



- Instructor: Osvaldo Contreras Martinez

Arquitectura cliente servidor.

La arquitectura cliente-servidor es un modelo de diseño en el que los dispositivos se dividen en dos tipos: clientes y servidores.

Los clientes son dispositivos que solicitan información y recursos, mientras que los servidores son dispositivos que proporcionan información y recursos a los clientes.

En este modelo, los clientes envían peticiones a los servidores, y los servidores responden con la información o recursos solicitados.

Este modelo se utiliza ampliamente en aplicaciones de red, incluyendo el correo electrónico, la navegación web y la transferencia de archivos.

Cliente-Servidor

Arquitectura, sirve para desarrollar software, cliente servidor es una distribución de cargas de trabajo, dividido en dos capas, servidor(lógica, basada en un lenguaje) y cliente(lógica basada en un lenguaje).

- Monoliticas → Se ejecuta en un lenguaje de lado del servidor → php, java, phynton, c#(.net), go, perl, ruby, etc
- Servidor → Back end → java, php, phynton, perl, c#, ruby, etc
- Cliente → Front end → JavaScript, TypeScript, jquery, angular, react, veujs

Cliente-Servidor

FullStack → Back + Front

Maven

- Es un repositorio donde podemos descargar librerías y proyectos compeltos.
- Jdbc → Conexión de Oracle
- Dependencias = Librerias

Maven.

Maven es una herramienta de construcción y gestión de dependencias para proyectos Java. Es un sistema de automatización de construcción que se basa en un modelo de proyecto y una estructura de directorios estándar para facilitar la construcción, el despliegue y la documentación de proyectos Java.

Maven proporciona una serie de características útiles como:

- Administración de dependencias: Maven permite definir las dependencias de un proyecto en un archivo de configuración (pom.xml) y automatiza la descarga e integración de las mismas en el proyecto.
- Compilación y empaquetado: Maven proporciona una serie de plugins para automatizar la compilación y el empaquetado del código fuente en un archivo ejecutable como un archivo JAR o WAR.
- Integración con herramientas de pruebas y despliegue: Maven permite integrarse con herramientas de pruebas y despliegue automatizado, como JUnit, Selenium y Jenkins.
- Documentación: Maven proporciona una serie de plugins para generar automáticamente la documentación del código fuente y la documentación del proyecto.
- En resumen, Maven es una herramienta de automatización de construcción y gestión de dependencias que ayuda a los desarrolladores a manejar y organizar las dependencias de un proyecto, automatizar tareas de construcción y documentación, y facilitar la integración con herramientas de pruebas y despliegue.

Para Que Sirve Maven.

Maven sirve para automatizar varias tareas en el desarrollo de proyectos Java.

Webservice

Es una forma de comunicar dos sistemas, esto funciona gracias al protocolo de transferencia de datos http, no importa los lenguajes siempre y cuando trabajen con http puedan ser consumidos unos de los otros, existen dos tipos de WebServices, los soap y los rest.

- Ale – java
- Aristereo – kotlin – java
- Carolina F java – angular
- Carolina G .net -- java

Ejemplo:

Crea una clase de controlador para manejar las solicitudes HTTP:

```
@RestController  
public class HelloController {  
  
    @GetMapping("/hello")  
    public String hello() {  
        return "Hello from Spring Boot!";  
    }  
}
```

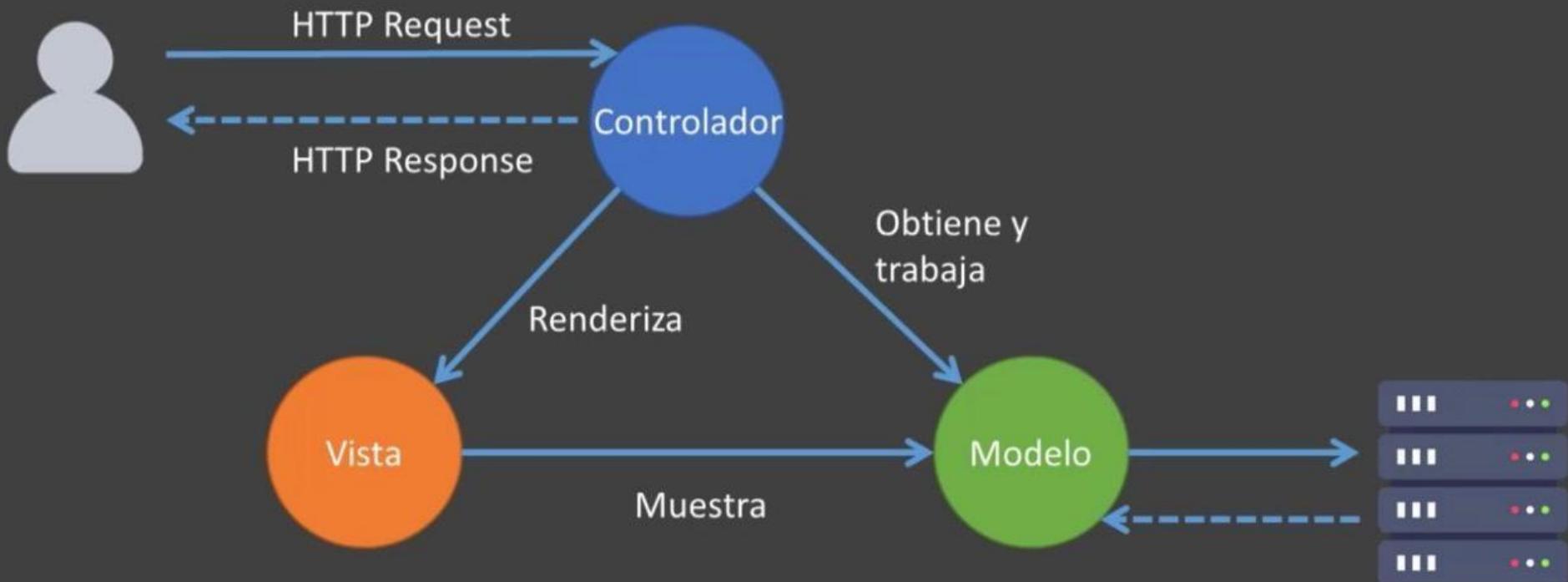
MVC

Arquitectura de software, sirve para desarrollar software, diviendo la programación en capas.

- Modelo– Base de datos
- Controller-- Es la parte lógica de la aplicación
- Vista – Interface Grafica

MVC

Modelo Vista y Controlador



SpringFramework

Es un framework modulado de java, para lo que se necesite existe un modulo de desarrollo.

- Web- SpringWeb
- Mvc- SpringMvc
- BaseDatos-SpringData, SpringJdbc
- Seguridad-Springsecurity
- Documentacion-SpringDoc
- Android- SpringAndroid
- Movil- SpringMovie
- Microservicios-Springcloud
- WebService-SpringRest

Que es Spring Boot.

Spring Boot es un marco de aplicaciones de código abierto basado en el marco Spring para el desarrollo de aplicaciones web. Es una herramienta que te permite crear aplicaciones web de manera rápida y sencilla, ya que te permite configurar automáticamente un gran número de cosas, como la configuración de un servidor web, la inyección de dependencias y la seguridad, entre otras.

Spring Boot se basa en el concepto de "convención sobre configuración", lo que significa que se basa en un conjunto de convenciones predefinidas para configurar automáticamente las cosas, en lugar de requerir que se configure manualmente cada aspecto de la aplicación. Esto hace que sea muy fácil de usar para desarrolladores principiantes y avanzados.

Además, Spring Boot proporciona una gran cantidad de características y herramientas útiles para el desarrollo de aplicaciones web, como soporte para bases de datos, integración con diferentes sistemas de autenticación y autorización, y una interfaz de línea de comandos para facilitar la creación y configuración de proyectos.

Para Que Sirve Spring Boot.

Spring Boot es un marco de aplicaciones de código abierto que se utiliza para el desarrollo de aplicaciones web. Sirve para facilitar el desarrollo de aplicaciones web mediante el uso de convenciones sobre configuración y una gran cantidad de características y herramientas útiles. Algunas de las funciones principales de Spring Boot son:

1. Automatizar la configuración: Spring Boot proporciona una gran cantidad de configuraciones automatizadas para cosas como el servidor web, la inyección de dependencias y la seguridad.
2. Facilitar el desarrollo: Spring Boot utiliza convenciones sobre configuración para facilitar el desarrollo de aplicaciones web, lo que significa que se basa en un conjunto de convenciones predefinidas para configurar automáticamente las cosas, en lugar de requerir que se configure manualmente cada aspecto de la aplicación.
3. Proporcionar una interfaz de línea de comandos: Spring Boot proporciona una interfaz de línea de comandos que facilita la creación y configuración de proyectos.
4. Integración con diferentes tecnologías: Spring Boot proporciona integración con diferentes tecnologías y herramientas como bases de datos, sistemas de autenticación y autorización, entre otras.
5. Facilitar el desarrollo de microservicios: Spring Boot es especialmente útil para el desarrollo de microservicios, ya que proporciona una gran cantidad de características y herramientas para simplificar la creación y configuración de microservicios.

SpringBoot

La configuración de SpringFramework es muy compleja y complicada, springBoot quita toda la configuración de SpringFramework

Diferencias entre Spring Framework y Spring Boot.

- Spring Framework es un marco de desarrollo de aplicaciones Java que proporciona una amplia gama de funcionalidades y herramientas para construir aplicaciones Java. Incluye una variedad de módulos que abarcan diferentes áreas, como seguridad, transacciones, acceso a datos y más.
- Spring Boot, por otro lado, es un marco de trabajo basado en Spring Framework que se enfoca en facilitar la creación de aplicaciones autoconfigurables y de inicio rápido. Proporciona una forma fácil de crear aplicaciones con una configuración mínima, utilizando una amplia gama de funciones y características incluidas en Spring Framework.
- En resumen, Spring Framework es un marco más amplio que ofrece una amplia gama de funcionalidades para construir aplicaciones Java, mientras que Spring Boot es una herramienta de fácil uso basada en Spring Framework que se enfoca en facilitar la creación de aplicaciones con una configuración mínima y un inicio rápido.

Patrones de diseño.

Los patrones de diseño son soluciones estandarizadas para problemas comunes en la programación de software. Son una especie de plantilla o receta que se puede aplicar a un problema específico. Los patrones de diseño se dividen en varios tipos, como patrones de creación, patrones estructurales y patrones de comportamiento.

En Spring Boot, los patrones de diseño son utilizados para resolver problemas comunes en el desarrollo de aplicaciones web, como la inyección de dependencias, la gestión de sesiones y la creación de una arquitectura de capas. Al utilizar patrones de diseño en Spring Boot, puedes aumentar la escalabilidad, el rendimiento y la mantenibilidad de tu aplicación.

Spring Tools.

Spring Tools es un conjunto de herramientas de desarrollo de software desarrolladas por Pivotal, la empresa detrás del marco de aplicaciones Spring. Estas herramientas están diseñadas para facilitar el desarrollo de aplicaciones web basadas en Spring, proporcionando una interfaz visual para configurar y administrar proyectos Spring.

Entre las herramientas que ofrece Spring Tools se encuentran:

Spring Tool Suite (STS): Es un entorno de desarrollo integrado (IDE) basado en Eclipse, especialmente diseñado para desarrolladores de aplicaciones Spring. STS proporciona un conjunto de herramientas de desarrollo, como un depurador, un analizador de código y una interfaz visual para configurar proyectos Spring.

Spring Initializer: Es una herramienta en línea que te permite generar un proyecto Spring con una configuración básica y las dependencias necesarias.

Spring Boot CLI: Es una interfaz de línea de comandos para Spring Boot, que te permite ejecutar y probar aplicaciones Spring Boot desde la línea de comandos.

Spring Cloud CLI: Es una herramienta para ayudar a desarrollar aplicaciones de nube basadas en Spring Boot.

En conjunto, estas herramientas ofrecen una interfaz de desarrollo unificada y fácil de usar para los desarrolladores de aplicaciones Spring, lo que permite ahorrar tiempo y esfuerzo en la configuración y el desarrollo de aplicaciones.

Anotaciones Spring Boot.

Las anotaciones en Spring Boot son etiquetas especiales que se colocan en el código fuente para indicarle al framework cómo debe configurar y utilizar los componentes de la aplicación. Algunas de las anotaciones más comunes en Spring Boot incluyen:

- `@SpringBootApplication`: esta anotación se utiliza para indicar al framework que esta clase es la clase principal de la aplicación, y que debe utilizarla para configurar automáticamente el resto de la aplicación.
- `@Autowired`: esta anotación se utiliza para indicar al framework que debe injectar automáticamente una dependencia en una propiedad o un constructor.
- `@RestController`: esta anotación se utiliza para indicar al framework que esta clase es un controlador REST y debe manejar peticiones HTTP.
- `@RequestMapping`: esta anotación se utiliza para indicar al framework qué ruta debe manejar un método específico en un controlador REST.
- `@Entity`: esta anotación se utiliza para indicar al framework que esta clase representa una entidad en una base de datos y debe ser mapeada a una tabla.
- `@Service`: esta anotación se utiliza para indicar al framework que esta clase es un servicio y debe ser injectada automáticamente en otras clases.

Existen muchas otras anotaciones en Spring Boot, cada una de ellas tiene una función específica. Estas anotaciones te permiten declarar y configurar automáticamente los componentes de tu aplicación, lo que hace que el desarrollo sea más fácil y rápido.

Anotaciones Básicas de Spring Boot.

Spring Boot proporciona una serie de anotaciones para ayudar en el desarrollo de aplicaciones web. Algunas de las anotaciones básicas en Spring Boot son:

`@SpringBootApplication`: esta anotación es una combinación de varias anotaciones, incluyendo `@Configuration`, `@EnableAutoConfiguration`, y `@ComponentScan`. Se utiliza para indicar al framework que esta clase es la clase principal de la aplicación y que debe utilizarla para configurar automáticamente el resto de la aplicación.

`@RestController`: esta anotación se utiliza para indicar al framework que esta clase es un controlador REST y debe manejar peticiones HTTP.

`@Autowired`: esta anotación se utiliza para indicar al framework que debe injectar automáticamente una dependencia en una propiedad o un constructor.

`@RequestMapping`: esta anotación se utiliza para indicar al framework qué ruta debe manejar un método específico en un controlador REST.

`@Value`: esta anotación se utiliza para indicar al framework que debe injectar un valor de una propiedad de configuración en una propiedad de la clase.

`@Service`: esta anotación se utiliza para indicar al framework que esta clase es un servicio y debe ser injectada automáticamente en otras clases.

`@Repository`: esta anotación se utiliza para indicar al framework que esta clase es un repositorio y debe ser injectada automáticamente en otras clases.

`@Bean`: esta anotación se utiliza para indicar al framework que debe registrar un bean específico en el contenedor de beans.

Existen muchas otras anotaciones en Spring Boot, cada una de ellas tiene una función específica, pero estas son algunas de las anotaciones básicas que se utilizan con mayor frecuencia en el desarrollo de aplicaciones web.

CrudRepository.

CRUDRepository es una interfaz en Spring Data que proporciona un conjunto básico de operaciones CRUD (crear, leer, actualizar y eliminar) para una entidad específica. Es una interfaz genérica que proporciona implementaciones básicas para las operaciones CRUD más comunes, como guardar, actualizar, eliminar y buscar entidades en un repositorio.

CRUDRepository se basa en la interfaz Repository de Spring Data, que proporciona un conjunto básico de operaciones de persistencia para entidades. CRUDRepository se extiende de Repository para proporcionar un conjunto específico de operaciones CRUD.

CRUDRepository proporciona los siguientes métodos:

- `save(S entity)`: guarda una entidad en el repositorio
- `findById(ID id)`: busca una entidad por su identificador
- `findAll()`: busca todas las entidades en el repositorio
- `delete(T entity)`: elimina una entidad del repositorio
- `deleteById(ID id)`: elimina una entidad por su identificador.

CRUDRepository se utiliza como una interfaz para crear un repositorio personalizado para una entidad específica.

En resumen, CRUDRepository es una interfaz en Spring Data que proporciona un conjunto básico de operaciones CRUD para una entidad específica, se basa en la interfaz Repository de Spring Data y se utiliza como una interfaz para crear un repositorio personalizado.

Anotaciones de CrudRepository.

CRUDRepository es una interfaz en Spring Data, no una anotación, no tiene anotaciones asociadas. Sin embargo, para utilizar un repositorio basado en la interfaz CRUDRepository en una aplicación Spring, se puede utilizar la anotación `@Repository` para indicar al framework que esta interfaz debe ser tratada como un bean y que las instancias deben ser injectadas automáticamente en otras clases.

La anotación `@Repository` se utiliza para indicar al framework que esta interfaz es un repositorio y debe ser injectada automáticamente en otras clases. Esta anotación no es específica de CRUDRepository, se puede utilizar con cualquier interfaz que extienda de Repository.

Además, para utilizar las funcionalidades de Spring Data JPA, se debe agregar al proyecto la dependencia de Spring Data JPA y configurar una conexión a una base de datos. La anotación `@EnableJpaRepositories` indica a Spring donde buscar los repositorios, y `@EntityScan` indica a Spring donde buscar las entidades JPA.

En resumen, CRUDRepository es una interfaz en Spring Data que proporciona un conjunto básico de operaciones CRUD para una entidad específica, no tiene anotaciones asociadas, pero para utilizar un repositorio basado en esta interfaz se puede utilizar la anotación `@Repository` para indicar al framework que esta interfaz debe ser tratada como un bean y injectada automáticamente en otras clases.

JpaRepository.

JPA Repository es una interfaz proporcionada por Spring Data que proporciona un conjunto de métodos para realizar operaciones CRUD (Crear, Leer, Actualizar y Eliminar) en entidades JPA. Esta interfaz se basa en el estándar JPA (Java Persistence API) y se utiliza para interactuar con una base de datos relacional.

Al extender la interfaz JpaRepository, se heredan los métodos básicos CRUD y también se pueden agregar métodos personalizados según sea necesario. Algunos de los métodos comunes disponibles en JpaRepository incluyen:

- save() para guardar una entidad en la base de datos
- findById() para buscar una entidad por su ID
- findAll() para recuperar todas las entidades de una tabla
- deleteById() para eliminar una entidad por su ID
- count() para contar el número de entidades en una tabla

Además, JpaRepository también proporciona varias formas de especificar consultas personalizadas para recuperar datos de la base de datos, como por ejemplo, mediante el uso de palabras clave específicas en el nombre de los métodos o mediante el uso de @Query anotaciones.

Anotaciones Básicas de JpaRepository.

Algunas de las anotaciones básicas proporcionadas por JpaRepository son:

- **@Repository**: Esta anotación se utiliza para indicar que una clase es un repositorio de Spring Data. La anotación **@Repository** ayuda a Spring a detectar automáticamente las clases de repositorio y a registrarlas como un bean en el contenedor de Spring.
- **@Entity**: Esta anotación se utiliza para indicar que una clase es una entidad JPA. Una entidad es un objeto que se almacena en una tabla de la base de datos y se utiliza para representar una fila de la tabla.
- **@Id**: Esta anotación se utiliza para indicar la propiedad de una entidad que se utiliza como clave principal.
- **@GeneratedValue**: Esta anotación se utiliza para indicar cómo se genera el valor de la clave principal. Por ejemplo, puede utilizar esta anotación para indicar que el valor de la clave principal se genera automáticamente por la base de datos.
- **@Column**: Esta anotación se utiliza para indicar que una propiedad de una entidad se almacena en una columna específica de una tabla.
- **@ManyToOne** y **@OneToMany**: Estas anotaciones se utilizan para indicar relaciones uno a muchos entre entidades.
- **@Transactional**: Esta anotación se utiliza para indicar que un método o una clase necesita una transacción, es decir, que se realizarán varias operaciones en la base de datos y si algo falla se debe deshacer todas las operaciones realizadas.

Es importante recordar que estas son solo algunas de las anotaciones básicas proporcionadas por JpaRepository y hay muchas otras anotaciones disponibles para manejar diferentes escenarios de almacenamiento de datos.

Diferencia entre JPA y CRUD Repository.

CRUDRepository y JpaRepository son dos interfaces de Spring Data que proporcionan una implementación de repositorio genérica para realizar operaciones básicas de CRUD (crear, leer, actualizar y eliminar) en una base de datos. Sin embargo, hay algunas diferencias entre ellas.

- CRUDRepository es una interfaz más básica que proporciona operaciones CRUD básicas para una entidad dada. Ofrece métodos como save(), findOne(), findAll(), delete(), etc. Es adecuado para una implementación simple de repositorio.
- JpaRepository, por otro lado, es una interfaz más avanzada que extiende a CRUDRepository. Además de proporcionar operaciones CRUD básicas, también proporciona operaciones adicionales específicas de JPA (Java Persistence API) como findAll(Example), findAll(Example, Sort), findOne(Example), etc. JpaRepository es adecuado para una implementación más avanzada de repositorio y se utiliza para trabajar con JPA y bases de datos relacionales.

En resumen, CRUDRepository es una interfaz más básica y simple para operaciones CRUD básicas, mientras que JpaRepository es más avanzado y proporciona operaciones adicionales para trabajar con JPA y bases de datos relacionales.

Path Spring Boot.

En Spring Boot, el término "path" se refiere a la ruta o URL que se utiliza para acceder a un controlador o servicio específico. Cada controlador o servicio puede tener una ruta única asociada a él, que es utilizada para identificarlo y acceder a él a través de una petición HTTP.

```
@RestController  
class AutomovilController {  
  
    @Autowired  
    private Automovil automovil;  
  
    @GetMapping("/encender")  
    public void
```

Para asociar una ruta a un controlador o servicio en Spring Boot, se utilizan las anotaciones de mapeo de rutas, como `@GetMapping`, `@PostMapping`, `@RequestMapping`, etc. Por ejemplo, si queremos asociar la ruta `"/encender"` a un método `encender()` de un controlador `AutomovilController`, podemos utilizar la anotación `@GetMapping("/encender")` en el método:

Http.

HTTP (Hypertext Transfer Protocol) es un protocolo de red utilizado para transmitir información en la web. Es el protocolo que utilizan los navegadores web y los servidores web para comunicarse entre sí.

HTTP define cómo se deben transmitir los datos en la web, incluyendo cómo deben estructurarse las peticiones y las respuestas, y los códigos de estado que indican el resultado de una petición.

Las peticiones HTTP son mensajes enviados desde un cliente (como un navegador web) a un servidor para solicitar un recurso. El mensaje de petición consta de una línea de petición (que especifica el método HTTP, la URI y la versión del protocolo), un encabezado (que proporciona información adicional sobre la petición) y un cuerpo opcional (que proporciona datos adicionales para la petición).

Las respuestas HTTP son mensajes enviados desde un servidor a un cliente después de recibir una petición. El mensaje de respuesta consta de una línea de estado (que indica el resultado de la petición), un encabezado (que proporciona información adicional sobre la respuesta) y un cuerpo (que proporciona el contenido de la respuesta).

HTTP es un protocolo estandarizado y ampliamente utilizado, es importante conocerlo para poder desarrollar aplicaciones web, servicios web y aplicaciones móviles.

Métodos HTTP.

El protocolo HTTP define múltiples métodos listados a continuación:

GET: Es utilizado para obtener información del servidor

HEAD: Es igual que GET pero solo devuelve el status y los headers.

POST: Es utilizado para enviar información al servidor

PUT: Reemplaza el contenido del recurso con el nuevo

PATCH: Aplica modificaciones parciales al recurso especificado

DELETE: Borra la información del recurso especificado

TRACE: Es utilizado para debugging enviando un echo devuelta al usuario

OPTIONS: Describe las opciones de comunicación con el servidor

CONNECT: Establece un túnel con el servidor basado en la URL

URI.

URI (Uniform Resource Identifier) Es el nombre que se utiliza para identificar un recurso de forma única. Contiene 3 partes :

- Schema: Protocolo a utilizar para acceder al recurso
- Host: Ubicación del servidor (DNS, host name o IP)
- Path: Ruta al recurso en el servidor

Petición HTTP.

Una petición HTTP se conforma de lo siguiente:

- Método HTTP: Método HTTP, ejemplo GET
- URI: URI del recurso solicitado, ejemplo:
`http://www.twitter.com/search?q`
- Versión: Versión del protocolo utilizado, ejemplo 1.1
- Headers: Información adicional de la petición, ejemplo `Accept : text/html` .
- Body: Cuerpo de la petición HTTP

Status HTTP.

Un status HTTP es un número que representa el resultado de una petición.
Se clasifica en rangos :

Peticiones exitosas.

A continuación se presentan los status HTTP que se utilizan para representar peticiones exitosas de acuerdo a cada método HTTP:

-200 OK: La petición fue exitosa. Se puede aplicar a métodos como GET, HEAD, PUT y TRACE

-201 CREATED: La petición fue exitosa y creó un nuevo recurso. Se genera en métodos POST.

-204 NOT CONTENT: No hay contenido que devolver. Se utiliza como resultado de un método DELETE.

Errores del lado del cliente.

Algunos errores comunes del lado del cliente:

- 400 BAD REQUEST: Error genérico que indica que existe un error del lado del cliente
- 401 UNAUTHORIZED: Cuando el cliente no tiene suficientes privilegios para acceder a un recurso
- 404 NOT FOUND: El recurso solicitado no existe
- 405 METHOD NOT ALLOWED: El método HTTP no está soportado en el recurso solicitado
- 429 TOO MANY REQUESTS: Demasiadas peticiones . Se utiliza cuando se tiene una cantidad límite de peticiones por cliente.
- 409 CONFLICT: Existe un conflicto

Errores del lado del servidor

Algunos errores del lado del servidor :

- 500 INTERNAL SERVER ERROR: Error genérico que significa que existe un error del lado del servidor
- 501 NOT IMPLEMENTED: Indica que el servidor no soporta la funcionalidad solicitada
- 502 BAD GATEWAY: Error que indica que el servidor recibió un error al invocar a otro servidor
- 503 SERVICE UN AVAILABLE: El servidor no puede procesar la petición.

REST.

REST (Representational State Transfer) es un estilo de arquitectura de software para diseñar servicios web. Es un conjunto de principios y reglas para construir servicios web escalables, modulares y fácilmente evolutivos.

Los servicios web RESTful utilizan el protocolo HTTP para interactuar con los clientes y se basan en las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) para manipular recursos. Los recursos son representados por URIs (Uniform Resource Identifiers) y se pueden recuperar o modificar mediante peticiones HTTP como GET, POST, PUT y DELETE.

El diseño de un servicio web RESTful se basa en el uso de recursos identificables y la manipulación de estos recursos a través de operaciones estandarizadas. Los recursos son representados en formato JSON o XML y son accedidos a través de una URL única.

Spring Boot es una herramienta popular para desarrollar aplicaciones RESTful basadas en Spring Framework. Proporciona una serie de características para simplificar el desarrollo de servicios web RESTful, como el soporte para la creación de controladores RESTful, la integración con diferentes formatos de mensajes (JSON, XML, etc.) y la capacidad de configurar y personalizar la serialización y deserialización de datos.

Que es REST.

REST es un estilo de arquitectura de software utilizado para diseñar servicios web, su principal objetivo es proporcionar una interfaz estandarizada para acceder y manipular recursos en un sistema. Los recursos son representados mediante URIs (Uniform Resource Identifiers) y se accede a ellos mediante peticiones HTTP estandarizadas como GET, POST, PUT y DELETE.

REST es utilizado en diferentes aplicaciones, tales como:

- Creación de servicios web para aplicaciones móviles y web: Los servicios web RESTful son fáciles de consumir y pueden ser utilizados por cualquier plataforma o lenguaje de programación que soporte HTTP.
- Integración de sistemas: Los servicios web RESTful pueden utilizarse para integrar diferentes sistemas y aplicaciones, permitiendo la interacción entre ellos mediante una interfaz estandarizada.
- Microservicios: REST es una arquitectura de referencia para la creación de microservicios, ya que permite crear servicios pequeños, modulares y fácilmente escalables.
- Compartir recursos: Los servicios web RESTful permiten compartir recursos de un sistema con otros sistemas, permitiendo la colaboración y el intercambio de información.

En resumen REST es un estandar para construir servicios web escalables, modulares y fácilmente evolutivos, su objetivo es proporcionar una interfaz estandarizada para acceder y manipular recursos en un sistema, ya sea desde una aplicación móvil, web o integrando diferentes sistemas.

Anotaciones Básicas de REST.

Algunas de las anotaciones básicas utilizadas para crear servicios web RESTful en Spring Boot son:

- `@RestController`: Esta anotación se utiliza para indicar que una clase es un controlador RESTful. Un controlador es una clase que maneja las peticiones HTTP y proporciona una respuesta.
- `@RequestMapping`: Esta anotación se utiliza para mapear una URL a un método en un controlador. Por ejemplo, puedes utilizar esta anotación para indicar que una petición GET a la URL `/users` debe ser manejada por el método `getUsers()` en un controlador.
- `@GetMapping`, `@PostMapping`, `@PutMapping` y `@DeleteMapping`: Estas anotaciones son una forma abreviada de `@RequestMapping` que se utilizan para especificar el tipo de petición HTTP que debe ser manejada por un método.
- `@PathVariable`: Esta anotación se utiliza para indicar que un parámetro en un método debe ser recuperado de la URL. Por ejemplo, puedes utilizar esta anotación para indicar que una petición GET a la URL `/users/{id}` debe recuperar el valor de `{id}` como un parámetro en el método.
- `@RequestBody`: Esta anotación se utiliza para indicar que el cuerpo de una petición HTTP debe ser deserializado como un objeto Java.
- `@ResponseBody`: Esta anotación se utiliza para indicar que el objeto devuelto por un método debe ser serializado como el cuerpo de una respuesta HTTP.
- `@ResponseStatus`: Esta anotación se utiliza para indicar el estatus de la respuesta HTTP.

Estas son algunas de las anotaciones básicas utilizadas para crear servicios web RESTful en Spring Boot, pero hay muchas otras anotaciones disponibles para manejar diferentes escenarios de comunicación HTTP.

Swat.

SWAT (Spring Web Application Test) es un marco de pruebas de aplicaciones web para aplicaciones Spring. Es una extensión de Spring MVC Test y proporciona una interfaz para probar controladores Spring y otras clases que formen parte del modelo-vista-controlador (MVC) en una aplicación web Spring.

Con SWAT, puedes simular peticiones HTTP y verificar las respuestas generadas por los controladores. También te permite configurar automáticamente el contexto de la aplicación y proporciona una serie de utilidades para ayudarte a escribir pruebas de manera sencilla.

- Algunas de las características de SWAT son:
- Capacidad para simular peticiones HTTP y verificar las respuestas generadas por los controladores.
- Permite configurar automáticamente el contexto de la aplicación antes de cada prueba.
- Proporciona una serie de utilidades para ayudarte a escribir pruebas de manera sencilla.
- Es compatible con Spring Boot y se integra bien con otras herramientas de pruebas de Java.

En resumen, SWAT es una herramienta muy útil para probar aplicaciones web basadas en Spring, ya que te permite simular peticiones HTTP y verificar las respuestas generadas por los controladores, además de proporcionar una serie de utilidades para ayudarte a escribir pruebas de manera sencilla.

Soa.

SOA (Service-Oriented Architecture) es un enfoque para la construcción de aplicaciones que se basa en la utilización de servicios independientes y reutilizables que se comunican entre sí mediante protocolos estandarizados. Los servicios son componentes lógicos que ofrecen una funcionalidad específica y que son expuestos a través de una interfaz de programación de aplicaciones (API).

La idea detrás de SOA es que los servicios pueden ser utilizados por diferentes aplicaciones y sistemas, lo que permite una mayor reutilización del código y una mayor flexibilidad en la arquitectura de la aplicación. Además, SOA permite una mayor escalabilidad y facilidad de mantenimiento, ya que los servicios pueden ser desarrollados, modificados y escalados de forma independiente.

El uso de protocolos estandarizados como HTTP, SOAP, REST, etc. permite que los servicios sean accedidos de forma homogénea independientemente de la tecnología subyacente.

Diferencia entre REST y Soa.

REST (Representational State Transfer) es un estilo de arquitectura para la construcción de servicios web, mientras que SOA (Service-Oriented Architecture) es un enfoque para la construcción de aplicaciones.

Aunque ambos enfoques tienen algunas similitudes, también hay algunas diferencias fundamentales entre ellos:

- REST es un estilo de arquitectura que se basa en el uso de protocolos web estandarizados como HTTP, URI y XML/JSON para comunicar entre sistemas. Por otro lado, SOA se refiere a la arquitectura de una aplicación y su estructura, y puede utilizar diferentes protocolos y tecnologías para comunicarse.
- REST es más simple y ligero que SOA, ya que se basa en el uso de protocolos web estandarizados y no requiere una infraestructura compleja para su funcionamiento. SOA, por otro lado, requiere una infraestructura más compleja para el despliegue y gestión de servicios.
- REST se enfoca en la representación de recursos a través de URI y la manipulación de estos recursos mediante métodos HTTP. SOA se enfoca en la construcción de servicios independientes y reutilizables que se comunican entre sí mediante protocolos estandarizados.
- REST es una arquitectura y protocolo para el desarrollo de servicios web, mientras que SOA es una arquitectura para el desarrollo de aplicaciones.

En resumen, REST es un estilo de arquitectura específico para servicios web, y SOA es un enfoque más general para la construcción de aplicaciones, y REST puede ser usado en una arquitectura SOA.

Inyección de Dependencias Spring Boot.

Spring Boot es un marco de desarrollo basado en Spring Framework que proporciona una forma fácil y sencilla de utilizar la inyección de dependencias mediante el uso de anotaciones como `@Autowired` y `@Service`.

La anotación `@Autowired` se utiliza para indicar a Spring que un atributo o método debe ser injectado automáticamente con una instancia del objeto correspondiente. Por ejemplo, si tenemos una clase `Automovil` que tiene un atributo `motor` de tipo `Motor`, podemos utilizar la anotación `@Autowired` en el atributo `motor` para indicar a Spring que debe injectar automáticamente una instancia de `Motor` en ese atributo.

La anotación `@Service` se utiliza para indicar a Spring que una clase es un servicio y, por lo tanto, debe ser gestionada por el contenedor de inyección de dependencias. Por ejemplo, si tenemos una clase `Motor` que proporciona un servicio de encendido de automóvil, podemos utilizar la anotación `@Service` en esa clase para indicar a Spring que debe gestionar automáticamente las instancias de esa clase.

Además, Spring Boot proporciona una forma fácil y sencilla de configurar y personalizar el contenedor de inyección de dependencias mediante el uso de anotaciones como `@Configuration` y `@Bean`.

En resumen, Spring Boot proporciona una forma fácil y sencilla de utilizar la inyección de dependencias mediante el uso de anotaciones como `@Autowired` y `@Service`, lo que permite desacoplar los objetos entre sí y mejorar la escalabilidad, flexibilidad y mantenibilidad de la aplicación.

Bean.

En Spring Boot, un bean es un objeto que es gestionado por el contenedor de inyección de dependencias (IoC, por sus siglas en inglés) de Spring. Los beans son objetos que se crean y administran por el contenedor de IoC, y que pueden ser injectados automáticamente en otras clases mediante la inyección de dependencias.

Para definir un bean en Spring Boot, se utiliza la anotación `@Bean`. Por ejemplo, si tenemos una clase `Motor` que proporciona un servicio de encendido de automóvil, podemos utilizar la anotación `@Bean` para indicar a Spring que debe crear un bean de esa clase.

Bean.

Un bean en Spring Boot es un objeto que es gestionado por el contenedor de beans de Spring. Los beans son instancias de clases que se utilizan para representar componentes de la aplicación, como controladores, servicios y repositorios.

Los beans se definen mediante clases Java y se configuran en un archivo de configuración o mediante anotaciones. Una vez definidos, el contenedor de beans se encarga de crear y administrar las instancias de los beans, y de inyectar automáticamente las dependencias necesarias en cada bean.

Los beans pueden ser configurados mediante anotaciones como `@Component`, `@Service`, `@Repository` y `@Controller`, o mediante archivos de configuración XML o Java. Spring Boot utiliza por defecto la configuración mediante anotaciones, pero también es posible utilizar configuraciones alternativas.

En resumen, los beans son objetos que son gestionados por el contenedor de beans de Spring, se utilizan para representar componentes de la aplicación y son configurados mediante anotaciones o archivos de configuración, el contenedor se encarga de administrar las instancias y las dependencias necesarias en cada bean.

Ejemplo de Inyección de Dependencias Spring Boot.

Un ejemplo de cómo utilizar la inyección de dependencias en Spring Boot podría ser el siguiente:

Tenemos una clase llamada Automóvil que tiene un atributo motor de tipo Motor, y un método encender() que utiliza el motor para encender el automóvil.

```
class Automovil {  
    @Autowired  
    private Motor motor;  
  
    public void encender() {  
        motor.encender();  
    }  
}
```

En este ejemplo, utilizamos la anotación `@Autowired` en el atributo `motor` para indicar a Spring que este atributo debe ser injectado automáticamente al crear una instancia de la clase Automóvil.

También tenemos una clase Motor

```
@Service  
class Motor {  
  
    public void encender(){  
        // código de encendido  
    }  
}
```

En este ejemplo, utilizamos la anotación @Service para indicar a Spring que esta clase es un servicio y por lo tanto debe ser gestionado por el contenedor de inyección de dependencias.

Ahora, si queremos crear una instancia de Automovil podemos hacerlo mediante el uso de un controlador, por ejemplo:

```
@RestController  
class AutomovilController {  
    @Autowired  
    private Automovil automovil;  
  
    @GetMapping("/encender")  
    public void encender() {  
        automovil.encender();  
    }  
}
```

En este ejemplo, el controlador AutomovilController también tiene una dependencia de Automovil, la cual es inyectada automáticamente por Spring. Al hacer una petición GET a la ruta '/encender' el método encender() del servicio Automovil será invocado.

En resumen, Spring Boot proporciona una forma fácil y sencilla de utilizar la inyección de dependencias mediante el uso de anotaciones como @Autowired y @Service, lo que permite desacoplar los objetos entre sí y mejorar la escalabilidad, flexibilidad y mantenibilidad de la aplicación.

Una vez que se ha definido un bean, puede ser injectado automáticamente en otras clases mediante la anotación @Autowired

```
class Automovil {  
    @Autowired  
    private Motor motor;  
  
    public void encender() {  
        motor.encender();  
    }  
}
```

En resumen, un bean en Spring Boot es un objeto que es gestionado por el contenedor de inyección de dependencias y puede ser injectado automáticamente en otras clases mediante la anotación @Autowired. Los beans se definen mediante la anotación @Bean y se pueden configurar y personalizar mediante clases anotadas con @Configuration.

```
@Configuration  
public class BeanConfig {  
  
    @Bean  
    public Motor motor() {  
        return new Motor();  
    }  
}
```

En este ejemplo, la clase BeanConfig es anotada con `@Configuration` y contiene un método `motor()` que es anotado con `@Bean`, indicando a Spring que debe crear un bean de tipo `Motor` al momento de crear una instancia de esta clase.