionen correctamente, Si es necesario puedes adaptar los archivos `Dockerfile` y `requirements.txt` de cada microservicio.

También puedes realizar las pruebas con herramientas como Postman, Thunder Client o Insomnia.

46.10 Recursos

- [Documentación de Docker](#)
- [Documentación de Docker Compose](#)
- [Documentación de FastAPI](#)
- [Documentación de SQLite](#)
- [Respositorio de Código del Laboratorio](#)

46.11 Conclusión

Has aprendido cómo crear y conectar varios microservicios utilizando Docker y Docker Compose. Ahora puedes expandir este proyecto y agregar más microservicios según tus necesidades.

47 Laboratorio de Implementación de un Sistema de Búsqueda e Indexación de Datos parte 1



47.1 Objetivo:

Introducir a los estudiantes a la implementación de sistemas de búsqueda e indexación de datos, utilizando herramientas y técnicas modernas.

47.2 Descripción:

Los estudiantes desarrollarán un sistema de búsqueda para una aplicación web que permita a los usuarios buscar y filtrar productos, documentos o artículos. Implementarán un motor de búsqueda utilizando Elasticsearch y configurarán el índice de datos para optimizar la búsqueda.

47.3 Conceptos Cubiertos:

- Indexación y búsqueda de datos.
- Integración de motores de búsqueda con aplicaciones web.
- Optimización de índices y consultas de búsqueda.

Herramientas:

- Elasticsearch
- Node.js/NestJS
- Docker

47.4 Paso 1: Configuración del Entorno con Docker

47.4.1 1.1. Crear el archivo Dockerfile para Node.js/NestJS:

Crema un archivo llamado Dockerfile en el directorio raíz de tu proyecto:

```
# Usar la imagen oficial de Node.js
FROM node:18-alpine

# Crear el directorio de la aplicación
WORKDIR /app

# Copiar los archivos de package.json y package-lock.json desde search-app/
COPY search-app/package*.json ./

# Instalar las dependencias de la aplicación
RUN npm install

# Copiar el resto de los archivos de la aplicación desde search-app/
COPY search-app/ .

# Exponer el puerto de la aplicación
EXPOSE 3000

# Comando para iniciar la aplicación
CMD ["npm", "run", "start:dev"]
```

47.4.2 1.2. Crear el archivo docker-compose.yml:

Crema un archivo llamado docker-compose.yml en el directorio raíz del proyecto:

```

services:
  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:8.5.0
    container_name: elasticsearch
    environment:
      - discovery.type=single-node
      - network.host=0.0.0.0
      - ES_JAVA_OPTS=-Xms1g -Xmx1g # Ajusta según los recursos disponibles
      - xpack.security.enabled=false # Deshabilitar seguridad para desarrollo
    ports:
      - '9200:9200'
      - '9300:9300'
    networks:
      - default
    volumes:
      - elasticsearch_data:/usr/share/elasticsearch/data

  nestjs-app:
    build: .
    ports:
      - '3000:3000'
    volumes:
      - ./search-app:/app:delegated
      - /app/node_modules
    depends_on:
      - elasticsearch
    environment:
      - ELASTICSEARCH_HOST=${ELASTICSEARCH_HOST}
      - ELASTICSEARCH_PORT=${ELASTICSEARCH_PORT}
    networks:
      - default

networks:
  default:
    driver: bridge

volumes:
  elasticsearch_data:
    driver: local

```

Manejemos variables de entorno en el archivo docker-compose.yml:

```

ELASTICSEARCH_HOST=elasticsearch
ELASTICSEARCH_PORT=9200

```

Este archivo define dos servicios: uno para la aplicación NestJS y otro para Elasticsearch.

47.5 Paso 2: Crear la Aplicación NestJS

47.5.1 2.1. Crear una nueva aplicación NestJS

Usa npx para crear una nueva aplicación NestJS:

```
npx @nestjs/cli new search-app
cd search-app
```

47.5.2 2.2. Instalar Dependencias de Elasticsearch

Dentro del directorio de tu proyecto, instala las dependencias necesarias para conectarte a Elasticsearch:

```
npm install @nestjs/elasticsearch elasticsearch
```

47.6 Paso 3: Implementar el Motor de Búsqueda

47.6.1 3.1. Configurar el Módulo de Elasticsearch

Crema un módulo para Elasticsearch en src/elasticsearch/elasticsearch.module.ts:

```
import { Module } from '@nestjs/common';
import { ElasticsearchModule } from '@nestjs/elasticsearch';
import { MyElasticsearchService } from './elasticsearch.service'; // Cambia Elasticsearch

@Module({
  imports: [
    ElasticsearchModule.register({
      node: process.env.ELASTICSEARCH_HOST || 'http://localhost:9200',
    }),
  ],
  providers: [MyElasticsearchService],
  exports: [MyElasticsearchService],
})
export class MyElasticsearchModule {}
```

47.7 3.2. Crear el Servicio de Elasticsearch

Crema un servicio en src/elasticsearch/elasticsearch.service.ts:

```

import { Injectable } from '@nestjs/common';
import { ElasticsearchService } from '@nestjs/elasticsearch'; // Esto está bien

@Injectable()
export class MyElasticsearchService {
  constructor(private readonly elasticsearchService: ElasticsearchService) {}

  async indexData(index: string, document: any) {
    return await this.elasticsearchService.index({
      index,
      body: document,
    });
  }

  async search(index: string, query: any) {
    return await this.elasticsearchService.search({
      index,
      body: {
        query,
      },
    });
  }
}

```

Este servicio proporciona métodos para indexar datos y realizar búsquedas.

47.7.1 3.3. Crear un Controlador para Manejar las Búsquedas

Crema un controlador en src/search/search.controller.ts:

```

import { Controller, Get, Query } from '@nestjs/common';
import { MyElasticsearchService } from '../elasticsearch.service'; // Esto está bien

@Controller('search')
export class SearchController {
  constructor(private readonly elasticsearchService: MyElasticsearchService) {}

  @Get()
  async search(@Query('q') query: string) {
    const results = await this.elasticsearchService.search('products', {
      match: { name: query },
    });
    return results.hits.hits;
  }
}

```

Este controlador permite a los usuarios buscar productos por nombre.

47.8 Paso 4: Probar la Aplicación

47.8.1 4.1 Levanta los servicios con Docker Compose:

```
docker compose build --no-cache
```

En el comando anterior se construyen las imágenes de Docker para la aplicación NestJS y Elasticsearch.

```
docker compose up
```

En el comando anterior se inician los contenedores de Docker para la aplicación NestJS y Elasticsearch.

47.9 Paso 5: Configurar el Índice de Elasticsearch

47.9.1 5.1. Crear el Índice

Para crear un índice en Elasticsearch, puedes agregar un script que se ejecute cuando se inicie la aplicación, o hacerlo manualmente usando cURL:

47.10 Paso 6: Probar la Aplicación

Para probar si tu aplicación está funcionando correctamente, puedes seguir estos pasos:

47.10.1 6.1. Verificar el Estado de los Contenedores

Primero, asegúrate de que los contenedores están en ejecución:

```
docker compose ps
```

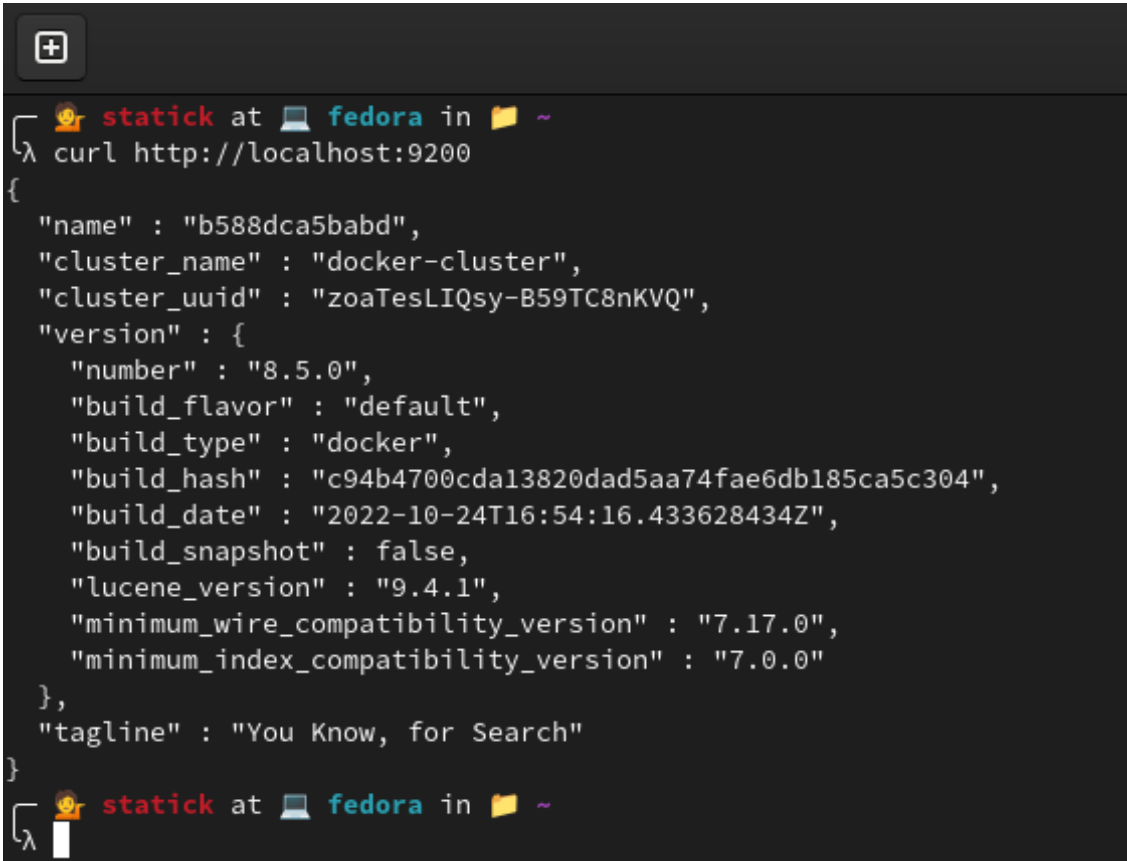
Esto debería mostrar que tanto elasticsearch como nestjs-app están en estado Up.

47.10.2 6.2. Probar Elasticsearch

Verificar la Disponibilidad del Servicio

Puedes verificar si Elasticsearch está funcionando enviando una solicitud HTTP al puerto 9200. Usa curl para hacer una solicitud simple:

```
curl http://localhost:9200
```



```
statick at fedora in ~
λ curl http://localhost:9200
{
  "name" : "b588dca5babd",
  "cluster_name" : "docker-cluster",
  "cluster_uuid" : "zoaTesLIQsy-B59TC8nKVQ",
  "version" : {
    "number" : "8.5.0",
    "build_flavor" : "default",
    "build_type" : "docker",
    "build_hash" : "c94b4700cda13820dad5aa74fae6db185ca5c304",
    "build_date" : "2022-10-24T16:54:16.433628434Z",
    "build_snapshot" : false,
    "lucene_version" : "9.4.1",
    "minimum_wire_compatibility_version" : "7.17.0",
    "minimum_index_compatibility_version" : "7.0.0"
  },
  "tagline" : "You Know, for Search"
}
```

Deberías recibir una respuesta JSON con información sobre el estado del clúster de Elasticsearch.

Consultar el Estado de Elasticsearch desde el Contenedor de NestJS

Para verificar la conectividad entre los servicios, puedes usar curl dentro del contenedor de NestJS:

```
docker exec -it laboratorio_indexacion_datos-nestjs-app-1 sh
# Dentro del contenedor
apk add --no-cache curl # Si `curl` no está instalado
curl http://elasticsearch:9200
```

Esto debería dar un resultado como este:

```
Terminal
[+]
statick at fedora in ~
λ docker exec -it laboratorio_indexacion_datos-nestjs-app-1 sh
/app # apk add --no-cache curl
fetch https://dl-cdn.alpinelinux.org/alpine/v3.20/main/x86_64/APKINDEX.tar.gz
fetch https://dl-cdn.alpinelinux.org/alpine/v3.20/community/x86_64/APKINDEX.tar.gz
OK: 16 MiB in 26 packages
/app # curl http://elasticsearch:9200
{
  "name" : "b588dca5babd",
  "cluster_name" : "docker-cluster",
  "cluster_uuid" : "zoaTesLIQsy-B59TC8nKVQ",
  "version" : {
    "number" : "8.5.0",
    "build_flavor" : "default",
    "build_type" : "docker",
    "build_hash" : "c94b4700cda13820dad5aa74fae6db185ca5c304",
    "build_date" : "2022-10-24T16:54:16.433628434Z",
    "build_snapshot" : false,
    "lucene_version" : "9.4.1",
    "minimum_wire_compatibility_version" : "7.17.0",
    "minimum_index_compatibility_version" : "7.0.0"
  },
  "tagline" : "You Know, for Search"
}
/app #
```

Esto verifica si el contenedor de NestJS puede conectarse al servicio de Elasticsearch por nombre de host.

47.10.3 6.3. Probar la Aplicación NestJS

Verificar la Disponibilidad del Servicio de NestJS

Puedes probar si tu aplicación NestJS está funcionando accediendo al puerto 3000:

```
curl http://localhost:3000
```

Esto debería devolver la respuesta "Hello World!" u otro mensaje configurado en tu aplicación NestJS.

```
Terminal
[+]
statick at fedora in ~
λ curl http://localhost:3000
Hello World!
λ
```

Verificar la Conectividad con Elasticsearch desde la Aplicación NestJS

```
Terminal
[+] statick at fedora in ~
λ docker exec -it laboratorio_indexacion_datos-nestjs-app-1 sh
/app # apk add --no-cache curl
fetch https://dl-cdn.alpinelinux.org/alpine/v3.20/main/x86_64/APKINDEX.tar.gz
fetch https://dl-cdn.alpinelinux.org/alpine/v3.20/community/x86_64/APKINDEX.tar.gz
OK: 16 MiB in 26 packages
/app # curl http://elasticsearch:9200
{
  "name" : "b588dca5babd",
  "cluster_name" : "docker-cluster",
  "cluster_uuid" : "zoaTesLIQsy-B59TC8nKVQ",
  "version" : {
    "number" : "8.5.0",
    "build_flavor" : "default",
    "build_type" : "docker",
    "build_hash" : "c94b4700cda13820dad5aa74fae6db185ca5c304",
    "build_date" : "2022-10-24T16:54:16.433628434Z",
    "build_snapshot" : false,
    "lucene_version" : "9.4.1",
    "minimum_wire_compatibility_version" : "7.17.0",
    "minimum_index_compatibility_version" : "7.0.0"
  },
  "tagline" : "You Know, for Search"
}
/app # curl http://localhost:3000
Hello World!/app #
```

Para probar la integración de tu aplicación NestJS con Elasticsearch:

- Asegúrate de que tu aplicación NestJS tiene configurado correctamente el host y puerto de Elasticsearch.
- Implementa una ruta en tu aplicación que realice una consulta a Elasticsearch y devuelva los resultados.

47.11 Revisar los Logs de los Contenedores

Si encuentras problemas, revisa los logs de los contenedores para identificar errores o mensajes informativos:

```
docker compose logs elasticsearch
docker compose logs nestjs-app
```

Esto te ayudará a diagnosticar problemas con la inicialización o la conexión entre los servicios.

47.12 Pruebas Adicionales

- Pruebas Unitarias/Integración:
- Si has escrito pruebas unitarias o de integración para tu aplicación, asegúrate de ejecutarlas para verificar que todos los componentes funcionan correctamente en conjunto.

47.13 Interfaz de Usuario:

- Si tu aplicación tiene una interfaz de usuario (UI), verifica que las páginas se carguen correctamente y que las funcionalidades (como la búsqueda en Elasticsearch) funcionen según lo esperado.

48 Reto

- Implementa datos de prueba para indexar en Elasticsearch, como productos, documentos o artículos.
- Implementa una funcionalidad de autocompletado en la barra de búsqueda de tu aplicación.
- Agrega filtros de búsqueda para permitir a los usuarios refinar los resultados por categoría, precio, etc.
- Implementa paginación para mostrar los resultados de búsqueda en varias páginas.

49 Conclusión

Este laboratorio proporciona a los estudiantes una introducción completa a la implementación de un sistema de búsqueda e indexación utilizando Elasticsearch y NestJS. Aprenden a configurar un entorno de desarrollo con Docker, indexar datos, y realizar búsquedas optimizadas. Al finalizar, los estudiantes tendrán una comprensión sólida de cómo integrar motores de búsqueda en aplicaciones web y cómo optimizar los índices y consultas para mejorar el rendimiento.

50 Laboratorio: Análisis de Texto con Python - Detección de Bigramas y Colocaciones.



50.1 Objetivo

El objetivo de este laboratorio es aprender a utilizar herramientas de procesamiento de lenguaje natural (NLP) en Python, como NLTK y Gensim, para analizar un corpus de texto, extraer bigramas significativos y detectar colocaciones. Al finalizar el laboratorio, podrás aplicar estos conocimientos en tareas de análisis de texto y minería de datos en diversos campos de aplicación.

50.2 Requisitos Previos

- Conocimiento básico de Python, incluyendo la sintaxis y las estructuras de datos.
- Familiaridad con el procesamiento de texto y conceptos de procesamiento de lenguaje natural (NLP), como tokenización, lematización y extracción de características.
- Experiencia en el uso de bibliotecas populares de Python para NLP, como NLTK, spaCy y Gensim.

- Conocimiento de técnicas de modelado de lenguaje, como n-gramas, modelos de bolsa de palabras y modelos de lenguaje basados en redes neuronales.
- Comprensión de conceptos estadísticos básicos, como la frecuencia de palabras y la medida de información mutua puntual (PMI).
- Familiaridad con el análisis de datos y la manipulación de datos en Python, utilizando bibliotecas como pandas y numpy.
- Conocimiento de visualización de datos utilizando bibliotecas como matplotlib y seaborn.
- Experiencia en la implementación de aplicaciones web utilizando frameworks como Flask o Django.
- Familiaridad con el uso de herramientas de línea de comandos y la instalación de paquetes de Python utilizando pip.

50.3 Instalación de Bibliotecas Necesarias

Empezamos creando un entorno virtual con venv y activándolo:

```
python -m venv env
source env/bin/activate
```

instalar las bibliotecas necesarias:

```
pip install matplotlib pandas numpy nltk gensim
```

50.4 Desarrollo del Laboratorio

51 Conceptos Básicos

51.1 Tokenización

La tokenización es el proceso de dividir un texto en unidades más pequeñas, como palabras o frases, llamadas tokens. Es un paso fundamental en el procesamiento de texto para la extracción de información y análisis lingüístico.

51.2 Bigrams

Un bigram es una secuencia de dos palabras consecutivas en un texto. Los bigrams son útiles para capturar relaciones entre palabras que no se pueden detectar con palabras individuales.

51.3 Colocaciones

Las colocaciones son combinaciones de palabras que ocurren juntas con mayor frecuencia de lo que se esperaría por azar. Detectar colocaciones es esencial para entender el contexto en el que se utilizan ciertas palabras.

52 Desarrollo del Laboratorio

52.1 Paso 0: Creamos el Notebook

Creamos un nuevo notebook de Jupyter y lo guardamos como `analisis_texto.ipynb`. A continuación, importamos las bibliotecas necesarias y descargamos los recursos de NLTK:

52.2 Paso 1: Importar Bibliotecas

Primero, importamos las bibliotecas necesarias para el análisis:

```
%matplotlib inline
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import re
import nltk
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.collocations import BigramAssocMeasures, BigramCollocationFinder
from gensim.models.phrases import Phrases, Phraser

# Descargar recursos necesarios de NLTK
nltk.download('punkt')
nltk.download('reuters')
from nltk.corpus import reuters
```

Explicación: Estas bibliotecas proporcionan las herramientas necesarias para la tokenización, la detección de bigrams y el análisis de colocaciones. También descargamos el corpus de Reuters para usarlo en nuestros ejemplos.

52.3 Paso 2: Cargar y Preparar el Corpus

Vamos a cargar el corpus de Reuters y preparar los documentos para el análisis:

```
documents = [reuters.raw(fileid).lower() for fileid in reuters.fileids()]

# Ver el primer documento
print(documents[0])
```

Explicación: El corpus de Reuters se compone de varios documentos de texto. En esta celda, cargamos todos los documentos y los convertimos a minúsculas para normalizar el texto.

52.4 Paso 3: Tokenización

Tokenizamos el corpus en palabras para procesar los bigramas:

```
tokens = [word for doc in documents for word in word_tokenize(doc)]
print(tokens[:15])
```

Explicación: Aquí convertimos cada documento en una lista de palabras (tokens). Esto es crucial para los análisis posteriores, ya que necesitamos trabajar con unidades básicas de texto.

52.5 Paso 4: Detección de Bigramas con NLTK

Utilizamos NLTK para encontrar bigramas y aplicamos un filtro para remover aquellos con una frecuencia menor a 10:

```
bigram_measures = BigramAssocMeasures()
finder = BigramCollocationFinder.from_words(tokens)
finder.apply_freq_filter(10)
bigramas = finder.nbest(bigram_measures.pmi, n=50)

print(bigramas)
```

Explicación: Usamos BigramCollocationFinder para encontrar bigramas en el texto tokenizado. Aplicamos un filtro para eliminar bigramas que aparezcan con poca frecuencia (menos de 10 veces) y seleccionamos los 50 más significativos según la medida PMI (Point-wise Mutual Information).

52.6 Paso 5: Detección de Colocaciones con Gensim

Ahora, utilizamos Gensim para detectar colocaciones en el texto:

```
sentences = [word_tokenize(sent) for sent in sent_tokenize("\n".join(documents).lower())]
sentences = [sent for sent in sentences if len(sent) > 1]

collocations = Phrases(sentences=sentences, min_count=10, threshold=0.5, scoring='npmi')
to_collocations = Phraser(collocations)

sent = 'new york is in united states of america. south africa and south america are in di
print(to_collocations[word_tokenize(sent)])
```

Explicación: Gensim se utiliza aquí para detectar colocaciones a partir de oraciones tokenizadas. El modelo de Phrases permite identificar secuencias de palabras que tienden a aparecer juntas. En este ejemplo, demostramos cómo funciona usando una oración de prueba.

52.7 Paso 6: Análisis de Colocaciones

Finalmente, analizamos y mostramos las colocaciones más significativas:

```
# Crear el objeto BigramCollocationFinder
collocations = BigramCollocationFinder.from_words(tokens)

# Usar BigramAssocMeasures para obtener las puntuaciones de los bigramas
scored = collocations.score_ngrams(BigramAssocMeasures().pmi)

# Crear un DataFrame con los bigramas y sus puntuaciones
df_collocations = pd.DataFrame(scored, columns=["bigram", "score"])

# Eliminar duplicados y ordenar por puntuación
df_collocations = df_collocations.drop_duplicates().sort_values(by="score", ascending=False)

# Imprimir los primeros 50 bigramas
print(df_collocations.head(50))
```

Explicación: Este paso combina el análisis de bigramas con la creación de un DataFrame para visualizar las colocaciones más significativas. Usamos el puntaje PMI para ordenar los bigramas según su relevancia.

53 Guardar el modelo de bigramas

```
to_collocations.save('bigram_model')
```

Explicación: Guardamos el modelo de bigramas para su uso posterior en la detección de colocaciones.

54 Implementación de una aplicación web

Para poner en práctica lo aprendido, puedes implementar una aplicación web que permita a los usuarios ingresar texto y detectar automáticamente las colocaciones más significativas. Esto puede ser útil en tareas de análisis de texto y minería de datos.

54.1 Paso 0: Creación de la aplicación web:

```
touch app.py
pip install streamlit
```

Implementación de la aplicación web:

54.2 Paso 1: Crear una Interfaz de Usuario

Utiliza una biblioteca como Streamlit para crear una interfaz de usuario simple donde los usuarios puedan ingresar texto y ver las colocaciones detectadas.

```
import streamlit as st

st.title("Detección de Colocaciones en Texto")

# Ingresar texto
text = st.text_area("Ingrese un texto:")
```

54.3 Paso 2: Procesar el Texto Ingresado

Utiliza el modelo de bigramas guardado para detectar colocaciones en el texto ingresado por el usuario.

```
from gensim.models.phrases import Phraser

# Cargar el modelo de bigramas
to_collocations = Phraser.load('bigram_model')

# Tokenizar el texto ingresado
tokens = word_tokenize(text.lower())
```

```
# Aplicar el modelo de bigramas
colocaciones = to_collocations[tokens]

# Mostrar las colocaciones detectadas
st.write("Colocaciones Detectadas:")
st.write(colocaciones)
```

54.4 Paso 3: Mostrar las Colocaciones Detectadas

Muestra las colocaciones más significativas en la interfaz de usuario para que los usuarios puedan verlas y analizarlas.

```
# Mostrar las colocaciones en una tabla
st.write("Colocaciones Detectadas:")
st.write(pd.Series(colocaciones).value_counts())
```

54.5 Paso 4: Ejecutar la Aplicación

Ejecuta la aplicación web y prueba la detección de colocaciones con diferentes textos de entrada.

```
streamlit run app.py
```

55 Probar la aplicación con estos ejemplos

Ejemplo 1: Artículo de Noticias

The stock market is showing signs of recovery as major indices reported gains for the thi

Ejemplo 2: Descripción de un Lugar

New York is a bustling metropolis known for its iconic landmarks such as the Statue of Li

Ejemplo 3: Texto Literario

It was the best of times, it was the worst of times, it was the age of wisdom, it was the

Ejemplo 4: Texto Técnico

Machine learning is a subset of artificial intelligence that focuses on the development o

Ejemplo 5: Texto de una Conversación

Hey, how are you doing today? I was thinking we could go to the park later if the weather

Cada uno de estos ejemplos te permitirá observar cómo el modelo detecta bigramas y colocaciones en distintos tipos de texto, desde descripciones técnicas hasta conversaciones informales.

56 Reto

Como reto adicional, puedes intentar mejorar la detección de colocaciones utilizando otros modelos de bigramas o trigramas, como los proporcionados por NLTK o spaCy. También puedes explorar la detección de colocaciones en otros idiomas o en textos especializados, como documentos médicos o legales.

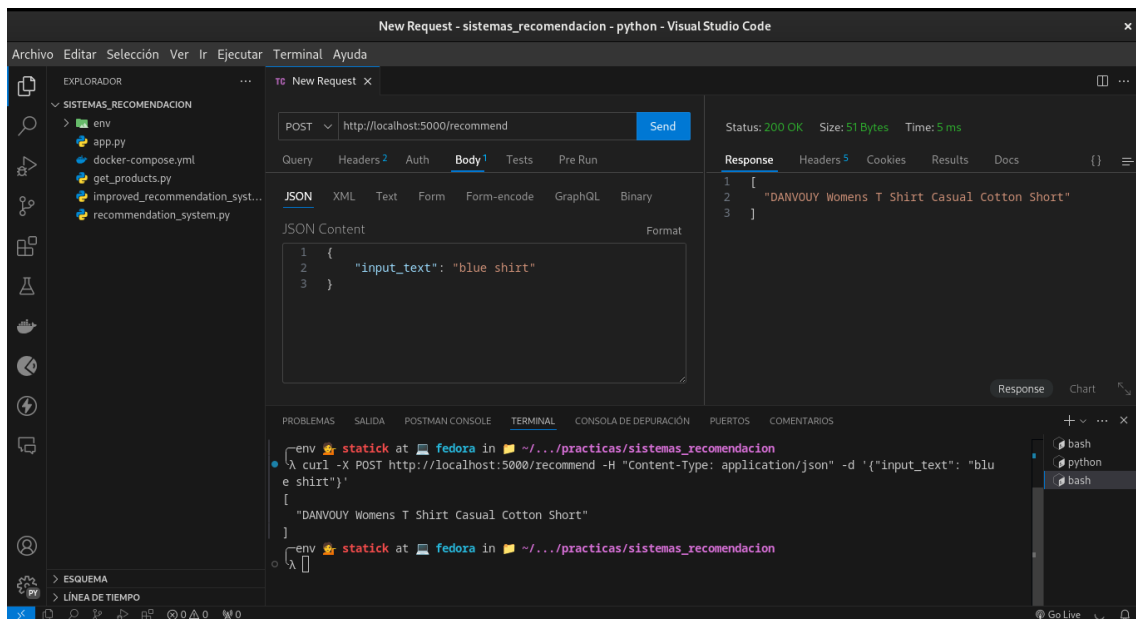
57 Recursos

- [NLTK Documentation](#)
- [Gensim Documentation](#)
- [Streamlit Documentation](#)
- [Corpus de Reuters en NLTK](#)
- [Bigram Collocations in NLTK](#)
- [Repositorio de Código en GitHub](#)

58 Conclusiones

En este laboratorio, hemos aprendido a utilizar herramientas de procesamiento de lenguaje natural (NLP) para analizar texto, detectar bigramas y colocaciones significativas. Hemos utilizado bibliotecas populares como NLTK y Gensim para realizar estas tareas y hemos implementado una aplicación web para demostrar la detección de colocaciones en texto ingresado por el usuario. Este conocimiento es fundamental para tareas de análisis de texto y minería de datos en diversos campos de aplicación.

59 Laboratorio: Introducción a los Sistemas de Recomendación



59.1 Objetivo

El objetivo de este laboratorio es aprender los conceptos básicos de los sistemas de recomendación y cómo implementarlos utilizando Python. En este laboratorio, construirás un sistema de recomendación simple que sugiere productos a los usuarios en función de una entrada de texto. Utilizarás herramientas como Redis para procesar datos y hacer recomendaciones. Además, aprenderás a utilizar la API de Fake Store para obtener datos de productos de ejemplo. Al finalizar el laboratorio, podrás mejorar el sistema de recomendación agregando más funcionalidades y refinando el algoritmo de recomendación. También podrás implementar el sistema de recomendación en un sistema web utilizando Flask para permitir a los usuarios realizar consultas y recibir recomendaciones de productos en tiempo real.

59.2 Conceptos clave

- **Sistemas de recomendación:** Los sistemas de recomendación son algoritmos que sugieren elementos a los usuarios en función de sus preferencias y comportamientos.

pasados. Estos sistemas son ampliamente utilizados en plataformas en línea para recomendar productos, películas, música, libros y otros elementos a los usuarios.

- **Redis:** Redis es una base de datos en memoria de código abierto que se utiliza para almacenar y procesar datos en tiempo real. Redis es ampliamente utilizado en aplicaciones web y sistemas de recomendación para almacenar y recuperar datos de forma eficiente.
- **API de Fake Store:** La API de Fake Store es una API de demostración que proporciona datos de productos ficticios para propósitos de demostración. La API de Fake Store se utiliza en este laboratorio para obtener datos de productos de ejemplo.
- **Flask:** Flask es un framework de desarrollo web ligero y flexible para Python. Flask se utiliza en este laboratorio para implementar un sistema de recomendación en un sistema web y permitir a los usuarios realizar consultas y recibir recomendaciones de productos en tiempo real.
- **Procesamiento de lenguaje natural (NLP):** El procesamiento de lenguaje natural es una rama de la inteligencia artificial que se ocupa de la interacción entre las computadoras y el lenguaje humano. En este laboratorio, se utiliza el procesamiento de lenguaje natural para analizar y comparar la entrada de texto y los títulos de los productos.
- **Docker:** Docker es una plataforma de código abierto que permite a los desarrolladores empaquetar, enviar y ejecutar aplicaciones en contenedores. Docker se utiliza en este laboratorio para instalar Redis en un contenedor y configurar un entorno de desarrollo.
- **Docker Compose:** Docker Compose es una herramienta que permite definir y ejecutar aplicaciones Docker multicontenedor. Docker Compose se utiliza en este laboratorio para configurar un contenedor de Redis y facilitar la instalación y configuración de Redis en un entorno de desarrollo.

59.3 Requisitos previos

En este laboratorio utilizamos la versión de python 3.12.4, el servidor redis será instalado en un contenedor de docker, por lo que necesitarás tener instalado Docker y Docker Compose en tu máquina. Además, necesitarás tener instaladas las bibliotecas **redis** y **requests** en tu entorno de Python. Puedes instalar las bibliotecas utilizando los siguientes comandos:

59.3.1 Creación de un entorno virtual

```
python3 -m venv env
source env/bin/activate
```


59.3.2 Instalación de Redis

```
pip install redis
```

Además, necesitarás una conexión a Internet para acceder a la API de Fake Store y obtener datos de productos de ejemplo.

59.3.3 Instalación de Redis

```
pip install requests
```

59.4 Introducción

Los sistemas de recomendación son algoritmos que sugieren elementos a los usuarios en función de sus preferencias y comportamientos pasados. Estos sistemas son ampliamente utilizados en plataformas en línea para recomendar productos, películas, música, libros y otros elementos a los usuarios. Los sistemas de recomendación pueden ser de varios tipos, como basados en contenido, filtrado colaborativo, filtrado basado en la popularidad, etc.

En este laboratorio, construirás un sistema de recomendación simple que sugiere productos a los usuarios en función de una entrada de texto. Utilizarás la API de Fake Store para obtener datos de productos de ejemplo y Redis para procesar datos y hacer recomendaciones.

59.5 Paso 0: Instalación de Redis con docker compose.

Para instalar Redis en tu máquina, puedes utilizar Docker Compose para configurar un contenedor de Redis. A continuación, se muestra un ejemplo de cómo instalar Redis con Docker Compose:

1. Crear un archivo llamado **docker-compose.yml** y agregar el siguiente código:

```
services:
  redis:
    image: redis
    container_name: redis
    ports:
      - "6379:6379"
```

2. Ejecutar el siguiente comando para iniciar el contenedor de Redis:

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

Con estos pasos, habrás configurado un contenedor de Redis en tu máquina y podrás utilizar Redis para almacenar y procesar datos en tu sistema de recomendación.

59.6 Paso 1: Obtener datos de productos de la API de Fake Store

En este paso, utilizarás la API de Fake Store para obtener datos de productos de ejemplo. La API de Fake Store proporciona datos de productos ficticios para propósitos de demostración. Puedes acceder a la API de Fake Store en la siguiente URL: <https://fakestoreapi.com/>.

Para obtener datos de productos de la API de Fake Store, puedes utilizar la biblioteca **requests** en Python. A continuación, se muestra un ejemplo de cómo obtener datos de productos de la API de Fake Store:

Para probar la obtención de datos de productos de la API de Fake Store, implementamos un script de Python que obtiene los datos de productos de la API y muestra los títulos de los productos en la consola.

Creemos un archivo llamado **get_products.py** y agregamos el siguiente código:

```
import requests

url = "https://fakestoreapi.com/products"
response = requests.get(url)
products = response.json()

for product in products:
    print(product["title"])
```

En este ejemplo, se utiliza la biblioteca **requests** para enviar una solicitud GET a la URL de la API de Fake Store y se obtienen los datos de productos en formato JSON. Luego, se imprime el título de cada producto en la consola.

59.7 Paso 2: Procesar datos de productos y construir un sistema de recomendación

En este paso, procesarás los datos de productos obtenidos de la API de Fake Store y construirás un sistema de recomendación simple que sugiere productos a los usuarios en función de una entrada de texto. Utilizarás Redis para almacenar y procesar los datos de productos y hacer recomendaciones.

Para construir el sistema de recomendación, puedes seguir los siguientes pasos:

1. Obtener datos de productos de la API de Fake Store.
2. Procesar los datos de productos y almacenarlos en Redis.
3. Implementar un algoritmo de recomendación simple que sugiere productos en función de una entrada de texto.

A continuación, se muestra un ejemplo de cómo procesar los datos de productos y construir un sistema de recomendación simple, creamos un archivo llamado **recommendation_system.py** y agregamos el siguiente código:

```
import redis
import requests

# Conectar a Redis
r = redis.Redis(host='localhost', port=6379, db=0)

# Obtener datos de productos de la API de Fake Store
url = "https://fakestoreapi.com/products"
response = requests.get(url)
products = response.json()

# Procesar los datos de productos y almacenarlos en Redis
for product in products:
    r.set(product["id"], product["title"])

# Implementar un algoritmo de recomendación simple
def recommend_products(input_text):
    recommendations = []
    for key in r.keys():
        title = r.get(key).decode('utf-8')
        if input_text.lower() in title.lower():
            recommendations.append(title)
    return recommendations

# Hacer recomendaciones basadas en una entrada de texto
input_text = "shirt"
recommendations = recommend_products(input_text)
print("Recomendaciones:")
for recommendation in recommendations:
    print(recommendation)
```

En este ejemplo, se utiliza Redis para almacenar los datos de productos obtenidos de la API de Fake Store. Luego, se implementa un algoritmo de recomendación simple que sugiere productos en función de una entrada de texto. El algoritmo busca coincidencias entre la entrada de texto y los títulos de los productos almacenados en Redis y devuelve una lista de recomendaciones.

59.7.1 Ejecución del código

Para ejecutar el código, puedes guardar el código en un archivo Python (por ejemplo, **recommendation_system.py**) y ejecutarlo desde la línea de comandos:

```
python recommendation_system.py
```

59.8 Paso 3: Mejorar el sistema de recomendación

En este paso, mejorarás el sistema de recomendación agregando más funcionalidades y refinando el algoritmo de recomendación. Puedes probar diferentes enfoques y técnicas para mejorar la precisión y relevancia de las recomendaciones. Algunas ideas para mejorar el sistema de recomendación incluyen:

- Utilizar técnicas de procesamiento de lenguaje natural (NLP) para analizar y comparar la entrada
- Implementar un algoritmo de filtrado colaborativo para hacer recomendaciones basadas en el comportamiento de los usuarios
- Utilizar técnicas de aprendizaje automático para mejorar la precisión de las recomendaciones

Puedes experimentar con diferentes enfoques y técnicas para mejorar el sistema de recomendación y hacer recomendaciones más precisas y relevantes.

59.9 Aplicación de mejoras

Para aplicar mejoras al sistema de recomendación, puedes probar diferentes enfoques y técnicas para mejorar la precisión y relevancia de las recomendaciones. Por ejemplo, puedes utilizar técnicas de procesamiento de lenguaje natural (NLP) para analizar y comparar la entrada, implementar un algoritmo de filtrado colaborativo para hacer recomendaciones basadas en el comportamiento de los usuarios, o utilizar técnicas de aprendizaje automático para mejorar la precisión de las recomendaciones.

A continuación, se muestra un ejemplo de cómo mejorar el sistema de recomendación utilizando técnicas de procesamiento de lenguaje natural (NLP) para analizar y comparar la entrada, creamos un archivo llamado **improved_recommendation_system.py** y agregamos el siguiente código:

```
import redis
import requests
import nltk
from nltk.tokenize import word_tokenize

# Descargar recursos de NLTK
nltk.download('punkt')

# Conectar a Redis
r = redis.Redis(host='localhost', port=6379, db=0)

# Obtener datos de productos de la API de Fake Store
url = "https://fakestoreapi.com/products"
```

```

response = requests.get(url)
products = response.json()

# Procesar los datos de productos y almacenarlos en Redis
for product in products:
    r.set(product["id"], product["title"])

# Implementar un algoritmo de recomendación mejorado
def recommend_products(input_text):
    recommendations = []
    input_tokens = word_tokenize(input_text.lower())
    for key in r.keys():
        title = r.get(key).decode('utf-8')
        title_tokens = word_tokenize(title.lower())
        if any(token in title_tokens for token in input_tokens):
            recommendations.append(title)
    return recommendations

# Hacer recomendaciones basadas en una entrada de texto
input_text = "blue shirt"
recommendations = recommend_products(input_text)
print("Recomendaciones:")
for recommendation in recommendations:
    print(recommendation)

```

En este ejemplo, se utiliza la biblioteca **nltk** para tokenizar la entrada de texto y los títulos de los productos. Luego, se compara si alguno de los tokens de la entrada de texto está presente en los títulos de los productos almacenados en Redis. Si se encuentra una coincidencia, se agrega el producto a la lista de recomendaciones.

59.9.1 Ejecución del código

Para ejecutar el código, puedes guardar el código en un archivo Python (por ejemplo, **improved_recommendation_system.py**) y ejecutarlo desde la línea de comandos:

```
python improved_recommendation_system.py
```

60 Implementación de recomendaciones en un sistema web

Para implementar recomendaciones en un sistema web, puedes utilizar un marco de desarrollo web como Flask para crear una aplicación web que permita a los usuarios ingresar una consulta y recibir recomendaciones de productos en función de la consulta. A continuación, se muestra un ejemplo de cómo implementar recomendaciones en un sistema web utilizando Flask:

1. Instalar Flask

```
pip install Flask
```

2. Crear una aplicación web con Flask

Creamos un archivo llamado **app.py** y agregamos el siguiente código:

```
from flask import Flask, request, jsonify
import redis
import nltk
from nltk.tokenize import word_tokenize

# Descargar recursos de NLTK
nltk.download('punkt')

# Conectar a Redis
r = redis.Redis(host='localhost', port=6379, db=0)

app = Flask(__name__)

# Implementar un algoritmo de recomendación mejorado

def recommend_products(input_text):
    recommendations = []
    input_tokens = word_tokenize(input_text.lower())
    for key in r.keys():
        title = r.get(key).decode('utf-8')
        title_tokens = word_tokenize(title.lower())
        if any(token in title_tokens for token in input_tokens):
            recommendations.append(title)
    return recommendations
```

```

@app.route('/recommend', methods=['POST'])
def get_recommendations():
    data = request.get_json()
    input_text = data['input_text']
    recommendations = recommend_products(input_text)
    return jsonify(recommendations)

if __name__ == '__main__':
    app.run(debug=True)

```

En este ejemplo, se utiliza Flask para crear una aplicación web que permite a los usuarios enviar una consulta a través de una solicitud POST y recibir recomendaciones de productos en función de la consulta. La aplicación web utiliza el algoritmo de recomendación mejorado implementado anteriormente para generar recomendaciones.

60.0.1 Ejecución de la aplicación web

Para ejecutar la aplicación web, puedes guardar el código en un archivo Python (por ejemplo, **app.py**) y ejecutarlo desde la línea de comandos:

```
python app.py
```

Luego, puedes enviar una solicitud POST a la URL de la aplicación web con la consulta de entrada para recibir recomendaciones de productos. Por ejemplo, puedes utilizar la herramienta **curl** para enviar una solicitud POST a la URL de la aplicación web:

```
curl -X POST http://localhost:5000/recommend -H "Content-Type: application/json" -d '{"in
```

La aplicación web responderá con una lista de recomendaciones de productos basadas en la consulta de entrada.

61 Probar la aplicación con Thunder Client

Para probar la aplicación web con Thunder Client, puedes instalar la extensión Thunder Client en Visual Studio Code y enviar una solicitud POST a la URL de la aplicación web con la consulta de entrada. A continuación, se muestra un ejemplo de cómo probar la aplicación web con Thunder Client:

1. Instalar la extensión Thunder Client en Visual Studio Code.
2. Abrir Thunder Client y crear una nueva solicitud POST con la URL de la aplicación web y la consulta de entrada.
3. Enviar la solicitud POST y recibir las recomendaciones de productos en la respuesta.

Para ello debes utilizar la siguiente URL: <http://localhost:5000/recommend>

61.0.1 Ejemplo

```
{  
  "input_text": "blue shirt"  
}
```

Se espera la salida de recomendaciones de productos basadas en la consulta de entrada.

Con Thunder Client, puedes probar la aplicación web y verificar si las recomendaciones de productos son precisas y relevantes.

61.1 Reto

Para el reto de este laboratorio, puedes mejorar el sistema de recomendación agregando más funcionalidades y refinando el algoritmo de recomendación. Puedes probar diferentes enfoques y técnicas para mejorar la precisión y relevancia de las recomendaciones. Algunas ideas para mejorar el sistema de recomendación incluyen:

- Utilizar técnicas de procesamiento de lenguaje natural (NLP) para analizar y comparar la entrada
- Implementar un algoritmo de filtrado colaborativo para hacer recomendaciones basadas en el comportamiento de los usuarios
- Utilizar técnicas de aprendizaje automático para mejorar la precisión de las recomendaciones

Puedes experimentar con diferentes enfoques y técnicas para mejorar el sistema de recomendación y hacer recomendaciones más precisas y relevantes.

También puedes implementar una aplicación de frontend para el sistema de recomendación utilizando un framework de desarrollo web como React o Angular. La aplicación de frontend puede permitir a los usuarios ingresar una consulta y recibir recomendaciones de productos en tiempo real.

61.2 Agregar .gitignore

Para evitar subir archivos innecesarios a tu repositorio, puedes agregar un archivo **.gitignore** para ignorar archivos y directorios específicos. A continuación se muestra un ejemplo de un archivo **.gitignore** para ignorar archivos y directorios generados por Python y Visual Studio Code:

```
# Byte-compiled / optimized / DLL files
__pycache__/
*.py[cod]
*$py.class

# C extensions
*.so

# Distribution / packaging
.Python
build/
develop-eggs/
dist/
downloads/
eggs/
.eggs/
lib/
lib64/
parts/
sdist/
var/
wheels/
share/python-wheels/
*.egg-info/
.installed.cfg
*.egg
MANIFEST

# Virtual environment
venv/
ENV/
env/
```

```
.venv/  
.env/  
env.bak/  
venv.bak/  
  
# PyInstaller  
# Usually these files are written by a python script from a template  
# before PyInstaller builds the exe, so as to inject date/other infos into it.  
*.manifest  
*.spec  
  
# Installer logs  
pip-log.txt  
pip-delete-this-directory.txt  
  
# Unit test / coverage reports  
htmlcov/  
.tox/  
.nox/  
.coverage  
.coverage.*  
.cache  
nosetests.xml  
coverage.xml  
*.cover  
*.py,cover  
.hypothesis/  
.pytest_cache/  
  
# Translations  
*.mo  
*.pot  
  
# Django stuff:  
*.log  
local_settings.py  
db.sqlite3  
db.sqlite3-journal  
  
# Flask stuff:  
instance/  
.webassets-cache  
  
# Scrapy stuff:  
.scrapy  
  
# Sphinx documentation  
docs/_build/
```

```
docs/_static/
docs/_templates/

# PyBuilder
target/

# Jupyter Notebook
.ipynb_checkpoints

# IPython
profile_default/
ipython_config.py

# pyenv
.python-version

# celery beat schedule file
celerybeat-schedule

# SageMath parsed files
*.sage.py

# Environments
.env
.venv
env/
venv/
ENV/
env.bak/
venv.bak/

# Spyder project settings
.spyderproject
.spyproject

# Rope project settings
.ropeproject

# mkdocs documentation
/site

# mypy
.mypy_cache/
.dmypy.json
dmypy.json

# Pyre type checker
.pyre/
```

```
# pytype static type analyzer
.pytype/

# Cython debug symbols
cython_debug/
```

61.3 Agregar README.md

Para documentar tu proyecto y proporcionar instrucciones de uso, puedes agregar un archivo **README.md** con información detallada sobre el sistema de recomendación y cómo utilizarlo. A continuación se muestra un ejemplo de un archivo **README.md** para un sistema de recomendación:

```
# Sistema de Recomendación

Este es un sistema de recomendación simple que sugiere productos a los usuarios en función de sus gustos.

## Requisitos previos

Para ejecutar el sistema de recomendación, necesitarás tener instalado Docker y Docker Compose.

## Ejecución del sistema de recomendación

1. Clona el repositorio:

bash
git clone https://github.com/statick88/SistemaRecomendacion.git

2. Instalación de bibliotecas desde el archivo requirements.txt

Para instalar las bibliotecas necesarias para ejecutar el sistema de recomendación, puedes ejecutar el siguiente comando:

bash
pip install -r requirements.txt

3. Levantar el servidor Redis con Docker Compose

Para iniciar el contenedor de Redis, puedes utilizar Docker Compose para configurar el contenedor de Redis.

bash
docker compose up --build -d

4. Ejecutar el sistema de recomendación

Par ejecutar el sistema de recomendación existen 3 formas en este repositorio:
```

- **recommendation_system.py**: Este script de Python se puede ejecutar para obtener reco

```
bash
python recommendation_system.py
```

- **improve_recommendations_system.py**: Este script de Python se puede ejecutar para mej

```
bash
python improve_recommendations_system.py
```

- **app.py**: Este script de Python contiene el código de la aplicación web que proporció

```
bash
python app.py
```

5. Acceder a la aplicación web para recibir recomendaciones de productos

Una vez que hayas ejecutado el script **app.py**, puedes acceder a la aplicación web para

La url de la aplicación web es: [http://localhost:5000/recommend] (http://localhost:5000/r

Ejemplo de solicitud POST:

```
json
{
  "input_text": "blue shirt"
}
```

Con estos pasos, habrás ejecutado el sistema de recomendación y accedido a la aplicación

El archivo **requirements.txt** contendrá una lista de las bibliotecas y sus versiones necesarias para ejecutar el sistema de recomendación. Puedes utilizar el archivo **requirements.txt** para instalar las bibliotecas necesarias en un entorno virtual de Python.

61.4 Inicializamos el repositorio git

Para inicializar un repositorio git en tu proyecto, puedes ejecutar los siguientes comandos en la terminal:

```
git init
git add .
git commit -m "Initial commit"
```

Con estos comandos, habrás inicializado un repositorio git en tu proyecto y realizado el primer commit con los archivos y código fuente del sistema de recomendación.

61.5 Recursos

- [API de Fake Store](#)
- [Documentación de Redis](#)
- [Documentación de Flask](#)
- [Documentación de NLTK](#)
- [Documentación de Thunder Client](#)
- [Tutorial de Git](#)
- [Repositorio de ejemplo en GitHub](#)

61.6 Conclusión

En este laboratorio, aprendiste los conceptos básicos de los sistemas de recomendación y cómo implementarlos utilizando Python. Construiste un sistema de recomendación simple que sugiere productos a los usuarios en función de una entrada de texto. Utilizaste herramientas como Redis para procesar datos y hacer recomendaciones. Además, aprendiste a utilizar la API de Fake Store para obtener datos de productos de ejemplo.

Durante el laboratorio, exploraste diferentes mejoras para el sistema de recomendación, como utilizar técnicas de procesamiento de lenguaje natural (NLP), implementar algoritmos de filtrado colaborativo y aplicar técnicas de aprendizaje automático. Estas mejoras pueden ayudar a mejorar la precisión y relevancia de las recomendaciones.

Además, consideraste la posibilidad de implementar una aplicación de frontend utilizando un framework de desarrollo web como React o Angular. Esto permitiría a los usuarios interactuar con el sistema de recomendación de manera más intuitiva y recibir recomendaciones en tiempo real.

En resumen, este laboratorio te brindó una introducción práctica a los sistemas de recomendación y te proporcionó las herramientas y conocimientos necesarios para construir y mejorar un sistema de recomendación utilizando Python.

¡Ahora estás listo para aplicar estos conceptos y técnicas en tus propios proyectos de recomendación!