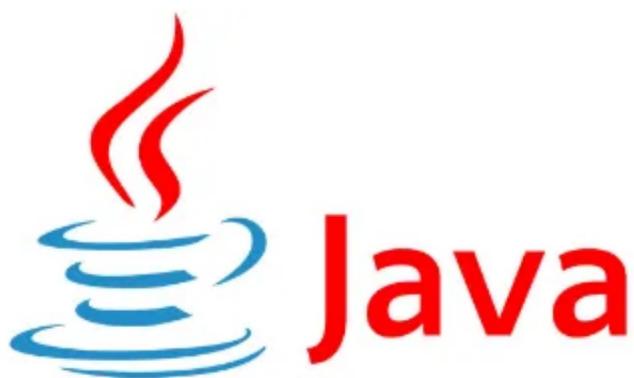




# Ruta para backend developer (Java)

Hugojose14 · [Follow](#)

12 min read · 1 day ago



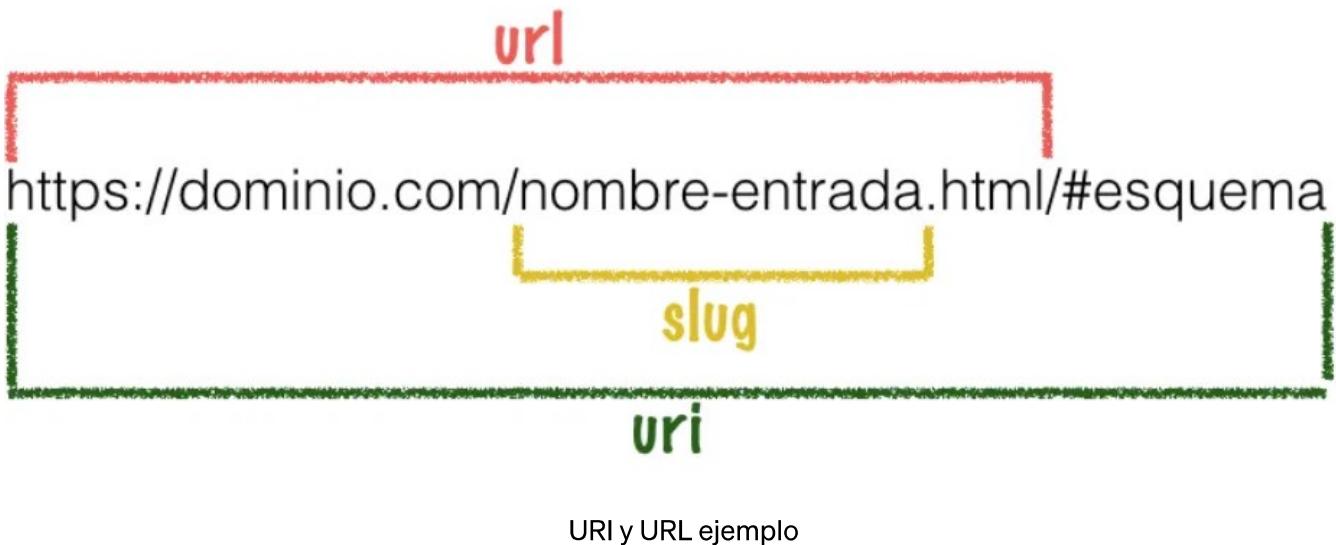
- **Java:** Es necesario tener conocimientos básicos de Java (Declaración de variables, tipos de datos, bucles, métodos, clases, objetos, wrappers, etc)
- **IDEA:** Para estudiar nuestro lenguaje favorito o utilizar nuestro framework, entonces queramos compilar nuestras aplicaciones realizadas en Java o Spring boot si o si necesitaremos de un IDE para trabajar en ellas y hacer esas compilaciones, entonces mi mayor sugerencia es aprender a utilizar IntelliJ IDEA, también cabe mencionar que existen otras alternativas como: Spring Tools Suite, Eclipse, entre otras, pero la más completa creo que a mí gusto es IntelliJ.

- **Server:** Una computadora a la que nos conectamos y que nos sirve para subir nuestra aplicación web, etc... Pero debemos contratar un host (que es algo donde podemos alojar nuestra aplicación)
- **Estructura básica de una URL:**



- **Próntocolo:** Define el método por el cual accedemos al recurso, en este caso tenemos dos opciones http y https (nos dice que va a utilizar un certificado ssl para enviar y recibir datos).
- **Simbolo #:** Es el acceso a un recurso en específico, puede ser posts, users, etc, entonces luego de que veamos una almohadilla “#” significa que es un recurso en específico al cual queremos acceder.
- **Recurso:** Puede ser un archivo, video, imagen, un html, etc...
- **URI:** Es la suma de todo el link (1-7) , pero para que quede más claro es la suma de muchos caracteres que nos permiten localizar un recurso en internet el cual debe ser único sin importar donde este se encuentre.
- **URL:** Abarca toda la dirección sin incluir el recurso (lo que continua

luego de la almohadilla “#”)



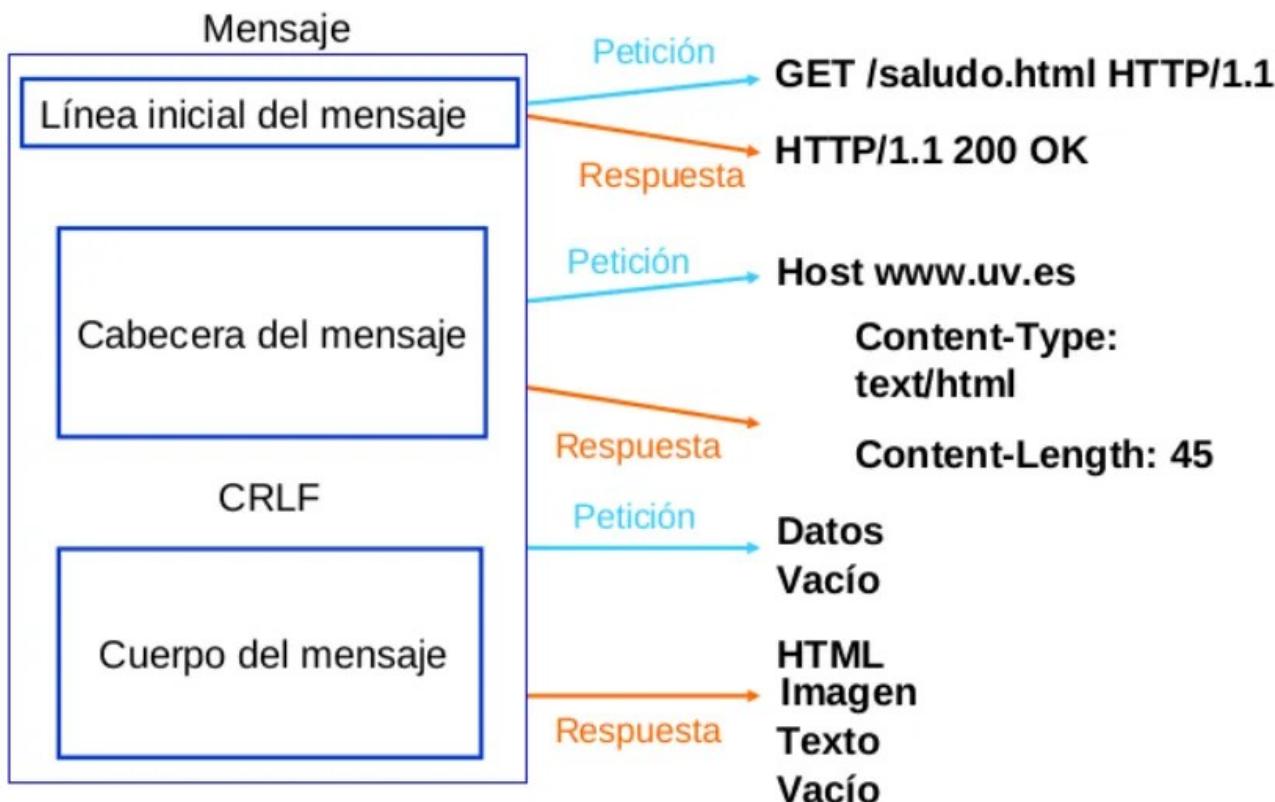
- Link avanzado:



- **Host:** Nombre de la pagina web
- **Query (params):** Consulta que quiero realizar clave1=valor, si queremos buscar por un usuario en específico entonces enviamos la clave y el valor de ese usuario, el simbolo "?" nos permite indicarle que será una busqueda por parámetros y el "&" nos va a permitir concatenar los parámetros. Ejemplo: quiero buscar el usuario con identificacion 1001896753 y tipo de identificación Cédula de Ciudadanía... Sería de la de la siguiente forma  
<http://wwwxxxxxxxx:80/user?id=10&type=CC>, entonces devolvemos el usuario con esa cédula y tipo de cédula
- **Puerto:** 80 ser el puerto donde estará mi aplicación, pero lo normal es

que cuando entramos a un sitio como google.com vemos que no hay puerto, pero el por defecto siempre tiene el puerto 80

- **usuario:contraseña:** Credenciales del usuario que está intentando entrar a cierto recurso.
- **Métodos HTTP:** Es un método de petición para decirle a X recurso que tipo de acción requiero. También cabe mencionar que el protocolo http o https son un conjunto o reglas que nos permiten acceder a un recurso en específico.
- **Próptocolo HTTP:** Son reglas que nos permiten acceder a cierto recurso.



Protocolo HTTP

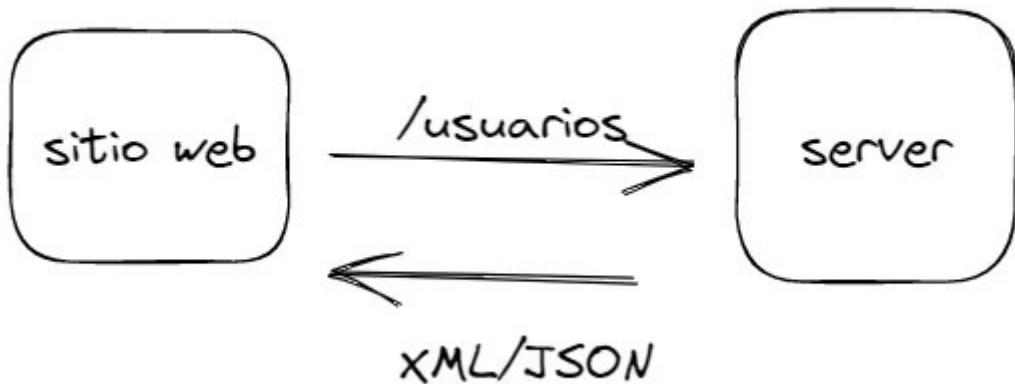
- **Códigos de estado:** Son códigos que vienen atados a una respuesta del servidor.

- **1xx:** Mensaje informativo.
- **2xx:** Exito
  - 200 OK
  - 201 Created
  - 202 Accepted
  - 204 No Content
- **3xx:** Redirección
  - 300 Multiple Choice
  - 301 Moved Permanently
  - 302 Found
  - 304 Not Modified
- **4xx:** Error del cliente
  - 400 Bad Request
  - 401 Unauthorized
  - 403 Forbidden
  - 404 Not Found
- **5xx:** Error del servidor
  - 500 Internal Server Error
  - 501 Not Implemented
  - 502 Bad Gateway
  - 503 Service Unavailable

Códigos de estado

- **Librería:** Un código creado de tercero para que lo reutilicemos nosotros, por ejemplo (conectarme a una base de datos, cambiar el formato de un vídeo de mp3 a mp4). Tenemos maven y gradle que son dependencias del proyecto también.
- **Maven:** Es una herramienta de gestión de proyectos de desarrollo utilizada principalmente en el entorno de computación Java utilizando conceptos provenientes de Apache Ant, dentro de maven se emplea el lenguaje XML.
- **Gradle:** es una herramienta de automatización de compilación del código abierto, que obtuvo una rápida popularidad ya que fue diseñada fundamentalmente para construir multiproyectos, utilizando conceptos provenientes de Apache Maven, dentro de gradle se utiliza el lenguaje DSL ya que está focalizado en la resolución de un problema en específico.
- **JAR:** Es más enfocado a aplicaciones con solo backend (empaquetado)
- **WAR:** Más enfocado a aplicaciones web (empaquetado)

- **Controller:** Sirve para manejar las url's de nuestras aplicaciones, entonces nos permiten exponer nuestros endpoints, por ejemplo el endpoint de /usuarios que nos extrae todo los usuarios de nuestra aplicación.
- **YAML:** Es un lenguaje de serialización de datos que las personas pueden comprender y suele utilizarse en el diseño de archivos de configuración. Para algunas personas, YAML significa otro lenguaje de marcado más; para otras, es un acrónimo recursivo que quiere decir “YAML no es un lenguaje de marcado”, lo que enfatiza la idea de que se utiliza para los datos, no para los documentos.
- **Json:** Es un formato de intercambio de datos. Es independiente del lenguaje, lo que significa que se puede usar con cualquier lenguaje de programación, y la estructura de datos subyacente es independiente de la plataforma. <https://json.org/example.html>
- **XML:** Es un estándar abierto para almacenar e intercambiar datos. Es un lenguaje de marcado para describir la estructura y el contenido de cualquier XML archivo, como documentos, páginas web o bases de datos. Tu puedes pensar en XML como HTML, pero mejor: le permite adjuntar información adicional a los nodos de su documento sin cambiar el formato subyacente.
- **XML – JSON:** En resumidas sirven para la comunicación entre servicios.



Obtengo los usuarios del servidor y la información me la puede retornar en un json o en un xml

**Base de datos:** Hoy día existen muchas bases de datos y lenguajes, pero iniciar con **Oracle** sería una buena opción, también depende mucho lo que deseas realizar, por lo tanto puedes empezar con Oracle en cuanto a base de datos SQL o relacional, pero cuando es la opción no SQL lo mejor es iniciar con **Mongo DB**, pero como hago enfasis todo dependerá del problema que quieras resolver y de la demanda de información que quieras almacenar, pero a mí parecer estas dos son unas buenas opciones.

**Testing:** Junit es una buena opción para empezar en el mundo del testing con Java y Spring Boot, por lo tanto es mi recomendación, además a la hora de realizar testing te recomiendo leer cosas como TDD y BDD, también hay temas apartes como Mockito, PowerMockito, Mutation test, entre otros.

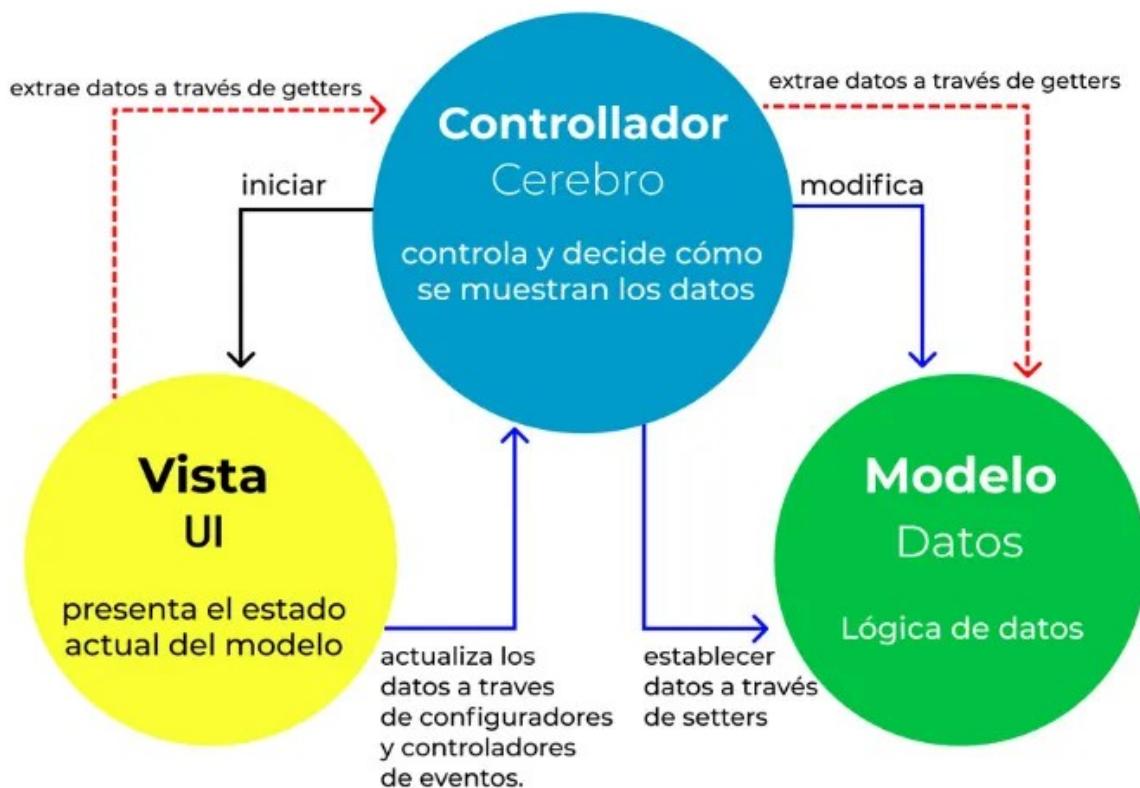
**Framework:** En mi recomendación y base a mi experiencia, hoy día el framework más utilizado de Java es Spring Boot, por lo que mi recomendación es que aprendas a utilizar Spring boot... Por lo tanto un framework no es más que una forma de trabajo organizada que nos brinda un conjunto de herramientas (anotaciones, implementaciones, estructura de paquetes, patrones de diseño, etc) para hacer el desarrollo más fácil y más

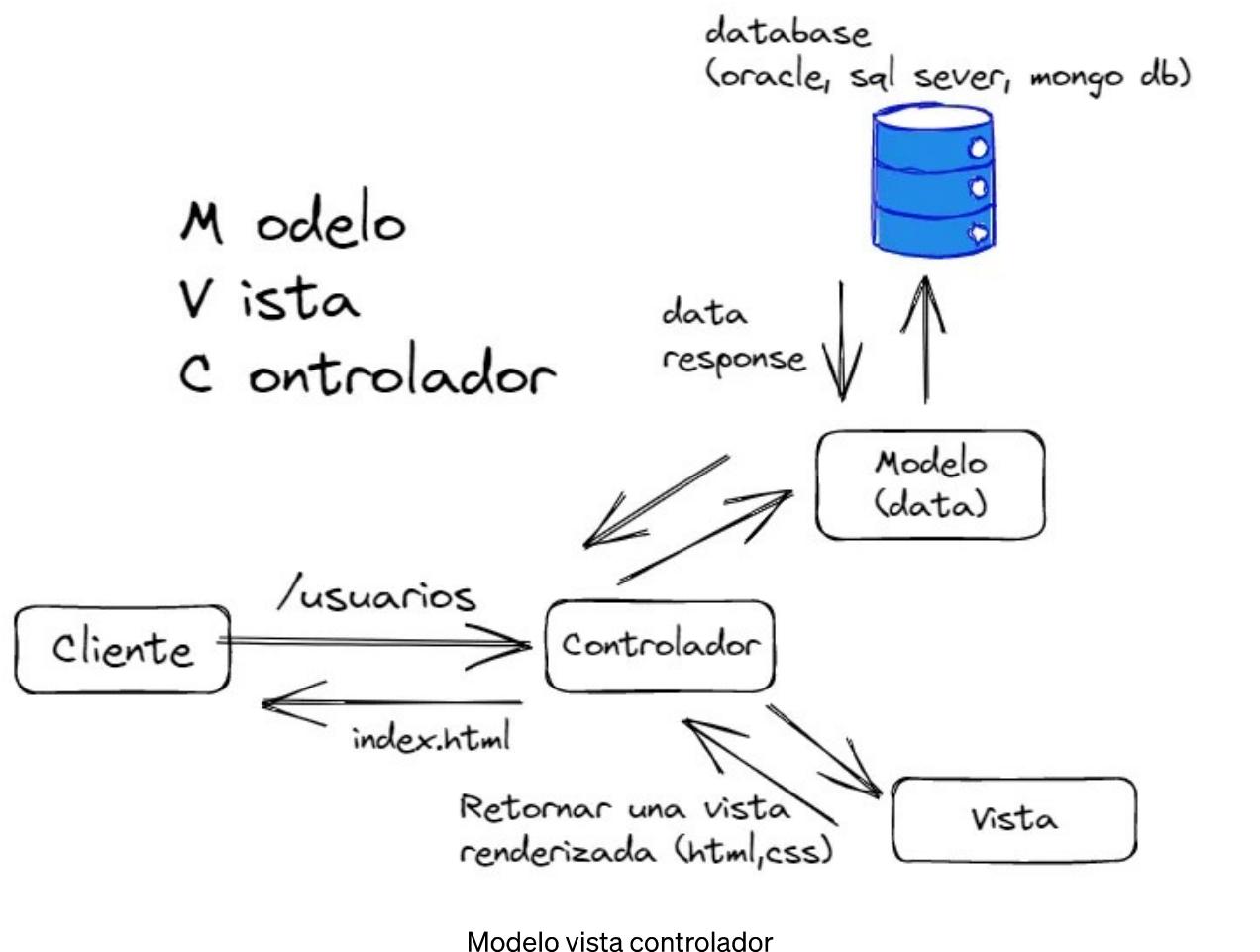
organizado. Es super importante que el framework que elijas tenga una comunidad amplia, porque a la hora de tener dudas o errores ya existen personas que les ha pasado y te será más fácil. La mejor página para aprender sobre spring boot a mí parecer es baeldung, por lo tanto te dejo el link <https://www.baeldung.com/spring-tutorial>.

**Spring security:** Es un framework que nos brinda la autentación en nuestra aplicación.

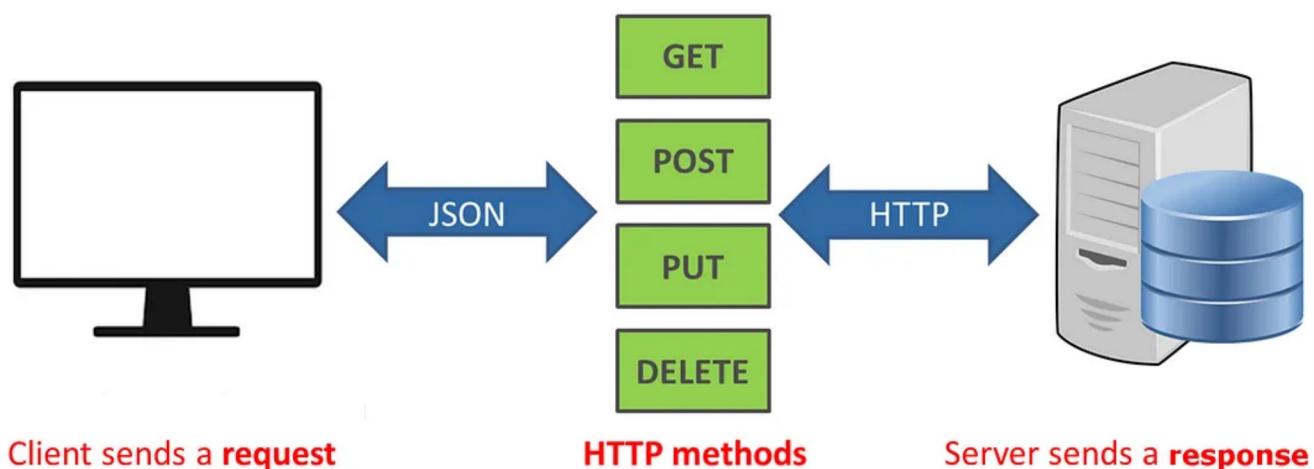
**MVC:** Es un patrón de arquitectura que nos va a servir para darle una mejor organización a nuestro código. Nos devuelve el HTML renderizado.

## Patrones de Arquitectura MVC





**API REST:** Por sus siglas “transferencia de representación de estado” que hace mención a una interfaz que nos permite conectar varios sistemas basándose en el protocolo HTTP.



**DTO:** Data transfer object, simplemente son objetos que nos van a permitir transmitir información.

**DAO:** Por su siglas también como Data Access Object, son la representación de nuestros objetos de base de datos, por ejemplo hacen enfasis a nuestras entitys en spring boot. Ejemplo si tenemos una tabla que se llama usuario en base de datos y queremos traer toda esa información, si o si deberíamos tener una clase que tenga el mismo nombre y todos los campos que requerimos mapear.

**Value Objects:** Los **value objects** (VO) son objetos que se identifican por su contenido y nos ayudan a modelar conceptos de negocio.

**Mapeos:** Llegaremos a un punto donde necesitemos mapear objetos entre capas, por ejemplo algo que nunca deberíamos realizar es retornar en nuestro controlador una entidad (DAO), jamás debería llegar a nuestro controlador, entonces en nuestro código responderemos en la mayoría de casos con DTO, entonces cuando vayamos a base de datos si o si necesitaremos transformar el objeto de base de datos a un DTO que viajará hacia mi controlador. Entonces mi recomendación es que le echen una leída a **mapstruct**. <https://mapstruct.org/> o **ModelMapper** que nos permitirán mapear objetos o JMapper son alternativas, pero lo que más he solidido encontrar en el mundo del software es **mapstruct**, es rápido y sencillo de implementar.

**JPA:** Es Java Persistence Api que es una API de persistencia desarrollada para Java.

**Paginación:** Imaginemos que tenemos 1000 registros en base de datos, entonces devolver esos registros de golpe no es la mejor opción debido a que tendremos una carga muy fuerte, por lo que es mejor devolver cierta

cantidad de elementos por pagina.

1

2

3

12

NEXT >

**Casos de declaración:** A la hora de declarar métodos, clases, variables, constantes, campos de bases de datos, de nuestra API, debemos estar seguro el estandar qué utilizamos, por lo tanto te invito a estudiar los siguientes: **UPPERCASE, lowercase, camelCase, PascalCase, snake\_case, SCREAMING\_SNAKE\_CASE, kebab-case...**

**UPPERCASE**

**lowercase**

**camelCase**

**PascalCase**

**snake\_case**

**SCREAMING\_SNAKE\_CASE**

**kebab-case**

**Principios para escribir código:** KISS, DRY son una buena opción para mantener nuestro código simple y no hacer demás a la hora de escribir nuestro código, por lo tanto un código simple debe ser matenible y fácil de

leer.

**SonarLint:** Te imaginas que alguien nos ayude a analizar nuestro código, a encontrar bugs, código innecesario, clases sin implementar, variables sin usar, entonces SonarLint es la mejor opción.

**Lombok:** Nos va a permitir a través de anotaciones escribir menos código, por ejemplo no necesitaremos colocar getter y setter manuales, si no que colocando anotaciones al inicio de nuestra clase bastará y se generarán automáticamente @Getter @Setter @NoArgsConstructor @Builder

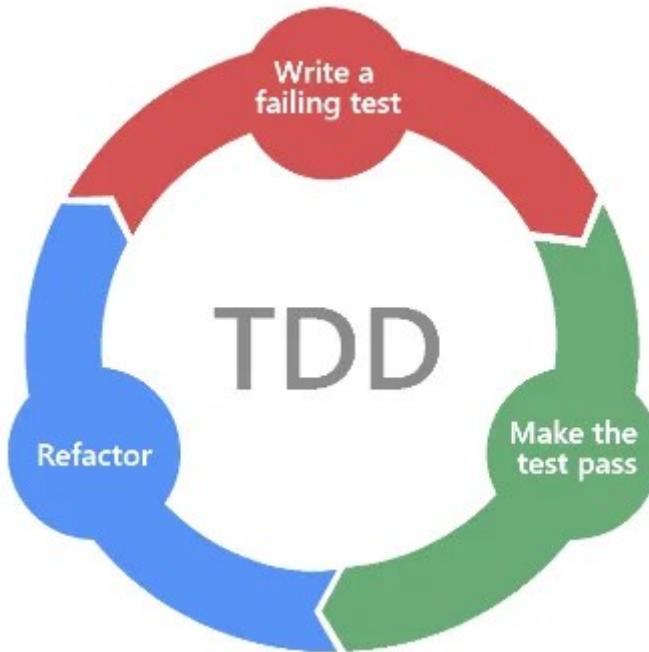
**Builder:** Es una forma de crear objetos sin tener necesidad de instanciarlo, entonces Lombok nos ahorra la vida, solo para utilizarlo necesitamos agregarle la anotación @Builder a nuestras clases y con eso bastaría para crear un objeto, cabe mencionar que para spring boot esto es mejor, debido a que maneja mejor los objetos que no son instanciados, además ayuda la memoria de java, ya que si tenemos un código complejo no vamos a estar creando muchos objetos y se nos va a llenar la memoria de Java.

**Inyección de dependencias:** Nos permite reutilizar código, facilidad de reutilización, enviarle a las clases los objetos necesarios.

**Liquibase:** Cuando se trata de guardar las versiones de los scripts que ejecutamos en base de datos liquibase es la mejor opción, por lo tanto si quieras tener un versionamiento de lo que ejecutas y que no tengas que buscar luego o armar todo es una buena opción, además que si tienes tu base de datos en oracle y quieras migrar a mysql, basta con especificarle el tipo de base de datos que deseas ejecutar y el te generará todos los scripts versionados.

**Postman:** Cuando se trata de probar nuestras aplicaciones, consumir servicios, automatizar las pruebas de nuestras aplicaciones, postman es la mejor opción, por lo tanto te invito a estudiar postman.

**TDD:** También por sus siglas como Test Driven Development, que es desarrollo orientado a las pruebas, por lo tanto siempre que iniciemos un desarrollo nos vamos a basar en las pruebas.



<https://www.hiberus.com/crecemos-contigo/todo-lo-que-necesitas-saber-de-tdd-en-3-minutos/>

**DDD:** De acuerdo con Paradigma Digital, Domain Driven Design (DDD) es una aproximación holística al diseño de software que pone en el centro el Domain, es decir, el dominio o problema de negocio.

**Autenticación:** Nos va a permitir validar quiénes somos para cierto recurso, imaginamos que estamos en un login de cualquier pagina, entonces debemos autenticarnos para que el sistema sepa quiénes somos dentro de esa pagina, entonces autenticación es “quiénes somos para ese recurso”.

**Autorización:** Mientras que autorización son los permisos o privilegios que tenemos dentro de ese sistema, imaginemos que ya estamos autenticados en una web de venta de productos (mercado libre), entonces nosotros solo estamos autorizados para agregar y eliminar productos a nuestro carrito, pero no podemos eliminar usuarios ni actualizarlos, solo nuestro usuario.

**JWT:** Por sus siglas conocidas con Json web token, en resumidas cuentas nos va a permitir transmitir nuestra información de forma segura entre dos sistemas. Imagina que necesitas acceder a un sistema, pero tu contraseña viaja en claro, lo que significa que cualquier atacante puede tomarla y acceder como si fuieras tú, entonces allí es donde nace jwt para transmitir información entre dos sistemas de forma segura. <https://jwt.io/>

**Clean Code:** Escribir nuestro código requiere de buenas prácticas, debido a que será mantenible en el tiempo, además nos permitirá que otros desarrolladores lo lean de forma sencilla, además que si no tocamos un componente hace mucho tiempo nos será fácil de leer y volver a entenderlo, por lo tanto la forma en que escribimos nuestro código es importante para mantener nuestro proyecto. Mi recomendación para tener un buen código es leer el libro de “Clean code” de Robert C. Martin, pero esto no nos garantiza que tengamos un buen código si no ponemos en práctica lo aprendido.

**Patrones de diseño:** Nos van a permitir resolver un problema que es repetitivo, además se dividen en creaciones y estructurales. Te invito a leer el libro de patrones de diseño “**Sumergete en los patrones de diseño**” de *Alexander Shvets*, además hay una web que es súper interesante y es <https://refactoring.guru/> allí puedes encontrar todos los patrones de diseños explicados y con ejemplos, pero además puedes encontrarlos en diferentes lenguajes como: python, java, etc...

**Arquitecturas limpias:** La arquitectura limpia es una filosofía de diseño de software presentada por Robert C. Martin en 2017 en un libro con el mismo nombre <https://nescalro.medium.com/entendiendo-a-la-arquitectura-limpia-7877ad3a0a47>

**Arquitectura Hexagonal:** La Arquitectura Hexagonal propone que nuestro dominio sea el núcleo de las capas y que este no se acople a nada externo. En lugar

*de hacer uso explícito y mediante el principio de inversión de dependencias nos acoplamos a contratos (interfaces o puertos) y no a implementaciones concretas.*

<https://medium.com/@edusalguero/arquitectura-hexagonal-59834bb44b7f>

**¿Dónde puedo aprender lo anterior mencionado?**

**Java:**

<https://www.baeldung.com/java-tutorial>

### **Baeldung - Java and Spring**

In-depth, to-the-point tutorials on Java, Spring, Spring Boot, Security, and REST.

[www.youtube.com](http://www.youtube.com)