UNIVERSIDAD NACIONAL DE CHIMBORAZO

FACULTAD DE INGENIERÍA

INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN

NOMBRE:

Elvis Santiago Pilco Paucar

FECHA: 2024/07/15

INTEROPERABILIDAD DE PLATAFORMAS

PRUEBA TEORICA - VIDEO

PERIODO ACADÉMICO 2024-1S

1. OBJETIVO GENERAL, ESPECÍFICOS, RECURSOS

1.1. OBJETIVO GENERAL

El objetivo general del semestre es aplicar los conocimientos teóricos y prácticos adquiridos en la unidad sobre arquitectura y diseño de sistemas distribuidos. Esto se logrará a través del desarrollo de varias aplicaciones y análisis tecnológicos, que incluyen el uso de tecnologías avanzadas y el desarrollo de aplicaciones distribuidas interoperables.

1.2. OBJETIVOS ESPECÍFICOS

- Aplicar los conocimientos teóricos relacionados con la arquitectura y el diseño de sistemas distribuidos adquiridos durante el semestre mediante el desarrollo de una aplicación distribuida e interoperable utilizando tecnología RPC (Remote Procedure Call).
- Crear dos aplicaciones que utilicen las API públicas de OpenWeatherMap y de geolocalización para obtener y analizar datos meteorológicos y de ubicación, respectivamente. Estas aplicaciones están diseñadas para ofrecer a los usuarios herramientas que permitan visualizar tendencias y patrones climáticos, así como obtener la ubicación mediante coordenadas de latitud y longitud.
- Analizar y evaluar el uso de tecnologías .NET para el desarrollo de servicios REST, destacando sus ventajas, desventajas y mejores prácticas, con el fin de proporcionar una guía completa para desarrolladores interesados en implementar soluciones web escalables y eficientes.
- Desarrollar una comprensión profunda y práctica de la creación, gestión y despliegue de aplicaciones basadas en microservicios utilizando Spring Boot, Spring Boot Eureka y Spring Boot Gateway, con el fin de construir sistemas escalables, flexibles y resilientes que cumplan con los estándares industriales y las mejores prácticas en desarrollo de software.

1.3. RECURSOS

- Computador
- XAMPP
- Composer
- GIT

- VirtualBox
- Navegador web
- Visual Studio Code
- Open Weather Map
- Google Maps
- Visual Studio
- Postman
- SpringToolSuite4

2. INTRODUCCIÓN.

En el escenario tecnológico actual, los sistemas distribuidos son indiscutiblemente un pilar importante para la informática moderna. Estos sistemas, definidos como una colección de computadoras independientes que aparecen frente a los usuarios como una única computadora, brindan una solución innovadora para enfrentar los desafíos de escalabilidad, disponibilidad y rendimiento en aplicaciones y servicios. La naturaleza distribuida de estos sistemas les permite ser altamente resilientes y capaces de soportar fallos individuales sin interrumpir el funcionamiento general. Asimismo, su arquitectura descentralizada permite agregar más nodos al sistema para aumentar su capacidad, al mismo tiempo que se aprovechan mejor los recursos disponibles.

Este tipo de sistemas se utilizan en una amplia variedad de contextos, desde aplicaciones web y móviles hasta sistemas de control de procesos y redes de sensores. Ejemplos de su uso incluyen la gestión de bases de datos, el procesamiento de imágenes y vídeos, el análisis de datos, y el procesamiento de transacciones financieras. En un futuro cada vez más interconectado, los sistemas distribuidos serán fundamentales en la infraestructura tecnológica, siendo esencial comprender sus principios y desafíos para diseñar y desarrollar soluciones eficientes y robustas en un mundo digital en constante evolución.

La era digital ha traído consigo una diversidad de plataformas y servicios interconectados, cada uno con sus propias interfaces de programación de aplicaciones (APIs). Estas APIs permiten que los sistemas se comuniquen entre sí, posibilitando el intercambio de datos y la automatización de tareas. En este proyecto, se explorará la implementación y el consumo de APIs REST públicos,

específicamente el API de OpenWeatherMap y el API de Google Maps, para demostrar su eficacia y versatilidad en la gestión de datos y la interacción con plataformas de terceros.

OpenWeatherMap proporciona acceso a datos meteorológicos de todo el mundo a través de su API REST público, permitiendo a los desarrolladores solicitar datos climáticos actuales y previsiones para cualquier ubicación. En este proyecto, esta API se usará para obtener información sobre el clima de las ciudades de Ecuador. Los datos recogidos se procesarán y almacenarán para su posterior uso, demostrando el consumo efectivo de APIs REST públicos y la gestión de datos.

Por otro lado, Google Maps, una de las plataformas de mapeo y geolocalización más populares del mundo, también proporciona un API REST público que permite obtener la ubicación mediante coordenadas de latitud y longitud. En el proyecto, se utilizará el API de Google Maps para obtener y mostrar la ubicación precisa de los usuarios, demostrando la utilidad del API y su capacidad para automatizar la interacción con servicios de geolocalización. El objetivo último es integrar los dos servicios, proporcionando una solución completa de información meteorológica y geolocalización.

El desarrollo de aplicaciones y servicios web es crucial para la operación, la competitividad y el crecimiento sostenido de las empresas en la era digital actual. Las tecnologías .NET y los servicios REST se han consolidado como componentes esenciales en la arquitectura moderna de software, ofreciendo soluciones robustas y escalables. .NET, un marco de desarrollo de software creado por Microsoft, proporciona una plataforma flexible y potente para la construcción de aplicaciones de alto rendimiento. Los servicios REST, por su parte, permiten la comunicación entre sistemas mediante operaciones CRUD, facilitando la integración de diferentes sistemas y aplicaciones.

Este informe explorará en profundidad las tecnologías .NET y su aplicación en la creación de servicios REST. Se abordarán las características distintivas y los beneficios de .NET, y se presentará una guía práctica para la implementación de un servicio REST utilizando .NET, incluyendo ejemplos de código y proyectos de referencia.

El desarrollo de aplicaciones escalables, mantenibles y de alto rendimiento es fundamental para satisfacer las demandas del mercado y las expectativas de los usuarios. La arquitectura de microservicios, que permite dividir una aplicación monolítica en pequeños servicios independientes,

se ha adoptado ampliamente para abordar estos desafíos. Spring Boot es un marco de desarrollo que simplifica el proceso de configuración y lanzamiento de aplicaciones Java, promoviendo prácticas de desarrollo limpio y estructurado.

En una arquitectura de microservicios, la gestión de servicios y la comunicación entre ellos es crucial. Spring Boot Eureka facilita la localización y el registro de microservicios en un sistema distribuido, permitiendo una arquitectura dinámica y resiliente. Spring Boot Gateway actúa como un punto de entrada único para todas las solicitudes dirigidas a los microservicios, centralizando funciones de enrutamiento, balanceo de carga, y políticas de seguridad y autenticación.

La combinación de Spring Boot, Spring Boot Eureka y Spring Boot Gateway forma una poderosa infraestructura para desarrollar aplicaciones basadas en microservicios, abordando los desafíos de escalabilidad, resiliencia y mantenibilidad en el desarrollo de software moderno. A través de un enfoque integral que combina teoría y práctica, los desarrolladores podrán construir sistemas que cumplen con los requisitos actuales y están preparados para evolucionar con las demandas futuras del mercado.

3. MARCO TEÓRICO.

3.1. SISTEMA DISTRIBUIDO

Un sistema distribuido es una colección de computadoras interconectadas a través de una red que trabajan juntas para lograr una tarea común [4]. El objetivo de estos sistemas es compartir recursos y coordinar la ejecución de tareas de manera eficiente, distribuyendo la carga de trabajo entre múltiples ordenadores. La implementación sistemas distribuidos presenta desafíos técnicos relacionados con la gestión de concurrencia, y la garantía de la consistencia en la distribución de datos en tiempo real. [4] Aún así, este tipo de sistemas pueden proporcionar ventajas significativas sobre otras tecnologías entre las que destacan la redundancia y tolerancia a fallos; con el objetivo de mejorar la eficiencia de las operaciones y reducir costos en diversas aplicaciones empresariales.

3.2. RPC

Un sistema RPC (Remote Procedure Call) es un modelo que permite a una aplicación solicitar a otra aplicación o proceso la ejecución de una función en otra máquina a través de la red. En este modelo, la máquina solicitante envía una solicitud al servidor, que procesa la solicitud y devuelve el resultado

a través de la red [5]. De esta manera, las aplicaciones pueden interactuar y trabajar juntas de manera eficiente y coordinada, sin importar su ubicación física. El uso de sistemas RPC puede ayudar a simplificar el diseño y la implementación de sistemas distribuidos dado que permite una comunicación segura y eficiente entre diferentes aplicaciones. Los sistemas RPC se utilizan comúnmente en entornos empresariales para permitir la interacción entre diferentes sistemas y aplicaciones, y existen diferentes implementaciones y protocolos entre los que están RPC, XMLRPC, JSON-RPC, etc [5].

3.3. WEB SERVICE

Un servicio web es una tecnología de comunicación que permite a diferentes aplicaciones y sistemas intercambiar datos a través de una red, utilizando un conjunto de estándares y protocolos comunes. El objetivo principal de los servicios web es proporcionar una plataforma independiente del lenguaje de programación y la plataforma [9], lo que permite que diferentes sistemas se comuniquen de manera efectiva, se integren fácilmente y se reutilicen. Los servicios web se utilizan en una amplia variedad de aplicaciones, desde la integración de sistemas empresariales hasta el intercambio de información entre diferentes plataformas web y móviles, mediante dos estándares tecnológicos principales: SOAP y REST, que permiten estructurar la información intercambiada mediante protocolos como HTTP.

3.4. SERVICIOS WEB EXTERNO Y SU CONSUMO

A servicio web externo se refiere a aquellos servicios web que se encuentran fuera de una empresa u organización, es decir, que son proporcionados por terceros. Estos servicios permiten a las empresas acceder a una amplia variedad de recursos y funcionalidades, lo que puede resultar en una mayor eficiencia y competitividad en el mercado. Los servicios web externos utilizan estándares comunes como SOAP y REST y se comunican a través de protocolos de comunicación como HTTP

3.5. API

Un API (Application Programming Interface) es un conjunto de reglas y protocolos que permite la comunicación entre diferentes sistemas de software. Sirve como un intermediario que permite a las aplicaciones interactuar entre sí, solicitando datos o servicios específicos. Los APIs definen la forma en que se deben estructurar las solicitudes y las respuestas, y proporcionan una interfaz estandarizada para acceder y manipular datos [3]. Estos pueden ser utilizados para acceder a funcionalidades de otros programas, servicios web o sistemas operativos. Los APIs facilitan la integración y el

intercambio de información entre diferentes aplicaciones, lo que permite la creación de soluciones más robustas y flexibles.

3.6.API REST:

Un API REST (Representational State Transfer) es un estilo de arquitectura para el desarrollo de servicios web que se basa en el protocolo HTTP. [4]Este enfoque utiliza los métodos estándar de HTTP (GET, POST, PUT, DELETE) para operar sobre los recursos y manipular su estado. Un API REST se centra en los recursos y utiliza URLs para identificarlos de manera única. Los datos suelen ser transferidos en formato JSON o XML. Los APIs REST son ampliamente utilizados debido a su simplicidad, escalabilidad y facilidad de consumo. Proporcionan una forma estándarizada de acceso a los recursos y permiten la interoperabilidad entre diferentes sistemas.

3.7. API REST públicos:

Los APIs REST públicos son aquellos que se ofrecen de forma abierta y gratuita para que los desarrolladores puedan acceder y utilizar sus servicios. Estos APIs permiten a los desarrolladores interactuar con sistemas y plataformas externas sin necesidad de tener acceso directo a su infraestructura interna [5]. Los APIs REST públicos proporcionan una forma estandarizada de acceso a datos y funcionalidades específicas. Al ser públicos, suelen requerir una clave de API para controlar el acceso y limitar el uso indebido. Estos APIs son utilizados por desarrolladores de todo el mundo para crear aplicaciones y servicios que aprovechan la funcionalidad y los datos ofrecidos por las organizaciones propietarias de los APIs.

3.8. Consumo de API

El consumo de un API implica utilizar sus endpoints y métodos para realizar solicitudes y obtener datos o servicios específicos. Los desarrolladores deben seguir la documentación proporcionada por el proveedor del API para comprender cómo autenticarse, estructurar las solicitudes, y procesar las respuestas [6]. El consumo de un API implica enviar solicitudes HTTP utilizando métodos como GET, POST, PUT o DELETE, y recibir respuestas en formatos como JSON o XML. El proceso de consumo puede incluir la autenticación, la manipulación de parámetros de consulta, el manejo de errores y la interpretación de los datos recibidos. El consumo de APIs es fundamental en la integración de servicios y en el desarrollo de aplicaciones que requieren acceso a datos externos.

3.9. OpenWeathermap API

El API de OpenWeathermap es un servicio web que proporciona datos meteorológicos actuales y

pronósticos para ubicaciones en todo el mundo [1]. Permite a los desarrolladores obtener información sobre temperatura, humedad, velocidad del viento, condiciones climáticas y otros datos relacionados con el clima. Para utilizar el API de OpenWeathermap, los desarrolladores deben registrarse en el sitio web de OpenWeathermap y obtener una clave de API. Luego, pueden realizar solicitudes al API utilizando la clave de API, especificando la ubicación deseada. Las respuestas del API contienen los datos climáticos correspondientes que se pueden utilizar en aplicaciones y servicios relacionados con el clima.

3.10. Google Maps API

La API de Google Maps es un conjunto de servicios proporcionados por Google que permite a los desarrolladores integrar las capacidades de Google Maps en sus aplicaciones web o móviles. Estas APIs ofrecen diversas funcionalidades relacionadas con los mapas, geolocalización, y servicios de dirección. Algunas de las principales APIs de Google Maps incluyen:

1. Maps JavaScript API**: Permite insertar mapas interactivos en páginas web, proporcionando funcionalidades como la visualización de mapas, geocodificación (conversión de direcciones en coordenadas geográficas), marcadores personalizados, y capas de información adicional.
2. Geocoding API: Convierte direcciones en coordenadas de latitud y longitud, y viceversa. Esto es útil para aplicaciones que necesitan traducir una dirección postal en una ubicación geográfica precisa o encontrar la dirección correspondiente a unas coordenadas específicas.
3. Places API: Ofrece información sobre millones de lugares en todo el mundo, como restaurantes, tiendas, parques, y otros puntos de interés. Los desarrolladores pueden buscar lugares, obtener detalles específicos sobre un lugar y realizar consultas autocompletadas.
4. Directions API: Proporciona rutas entre dos o más ubicaciones, incluyendo opciones para distintos modos de transporte como automóvil, bicicleta, transporte público y a pie. También incluye información sobre el tráfico y la duración estimada del viaje.
5. Distance Matrix API: Calcula las distancias y tiempos de viaje entre múltiples puntos, útil para aplicaciones de logística y transporte.
6. Elevation API: Proporciona información sobre la altitud de una ubicación específica, útil para aplicaciones que requieren datos de elevación.
7. Street View API: Permite insertar vistas panorámicas de 360 grados a nivel de calle en aplicaciones, proporcionando una experiencia visual detallada de lugares específicos.

8. Geolocation API: Determina la ubicación del usuario en tiempo real mediante la recopilación de datos de redes inalámbricas y torres de telefonía móvil. Estas APIs son altamente versátiles y se utilizan en una amplia gama de aplicaciones, desde servicios de entrega y navegación hasta redes sociales y aplicaciones de turismo. Los desarrolladores pueden acceder a estas APIs mediante solicitudes HTTP y utilizar las respuestas en formato JSON o XML para integrar funcionalidades de mapeo y geolocalización en sus proyectos.

3.11. Aplicación web

Una aplicación web es una aplicación que se ejecuta en un navegador web y utiliza tecnologías web como HTML, CSS y JavaScript. Proporciona una interfaz de usuario interactiva y permite a los usuarios interactuar con servicios y datos a través de Internet. Las aplicaciones web se pueden acceder desde cualquier dispositivo con un navegador y una conexión a Internet, lo que las hace altamente accesibles. Estas aplicaciones se ejecutan en un servidor y utilizan el protocolo HTTP para comunicarse con los clientes. En el contexto de este proyecto, se desarrollará una aplicación web que consume los APIs de OpenWeathermap y Twitter para mostrar información climatológica y publicar tweets en una cuenta específica [8].

3.12. XAMPP:

XAMPP es un paquete de software gratuito y de código abierto que proporciona un entorno de desarrollo web local. El acrónimo XAMPP se refiere a los componentes principales que incluye: Apache, MySQL, PHP y Perl. Estos componentes trabajan en conjunto para permitir el desarrollo y la ejecución de aplicaciones web en un entorno de servidor local. XAMPP es compatible con varios sistemas operativos y proporciona una solución completa para la configuración de un servidor web local, lo que facilita el desarrollo y la prueba de aplicaciones antes de ser implementadas en un entorno de producción. XAMPP también incluye herramientas adicionales como phpMyAdmin para administrar bases de datos MySQL y FileZilla FTP Server para facilitar la transferencia de archivos.

3.13 Tecnologías .NET

.NET es un marco de desarrollo de software creado por Microsoft que proporciona un entorno de programación unificado para construir una amplia variedad de aplicaciones, desde aplicaciones web y de escritorio hasta servicios en la nube y aplicaciones móviles. Desde su lanzamiento inicial en 2002, .NET ha evolucionado significativamente, culminando en su versión más reciente, .NET 6, que unifica las diferentes plataformas .NET bajo un solo marco.

3.14. Componentes de .NET

.NET Core: Una versión multiplataforma, de código abierto y modular de .NET que permite desarrollar aplicaciones que se ejecutan en Windows, macOS y Linux. Es especialmente adecuado para aplicaciones en la nube y microservicios. ASP.NET Core: Un marco para construir aplicaciones web y servicios API, conocido por su alto rendimiento y capacidad para manejar aplicaciones de gran escala. Entity Framework Core: Un ORM (Object-Relational Mapper) que simplifica el acceso a bases de datos mediante la programación orientada a objetos, permitiendo a los desarrolladores trabajar con datos utilizando objetos .NET. Xamarin: Una plataforma que permite desarrollar aplicaciones móviles para iOS y Android utilizando .NET y C#.

3.15. Ventajas de .NET Multiplataforma:

Permite desarrollar aplicaciones que se pueden ejecutar en diferentes sistemas operativos. Rendimiento: Ofrece un alto rendimiento y una capacidad de respuesta rápida para aplicaciones web y servicios. Seguridad: Incorpora características avanzadas de seguridad para proteger las aplicaciones contra amenazas y vulnerabilidades. Productividad: Herramientas integradas como Visual Studio y una amplia gama de bibliotecas y frameworks facilitan el desarrollo rápido y eficiente.

3.16 Servicios REST

REST (Representational State Transfer) es un estilo arquitectónico para sistemas distribuidos, introducido por Roy Fielding en su tesis doctoral en el año 2000. REST se basa en un conjunto de principios y restricciones que utilizan el protocolo HTTP para facilitar la comunicación entre clientes y servidores. 3.5. Principios de REST Stateless: Cada solicitud del cliente al servidor debe contener toda la información necesaria para entender y procesar la solicitud. El servidor no almacena el estado del cliente entre solicitudes. Client-Server: La arquitectura REST separa las preocupaciones del cliente y el servidor, permitiendo que cada uno evolucione de manera independiente. Cacheable: Las respuestas de las solicitudes deben ser explícitamente marcadas como cacheables o no cacheables, para mejorar la eficiencia y escalabilidad. Layered System: La arquitectura puede tener capas jerárquicas que limitan el comportamiento de los componentes a ciertos niveles. Uniform Interface: REST impone una interfaz uniforme que permite que las interacciones entre diferentes sistemas sean predecibles y consistentes.

3.17. Métodos HTTP en REST

GET: Recupera información del servidor.

POST: Envía datos al servidor para crear un nuevo recurso.

PUT: Actualiza un recurso existente en el servidor.

DELETE: Elimina un recurso del servidor.

PATCH: Aplica modificaciones parciales a un recurso.

3.18 ASP.NET Core para Servicios RESTful

ASP.NET Core es el marco principal dentro del ecosistema .NET para desarrollar servicios RESTful. Proporciona una infraestructura robusta para crear APIs que cumplen con los principios REST. Configuración Inicial: Configurar un proyecto ASP.NET Core para desarrollar una API RESTful incluye definir rutas, controladores y acciones. Controladores y Acciones: Los controladores en ASP.NET Core gestionan las solicitudes HTTP y retornan las respuestas adecuadas. Cada acción en el controlador está vinculada a un método HTTP (GET, POST, PUT, DELETE). Model Binding y Validación: ASP.NET Core facilita el enlace de datos de las solicitudes a objetos de modelo, así como la validación de los mismos antes de procesar las solicitudes.

3.19. Entity Framework Core en Servicios REST

Entity Framework Core (EF Core) es una herramienta clave para gestionar datos en servicios RESTful construidos con .NET. Mapeo de Entidades: EF Core permite mapear clases de .NET a tablas en una base de datos, facilitando las operaciones CRUD. Consultas LINQ: Utilizando LINQ (Language Integrated Query), los desarrolladores pueden escribir consultas de manera intuitiva y eficiente sobre las entidades mapeadas. Migraciones: EF Core soporta migraciones que permiten actualizar el esquema de la base de datos de forma incremental y controlada.

3.20. Seguridad y Autenticación

La seguridad es un aspecto crítico en el desarrollo de servicios RESTful. ASP.NET Core ofrece varios mecanismos para proteger APIs, como autenticación basada en tokens JWT (JSON Web Tokens) y autorización mediante políticas y roles. Autenticación: Implementación de OAuth 2.0 y OpenID Connect para autenticar usuarios y aplicaciones. Autorización: Definición de políticas de acceso y roles para controlar quién puede acceder a qué recursos y operaciones.

3.21. Documentación y Versionado

La documentación y el versionado son esenciales para la mantenibilidad y la evolución de las APIs RESTful. Swagger/OpenAPI: ASP.NET Core integra herramientas como Swashbuckle para generar

documentación interactiva de las APIs mediante la especificación OpenAPI. Versionado: Estrategias para versionar las APIs, asegurando compatibilidad y facilitando la introducción de nuevas funcionalidades sin interrumpir los clientes existentes.

3.22.Spring Tool Suite 4 (STS4)

Spring Tool Suite 4 (STS4) es un entorno de desarrollo integrado (IDE) basado en Eclipse, optimizado para el desarrollo de aplicaciones basadas en el marco de trabajo Spring. Ofrece herramientas y funcionalidades específicas que facilitan el desarrollo, prueba y despliegue de aplicaciones Spring.

Componentes principales:

Spring Boot: Soporte integrado para proyectos Spring Boot, que facilita la creación de aplicaciones basadas en Spring con mínima configuración. Spring Initializr: Herramienta para inicializar nuevos proyectos Spring de manera rápida y sencilla. Spring Boot Dashboard: Permite gestionar y monitorear las aplicaciones Spring Boot directamente desde el IDE. Editor de Propiedades: Un editor especializado para archivos de propiedades y configuración de Spring Boot. Spring Cloud: Soporte para microservicios y aplicaciones distribuidas utilizando Spring Cloud. Instalación y configuración: STS4 se puede instalar como una distribución independiente basada en Eclipse o como un plugin en una instalación existente de Eclipse. Uso y Aplicaciones: Desarrollo de Aplicaciones Spring: Proporciona herramientas específicas para trabajar con el ecosistema Spring. Integración Continua: Se integra con herramientas de construcción como Maven y Gradle, así como con sistemas de control de versiones como Git. Depuración y Pruebas: Facilita la depuración y pruebas de aplicaciones Spring con características avanzadas de monitoreo y logging. Despliegue: Permite el despliegue directo de aplicaciones en servidores locales o en la nube.

Ventajas:

• Optimizado para el desarrollo de aplicaciones Spring.

• Herramientas integradas que mejoran la productividad.

• Soporte para proyectos Spring Boot desde el inicio.

Desventajas:

• Basado en Eclipse, lo que puede resultar en una curva de aprendizaje si no se está familiarizado con este IDE.

• Requiere una buena cantidad de recursos del sistema para un rendimiento óptimo.

3.23. LARAVEL

Laravel es un framework PHP gratis y de código abierto que brinda un conjunto de herramientas y recursos para crear aplicaciones modernas. Posee un ecosistema integral que combina funciones integradas y una variedad de paquetes y extensiones compatibles.

Este framework de PHP creció en popularidad rápidamente en los últimos años, y muchos desarrolladores lo adoptaron como su framework de trabajo favorito para lograr un proceso de desarrollo optimizado.

3.24. Composer

Composer es un sistema de gestión de paquetes para programar en PHP el cual provee los formatos estándar necesarios para manejar dependencias y librerías de PHP. Fue desarrollado por Nils Adermann y Jordi Boggiano quienes continúan dirigiendo el proyecto. Ambos comenzaron el desarrollo en abril de 2011 y en marzo de 2012 presentaron la primera versión.1Composer está inspirado en Node.js, npm y en Bundler Ruby. Composer trabaja e instala dependencias o librerías desde la línea de comandos. También permite al usuario instalar las aplicaciones PHP que estén disponibles en el "Packagist", el repositorio principal que contiene todos los paquetes disponibles. También dispone de capacidad de auto-descarga para las librerías necesarias que se especifiquen en la información de arranque para así facilitar el uso del código de terceros.de comunicación abiertos y protocolos interoperables, lo que permite la integración de sistemas heterogéneos y la creación de sistemas distribuidos escalables y flexibles.

4. METODOLOGÍA DE DESARROLLO.

Para el desarrollo de la práctica acerca de la instalación paso a paso del framework, se sigue una metodología de desarrollo secuencial como se describe a continuación:

- **Análisis:** Con antecedente en la información recabada en la cátedra de la asignatura de Interoperabilidad de Plataformas, se hizo una revisión sobre la información relevante y las características de los sistemas basados en servicios web.
- **Diseño:** Con una comprensión del fundamento teórico se diseñó la aplicación, se dividió la creación en tres partes principales: descarga del framework, instalación y los comandos para la implementación.
- **Implementación:** Las fases de creación de la instalación se desarrollaron previamente a las horas

correspondientes a la cátedra, para con el apoyo del docente y en función a las guías proporcionadas solventar cualquier inquietud.

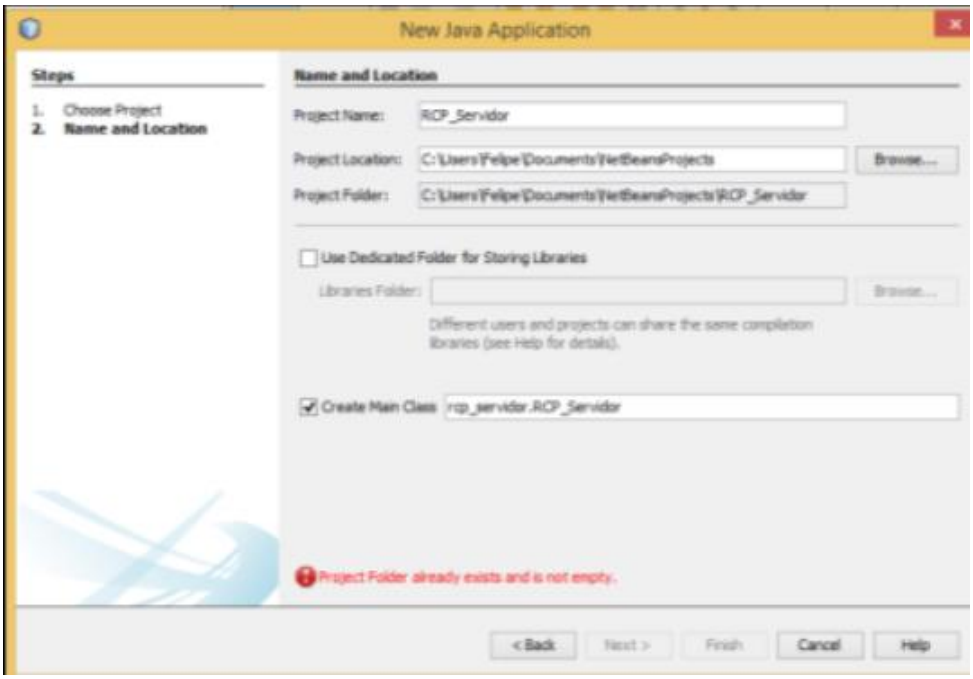
• **Evaluación:** Cada una de las fases fue evaluada por el docente en las horas correspondientes de la asignatura para verificar comprensión del tema, aclarar dudas y demostrar los conocimientos.

• **Documentación:** Finalmente, se elaboró un informe que recabe la información relativa a la instalación de Laravel, en donde se detalla la información de todo el proceso de desarrollo.

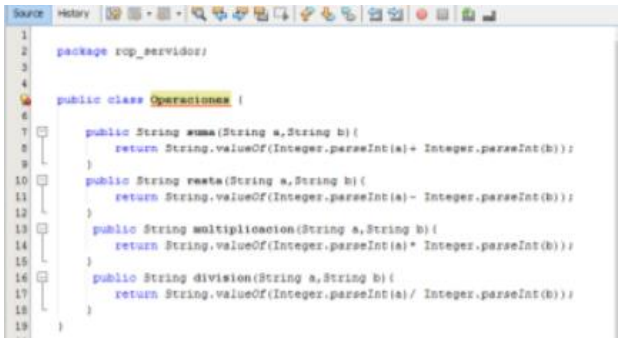
5. DESARROLLO DE LAS PRACTICAS.

5.1. PRACTICA SISTEMAS DISTRIBUIDOS

1. Como primer paso, añadir un proyecto para nuestro servidor, en este caso lo llamamos "RCP_Servidor"

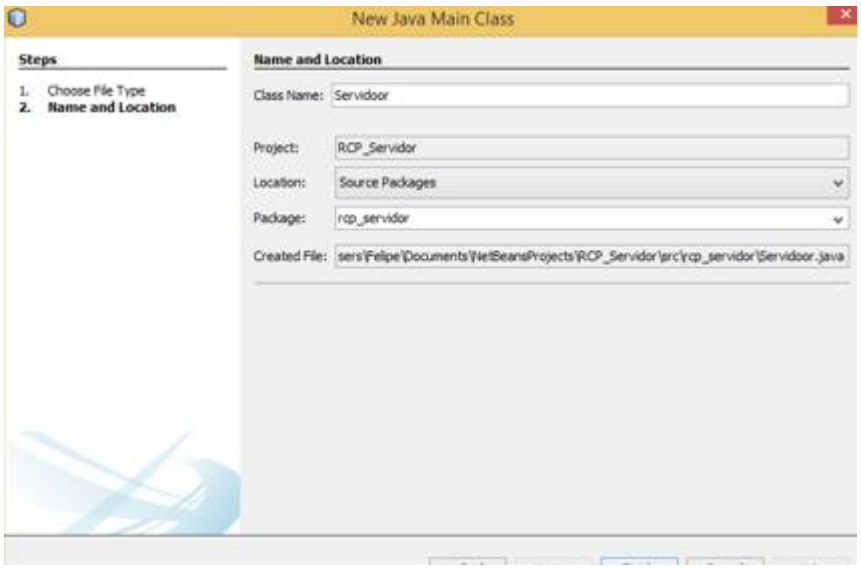


2. Crear una clase para la funcionalidad de nuestro sistema en este caso, "Operaciones" aquí estarán los métodos de suma, resta, multiplicar y división.



```
package rcp_servidor;

public class Operaciones {

    public String suma(String a,String b){
        return String.valueOf(Integer.parseInt(a)+ Integer.parseInt(b));
    }
    public String resta(String a,String b){
        return String.valueOf(Integer.parseInt(a)- Integer.parseInt(b));
    }
     public String multiplicacion(String a,String b){
        return String.valueOf(Integer.parseInt(a)* Integer.parseInt(b));
    }
     public String division(String a,String b){
        return String.valueOf(Integer.parseInt(a)/ Integer.parseInt(b));
    }
}
```

3. Añadir una Main Class llamada “Servidor” para llamar a los métodos y manejar las funciones de nuestra clase Operaciones.



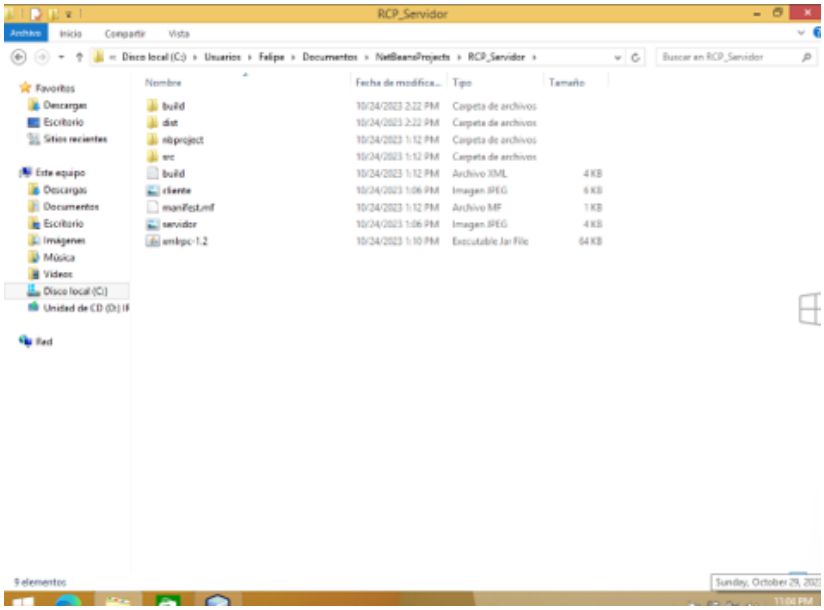
4. Primero se debe declarar las librerías necesarias y también el código necesario para la iniciar el servidor y llamar a los métodos de nuestra clase operaciones.

```java
package rcp_servidor;

import javax.swing.JOptionPane;
import org.apache.xmlrpc.WebServer;

public class Servidor {


    public static void main(String[] args) {

        try {
            JOptionPane.showMessageDialog(null,"Iniciando el servidor");
            WebServer server = new WebServer(8080);
            Suma suma = new Suma();
            Suma resta = new Suma();
            Suma multiplicacion = new Suma();
            Suma division = new Suma();
            server.addHandler("miServidorRpc",suma);
            server.addHandler("miServidorRpc",resta);
            server.addHandler("miServidorRpc",multiplicacion);
            server.addHandler("miServidorRpc",division);
            server.start();
            JOptionPane.showMessageDialog(null,"Servidor en linea");
        } catch (Exception e){
            JOptionPane.showMessageDialog(null,"Error"+e.getMessage());
        }
    }
}
```

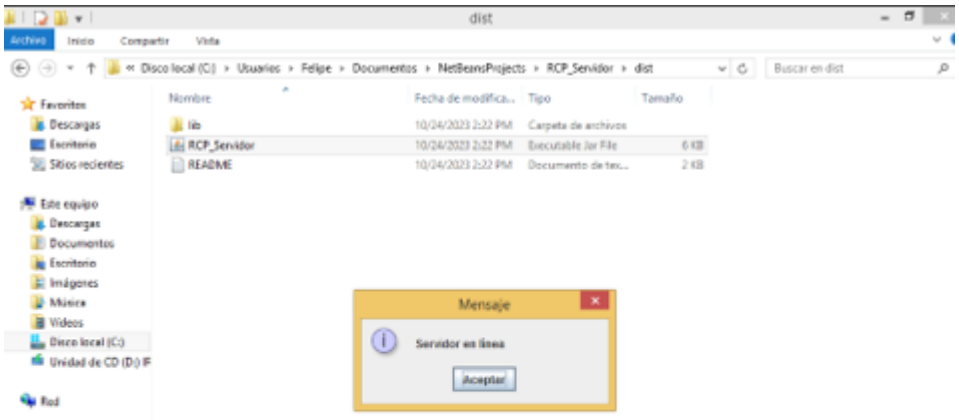
5. Añadir un nuevo proyecto para nuestro servidor, en este caso llamado “RCP_Cliente” donde crearemos nuestro formulario y conexión entre el cliente y servidor.

```java
RpcClient client = new XmlRpcClient("http://10.0.2.15:8080");
tor<String> parametros = new Vector<String>();
tionPane.showMessageDialog(null,"El cliente se ha conectado");
le(true){
 String menu = JOptionPane.showInputDialog(null,"Operaciones con 2 numeros\n"
                + "1. Suma \n"
                + "2. Resta \n"
                + "3. Multiplicacion \n"
                + "4. Division \n"
                + "5. Salir \n","Cliente Servidor RPC", JOptionPane.DEFAULT_OPTION);
    switch(menu){
        case "1":
            x=JOptionPane.showInputDialog(null,"Primer numero","Suma",JOptionPane.QUESTION
            y=JOptionPane.showInputDialog(null,"Segundo numero","Suma",JOptionPane.QUESTIO
            parametros.addElement(x);
            parametros.addElement(y);
            resultado = client.execute("miServidorRpc.suma",parametros);
            JOptionPane.showMessageDialog(null,"La suma es "+resultado);
            parametros.clear();
            break;
        case "2":
            x=JOptionPane.showInputDialog(null,"Primer numero","Resta",JOptionPane.QUESTIO
            y=JOptionPane.showInputDialog(null,"Segundo numero","Resta",JOptionPane.QUESTI
            parametros.addElement(x);
            parametros.addElement(y);
            resultado = client.execute("miServidorRpc.resta",parametros);
            JOptionPane.showMessageDialog(null,"La resta es "+resultado);
            parametros.clear();
            break;
```

6. Editar las propiedades de estos, señalar todas las opciones de las categorías de Compiling, Packging, Run y Applicacion, donde se debe asociar una imagen a través de splash screen y al final generar nuestros proyectos.



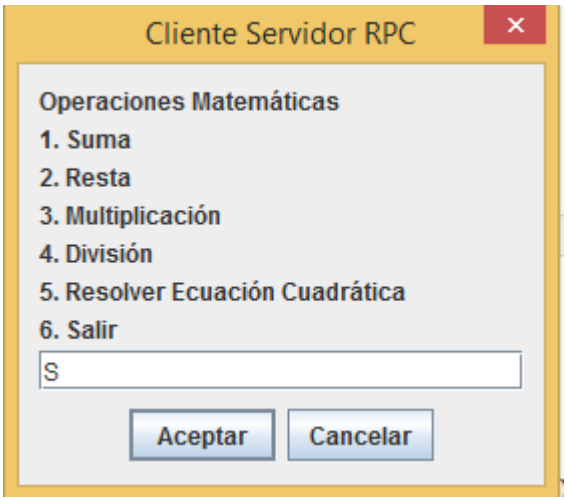
7. A los archivos .jar de ambos proyectos se inician para usar nuestra aplicación de sistemas distribuidos RPC



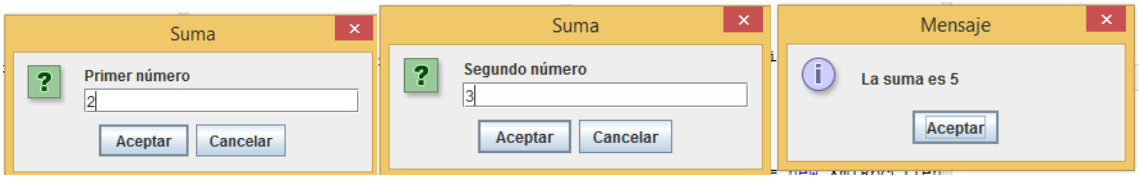
RESULTADOS.

PRUEBAS DE LOS MÉTODOS DEL SERVICIO WEB ASP .NET

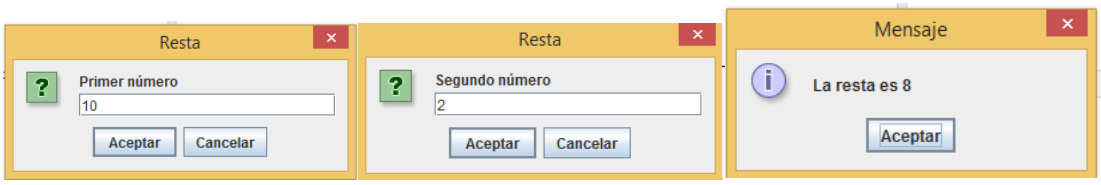
A. MENU PRINCIPAL



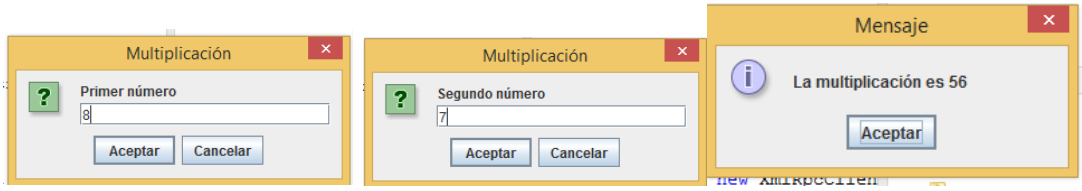
B. SUMAR



C. RESTAR

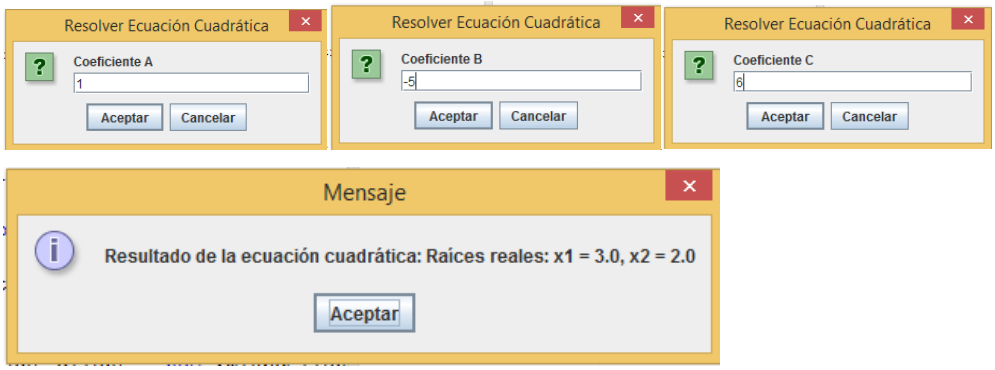


D. MULTIPLICAR



E. DIVIDIR

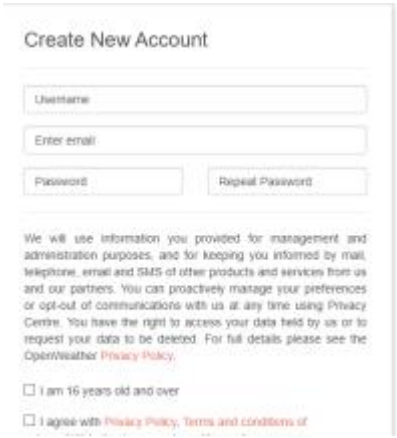
F. ECUACIÓN CUADRÁTICA



5.2. Informe API

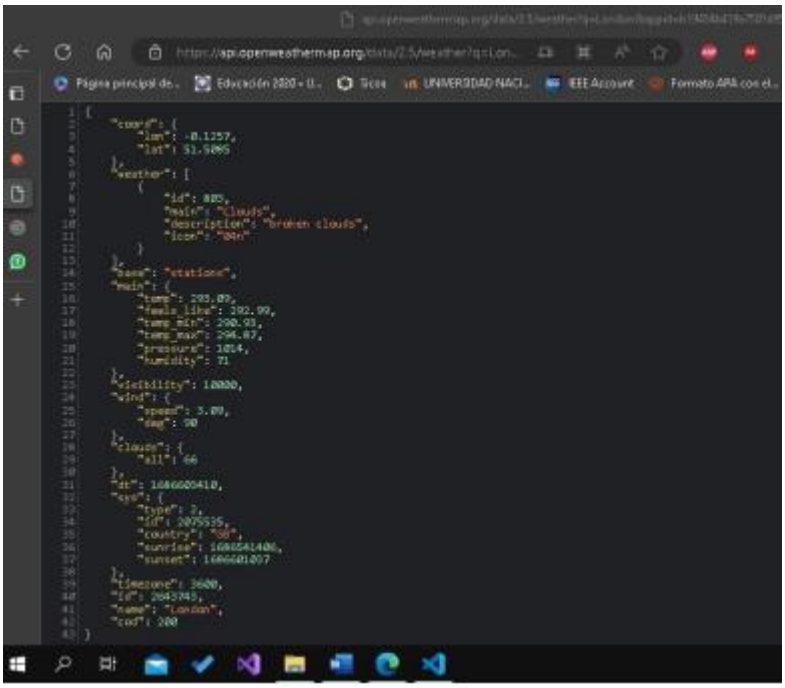
En primer instancias se va a utilizar la API de Open Weather Map como caso de estudio del consumo de API comerciales o públicas. Para ello se siguió las siguientes etapas:

a) Se creó una cuenta en de Open Weather Map en https://openweathermap.org/api conforme a los requerimientos de la página web para su uso.



b) Tras crear la cuenta es posible solicitar una llave o API KEY que se usará para la autenticación y acceso al servicio. Esta clave se habilita en un periodo de dos horas tras la solicitud.

c) Para comprobar el funcionamiento de la llave se probó la misma al acceder al enlace: api.openweathermap.org/data/2.5/weather?q={city name}&appid={API key}

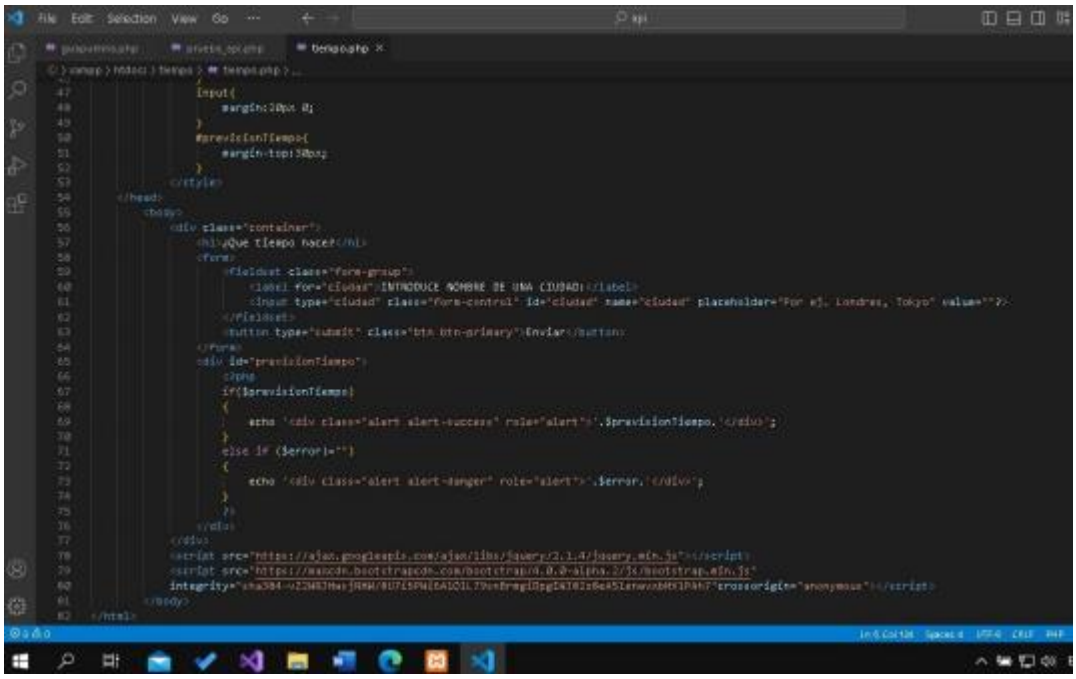


Nótese que los parámetros API KEY y City Name deben ser ingresados por el usuario. En el caso particular de esta prueba se utilizó la API KEY generada y la ciudad de Londres.

d) Una vez ya se comprobó el correcto funcionamiento se procedió a crear una una aplicación web en php que consuma el servicio. Para ello se utilizó el servidor web Apache integrado en XAMPP y por ende la aplicación se desplegó en la carpeta htdocs como servidor virtual de esta herramienta. Así pues, se creó un archivo tiempo.php en donde se definió la lógica de negocio para consumir el servicio mediante la API.

```
C: > xampp > htdocs > tiempo > tiempo.php > ...
<?php
    header("Content-type:text/html;charset=\"utf-8\"");
    $previsionTiempo="";$error="";
    if (array_key_exists('ciudad',$_GET))
    {
        $urlContents = file_get_contents("https://api.openweathermap.org/data/2.5/weather?q=".urlencode($_GET['ciudad'])."&appid=b19434b419a7501d95a40f8a20ffca22");
        $array=json_decode($urlContents,true);
        $previsionTiempo = "El tiempo en ".$_GET['ciudad']." es actualmente ".$array['weather'][0]['description']."";
        $temperaturaEnCelsius=$array['main']['temp']-273;
        $previsionTiempo .= ".La temperatura es ".intval($temperaturaEnCelsius)."&deg;C";
        $tempMin=$array['main']['temp_min']-273;
        $tempMax=$array['main']['temp_max']-273;
        $previsionTiempo .=" oscilando entre " .intval($tempMin). "&deg;C de minima y " .intval($tempMax). "&deg;C de máxima.";
    }
```

En primera instancia, en el apartado php se especificó la lógica de negocios para el consumo de la API especificando la url del servicio y la clave de acceso para este cometido. Asimismo, se implementó el procesamiento de la petición y respuesta en formato json de modo que se obtenga los campos necesarios para esta práctica (temperatura actual, máxima, mínima) Tras ello se implementó la estructura HTML y los estilos con CSS y Bootstrap para la presentación de la aplicación web de la siguiente manera:

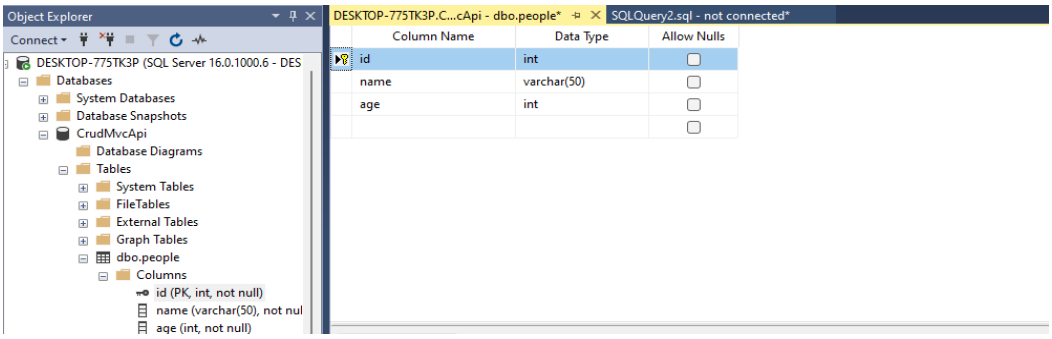


e) Finalmente se probó la aplicación en el servidor local con la ruta del recurso, en este caso localhost/tiempo.php

5.3. Tecnologías .net servicios REST

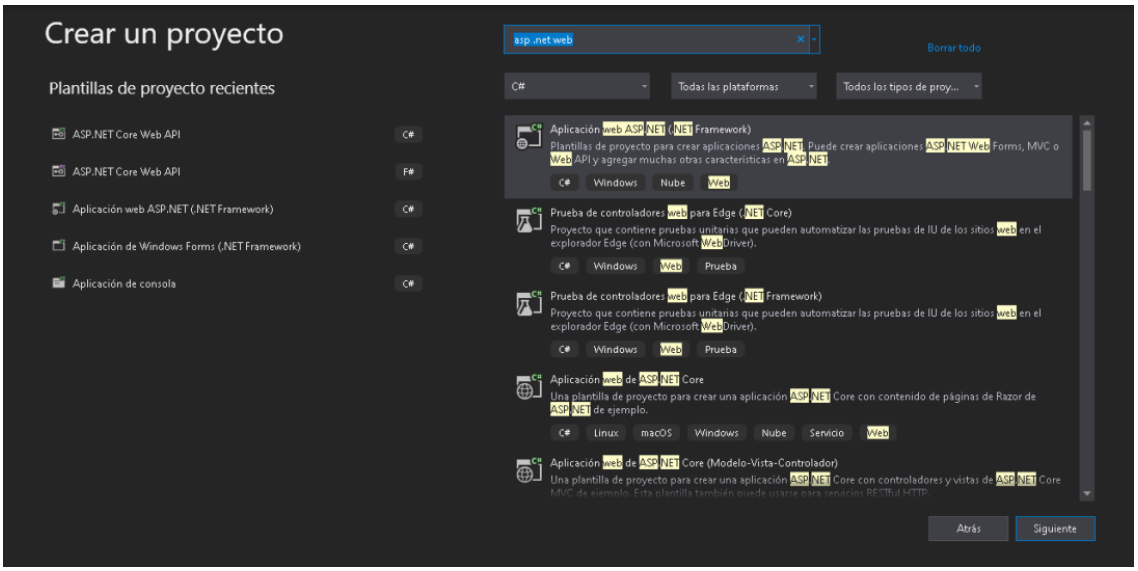
5.3.1 SQL server

En primer instancias se va a utilizar el servidor de base de datos de SQL server en el cual crearemos una base de datos llamada CrudMvcApi en el cual tenemos los atributos de id, nombre y edad.

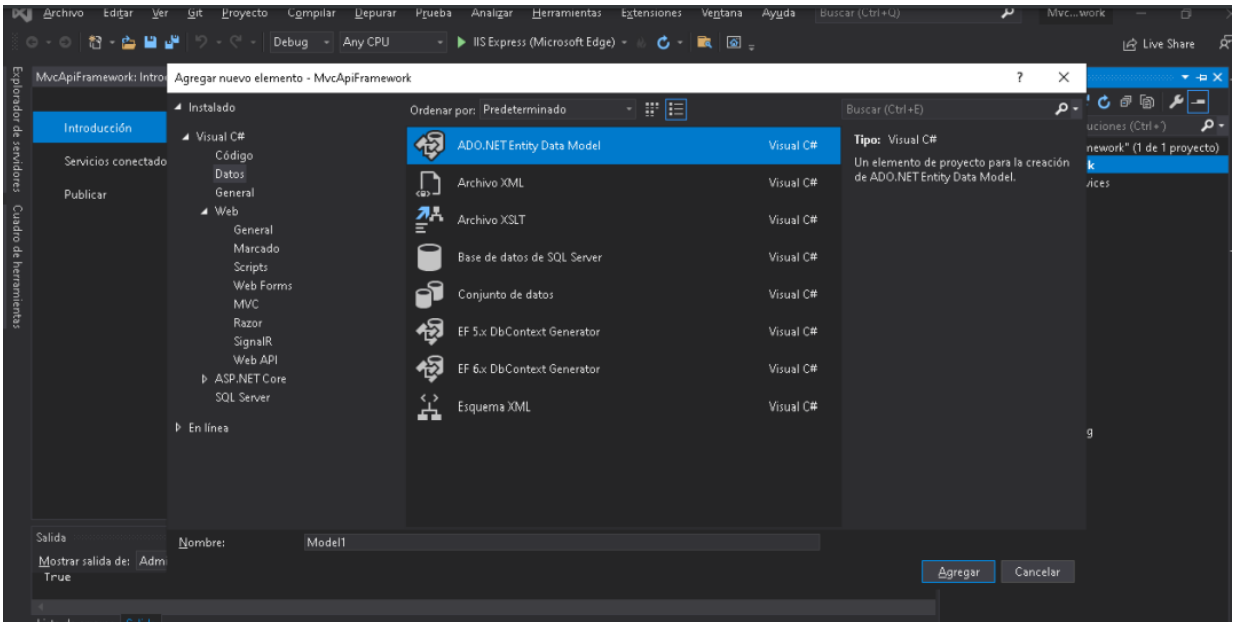


5.3.2 Visual Studio

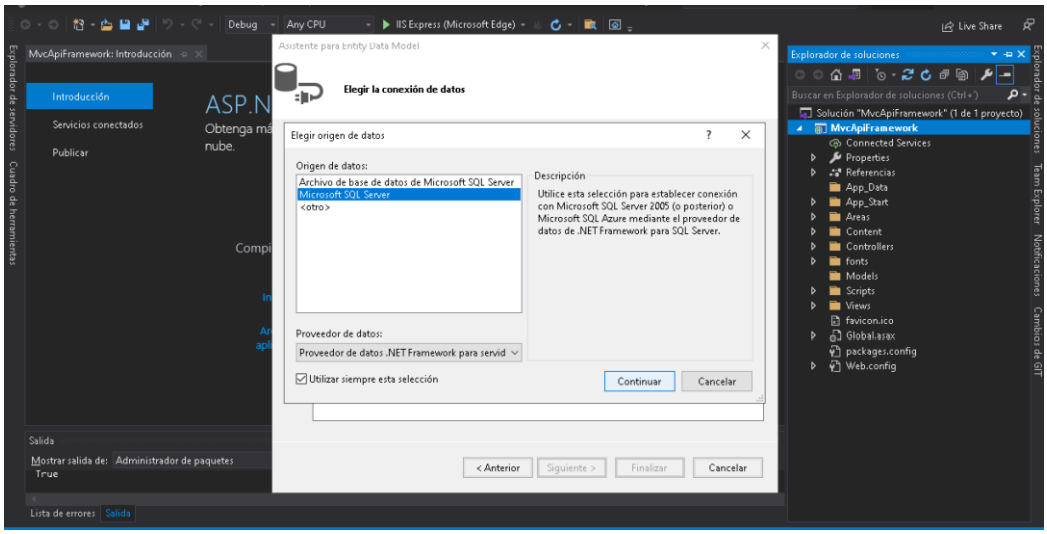
Ahora en visual studio creamos un nuevo proyecto en el cual crearemos nuestra aplicación web en cual se consumira nuestro .NET



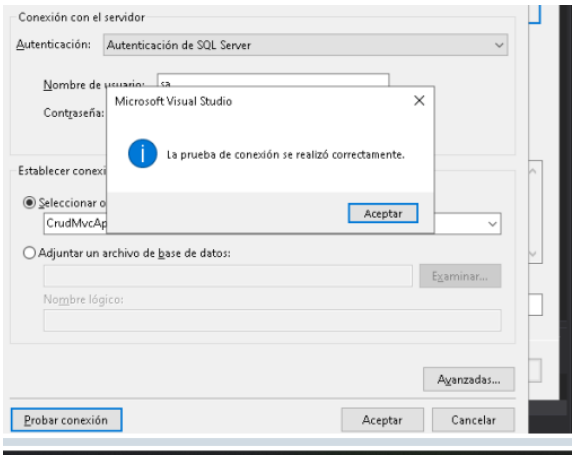
Una vez creado el proyecto hacemos la conexión de la base de datos creada anteriormente en SQL server



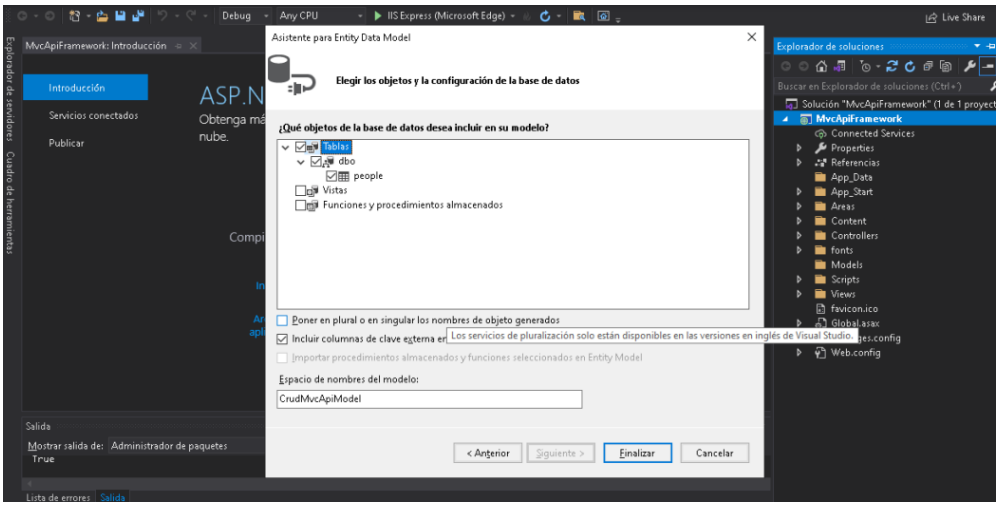
Aquí escogemos el servidor de SQL server



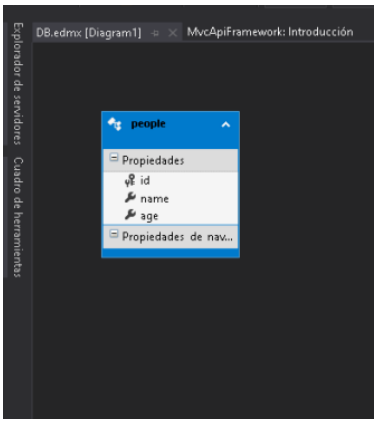
En este paso revisamos la conexión a la base de datos se realizo correctamente



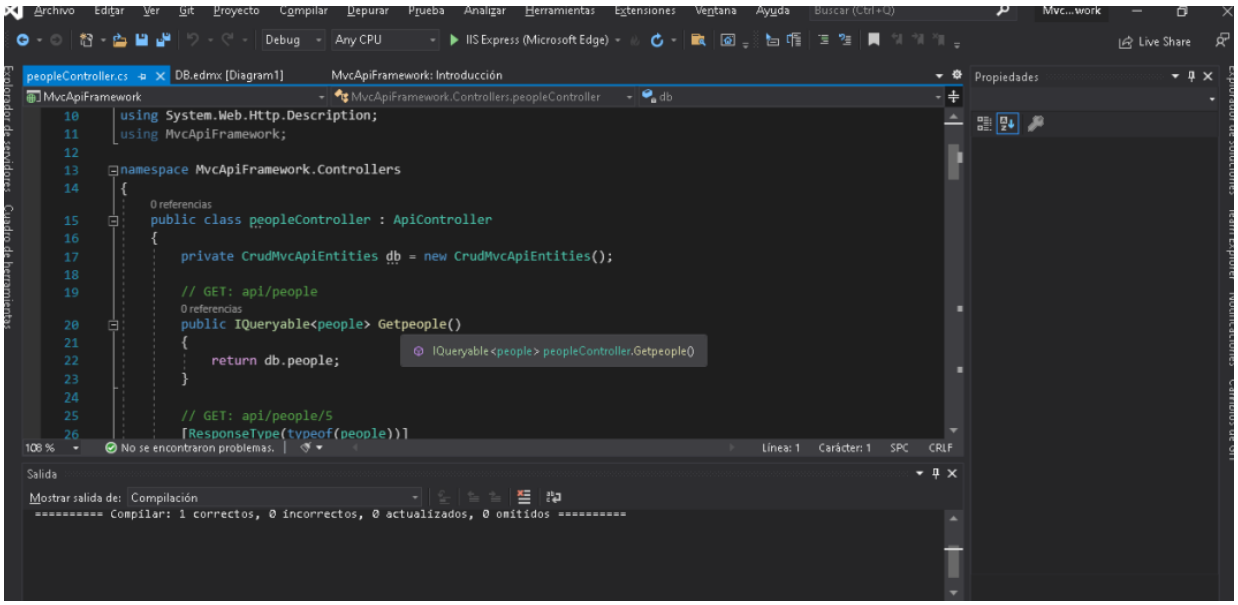
Una vez hecha la conexión importamos la tabla de datos people para que se nos guarde los datos en esa base de datos.



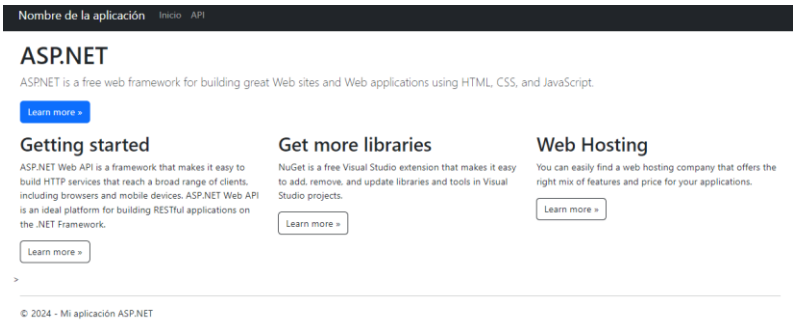
Una vez hecho todo esto y damos click en finalizar, simultáneamente observamos que se importó correctamente la tabla people.



Una vez importada correctamente la base de datos creamos un archivo scaffolding en la parte de Controller llamado peopleController en el cual ya nos da el código predeterminado

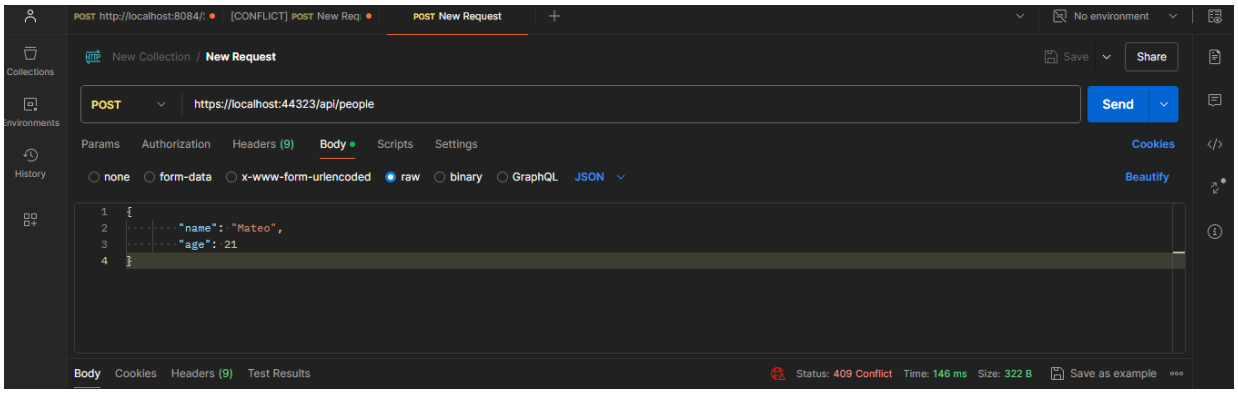


Posteriormente compilamos y observamos que se compile correctamente en nuestro navegador predeterminado.

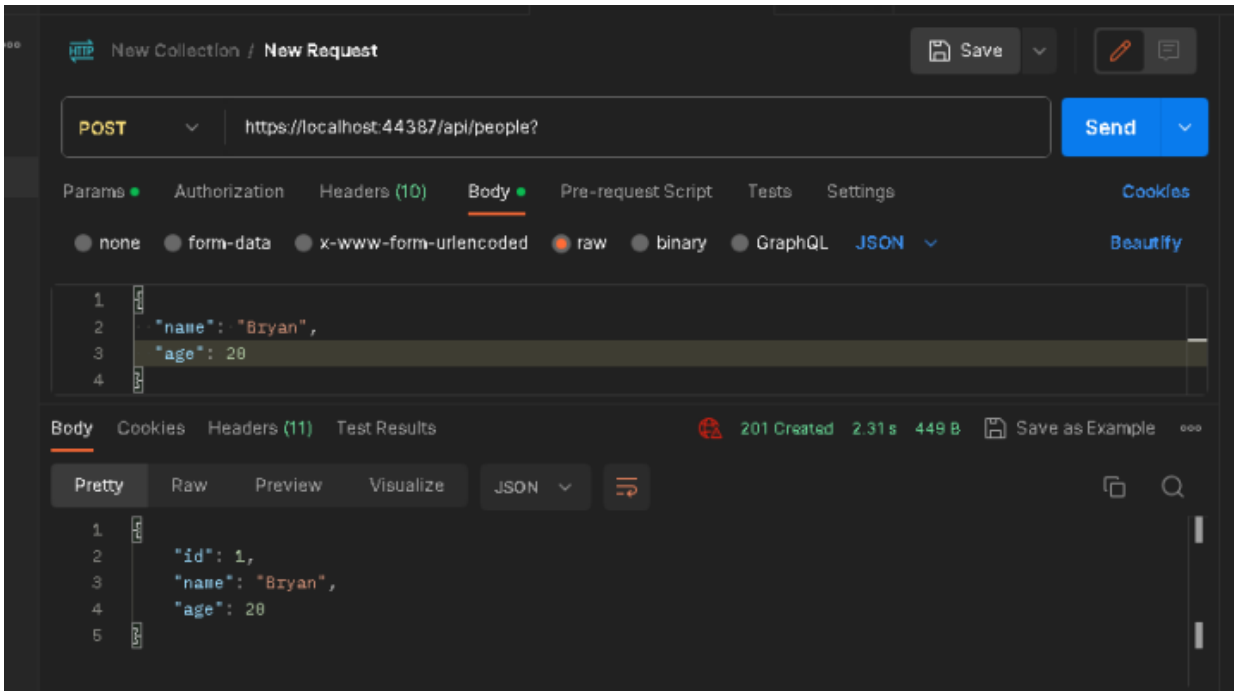


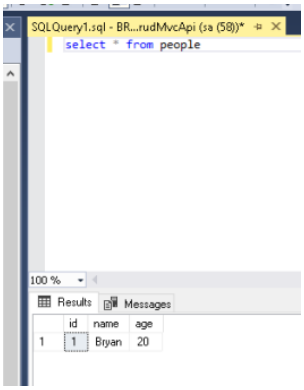
5.3.3 Postman

Una vez configurado todo en visual studio nos dirigimos a Postman a consumir el servicio y además comprobar si nuestra aplicación está funcionando correctamente.



Agregamos un registro y observamos que si se guardo correctamente en la base de datos de SQL server.

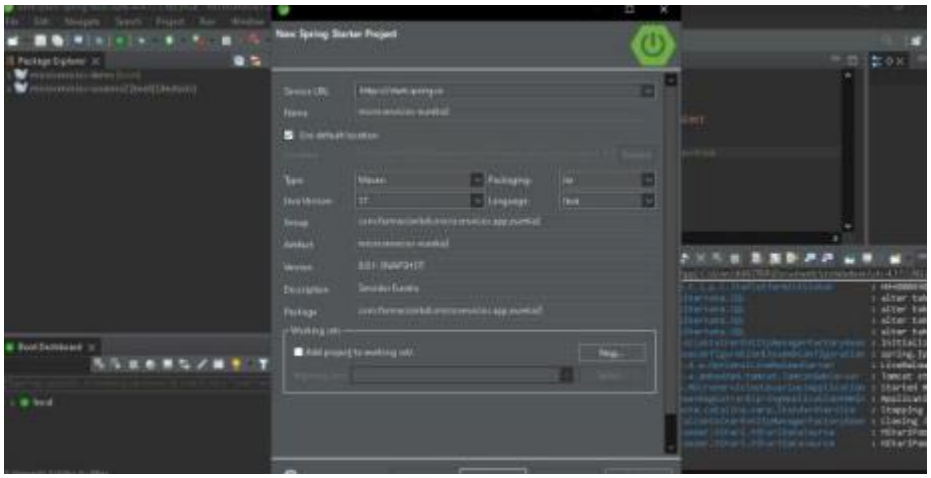




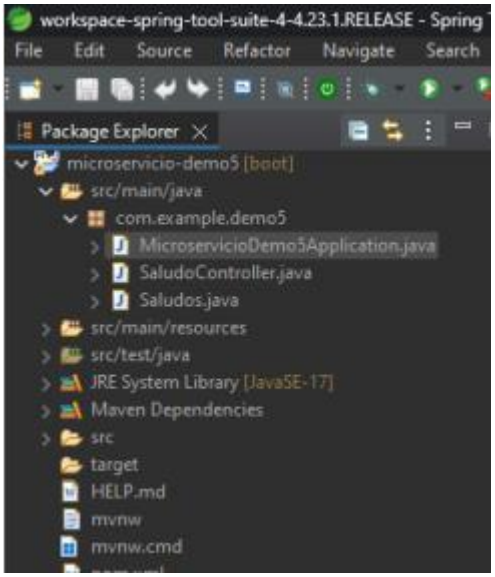
5.4. SPRINGBOOT

Para el desarrollo de la primera fase de la práctica se descargó todos los recursos para la correcta instalación de Laravel, siguiendo el proceso detallado:

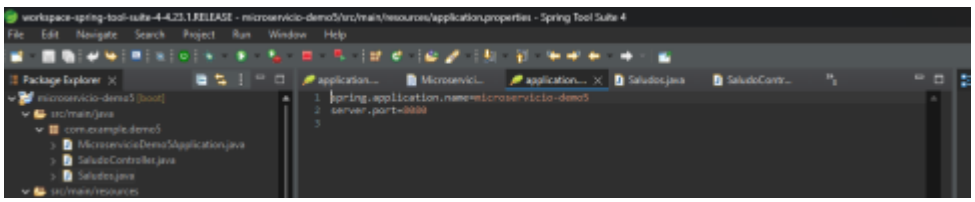
1. Creamos un nuevo proyecto en nuestra aplicación Springboot4



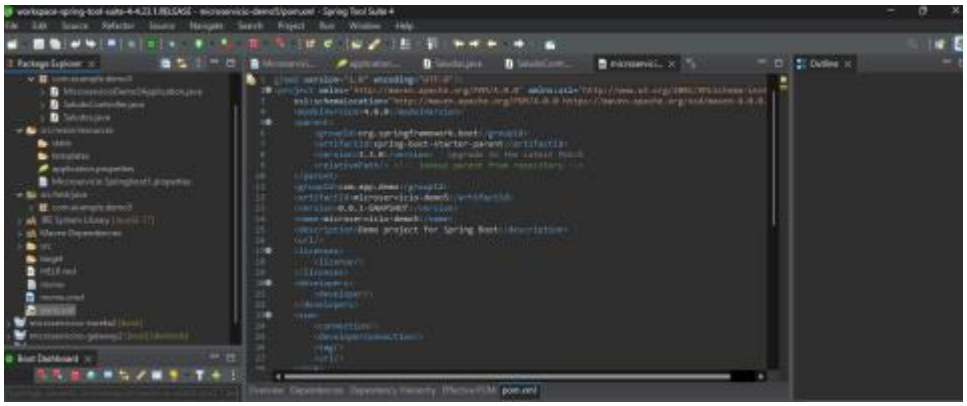
2. Al tener creado el proyecto creamos los siguientes archivos en nuestro src/main/java. Incluimos el código proporcionado por el docente.



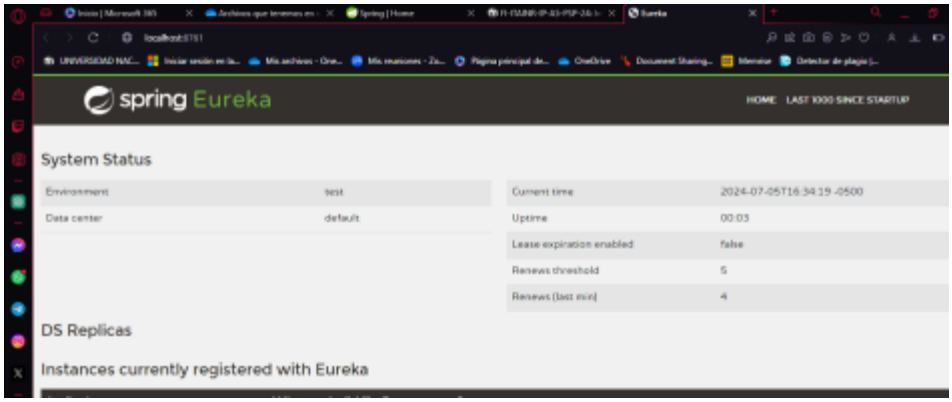
3. Como siguiente paso nos dirigimos a aplications propietaries que se encuentra en src/main/resources



4. Después nos dirigimos a nuestro pom que se encuentra en src y comprobamos que se encuentre la versión que hemos instalado antes de crear nuestros archivos.



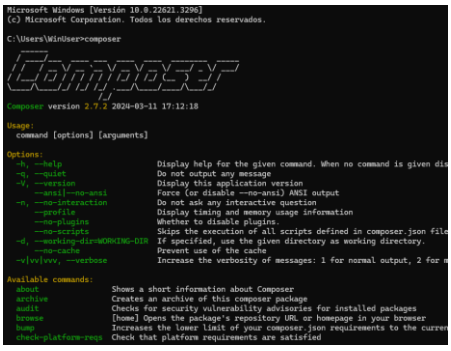
Levantamos el servicio y vemos que este corriendo perfectamente.



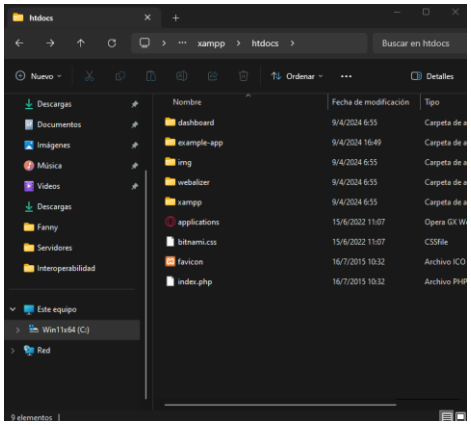
5.5. INSTALACION CORRECTA DE LARAVEL

Para el desarrollo de la primera fase de la práctica se descargó todos los recursos para la correcta instalación de Laravel, siguiendo el proceso detallado:

8. Para descargar Laravel primero debemos descargar composer, para ello nos dirigimos al navegador y escribimos composer y lo instalamos

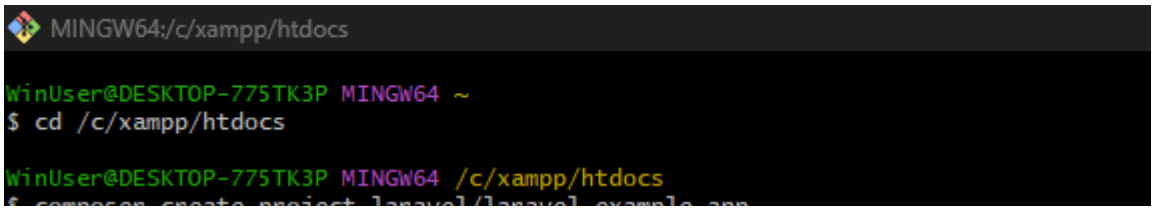


9. Teniendo instalado composer y XAMPP nos dirigimos a nuestra carpeta donde se va a guardar todos nuestros proyectos.

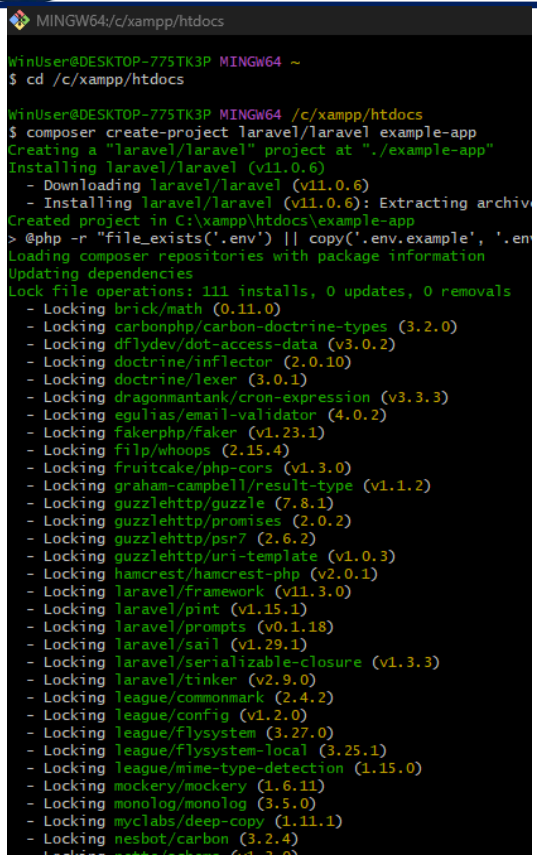


10. Teniendo la carpeta donde se va a guardar nuestros proyectos nos dirigimos a Git Bash sino tenemos descargado nos dirigimos a un navegador y damos en descargar.

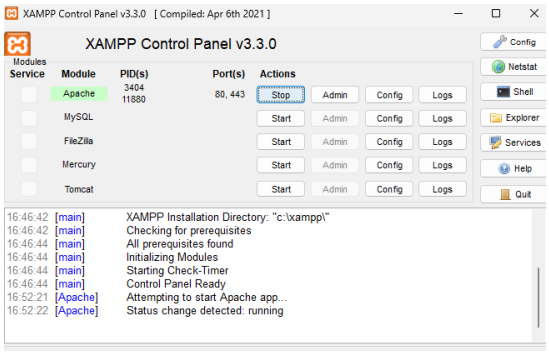
11. En Git Bash nos cambiamos de directorio a la carpeta donde se van a guardar nuestros proyectos.



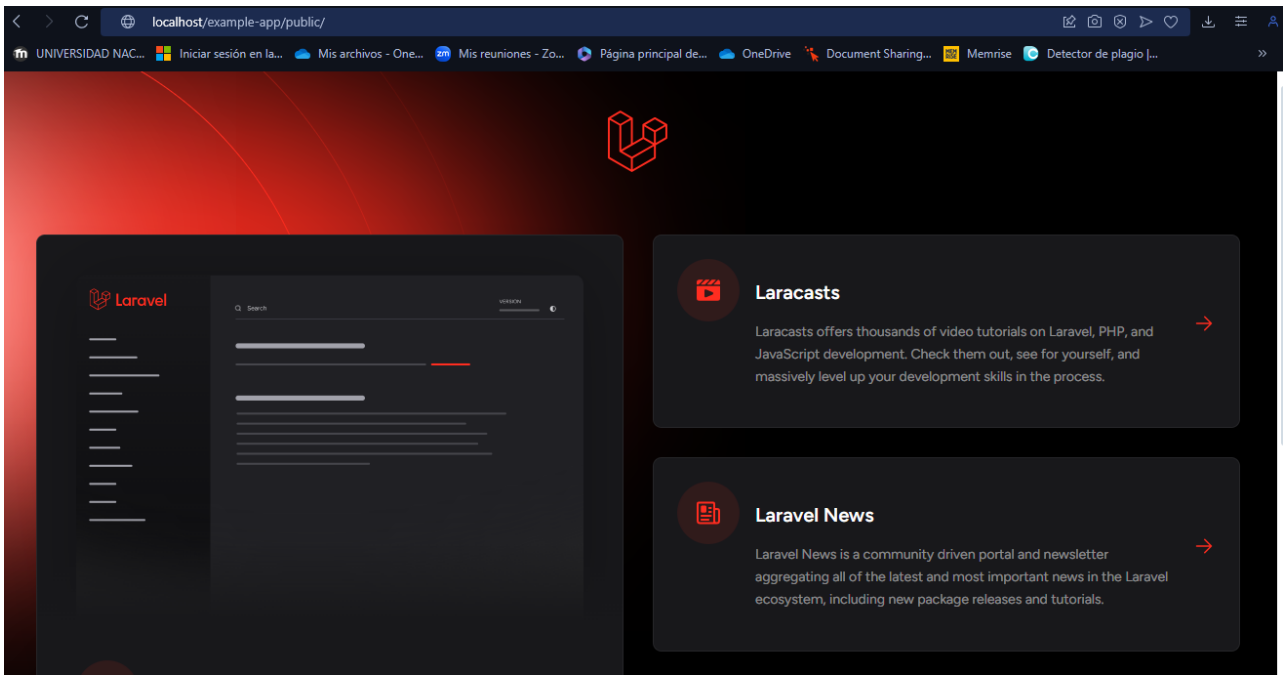
12. Nos dirigimos a la pagina oficial de laravel y nos dirigimos a instalación y copiamos la dirección y pegamos en nuestro Git y esperamos que se descargue laravel.


MINGW64:/c/xampp/htdocs
WinUser@DESKTOP-775TK3P MINGW64 ~
$ cd /c/xampp/htdocs
WinUser@DESKTOP-775TK3P MINGW64 /c/xampp/htdocs
$ composer create-project laravel/laravel example-app
Creating a "laravel/laravel" project at "./example-app"
Installing laravel/laravel (v11.0.6)
- Downloading laravel/laravel (v11.0.6)
- Installing laravel/laravel (v11.0.6): Extracting archiv
Created project in C:\xampp\htdocs\example-app
> @php -r "file_exists('.env') || copy('.env.example', '.en
Loading composer repositories with package information
Updating dependencies
Lock file operations: 111 installs, 0 updates, 0 removals
- Locking brick/math (0.11.0)
- Locking carbonphp/carbon-doctrine-types (3.2.0)
- Locking dflydev/dot-access-data (v3.0.2)
- Locking doctrine/inflector (2.0.10)
- Locking doctrine/lexer (3.0.1)
- Locking dragonmantank/cron-expression (v3.3.3)
- Locking egulias/email-validator (4.0.2)
- Locking fakerphp/faker (v1.23.1)
- Locking filp/whoops (2.15.4)
- Locking fruitcake/php-cors (v1.3.0)
- Locking graham-campbell/result-type (v1.1.2)
- Locking guzzlehttp/guzzle (7.8.1)
- Locking guzzlehttp/promises (2.0.2)
- Locking guzzlehttp/psr7 (2.6.2)
- Locking guzzlehttp/uri-template (v1.0.3)
- Locking hamcrest/hamcrest-php (v2.0.1)
- Locking laravel/framework (v11.3.0)
- Locking laravel/pint (v1.15.1)
- Locking laravel/prompts (v0.1.18)
- Locking laravel/sail (v1.29.1)
- Locking laravel/serializable-closure (v1.3.3)
- Locking laravel/tinker (v2.9.0)
- Locking league/commonmark (2.4.2)
- Locking league/config (v1.2.0)
- Locking league/flysystem (3.27.0)
- Locking league/flysystem-local (3.25.1)
- Locking league/mime-type-detection (1.15.0)
- Locking mockery/mockery (1.6.11)
- Locking monolog/monolog (3.5.0)
- Locking myclabs/deep-copy (1.11.1)
- Locking nesbot/carbon (3.2.4)

13. Terminada la descarga nos dirigimos a XAMPP y activamos Apache



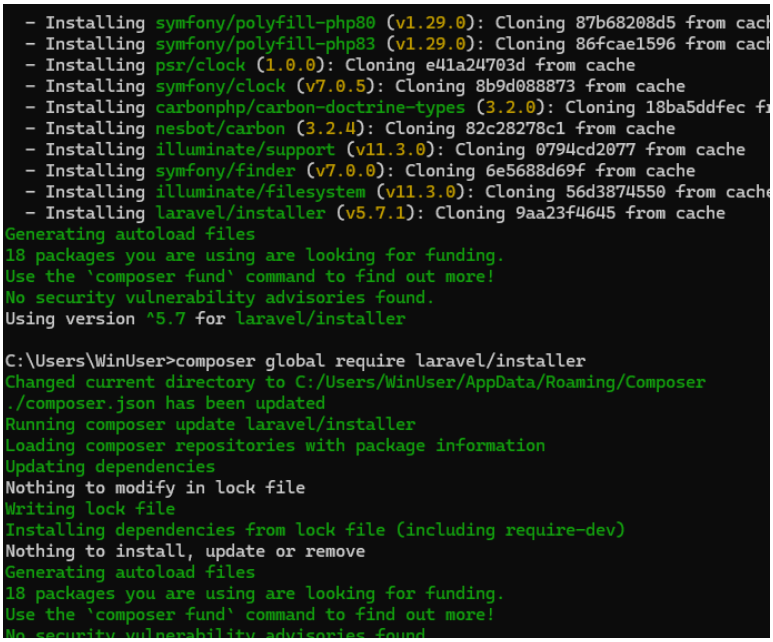
14. Para comprobar si esta funcionando laravel nos dirigimos a un navegador e ingresamos http://localhost/example-app/public/ teniendo en cuenta que creamos un proyecto example-app.



6. RESULTADOS.

6.2. PRUEBAS DE LA INSTALACION

A. Correcta instalacion



7. CONCLUSIONES.

Los sistemas distribuidos desempeñan un papel fundamental en el entorno empresarial actual, ya que permiten una mayor eficiencia en la gestión de la información y los recursos, aumentan la capacidad de procesamiento, reducen los tiempos de respuesta y, sobre todo, poseen escalabilidad y disponibilidad, lo que los hace ideales para la realidad tecnológica presente y futura.

- Los web services son una tecnología potente y escalable que permite el intercambio de datos entre sistemas de forma eficiente.
- La utilización de XML en sistemas distribuidos es de gran importancia, ya que permite la integración entre distintas plataformas y sistemas, facilita la comunicación entre aplicaciones y garantiza su compatibilidad y adaptabilidad a diferentes entornos y necesidades.
- A pesar de los beneficios, la implementación de un servicio web puede ser complicada, requiriendo una planificación cuidadosa y una coordinación adecuada para garantizar que el proyecto sea completado a tiempo y con un alto estándar.

Este proyecto ha mostrado eficazmente cómo implementar y consumir APIs REST públicas, específicamente las de OpenWeatherMap y Google Maps. Se ha conseguido consultar información meteorológica de ciudades en Ecuador utilizando la API de OpenWeatherMap y se ha demostrado la capacidad de obtener ubicaciones basadas en latitud y longitud con la API de Google Maps. Esto resalta la importancia y utilidad de las APIs REST públicas en el desarrollo de aplicaciones y servicios web.

- La integración exitosa de múltiples APIs ha sido un aspecto fundamental de este proyecto. Se ha desarrollado un sistema que combina los datos obtenidos de la API de OpenWeatherMap con los servicios de geolocalización proporcionados por la API de Google Maps, permitiendo la automatización de la visualización de ubicaciones climatológicas. Esta integración demuestra la capacidad de las APIs REST para trabajar en conjunto y la versatilidad de su uso en la creación de soluciones más completas y eficientes.
- El uso del formato JSON para el intercambio de datos ha sido crucial en este proyecto. Tanto la API de OpenWeatherMap como la API de Google Maps devuelven los datos en formato JSON, lo que ha permitido su fácil manipulación y procesamiento en el desarrollo de la aplicación. El uso de JSON como formato de intercambio de datos es ampliamente aceptado y compatible con diferentes lenguajes de programación, lo que facilita la integración de las APIs en aplicaciones web.
- La aplicación web desarrollada en este proyecto ha demostrado cómo las APIs REST públicas pueden enriquecer la experiencia del usuario al proporcionar información en tiempo real y permitir la interacción con plataformas externas. La posibilidad de consultar el clima

de ciudades de Ecuador y mostrar ubicaciones en un mapa desde una misma aplicación web muestra el potencial de las APIs para crear soluciones innovadoras y centradas en el usuario.

La plataforma .NET, especialmente con el uso de ASP.NET Core y Entity Framework Core, ofrece una infraestructura robusta y eficiente para la creación de servicios RESTful. Su capacidad para manejar aplicaciones de gran escala, junto con un ecosistema rico en herramientas y bibliotecas, permite a los desarrolladores construir APIs de alto rendimiento y altamente escalables. La flexibilidad de .NET para operar en diferentes entornos y sistemas operativos refuerza su utilidad en diversas situaciones de desarrollo.

- La implementación de servicios RESTful en .NET beneficia significativamente de la adopción de buenas prácticas y patrones de diseño establecidos. El uso de arquitecturas bien definidas, la separación de preocupaciones y la implementación de principios REST como la statelessness y la uniformidad de interfaces, aseguran que las APIs sean consistentes, mantenibles y fáciles de escalar. Además, la utilización de Entity Framework Core para la gestión de datos simplifica las operaciones CRUD y mejora la eficiencia del desarrollo.
- La seguridad en los servicios RESTful es fundamental y .NET proporciona diversas herramientas y mecanismos para asegurar APIs. La integración de autenticación mediante tokens JWT y la implementación de políticas de autorización robustas garantizan que solo usuarios y aplicaciones autorizadas puedan acceder a los recursos, protegiendo los datos y las operaciones sensibles. La capacidad de .NET para incorporar estándares como OAuth 2.0 y OpenID Connect facilita la implementación de soluciones seguras y confiables.
- La documentación clara y el versionado adecuado de las APIs son esenciales para la mantenibilidad y la evolución continua de los servicios RESTful. Herramientas como Swashbuckle y la especificación OpenAPI permiten generar documentación interactiva, lo que facilita la comprensión y el uso de las APIs por parte de los desarrolladores. Estrategias de versionado bien definidas aseguran que las APIs puedan evolucionar sin romper la compatibilidad con los clientes existentes, permitiendo una introducción gradual de nuevas funcionalidades.
- El uso de .NET para el desarrollo de servicios RESTful impacta positivamente en la eficiencia y productividad del proceso de desarrollo. La integración de herramientas de desarrollo como Visual Studio, junto con una comunidad activa y recursos de soporte abundantes, reduce significativamente el tiempo y el esfuerzo necesario para construir, probar y desplegar aplicaciones. Esto no solo acelera el ciclo de desarrollo, sino que también permite a las organizaciones responder rápidamente a las cambiantes necesidades del mercado y a las demandas de los usuarios.

La práctica de desarrollar aplicaciones con Spring Boot, Spring Boot Eureka y Spring Boot Gateway ofrece un panorama integral sobre la creación de sistemas basados en microservicios, abordando diversos aspectos cruciales como la simplicidad del desarrollo, la gestión de servicios y la optimización del enrutamiento y la seguridad. A continuación, se presentan las conclusiones detalladas derivadas de esta práctica:

1. **Eficiencia en el Desarrollo con Spring Boot:**
    - Simplicidad y Productividad: Spring Boot ha demostrado ser una herramienta esencial para aumentar la productividad del desarrollo de aplicaciones Java, eliminando la necesidad de configuraciones complicadas y permitiendo a los desarrolladores enfocarse en la lógica del negocio.
    - Estandarización y Buenas Prácticas: Al promover una estructura de proyecto estándar y el uso de buenas prácticas, Spring Boot facilita la mantenibilidad y la evolución de las aplicaciones a lo largo del tiempo.
2. **Escalabilidad y Gestión de Servicios con Spring Boot Eureka:**
    - Descubrimiento de Servicios Dinámico: Spring Boot Eureka proporciona una solución efectiva para el descubrimiento dinámico de servicios en una arquitectura de microservicios, permitiendo a las aplicaciones adaptarse a cambios y fallos sin interrupciones.
    - Escalabilidad Horizontal: La capacidad de registrar y desregistrar servicios dinámicamente facilita la escalabilidad horizontal, permitiendo la adición o eliminación de instancias de servicios según las necesidades de carga.
3. **Optimización del Enrutamiento y Seguridad con Spring Boot Gateway:**
    - Centralización del Enrutamiento: Spring Boot Gateway centraliza la gestión del enrutamiento, simplificando la comunicación entre los microservicios y mejorando la eficiencia del sistema.
    - Filtros y Políticas de Seguridad: La implementación de filtros personalizados y políticas de seguridad en el Gateway mejora la protección de los microservicios, asegurando que solo el tráfico autorizado acceda a los recursos.
4. **Resiliencia y Mantenibilidad de Microservicios:**
    - Resiliencia ante Fallos: La arquitectura de microservicios, apoyada por Spring Boot, Eureka y Gateway, mejora la resiliencia del sistema, permitiendo la recuperación rápida de fallos y minimizando el impacto en el usuario final.
    - Mantenibilidad y Evolución: La separación de responsabilidades en microservicios facilita la mantenibilidad y la evolución del sistema, permitiendo a los equipos de desarrollo trabajar de manera más eficiente y enfocada.

Finalmente, la instalación paso a paso de Laravel proporciona una base sólida para comenzar a desarrollar aplicaciones web modernas y escalables. Laravel ofrece un proceso de instalación sencillo y bien documentado, lo que facilita a los desarrolladores comenzar a trabajar rápidamente en sus proyectos. El uso de Composer para la instalación de Laravel simplifica la gestión de dependencias y la incorporación de paquetes. Laravel es un punto de partida sólido para los desarrolladores que buscan crear aplicaciones web modernas y escalables, ofreciendo facilidad, flexibilidad y un ecosistema robusto para el desarrollo eficiente de proyectos.

8. RECOMENDACIONES

Uso de Herramientas y Tecnologías Estándar:

- **Compatibilidad e Interoperabilidad:** Utilizar herramientas y tecnologías estándar para garantizar la compatibilidad con otros sistemas, aplicando estándares como XML y SOAP para asegurar la interoperabilidad.
- **Validación de Datos:** Validar los datos antes de utilizarlos y manejar adecuadamente los errores en el código. Asegurar la calidad de los datos recibidos de servicios web externos, evitando errores y vacíos.
- **Pruebas y Depuración:** Emplear herramientas para probar y depurar los web services, permitiendo realizar pruebas tanto en servidores locales como externos de manera fácil y efectiva.

Adopción Gradual y Planificada de Microservicios con Spring Boot:

- **Evaluación Inicial:** Realizar una evaluación exhaustiva de los requisitos del proyecto y las capacidades del equipo antes de adoptar una arquitectura de microservicios.
- **Implementación Gradual:** Comenzar con la separación de los servicios más críticos y complejos, extendiéndose progresivamente a otros componentes.

Capacitación y Formación Continua:

- **Formación Inicial:** Proveer formación y recursos educativos sobre Spring Boot, Eureka y Gateway a los equipos de desarrollo para asegurar una comprensión sólida de estas tecnologías.
- **Actualización Continua:** Fomentar la actualización de conocimientos mediante la participación en conferencias, talleres y cursos especializados.

Monitoreo y Observabilidad:

- **Implementación de Herramientas:** Utilizar herramientas de monitoreo y logging como Spring Boot Actuator, Prometheus y Grafana para mantener una visibilidad completa sobre el estado y el rendimiento de los microservicios.
- **Alertas y Análisis:** Configurar alertas para detectar problemas proactivamente y realizar análisis periódicos de rendimiento para identificar y resolver cuellos de botella.

Pruebas y Calidad de Código:

- **Pruebas Automatizadas:** Implementar pruebas unitarias, de integración y de aceptación automatizadas para asegurar la calidad del código y la funcionalidad de los microservicios.
- **Revisiones de Código:** Establecer revisiones de código regulares para mantener altos estándares de calidad y fomentar la colaboración entre desarrolladores.

Seguridad y Cumplimiento:

- **Autenticación y Autorización:** Utilizar Spring Security para implementar mecanismos robustos de autenticación y autorización, protegiendo los microservicios de accesos no autorizados.
- **Cumplimiento de Normativas:** Asegurar que las aplicaciones cumplan con las normativas y regulaciones relevantes, incluyendo protección de datos y privacidad.

Estrategia de Despliegue y CI/CD:

- **Automatización del Despliegue:** Adoptar prácticas de Integración Continua y Despliegue Continuo (CI/CD) para automatizar el proceso de despliegue, reduciendo errores humanos y acelerando la entrega de nuevas funcionalidades.
- **Entornos de Prueba:** Configurar entornos de prueba y staging que repliquen el entorno de producción para validar los cambios antes de su despliegue final.

Maximizar Beneficios de APIs REST Públicas:

- **Documentación y Buenas Prácticas:** Mantener una documentación clara y aplicar buenas prácticas en el diseño y consumo de APIs REST públicas para asegurar la interoperabilidad y facilidad de uso.
- **Gestión de Dependencias:** Utilizar gestores de dependencias como Composer en Laravel para simplificar la incorporación de paquetes y mantener el código organizado y eficiente.

9. BIBLIOGRAFÍA

[1] F. d. A. López, Sistemas Distribuidos, J. Rosas, Ed., Universidad Autónoma Metropolitana, 2015.

[2] G. Coulouris, J. Dollimore, T. Kindberg y G. Blair, DISTRIBUTED SYSTEMS: Concepts and Design, Quinta ed., M. Hirsch, Ed., Pearson Education Inc, 2012.

[3] S. Ghosh, Distributed Systems: An Algorithmic Approach, 2 ed., Lowa City: CRC Press, 2015.

[4] A. Kshemkalyani y M. Singhal, Distributed Computing: Principles, Algorithms, and Systems, Cambirdge: Cambridge University Press, 2011.

[5] g. Authors, «gRPC,» 2023. [En línea]. Available: https://grpc.io/about/.

[6] IBM, «IBM Topics,» 2023. [En línea]. Available: https://www.ibm.com/topics/soa?mhsrc=ibmsearch_a&mhq=soa.

[8] W3C, «SERVICIOS,» 2007. [En línea]. Available: https://www.w3.org/TR/soap12/.

[9] W3C, «Working Group Note,» 2004. [En línea]. Available: https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/.

[10] IBM, «Developer IBM,» 2023. [En línea]. Available: https://developer.ibm.com/articles/what-is-a-web-service/.

[11] Oracle, «Web Services Support,» 2023. [En línea]. Available: https://docs.oracle.com/cd/E19798-01/821-1841/bnabs/index.html.

[12] Orcale, «Java Documentation,» [En línea]. Available: https://docs.oracle.com/javase/tutorial/.

[13] Microsoft, «Microsoft Docs,» [En línea]. Available: https://dotnet.microsoft.com/es-es/learn/dotnet/what-is-dotnet.

[14] Mozilla, «Working with JSON,» 2023. [En línea]. Available: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON.

[15] Microsoft, «XML Documents and Data,» 2023. [En línea]. Available: https://learn.microsoft.com/en-us/dotnet/standard/data/xml/.

[16] W3C Schools, «JSON,» 2023. [En línea]. Available: https://www.w3schools.com/js/js_json_intro.asp.

[17] Apache, «Apache Netbeans,» 2023. [En línea]. Available: https://netbeans.apache.org/.

[18] Microsoft, «Visual Studio,» 2019. [En línea]. Available: https://visualstudio.microsoft.com/vs/.

[19] Postman INC, «Postman,» 2023. [En línea]. Available: https://www.postman.com/.