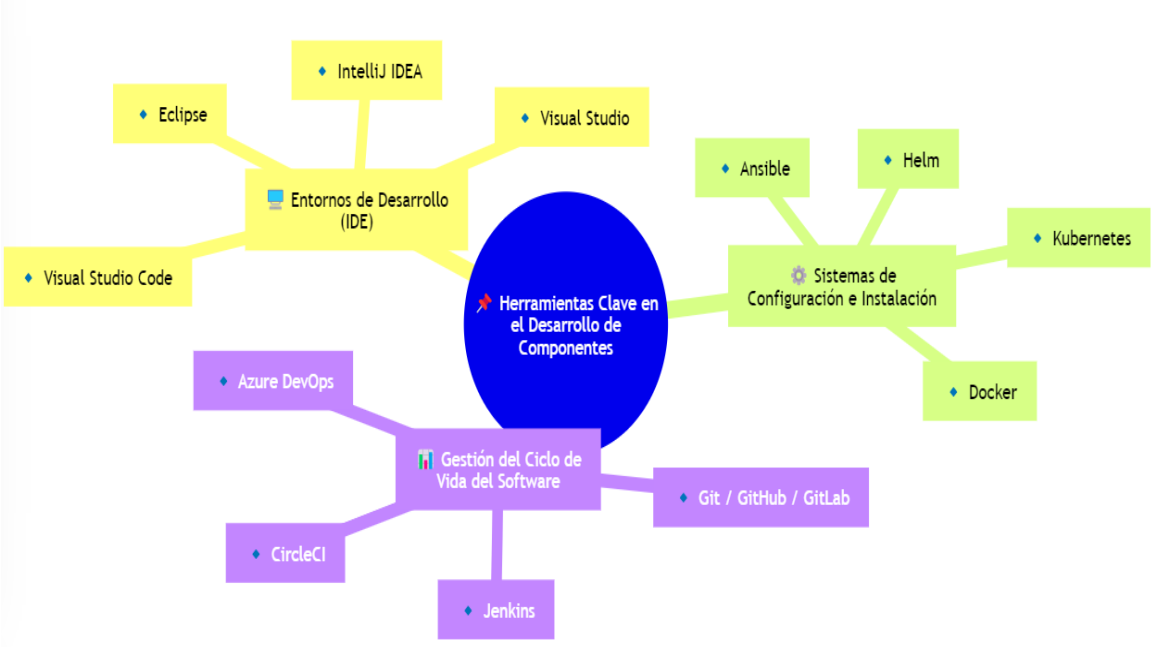
7. Herramientas para el desarrollo de componentes.

Cuando se trabaja con desarrollo basado en componentes, no basta con escribir código. También es necesario contar con las herramientas adecuadas para **programar, probar, depurar e integrar** los componentes dentro de una aplicación más grande. Estas herramientas facilitan el trabajo de los desarrolladores y permiten que los proyectos sean más organizados y eficientes.

Algunas de las más importantes son los **entornos integrados de desarrollo (IDE)**, **los sistemas de configuración e instalación de herramientas** y las plataformas para la **gestión del ciclo de vida del software**. Cada una de estas herramientas cumple un papel clave en el proceso de desarrollo, desde la escritura del código hasta la implementación final del software.



Herramientas clave en el desarrollo de componentes.

Los **entornos de desarrollo integrado (IDE)** son programas que ayudan a los desarrolladores a escribir y depurar código de forma más rápida y organizada. Algunos de los más conocidos son **IntelliJ IDEA**, ideal para proyectos en Java; **Visual Studio**, muy utilizado para desarrollar en .NET; **Eclipse**, otra opción popular para Java y otros lenguajes; y **Visual Studio Code**, una alternativa más ligera y flexible compatible con muchas tecnologías.

Por otro lado, los **sistemas de configuración e instalación** permiten gestionar cómo se instalan y despliegan las aplicaciones en distintos entornos. Herramientas como **Ansible** automatizan la configuración de servidores y aplicaciones, **Helm** facilita el despliegue de aplicaciones en Kubernetes, **Docker** permite empaquetar software en contenedores portátiles, y **Kubernetes** ayuda a administrar esos contenedores en sistemas distribuidos.

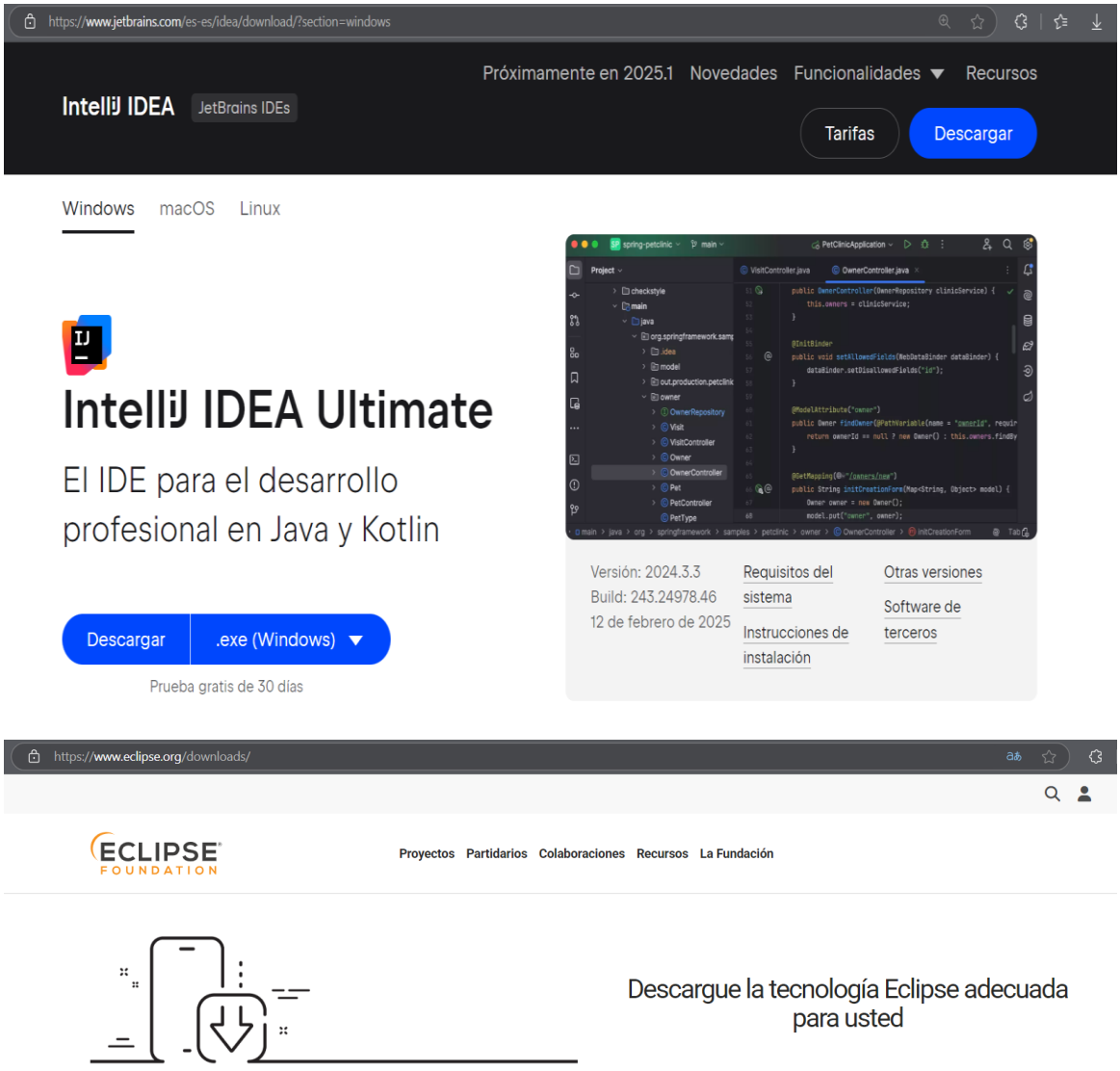
Además, para mantener un desarrollo fluido, es clave contar con herramientas para el **ciclo de vida del software**. **GitHub, GitLab y Git** permiten gestionar el código y colaborar en equipo, mientras que **Jenkins, Azure DevOps y CircleCI** se encargan de automatizar pruebas y despliegues, lo que ahorra tiempo y reduce errores en el proceso de desarrollo.

7.1. Entornos integrados de desarrollo de componentes.

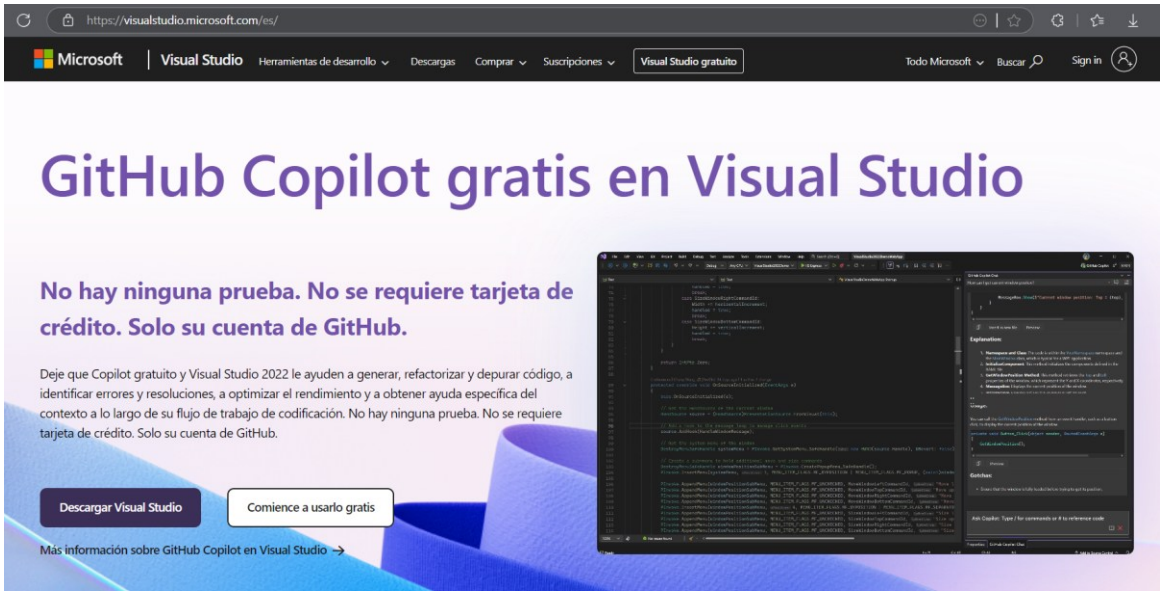
Un **entorno de desarrollo integrado (IDE)** es un software que proporciona todo lo necesario para programar en un solo lugar. En un IDE, los desarrolladores pueden **escribir código, compilarlo, ejecutarlo, depurarlo y gestionar dependencias** sin necesidad de usar múltiples herramientas separadas.

En el desarrollo de componentes, un buen IDE permite programar y también ayuda a organizar el proyecto, administrar bibliotecas y automatizar tareas repetitivas. Algunos de los más utilizados hoy en día son:

- **IntelliJ IDEA y Eclipse**: muy populares en el desarrollo con Java y frameworks como Spring Boot.



- **Visual Studio y Visual Studio Code**: usados en el ecosistema .NET, pero también compatibles con otros lenguajes como JavaScript, Python y Go.



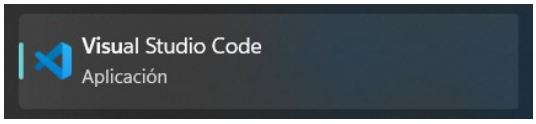
Saber más

Visual Studio Code (VS Code) es uno de los editores más populares entre los desarrolladores web. Ofrece una excelente integración con herramientas de desarrollo web y es altamente personalizable mediante extensiones. Es un editor de código fuente desarrollado por Microsoft. Es gratuito, de código abierto y compatible con Windows, macOS y Linux.

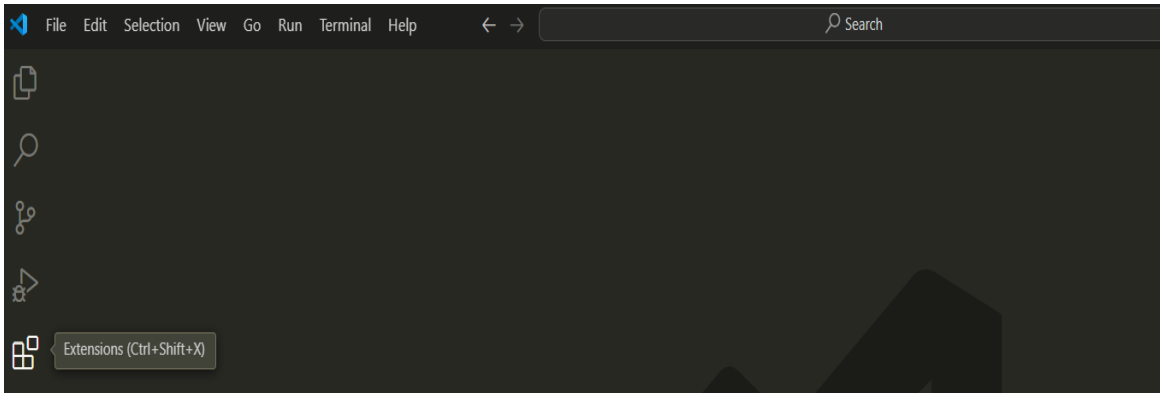
A continuación, se presentan los pasos para instalar este editor y algunas de sus extensiones:

1. Ve al sitio oficial de Visual Studio Code en tu navegador web.
2. Haz clic en el botón de descarga para tu sistema operativo (Windows, MacOS, Linux):

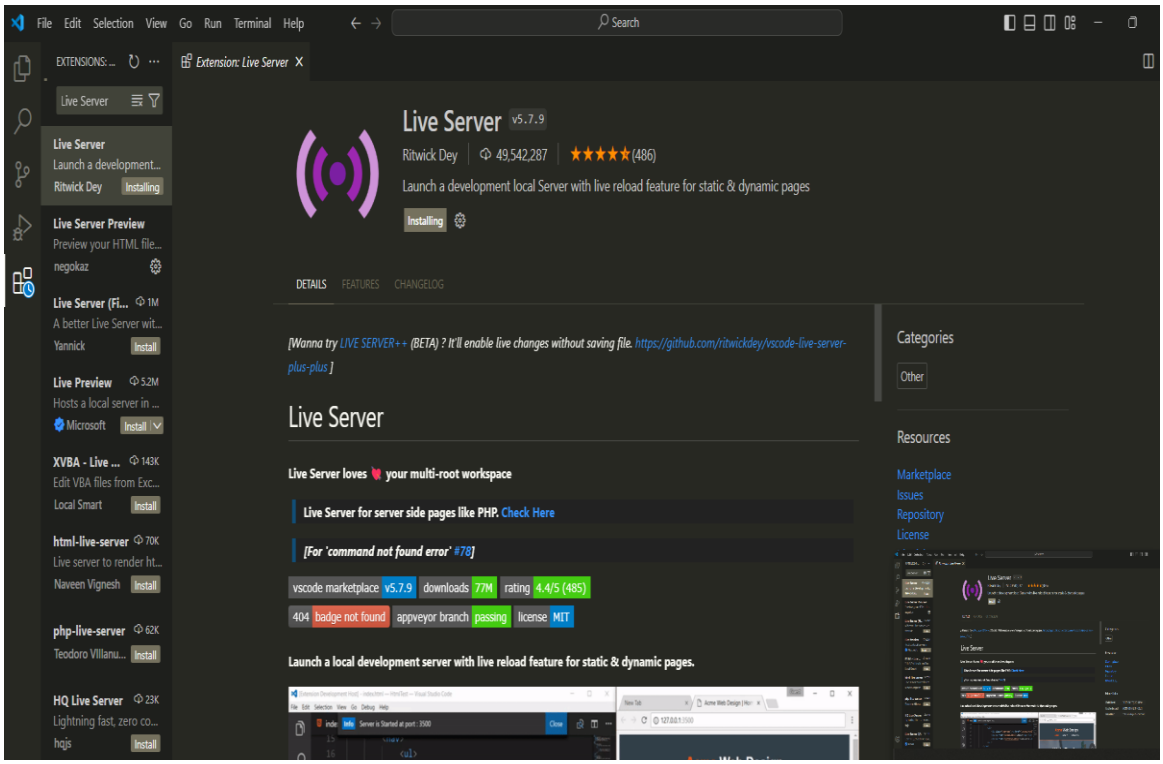
3. Una vez descargado el archivo, ábrelo para iniciar el proceso de instalación.
4. Sigue las instrucciones del instalador, que incluirán aceptar los términos de servicio y elegir la ubicación de instalación.
5. Una vez completada la instalación, puedes abrir Visual Studio Code:



6. Haz clic en el icono de extensiones en la barra lateral izquierda (o presiona Ctrl+Shift+X).

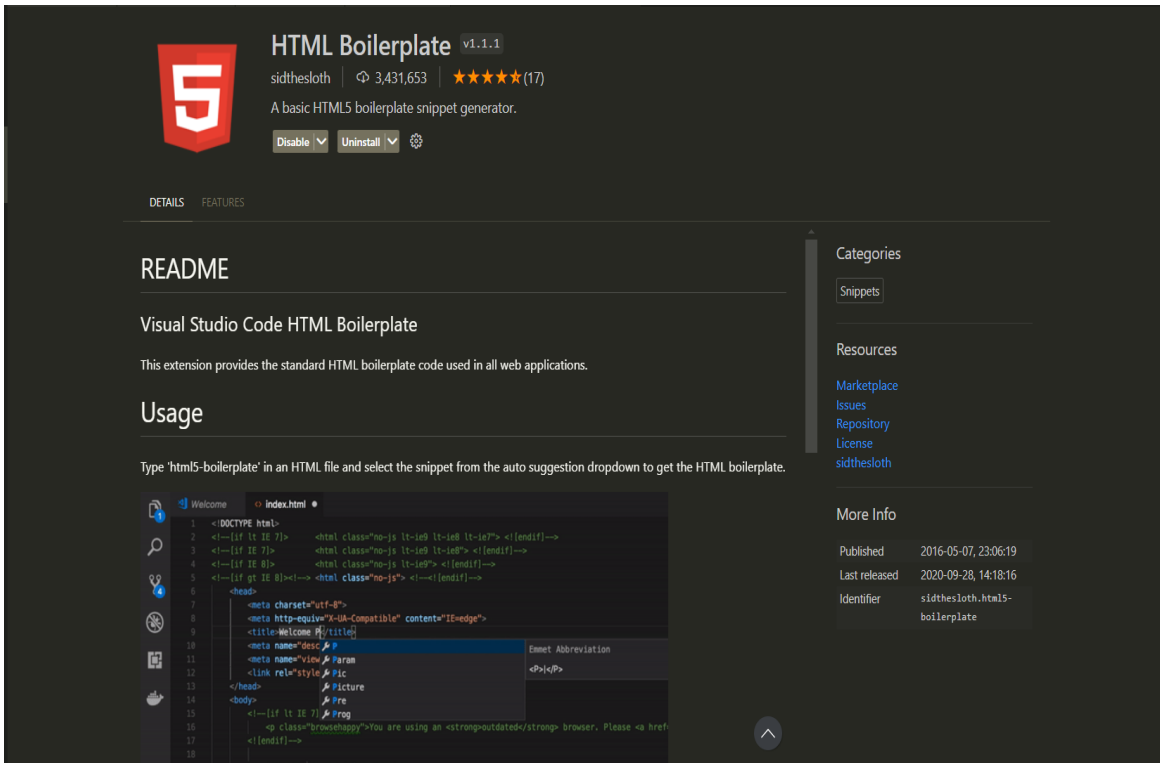


7. En la barra de búsqueda en la parte superior, escribe el nombre de la extensión que deseas instalar. Por ejemplo, podrías buscar “Live Server”, "HTML Boilerplate" o "Prettier - Code formatter":
8. En los resultados de la búsqueda, haz clic en el botón “Instalar” para la extensión que deseas:



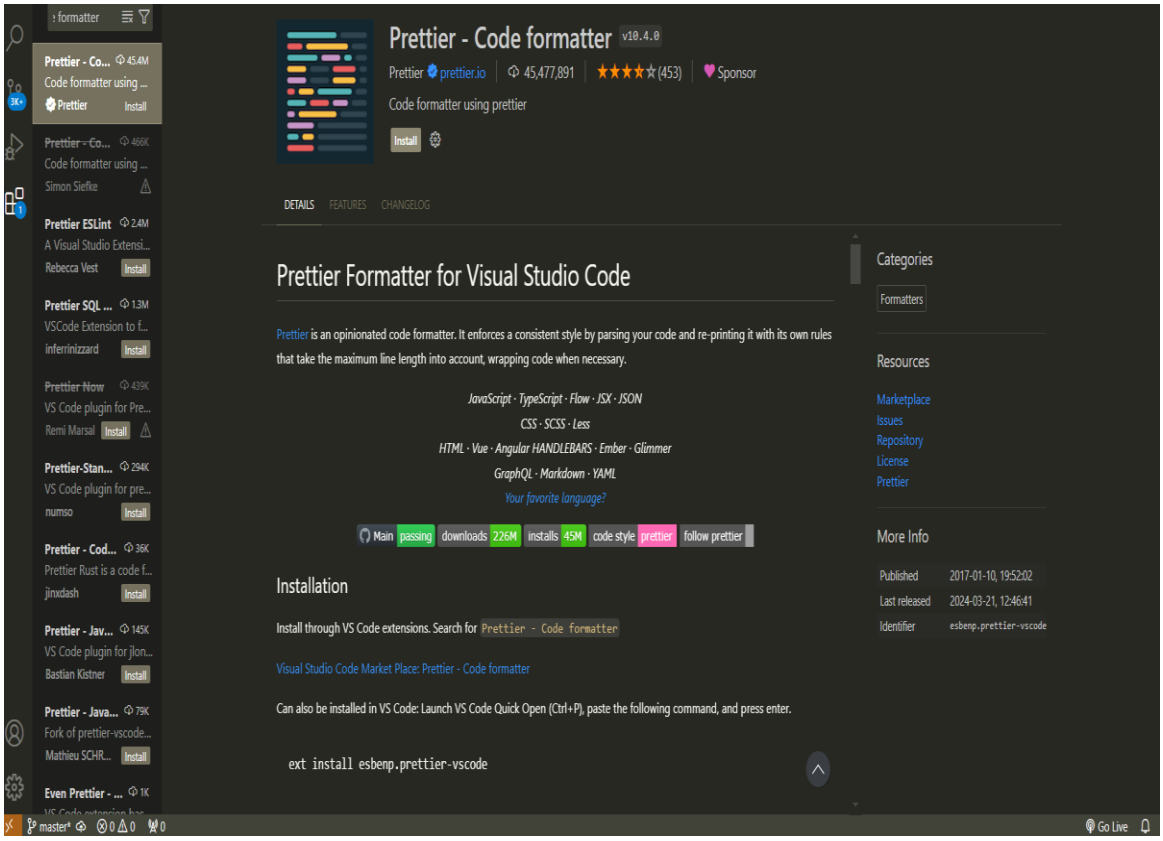
Nota

- Live Server: Esta extensión permite iniciar un servidor de desarrollo local con la funcionalidad de recarga en vivo para páginas estáticas y dinámicas. Facilita la previsualización de páginas web en una ventana del navegador en tiempo real, reflejando automáticamente los cambios realizados en el código sin necesidad de recargar la página. Para utilizarla, basta con abrir un archivo HTML y hacer clic en "Go Live" desde la barra de estado para encender o apagar el servidor.



Nota

- HTML Boilerplate: Esta herramienta genera una estructura básica de HTML5. Para emplearla, se debe escribir ‘html5-boilerplate’ en un archivo HTML y seleccionar el fragmento correspondiente de la lista desplegable de autocompletado para obtener la estructura HTML5. Alternativamente, se puede usar el atajo de teclado “!” en un archivo HTML y luego presionar “Tab” o “Enter” para generar el esqueleto HTML.



Nota

- Prettier - Code Formatter: Prettier es un formateador de código que aplica un estilo consistente al analizar y reimprimir el código según sus propias reglas, considerando la longitud máxima de línea y envolviendo el código cuando es necesario. Soporta una variedad de lenguajes como JavaScript, TypeScript, CSS, HTML, entre otros. Para asegurar su uso sobre otras extensiones, es necesario configurarlo como el formateador predeterminado en la configuración de VS Code.

9. Una vez instaladas las extensiones, reinicia Visual Studio Code para que los cambios surtan efecto.

- **PyCharm**: especializado en Python, útil para desarrollar componentes en frameworks como Django o FastAPI.

- **WebStorm**: optimizado para JavaScript y TypeScript, muy usado en el desarrollo de componentes frontend con React, Angular y Vue.js.

Un buen IDE puede hacer una gran diferencia en la productividad del desarrollador, ya que incluye características como **autocompletado de código, integración con herramientas de control de versiones (Git) y depuración avanzada**.

7.2. Configuración e instalación de herramientas de uso común.

Dependiendo del lenguaje y la plataforma en la que se trabaje, es necesario instalar herramientas específicas que faciliten el desarrollo de componentes. Entre los entornos más utilizados están **Java y .NET**, que cuentan con ecosistemas completos para la creación de software modular.

7.2.1. Entorno Java.

Para desarrollar componentes en **Java**, es fundamental configurar un entorno de desarrollo adecuado. Esto incluye:

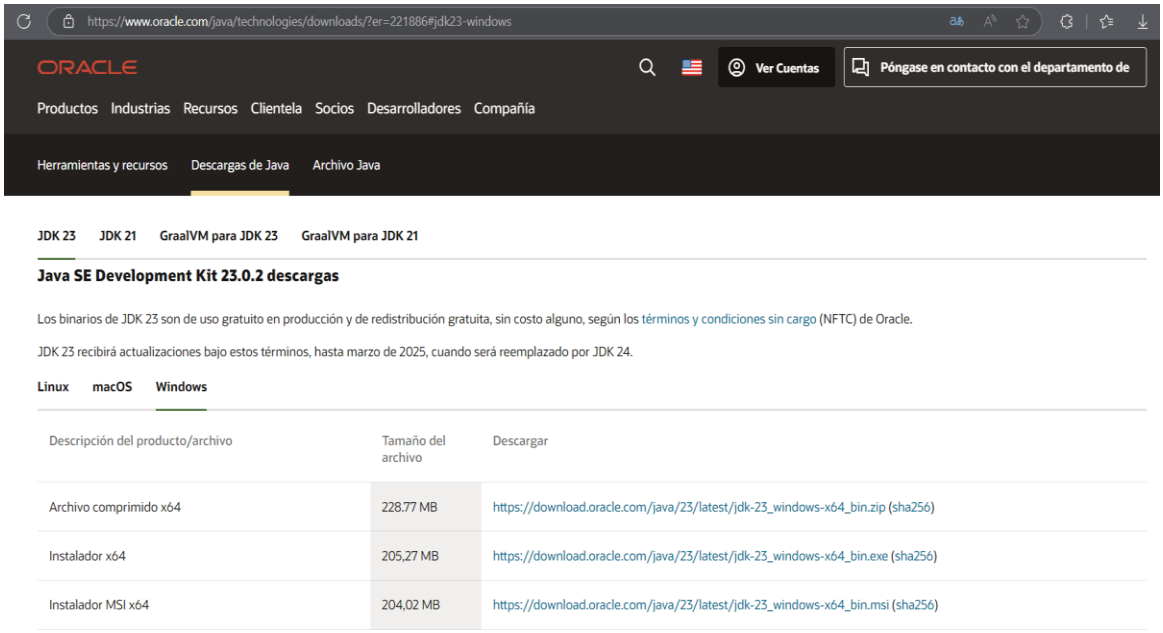
- **Java Development Kit (JDK)**: el conjunto de herramientas necesario para compilar y ejecutar aplicaciones en Java.
- **Maven y Gradle**: sistemas de gestión de dependencias y automatización de compilación.
- **Spring Boot**: un framework muy utilizado para crear aplicaciones basadas en componentes, especialmente en el desarrollo de microservicios.
- **IntelliJ IDEA o Eclipse**: los IDE más usados en el ecosistema Java.

A continuación, se explica paso por paso cómo configurar el entorno de desarrollo para trabajar con Java y crear aplicaciones modulares con Spring Boot:

Instalar el JDK (Java Development Kit)

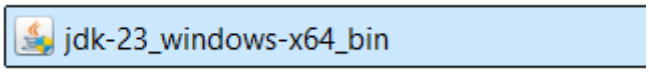
El JDK es lo primero que necesitas, ya que incluye todo lo necesario para compilar y ejecutar programas en Java.

1. Visita la página oficial de Oracle o Adoptium para descargar el **JDK:**
    1. Oracle JDK: https://www.oracle.com/java/technologies/downloads/?er=221886#jdk23-windows

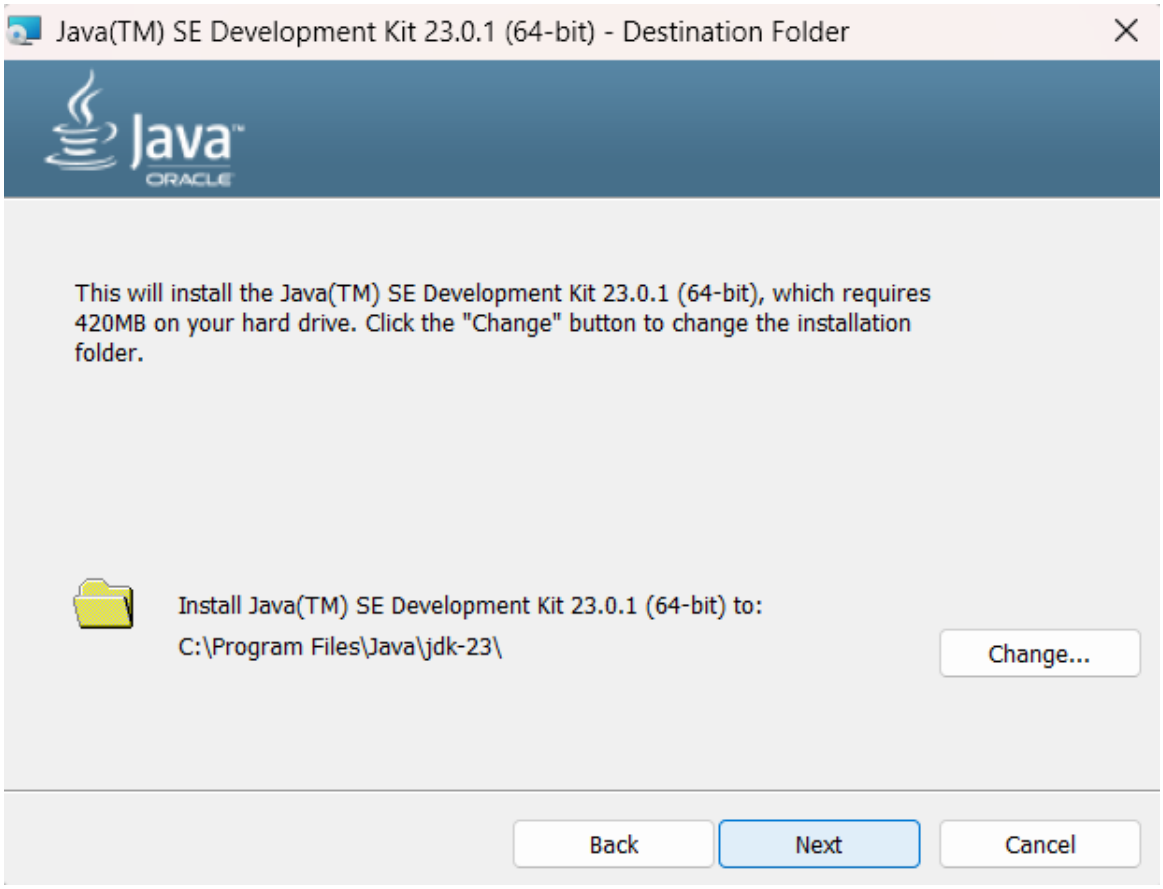


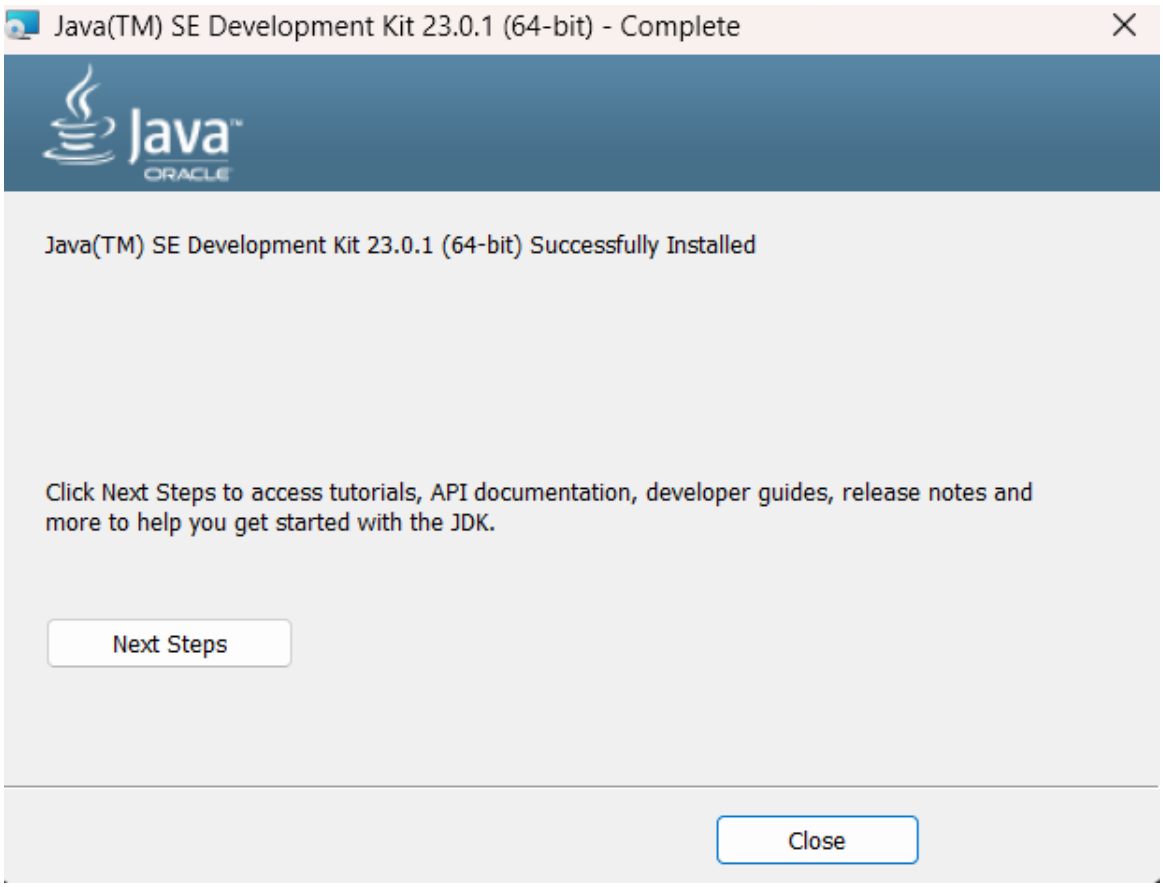
2. OpenJDK (Adoptium): https://adoptium.net/es/

2. Descarga el instalador según tu sistema operativo (Windows, macOS o Linux):



3. Ejecuta el instalador y sigue las instrucciones de instalación:

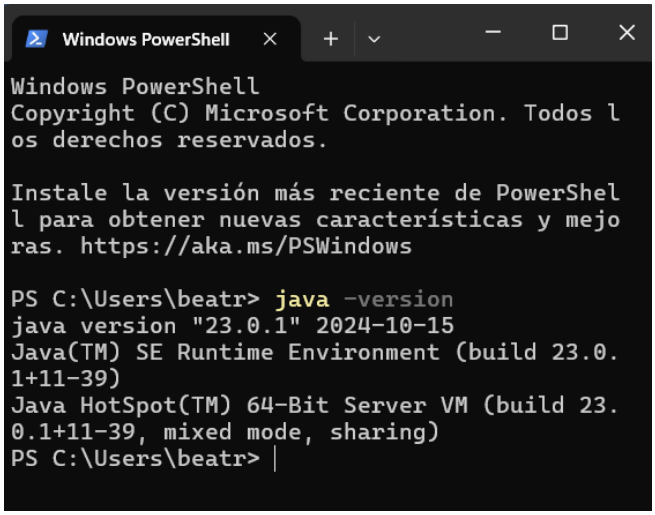

Java(TM) SE Development Kit 23.0.1 (64-bit) - Destination Folder
Java
ORACLE
This will install the Java(TM) SE Development Kit 23.0.1 (64-bit), which requires 420MB on your hard drive. Click the "Change" button to change the installation folder.
Install Java(TM) SE Development Kit 23.0.1 (64-bit) to:
C:\Program Files\Java\jdk-23\
Change...
Back
Next
Cancel


Java(TM) SE Development Kit 23.0.1 (64-bit) - Complete
Java
ORACLE
Java(TM) SE Development Kit 23.0.1 (64-bit) Successfully Installed
Click Next Steps to access tutorials, API documentation, developer guides, release notes and more to help you get started with the JDK.
Next Steps
Close

4. Para verificar que la instalación fue correcta, abre una terminal o consola y escribe:

```
java -version
```

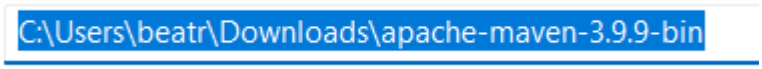
Si ves un mensaje con la versión instalada, significa que todo está listo:



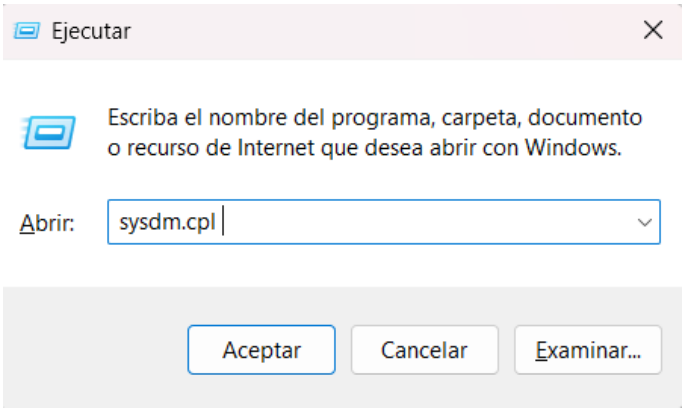
Instalar Maven

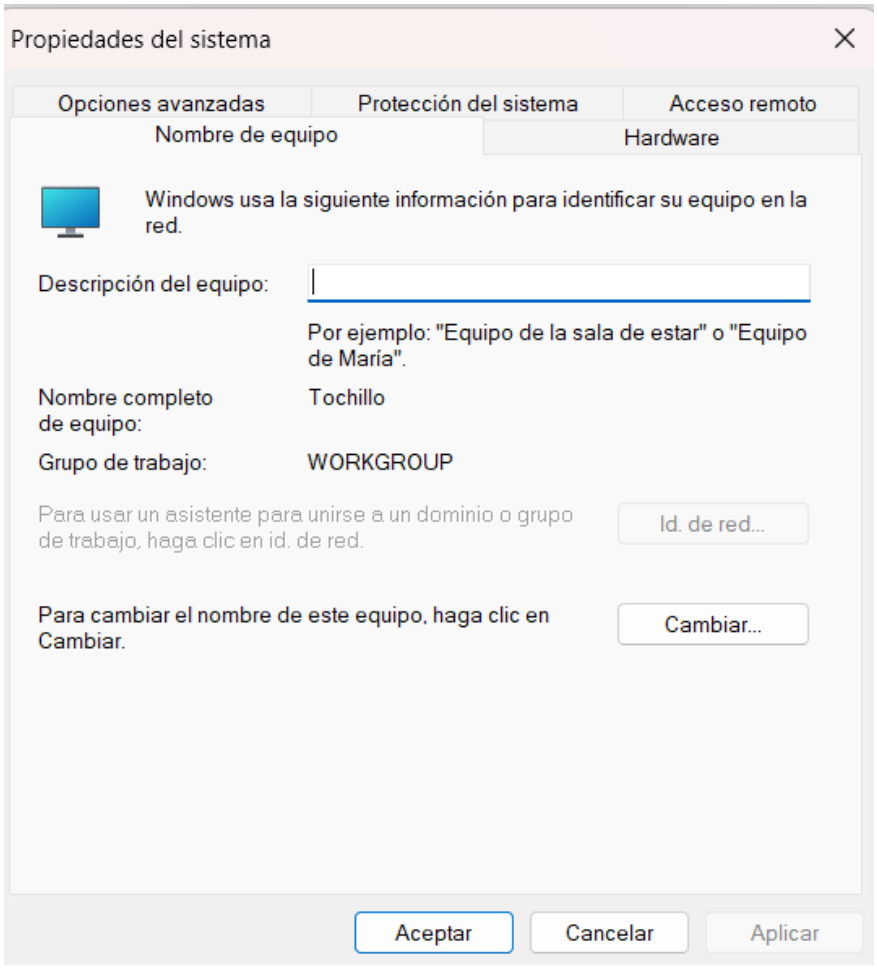
1. Descarga Maven desde su página oficial: https://maven.apache.org/download.cgi

2. Extrae el contenido del archivo descargado en una ubicación de tu elección:

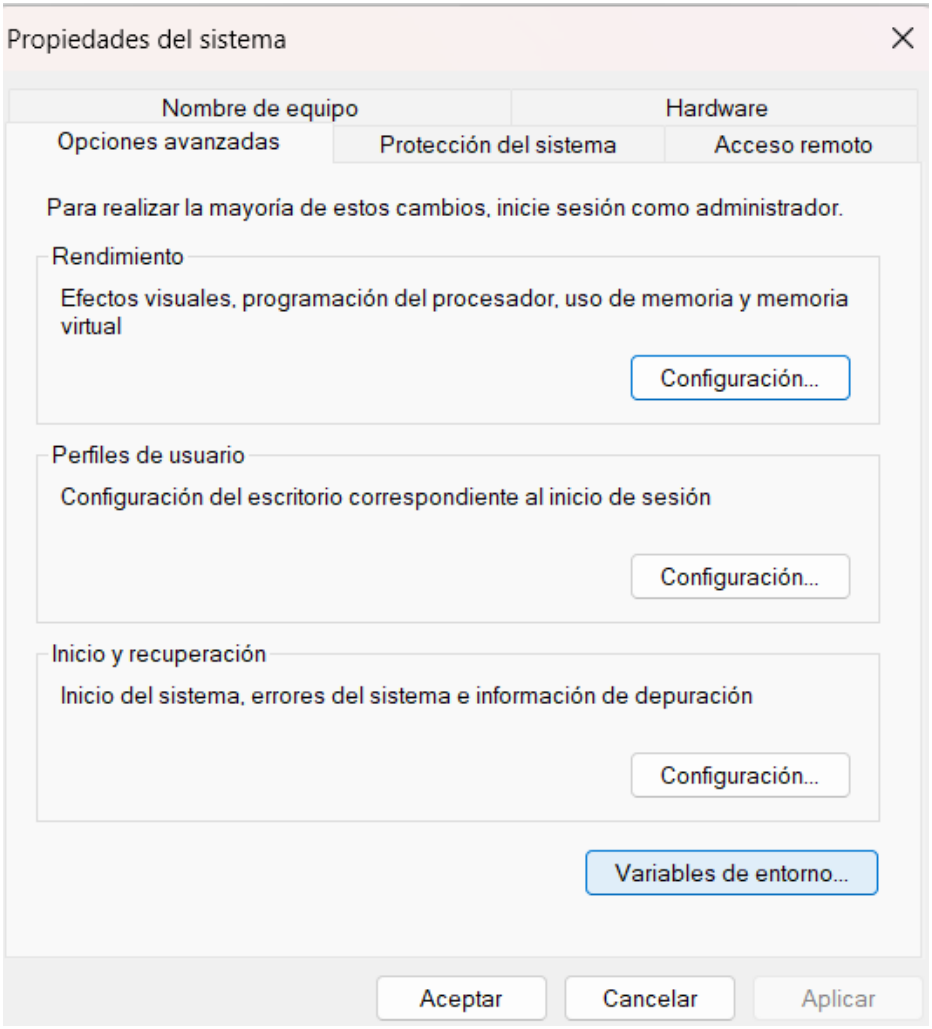


3. Configura la variable de entorno MAVEN_HOME apuntando a la carpeta donde extrajiste Maven. Para configurar Maven en tu sistema, sigue estos pasos:
    - Presiona Win + R, escribe sysdm.cpl y presiona Enter:

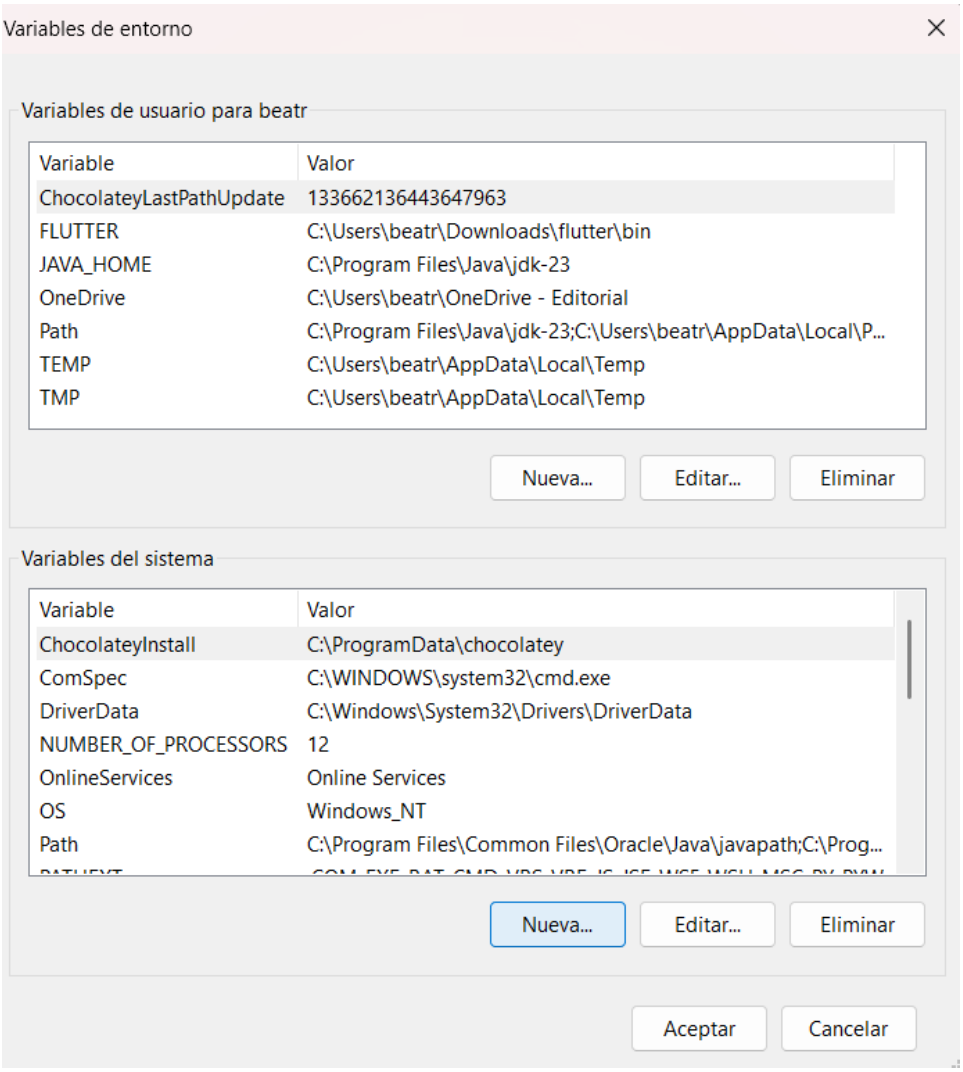




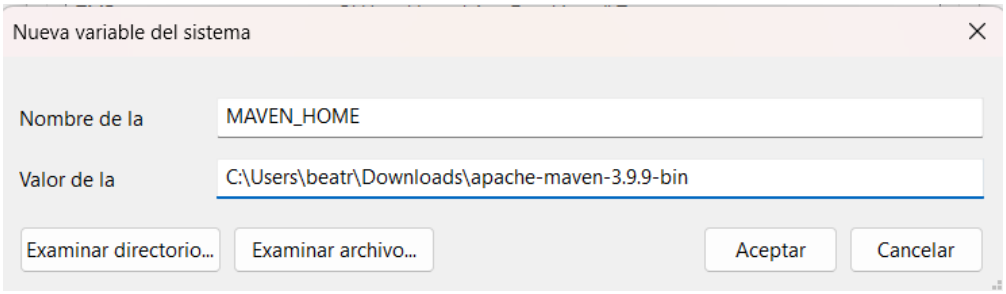
- Ve a la pestaña "Opciones avanzadas" y haz clic en "Variables de entorno":



- En la sección de Variables del sistema, haz clic en "Nueva".



- En Nombre de la variable, escribe: MAVEN_HOME.
- En Valor de la variable, ingresa la ruta donde extrajiste Maven:

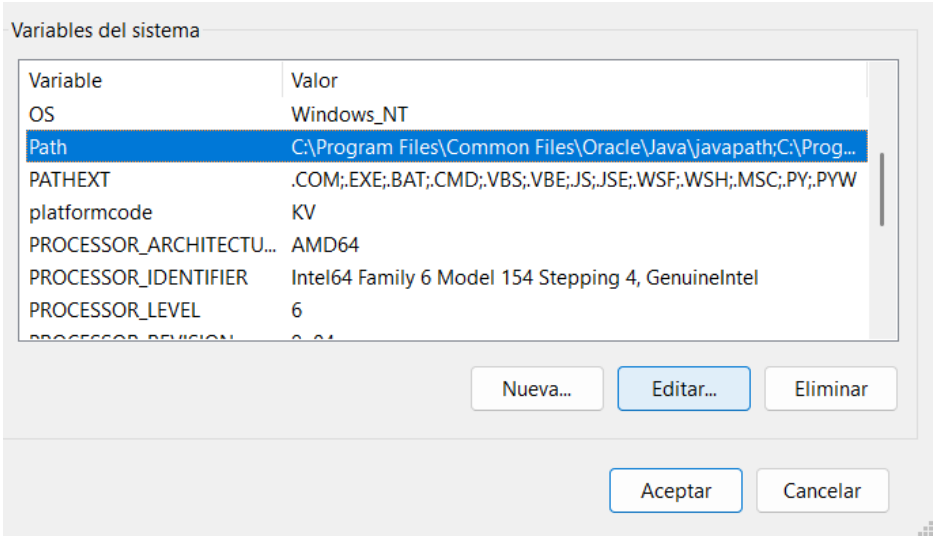


- Haz clic en Aceptar.

4. Agrega la carpeta bin de Maven al PATH del sistema para poder ejecutarlo desde la terminal. Para ello:

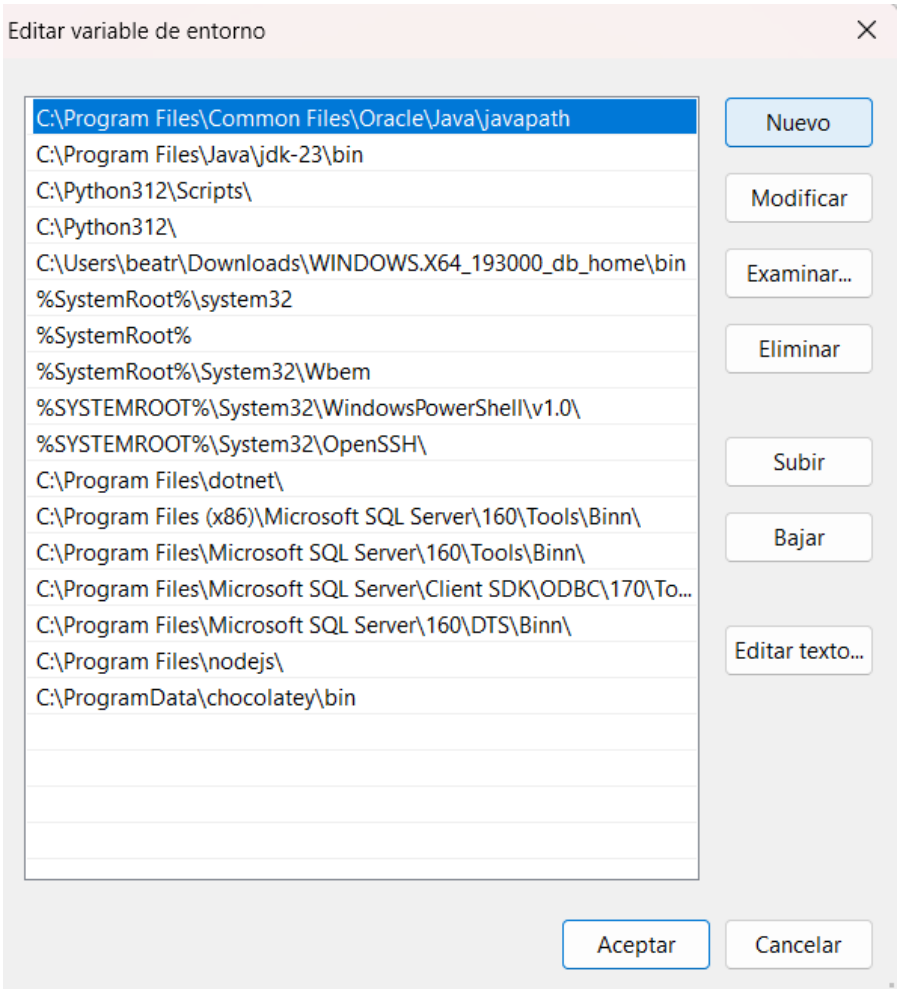
   Edita la variable Path

   - En la misma ventana de Variables de entorno, busca la variable llamada Path en la sección Variables del sistema y selecciona Editar:

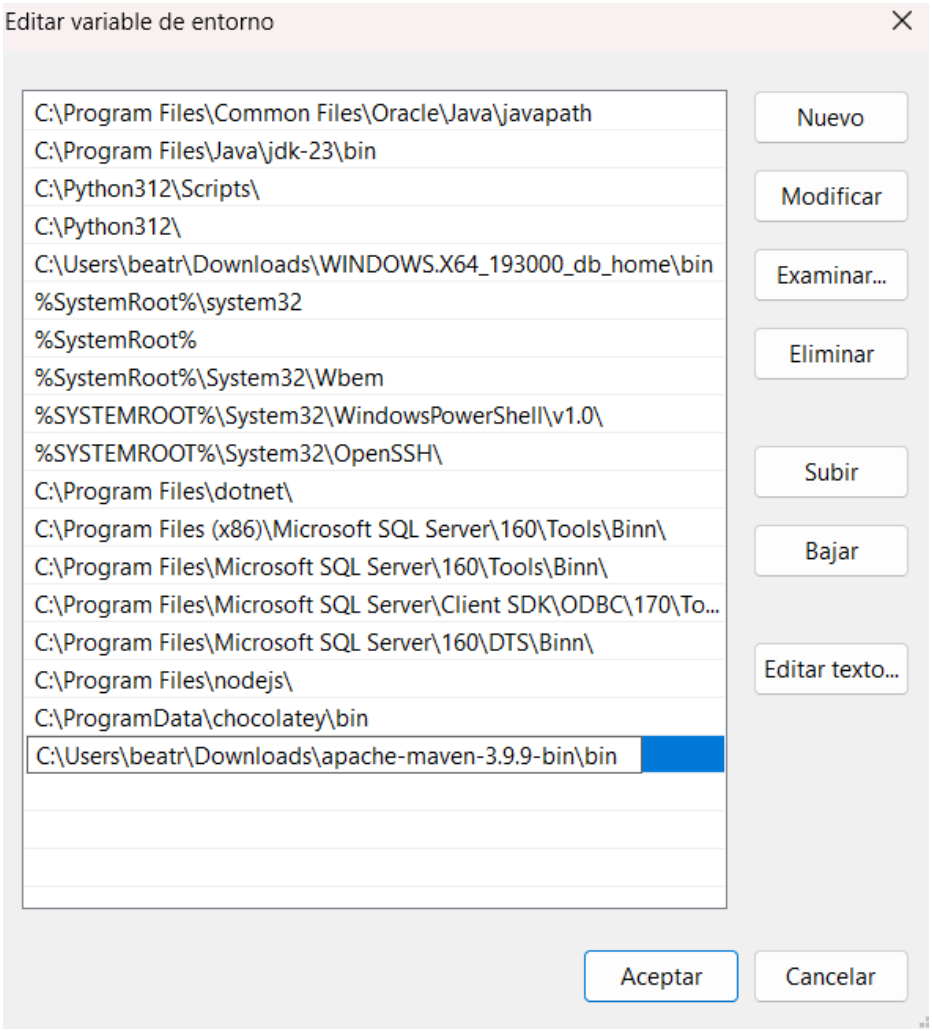


Añade la ruta de Maven

- Haz clic en "Nuevo" y agrega la siguiente ruta: C:\Users\beatr\Downloads\apache-maven-3.9.9-bin\bin



- Asegúrate de que la ruta esté correctamente escrita y presiona Aceptar.



5. Para comprobar que está funcionando, abre la terminal y ejecuta:

```
mvn -version
```

Si ves información sobre la versión instalada, ya está listo para usarse.

Instalar Gradle (alternativa a Maven)

1. Descarga Gradle desde https://gradle.org/install/.
2. Extrae los archivos en una ubicación de tu elección.
3. Configura la variable de entorno GRADLE_HOME y agrega su carpeta bin al **PATH** del sistema.
4. Para verificar la instalación, abre la terminal y ejecuta:

```
gradle -v
```

Si ves la versión de Gradle, la instalación fue exitosa.

Instalar Spring Boot

1. Instala Maven o Gradle, ya que Spring Boot se gestiona a través de estas herramientas.
2. Descarga **Spring Boot CLI** desde https://spring.io/projects/spring-boot.
3. Extrae los archivos y configura la variable de entorno SPRING_HOME apuntando a la carpeta de Spring Boot.
4. Agrega la carpeta bin de Spring Boot al **PATH** del sistema.
5. Para verificar la instalación, ejecuta en la terminal:

```
spring --version
```

Si ves un número de versión, significa que está funcionando correctamente.

Instalar IntelliJ IDEA

1. Ve a la página de JetBrains: https://www.jetbrains.com/idea/download/.
2. Descarga la versión **Community** (gratuita) o la versión **Ultimate** (de pago, con más funciones).
3. Ejecuta el instalador y sigue los pasos para completar la instalación.
4. Una vez instalado, abre IntelliJ IDEA y configura el JDK en **File > Project Structure > SDKs**.

Instalar Eclipse (alternativa a IntelliJ IDEA)

1. Descarga Eclipse desde https://www.eclipse.org/downloads/.
2. Ejecuta el instalador y elige la versión **Eclipse IDE for Java Developers**.
3. Completa la instalación siguiendo las instrucciones.
4. Abre Eclipse y configura el JDK en **Window > Preferences > Java > Installed JREs**.

En proyectos modernos, muchas aplicaciones Java se ejecutan en contenedores **Docker**, lo que permite empaquetar componentes junto con sus dependencias y desplegarlos en cualquier servidor sin preocuparse por configuraciones locales. Un contenedor es un entorno ligero y aislado que incluye todo lo necesario para ejecutar la aplicación, como el código, las bibliotecas y las configuraciones requeridas.

¿Por qué usar Docker en aplicaciones Java?

Al empaquetar la aplicación junto con sus dependencias en un contenedor, se garantiza que funcionará igual en cualquier servidor sin importar la configuración del sistema operativo o las versiones de Java instaladas.

Además, la misma imagen de Docker puede ejecutarse en cualquier entorno, ya sea en una máquina local, un servidor en la nube o un clúster Kubernetes.

Con Docker, se pueden automatizar despliegues en diferentes entornos (desarrollo, pruebas y producción) sin necesidad de configurar cada servidor manualmente.

Por ejemplo, si tienes una aplicación Java con Spring Boot, en lugar de ejecutarla manualmente con java -jar, puedes crear un archivo Dockerfile para empaquetarla y ejecutarla en un contenedor:

```
# Usa una imagen base con Java 17
FROM openjdk:17-jdk-slim
```

```
# Copia el archivo JAR de la aplicación al contenedor
COPY target/mi-aplicacion.jar /app.jar

# Expone el puerto en el que se ejecutará la aplicación
EXPOSE 8080

# Comando para ejecutar la aplicación dentro del contenedor
CMD ["java", "-jar", "/app.jar"]
```

Luego, con Docker instalado, puedes construir y ejecutar el contenedor:

sh

```
docker build -t mi-aplicacion .
docker run -p 8080:8080 mi-aplicacion
```

Esto permite que la aplicación se ejecute en cualquier servidor con Docker sin necesidad de instalar Java manualmente, asegurando que siempre funcione con la misma configuración.

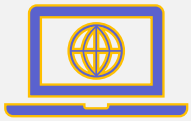
7.2.2. Entorno .NET.

En el ecosistema **.NET**, el entorno de desarrollo también requiere ciertas herramientas específicas:

- **.NET SDK**: el kit de desarrollo necesario para programar aplicaciones con C#. Actualmente, **.NET 8** es la versión más recomendada para nuevos proyectos.
- **Visual Studio**: el IDE más completo para desarrollar aplicaciones .NET, aunque Visual Studio Code también es una opción ligera y potente.
- **NuGet**: el gestor de paquetes de .NET, similar a Maven en Java.
- **Blazor y MAUI**: frameworks modernos para desarrollar aplicaciones web y móviles basadas en componentes dentro del ecosistema .NET.

Como en Java, muchas aplicaciones .NET también se ejecutan en contenedores **Docker** y pueden desplegarse en plataformas como **Azure y AWS**.

Actividad 7



Instala Visual Studio Code (VS Code), configura algunas extensiones útiles y crea un componente sencillo en JavaScript o Python, reforzando así el uso de herramientas para el desarrollo basado en componentes.

Parte 1: Instalación de Visual Studio Code

Descarga Visual Studio Code desde su sitio oficial: https://code.visualstudio.com/

Instala el programa siguiendo las instrucciones para tu sistema operativo (Windows, macOS o Linux).

Abre Visual Studio Code una vez completada la instalación.

Parte 2: Instalación de extensiones

Haz clic en el icono de extensiones en la barra lateral izquierda o presiona Ctrl + Shift + X.

Busca e instala las siguientes extensiones según el lenguaje que vayas a usar. Para JavaScript/React: "Live Server" y "ES7+ React/Redux/React-Native snippets". Para Python: "Python" y "Pylance".

Reinicia VS Code para aplicar los cambios.

Parte 3: Creación de un componente simple

Elige uno de los siguientes ejercicios y sigue los pasos en Visual Studio Code.

Opción 1 (JavaScript - React): Crear un componente de botón

Crea una nueva carpeta y ábrela en VS Code.

Dentro de la carpeta, crea un archivo llamado Boton.js y copia el siguiente código:

```
const Boton = ({ texto }) => {
    return <button style={{ padding: "10px", fontSize: "16px" }}>{texto}</button>;
};

export default Boton;
```

En otro archivo App.js, importa y usa el componente:

```
import Boton from "./Boton";

function App() {
    return (
        <div>
            <h1>Mi Aplicación con Componentes</h1>
            <Boton texto="Haz clic aquí" />
```

```
        </div>
    );
}

export default App;
```

Ejecuta el proyecto con npm start (si tienes un entorno React configurado).

Opción 2 (Python - Uso de Clases para Componentes)

Crea una nueva carpeta y ábrela en VS Code.

Dentro de la carpeta, crea un archivo llamado componente.py y copia el siguiente código:

```
class Componente:
    def __init__(self, nombre):
        self.nombre = nombre

    def mostrar(self):
        print(f"Este es un componente llamado {self.nombre}")

# Creación de un componente
mi_componente = Componente("Botón de Inicio")
mi_componente.mostrar()
```

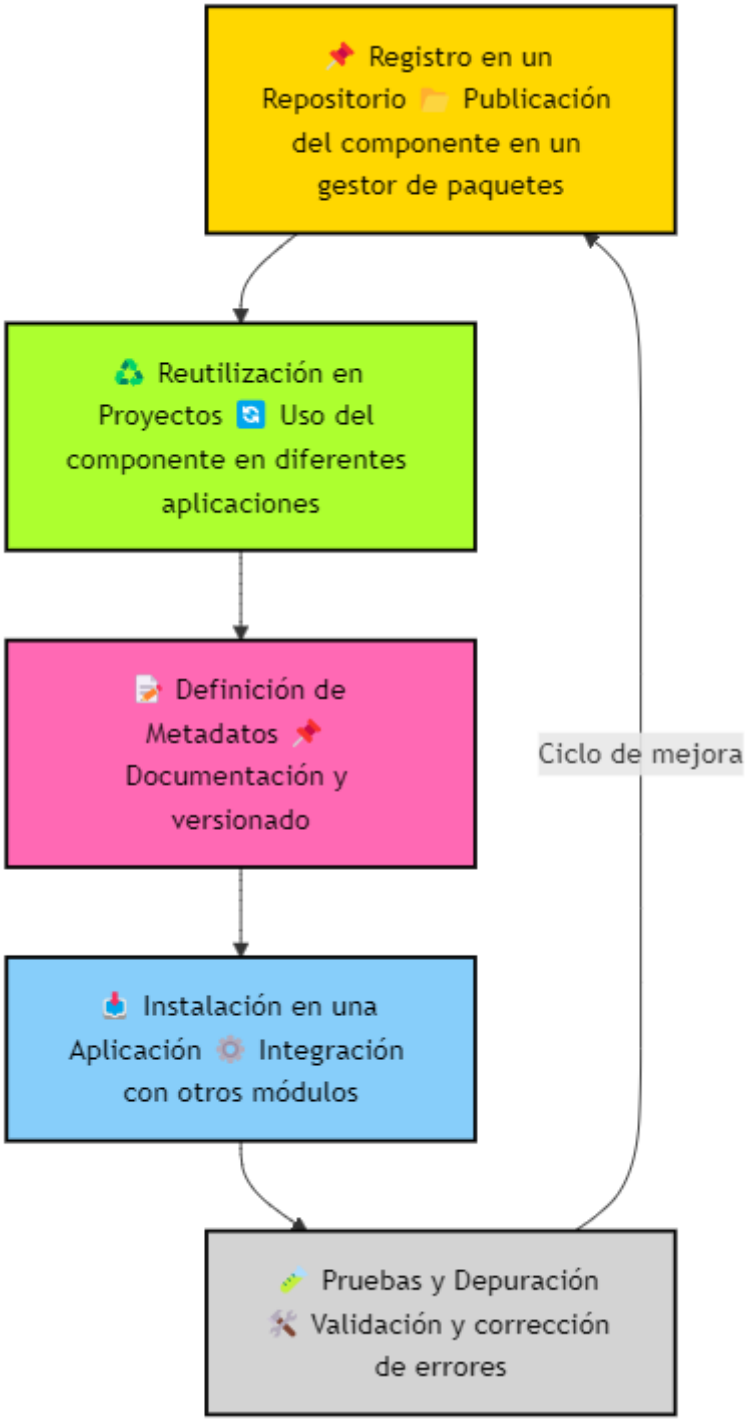
Guarda los cambios y ejecuta el script presionando F5 o ejecutando en la terminal:

```
python componente.py
```

7.3. Gestión del ciclo de vida en el desarrollo de componentes mediante herramientas de uso común.

Cuando desarrollamos software basado en componentes, no basta con escribir código y esperar que funcione. Cada componente tiene un **ciclo de vida**, desde que se diseña hasta que se pone en producción, se actualiza y se mantiene a lo largo del tiempo.

Esquema del ciclo de vida de un componente de software:



El primer paso es **registrar el componente en un repositorio**. Esto significa publicarlo en un gestor de paquetes como NPM (para JavaScript), Maven (para Java) o NuGet (para .NET). Esto permite que otros desarrolladores puedan acceder a él fácilmente e integrarlo en sus proyectos.

Una vez publicado, el componente **puede ser reutilizado en diferentes proyectos**. Es decir, otros desarrolladores pueden descargarlo e incorporarlo en sus aplicaciones sin necesidad de volver a escribir código desde cero. Esto ahorra tiempo y esfuerzo, además de garantizar consistencia en el desarrollo de software.

Para que un componente sea realmente útil, necesita **una buena documentación y un sistema de versionado**. Esto significa definir los metadatos, como el nombre, la descripción, las dependencias necesarias y la forma correcta de usarlo. También es importante que tenga un sistema de versiones para que los usuarios puedan saber qué cambios se han hecho y si hay nuevas actualizaciones.

Cuando un componente se va a utilizar en una aplicación, debe ser **instalado e integrado con otros módulos**. Esto puede hacerse manualmente o de forma automatizada con herramientas de gestión de dependencias. Aquí es donde el componente empieza a desempeñar su función dentro de un software más grande.

Antes de considerarlo finalizado, el componente pasa por un proceso de **pruebas y depuración**. Esto significa validar que funciona correctamente en distintos entornos y corregir cualquier error que pueda aparecer. Si se encuentran fallos o se identifican mejoras, se vuelve a actualizar y el ciclo comienza de nuevo, asegurando así su calidad y evolución constante.

Si este proceso no se gestiona bien, pueden aparecer problemas como errores difíciles de rastrear, incompatibilidades con otros módulos o dificultades para reutilizar el código. Para evitar estos problemas, se utilizan herramientas especializadas que permiten **organizar, probar, instalar y mantener los componentes** de manera eficiente. Vamos a ver algunos aspectos clave en la gestión del ciclo de vida de los componentes y cómo se utilizan en la práctica.

7.3.1. Uso de repositorios de componentes. Registro de componentes.

Uno de los pilares del desarrollo basado en componentes es la reutilización. Pero para que esto funcione bien, los componentes deben estar organizados en un lugar donde los desarrolladores puedan acceder a ellos fácilmente. Para esto existen los **repositorios de componentes**, que son plataformas donde se almacenan y gestionan los módulos reutilizables.

Por ejemplo, en **Java** se utilizan repositorios como **Maven Central o JFrog Artifactory**, en **.NET** se usa **NuGet**, y en **JavaScript** es muy común encontrar paquetes en **npm**. Estos sistemas permiten que cualquier desarrollador descargue e integre un componente en su aplicación sin necesidad de programarlo desde cero.

Además, en entornos empresariales o proyectos más grandes, suele ser necesario un **registro de componentes interno**, donde se almacenan módulos desarrollados dentro de la empresa y que no deben ser accesibles públicamente. Herramientas como **Sonatype Nexus o JFrog Artifactory** permiten gestionar estos registros y asegurarse de que los desarrolladores usen siempre versiones seguras y actualizadas de los componentes.

7.3.2. Reutilización de componentes para la construcción de sistemas software.

Uno de los mayores beneficios del desarrollo basado en componentes es que no es necesario reinventar la rueda. Si ya existe un módulo que resuelve un problema específico, lo ideal es reutilizarlo en lugar de programarlo otra vez. Esto ahorra tiempo y reduce errores.

Por ejemplo, en un proyecto web, en lugar de programar desde cero un sistema de autenticación, se puede reutilizar un componente que ya implemente OAuth 2.0 o JWT (JSON Web Token). Lo mismo ocurre en interfaces de usuario, donde frameworks como **React y Angular** permiten reutilizar componentes visuales en distintos proyectos sin necesidad de escribir el código otra vez.

Para que la reutilización sea efectiva, los componentes deben estar bien documentados y seguir estándares claros de integración. Aquí es donde entran los **metadatos de los componentes**, que permiten describir cómo deben utilizarse y qué funcionalidades ofrecen.

7.3.3. Definición de metadatos de componente. Descriptores de interfaces.

Los metadatos de un componente son información adicional que describe su funcionalidad, dependencias y configuración. Son como una etiqueta que le dice a los desarrolladores (y a las herramientas de gestión de software) qué hace el componente y cómo debe ser utilizado.

En muchos lenguajes, estos metadatos se incluyen en archivos específicos. Por ejemplo, en **Java con Maven**, se utiliza el archivo pom.xml para definir las dependencias y versiones. En **JavaScript con npm**, se usa package.json, mientras que en **.NET con NuGet**, el archivo de metadatos es un .nuspec.

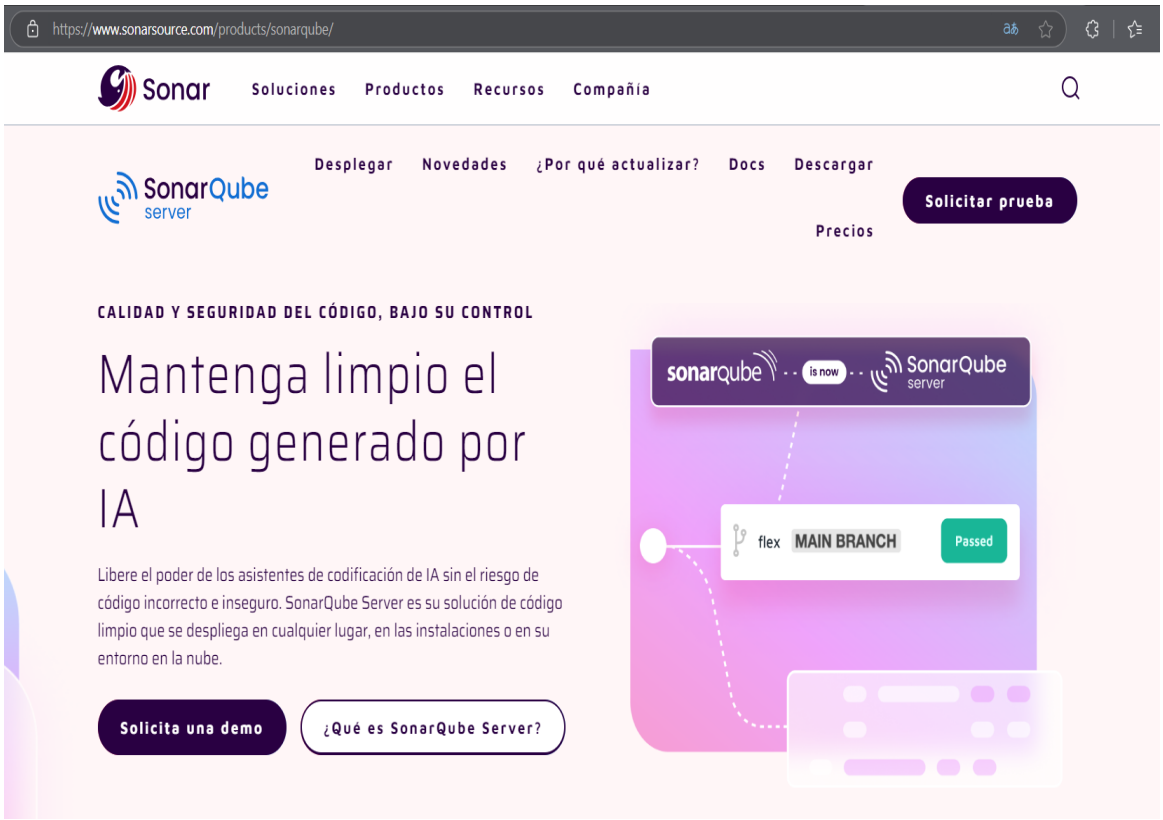
Además, en entornos donde los componentes interactúan con otros servicios, es común definir **descriptores de interfaces**, que especifican cómo deben comunicarse los módulos entre sí. Herramientas como **OpenAPI (para APIs REST) y Protocol Buffers (para gRPC)** permiten describir estas interfaces de forma clara y estandarizada.

7.3.4. Modelo de seguridad

Cuando se reutilizan componentes o se integran módulos de terceros en una aplicación, la seguridad es un aspecto que no se puede ignorar. Un componente mal diseñado o con vulnerabilidades puede poner en riesgo todo el sistema.

Para evitar esto, es importante seguir buenas prácticas de seguridad en el desarrollo e integración de componentes. Algunas de estas prácticas incluyen:

- **Firmas digitales y verificación de integridad**: herramientas como **SLSA (Supply Chain Levels for Software Artifacts)** permiten verificar que un componente no ha sido modificado maliciosamente.
- **Escaneo de vulnerabilidades**: plataformas como **Snyk, Dependabot o SonarQube** analizan los componentes en busca de problemas de seguridad.



- **Autorización y autenticación**: al integrar componentes en una aplicación, es fundamental asegurarse de que solo los usuarios y sistemas autorizados puedan acceder a ellos. Tecnologías como **OAuth 2.0 y JWT** son ampliamente utilizadas para esto.

Un error común en el desarrollo basado en componentes es confiar ciegamente en paquetes externos sin revisar su seguridad. Por eso, muchas empresas implementan **repositorios internos de confianza**, donde solo se almacenan componentes que han sido verificados.

7.3.5. Instalación de componentes.

Para que un componente pueda ser utilizado en una aplicación, debe instalarse correctamente. Dependiendo del lenguaje y el ecosistema en el que se trabaje, el proceso de instalación varía.

Como ya hemos visto, en **Java**, los componentes suelen instalarse a través de **Maven o Gradle**, que descargan automáticamente las dependencias necesarias. En **.NET**, se utiliza **NuGet**, y en **JavaScript**, los paquetes se instalan con **npm o Yarn**.

En entornos más avanzados, donde los componentes forman parte de una infraestructura de microservicios, se utilizan herramientas como **Docker y Kubernetes** para gestionar la instalación y ejecución de los módulos. Esto permite desplegar los componentes en cualquier entorno sin preocuparse por configuraciones específicas de cada servidor.

Un aspecto importante de la instalación de componentes es asegurarse de que siempre se utilicen **versiones compatibles**. Muchas herramientas permiten fijar versiones específicas o actualizar automáticamente a la última versión disponible sin romper compatibilidad con el resto del sistema.

7.3.6. Depuración y prueba de componentes.

Antes de que un componente se integre en una aplicación, debe ser probado a fondo para asegurarse de que funciona correctamente. La depuración y las pruebas son una parte esencial del ciclo de vida del desarrollo de software.

Para detectar errores y corregirlos, los desarrolladores suelen utilizar **depuradores (debuggers)** incluidos en IDEs como IntelliJ, Visual Studio Code o PyCharm. Estas herramientas permiten ejecutar el código paso a paso y analizar su comportamiento en tiempo real.

Además, se implementan distintos tipos de pruebas para validar que los componentes funcionen correctamente:

- **Pruebas unitarias**: verifican que cada componente funcione bien de manera aislada. Frameworks como **JUnit (Java), NUnit (.NET) y Jest (JavaScript)** se usan para esto.
- **Pruebas de integración**: comprueban que los componentes funcionan correctamente cuando se combinan con otros módulos.
- **Pruebas automatizadas**: herramientas como **Selenium (para aplicaciones web) o Postman (para APIs)** permiten automatizar pruebas y asegurarse de que no aparezcan errores en futuras actualizaciones.

Un buen proceso de pruebas evita que los componentes lleguen a producción con errores y ayuda a detectar problemas antes de que afecten a los usuarios.

8. Prueba de autoevaluación.

¿Cuál es una de las principales ventajas del desarrollo basado en componentes?

a) Obliga a reutilizar siempre el mismo código sin modificaciones.

b) Permite dividir la aplicación en módulos independientes y reutilizables.

c) Depende exclusivamente de la programación orientada a objetos.

¿Qué elemento permite que los componentes interactúen sin necesidad de conocer su implementación interna?

a) Las interfaces.

b) Los bucles.

c) Las variables globales.

¿Cuál de estas infraestructuras modernas facilita la gestión de microservicios en aplicaciones basadas en componentes?

a) CORBA y EJB.

b) Kubernetes e Istio.

c) ActiveX y COM.

¿Cómo se consigue la separación entre interfaz e implementación en el desarrollo basado en componentes?

a) Ocultando los detalles internos del componente y definiendo un contrato claro de comunicación.

b) Haciendo que todos los módulos compartan su código fuente para evitar inconsistencias.

c) Usando exclusivamente lenguajes de bajo nivel como ensamblador.

En el ensamblado de componentes, ¿qué técnica permite que los módulos reciban automáticamente sus dependencias sin crearlas manualmente?

a) Inyección de dependencias.

b) Uso de variables globales.

c) Copia directa del código de un módulo a otro.

En el desarrollo basado en componentes, cada módulo debe ser _____ para que pueda reutilizarse en distintos sistemas sin problemas.

La técnica que permite que un componente reciba sus dependencias sin necesidad de crearlas manualmente se llama _____.

Un componente debe poder _____ sin afectar al resto del sistema, lo que facilita su mantenimiento y actualización.

Una de las ventajas de separar la interfaz de la implementación es que los módulos pueden comunicarse a través de _____ sin conocer sus detalles internos.

Spring Boot, .NET Core y WebAssembly Components son ejemplos de tecnologías diseñadas para trabajar con _____.