**x¿Cuál es la relación entre South MicroSystems y Java?**

South MicroSystems, conocida por su destacada participación en proyectos como Java, OpenOffice, MySQL y Solaris, ha sido un pilar en el mundo del software de código abierto. En 2009, la empresa fue adquirida por Oracle, lo que causó un gran revuelo, especialmente porque Oracle no tenía una buena relación histórica con el mundo del open source. En este contexto, Java ha sido uno de los productos más afectados por estos cambios corporativos.

**¿Qué novedades trajo Java 11 y cuál es su relación con Oracle?**

En 2017, Oracle lanzó la versión Java 11, la cual introdujo un cambio significativo: la modificación de su licencia. A partir de entonces, Oracle anunció que aquellos que quisieran utilizar el JDK (Java Development Kit) para producción o con fines comerciales debían pagar una tarifa. Las tarifas eran de $2.5 dólares mensuales por cada equipo de usuario final o $25 dólares mensuales por cada aplicación instalada en un servidor, y $25 dólares mensuales por procesador.

**¿Cómo afectó el cambio de licencia al uso de Java?**

Antes del cambio, los desarrolladores podían utilizar el JDK de forma gratuita. El nuevo esquema de tarifas implicó que las empresas y desarrolladores tuvieran que considerar el costo de usar Java para sus aplicaciones. Sin embargo, esta nueva política de Oracle no significó el fin del uso libre de Java gracias a las alternativas que proporciona la comunidad open source.

**¿Qué es OpenJDK y cuál es su relevancia en este contexto?**

OpenJDK es un proyecto mantenido también por Oracle, pero bajo una licencia GPL, lo que permite su uso gratuito y libre por parte de la comunidad. Aunque antiguamente se consideraba una versión menos deseable en comparación con el JDK oficial, Oracle trabajó para que OpenJDK alcanzara un nivel técnico similar al JDK, logrando que OpenJDK fuera una alternativa válida para quienes deseaban evitar las tarifas de licencias.

¿Qué empresas utilizan OpenJDK?

Actualmente, OpenJDK es utilizado en multitud de aplicaciones, siendo el ejemplo más destacado Google, que ha basado la implementación de Android en OpenJDK. Además, existen otras versiones de OpenJDK mantenidas por empresas como Amazon y RedHat, las cuales proporcionan soporte a estas implementaciones específicas.

Recomendaciones para los desarrolladores que quieren usar Java sin costo

Conocer OpenJDK: Aprovechar el potencial de OpenJDK y familiarizarse con su estructura y funcionalidades.

Seguir actualizaciones de la comunidad: Mantenerse informado sobre las actualizaciones y mejoras que se realicen dentro del proyecto OpenJDK.

Explorar alternativas corporativas: Analizar ofertas de compañías que trabajan sobre OpenJDK, como Amazon o RedHat, que pueden ofrecer un soporte más sólido dependiendo de las necesidades.

Contribuir al open source: Participar en la comunidad OpenJDK puede ser una excelente oportunidad para mejorar habilidades y contribuir al ecosistema open source, a la vez que se elude el costo de las licencias.

**¿Qué es una aplicación autocontenida y cómo se relaciona con SpringBoot?**

Hoy día, las tendencias en desarrollo de aplicaciones nos llevan hacia modelos más ágiles y modulares, dejando atrás las grandes arquitecturas monolíticas. Antes, las aplicaciones empresariales solían desplegarse en un servidor único que gestionaba configuraciones, bases de datos, y servicios externos. Este enfoque puede tener sus ventajas, pero también complica el despliegue y mantenimiento.

**Ventajas de una arquitectura modular**

Las arquitecturas modernas optan por dividir una gran aplicación en pequeños servicios o microservicios independientes que colaboran entre sí. Entre las ventajas de este enfoque, se destacan:

**Facilidad de desarrollo:** Cada microservicio se desarrolla como una pequeña aplicación autocontenida.

**Mantenimiento sencillo:** Las actualizaciones se llevan a cabo en módulos específicos sin afectar a toda la aplicación.

**Escalabilidad:** Los servicios pueden escalarse de forma independiente según su demanda específica.

**Spring Boot y su función**

Spring Boot surge como la solución ideal para construir aplicaciones autocontenidas. A diferencia de otros proyectos del ecosistema Spring, Spring Boot permite olvidar la configuración complicada del servidor de aplicaciones para enfocarnos en escribir el mejor código posible.

Para lograrlo, Spring Boot incluye un servidor de aplicaciones embebido, siendo Apache Tomcat su opción por defecto, pero también permite usar Jetty o Undertow. Además, Spring Boot ofrece:

**Gestor de dependencias**: Opciones como Maven o Gradle para manejar las dependencias del proyecto.

**Configuraciones automáticas:** Adaptaciones de la aplicación de manera automática según las necesidades del desarrollador.

**Selección de dependencias**

Al elegir las dependencias iniciales, considera:

Spring Web: Essential para crear APIs RESTful usando Apache Tomcat como contenedor.

Otras dependencias: A medida que tu proyecto evolucione, podrás añadir otras necesarias directamente en el archivo build.gradle.

**Archivo build.gradle**

Es el corazón de tu gestión de dependencias. Aquí defines:

**Plugin y versión:** Los plugins que utilizarás y la versión de Java.

**Repositorios**: De dónde se obtendrán las dependencias.

**Dependencias y testing:** Toda dependencia necesaria para el correcto funcionamiento y pruebas de la aplicación.

@RestController y @RequestMapping para definir las rutas y la forma en que el controlador responderá.

Con @RequestMapping("/saludar") definimos la ruta principal.

Con @GetMapping("/hola") indicamos que el método responde a una petición GET.

**¿QUÉ ES EL ARCHIVO APPLICATION.PROPERTIES?**

El archivo application.properties es un recurso vital en proyectos de SpringBoot, ya que nos permite ajustar diversas configuraciones de la aplicación. Dentro de las posibilidades más comunes, podemos:

* Modificar el puerto por el cual se ejecuta la aplicación.
* Alterar el context path de la aplicación.
* Añadir variables o atributos personalizados.
* Gestionar diferentes perfiles para distintos entornos.

**¿Cómo gestionar diferentes perfiles de desarrollo?**

SpringBoot facilita la creación de perfiles separados para desarrollo y producción, permitiéndonos adaptar la configuración según el entorno. Para hacerlo:

Crea un nuevo archivo para el entorno de desarrollo:

Nombra el archivo application-dev.properties.

Crea otro archivo para el entorno de producción:

Nombra el archivo application-prd.properties.

En el archivo application.properties, especifica el perfil activo:

spring.profiles.active=dev

Define configuraciones específicas dentro de cada archivo, por ejemplo:

En application-dev.properties:

server.port=8090

En application-prd.properties:

server.port=80

**¿Cómo estructurar arquitectónicamente una aplicación?**

La estructura arquitectónica de una aplicación es fundamental para su correcto funcionamiento y mantenimiento a largo plazo. Una organización clara y bien definida facilita los procesos de desarrollo, depuración y actualización de la aplicación.

**¿Qué es la capa de dominio?**

La capa de dominio es el núcleo de la aplicación. En esta capa, encontramos los objetos de dominio y los DTO (Data Transfer Objects). Los objetos de dominio son representaciones abstractas de elementos dentro del contexto de la aplicación. En el caso de un supermercado, por ejemplo, podríamos tener productos, categorías, clientes, entre otros.

Además, en la capa de dominio, se definen los servicios que actúan como puentes entre los controladores de la API y la capa de persistencia. Finalmente, encontraremos las especificaciones de los repositorios, que son interfaces definiendo los contratos que la capa de persistencia debe cumplir.

**¿Qué roles juegan la capa web y la capa de persistencia?**

La capa web es donde se sitúan los controladores de la API. Estos controladores gestionan las solicitudes y respuestas de los clientes y actúan como la entrada al sistema.

Por su parte, la capa de persistencia interactúa directamente con la base de datos. Incluye los repositorios que implementan las especificaciones del dominio y las entidades, que actúan como mapeos entre las clases y las tablas de la base de datos.

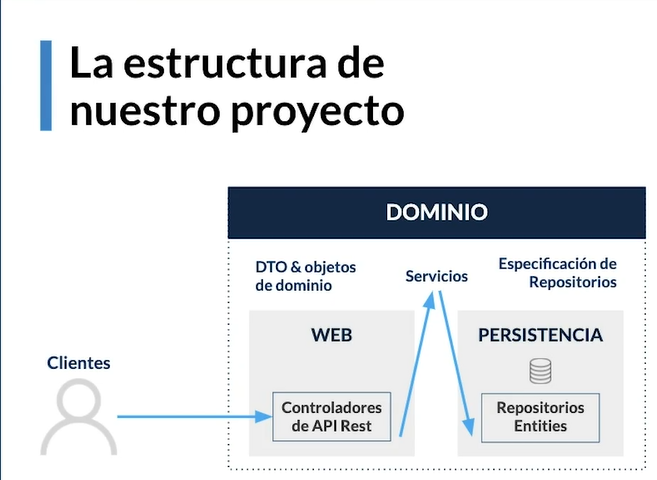
**¿Cómo fluye la información en esta arquitectura?**

Un cliente hace una solicitud a través de un controlador en la API.

* El controlador se comunica con los servicios en la capa de dominio para procesar la lógica de negocio.
* Los servicios contactan a los repositorios en la capa de persistencia para manejar operaciones de la base de datos.
* Finalmente, los datos son recuperados, transformados en entidades, y enviados de vuelta al cliente a través de la API.

Este flujo permite un desarrollo modular y escalable, facilitando el mantenimiento y crecimiento del proyecto. Así, no solo optimizas la arquitectura interna, sino que también garantizas un desempeño eficiente a largo plazo.

Este tipo de organización estructural beneficia tanto a desarrolladores como a usuarios, permitiendo un código claro y fácilmente extensible.



**¿Qué es JPA y por qué es importante?**

Jpa, o Java Persistence API, es una especificación en Java esencial para aquellos que buscan trabajar con bases de datos desde aplicaciones Java. JPA define un conjunto de reglas para el mapeo objeto-relacional (ORM), permitiendo que los desarrolladores interactúen con la base de datos sin necesidad de escribir consultas SQL nativas. Los frameworks más populares que siguen esta especificación incluyen Hibernate, EclipseLink, TopLink y ObjectDB. Además, JPA utiliza anotaciones para conectar las clases de Java con las tablas de la base de datos, facilitando así una integración fluida y eficiente.

**¿Cuáles son las anotaciones clave de JPA?**

JPA hace uso extensivo de anotaciones para simplificar el trabajo con bases de datos. Algunas de las más importantes incluyen:

**@Entity**: Indica que una clase Java representa una tabla en la base de datos.

**@Table:** Especifica el nombre de la tabla a la cual está mapeando nuestra clase.

**@Column:** Se utiliza en atributos dentro de una clase. Es opcional y solo necesaria cuando el nombre de la columna difiere del nombre del atributo en la clase.

**@Id**: Marca el campo que representa la clave primaria de la tabla en la clase. Se usa para claves primarias simples.

**@EmbeddedId**: Utilizada para claves primarias compuestas.

**@GeneratedValue**: Automáticamente genera valores para las claves primarias.

**@OneToMany y @ManyToOne**: Permiten representar relaciones complejas entre tablas, como existen en la base de datos, directamente en nuestras clases Java.

¿Cómo se integran Java y las bases de datos mediante JPA?

Gracias a JPA, Java ofrece una integración poderosa y flexible con las bases de datos. Al utilizar anotaciones, los desarrolladores pueden mapear rápidamente clases Java a tablas en la base de datos, simplificando el código y reduciendo la necesidad de SQL explícito. Esta facilidad de uso es fundamental para mejorar la productividad del desarrollo de aplicaciones Java y garantizar que las interacciones con las bases de datos sean seguras y eficientes.

En futuras lecciones, se ampliará el conocimiento sobre Spring Data, un proyecto del ecosistema Spring que aprovecha JPA para ofrecer herramientas adicionales y mejorar aún más la interacción de las aplicaciones Java con las bases de datos

**¿Qué es Spring Data y cómo se relaciona con JPA?**

Spring Data es un conjunto de proyectos que optimizan y facilitan la gestión de bases de datos en aplicaciones Java. Aunque no es una implementación de JPA (Java Persistence API), utiliza JPA para proporcionar una potente integración con bases de datos y ampliar sus capacidades. Este conjunto de herramientas permite conectar fácilmente aplicaciones a bases de datos SQL y NoSQL, gracias a subproyectos específicos como Spring Data JPA, Spring Data JDBC, Spring Data MongoDB y Spring Data Cassandra.

**¿Cuáles son las ventajas de usar Spring Data?**

**Repositorios sin código**

Una de las principales ventajas de utilizar Spring Data son los repositorios "sin código". Estos repositorios permiten ejecutar operaciones de lectura, escritura, actualización y borrado en bases de datos sin necesidad de escribir una sola línea de código. Parece magia, ¡y casi lo es!

**Auditorías transparentes**

También proporciona un sistema de auditorías potente y transparente que registra automáticamente cuándo se insertan, actualizan o eliminan registros en la base de datos. Los desarrolladores no necesitan preocuparse por implementar estas auditorías, ya que Spring Data maneja todo internamente.

**¿Cómo solucionar errores comunes al configurar Spring Data?**

Cuando inicias tu aplicación, podrías encontrar un error si no tienes configurada una base de datos. El error podría sugerir conectarte a una base de datos inmediata, como H2 o Derby. En este caso, no te preocupes, ya que el siguiente paso será configurar una conexión a una base de datos.

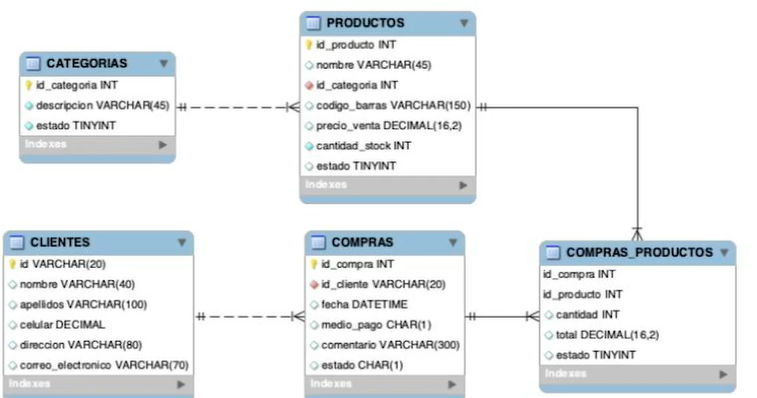


IMAGEN MER CURSO

**¿Qué son las claves primarias compuestas?**

En una base de datos, una clave primaria es un conjunto de uno o más campos o columnas que identifican de manera única cada registro en una tabla. Las claves primarias compuestas se utilizan cuando más de una columna es necesaria para garantizar la unicidad de un registro.

**Anotación @Embeddable:** Indica que la clase puede ser incrustada en una entidad.

**Implementación de Serializable:** Requerida para que la clase pueda ser utilizada como clave primaria.

**Implementación de la entidad con la clase Embedded**

Dentro de la entidad principal, se debe utilizar la anotación @**EmbeddedId**:

@**EmbeddedId**: Se emplea cuando una clase contiene la clave primaria compuesta. Diferente de @**Id**, que se usaría para claves simples.

Este método garantiza que las columnas relevantes de la base de datos se vinculen correctamente a los atributos de la clase.

**Asegurando la integridad del modelo de datos**

Finalmente, el siguiente paso en el desarrollo es establecer relaciones entre entidades para cumplir con el modelo de datos. Las relaciones, como @OneToMany, @ManyToOne, entre otras, configuran la manera en que las tablas interactúan entre sí, brindando un sistema coherente y robusto.

**¿Qué son los repositorios de Spring Data y cómo nos ayudan?**

Spring Data nos ofrece herramientas poderosas y los repositorios son una de las más destacadas. Diseñados para simplificar el desarrollo de aplicaciones empresariales en Java, estos repositorios nos permiten realizar operaciones en bases de datos sin tener que escribir código complejo. A través de ellos, operaciones comunes como eliminar, encontrar registros o guardar información se ejecutan de manera automática. Los tipos de repositorios que Spring Data ofrece incluyen:

* **CrudRepository**: Permite realizar operaciones básicas como crear, leer, actualizar y borrar.
* **PagingAndSortingRepository**: Además de las capacidades del CrudRepository, permite el paginado y ordenamiento de datos.
* **JpaRepository**: Combina las funcionalidades anteriores y ofrece características específicas de JPA, como Flush.

**¿Qué métodos proporciona un CrudRepository?**

Los métodos que incluye CrudRepository son variados y útiles para gestionar datos:

**save**: Guarda una entidad específica.

**saveAll**: Guarda una lista completa de entidades.

**findById**: Recupera una entidad usando su ID.

**existsById**: Verifica la existencia de un registro en la base de datos.

**findAll**: Recupera todos los registros de una tabla.

**findAllById**: Recupera los registros que pertenecen a una lista de IDs.

**count**: Cuenta cuántos registros existen en una tabla.

**deleteById**: Borra un registro específico por su ID.

**deleteAll**: Borra todos los registros o una lista específica de registros.

Estos métodos simplifican enormemente el manejo de datos y son inmediatos de usar sin necesidad de implementación adicional.

**Ventajas de usar Spring Data**

**Simplificación del desarrollo**: Al abstraer las operaciones de base de datos, Spring Data reduce significativamente la cantidad de código necesario.

**Flexibilidad y eficiencia:** Permite implementar operaciones complejas de manera simple.

**Mantenibilidad**: El uso de interfaces y métodos predefinidos facilita la actualización y el mantenimiento del código a lo largo del tiempo.

**¿Cómo utilizar queryMetods en Spring Data para consultas sin SQL?**

Los queryMetods son una herramienta valiosa y potente en Spring Data que facilita la realización de consultas sin la necesidad de escribir código SQL directamente. Esta alternativa es particularmente útil en situaciones donde los repositorios estándar de Spring Data no pueden proporcionar la información que necesitas.

**¿Qué son y cómo funcionan los queryMetods?**

Los queryMetods permiten generar consultas simplemente mediante el nombramiento de métodos de una forma específica. Esto se hace sin tener que escribir consultas en SQL, lo cual es posible gracias a ciertas convenciones de nomenclatura que el framework Spring interpreta para generar la consulta adecuada.

**¿Qué ventajas ofrecen los queryMetods?**

Los queryMetods no solo eliminan la necesidad de escribir SQL, sino que también:

* Mejoran la legibilidad del código al usar nombres de métodos intuitivos que describen la operación.
* Permiten la flexibilidad de retornar tipos de dato opcional, adaptándose a la programación funcional.
* Facilitan la implementación de varios tipos de consultas gracias a su estructura.

**¿Query nativos o queryMetods?**

Aunque el uso de queryMetods es generalmente preferible por su flexibilidad y mejor práctica, existe la opción de utilizar consultas nativas a través de anotaciones. Por ejemplo:

// Uso de consulta nativa para encontrar productos por ID de categoría

@Query(value = "SELECT \* FROM productos WHERE id\_categoria = ?1", nativeQuery = true)

public List<Product> getByCategoria(int idCategoria);

Este enfoque puede ser útil en situaciones donde se requiere mayor control sobre la consulta SQL.

**¿Cómo crear métodos más complejos con queryMetods?**

Los queryMetods también te permiten combinar varias condiciones de búsqueda. Por ejemplo, buscar productos cuya cantidad en stock sea menor a un umbral y que estén activos:

// Uso de operadores para encontrar productos con bajo stock y activos

public Optional<List<Product>> findByCantidadDeStockLessThanAndEstado(int cantidad, boolean estado);

Para implementar este método en tu repositorio, podrías hacer algo así:

// Método en el repositorio de productos

public Optional<List<Product>> getEscasos(int cantidad, boolean estado) {

return productoRepository.findByCantidadDeStockLessThanAndEstado(cantidad, true);

}

**¿Qué es un repositorio en Spring?**

Un repositorio en Spring es una clase que se encarga de realizar operaciones sobre la base de datos. Para lograr esto, utilizamos la anotación @Repository, que indica a Spring que la clase interactúa directamente con la base de datos. Además, podemos usar @Component, que es más genérica, pero @Repository es más específica y adecuada para nuestras necesidades.

**Importancia de las anotaciones en Spring**

La anotación @Repository es un estereotipo de Spring, crucial para indicar que una clase maneja operaciones de persistencia. Esto facilita la organización y el mantenimiento del código, ayudando a Spring a gestionar las excepciones automáticamente y proporcionando una capa adicional para la gestión de datos.

Incorporar estas buenas prácticas y comprender la diferencia entre @Component y @Repository refuerza el código y asegura su claridad y eficacia. Con esta base, estás preparado para seguir avanzando en las otras capas de tu aplicación de manera organizada y eficaz.

**¿Qué es el patrón DataMapper y cómo se aplica?**

El patrón DataMapper es un concepto fundamental en el desarrollo de software, especialmente en aplicaciones que siguen el enfoque de dominio. Este patrón se encarga de traducir o convertir entre dos objetos que realizan funciones similares. Por ejemplo, podríamos tener un objeto Producto y un objeto Product, donde sus atributos se traducen entre sí para cumplir roles equivalentes en diferentes contextos.

**Beneficios del DataMapper**

Implementar el patrón DataMapper ofrece varias ventajas:

**Protección de la base de datos**: No expone directamente el diseño de las tablas al API, protegiendo así la arquitectura interna.

**Desacoplamiento de la API:** Facilita el cambio de bases de datos sin modificar todo el código, simplemente ajustando el traductor.

**Evita campos innecesarios:** Permite omitir atributos que no son relevantes en la API, solo mantienen su utilidad dentro del contexto de la persistencia.

**Homogeneidad del idioma**: Permite mantener un único idioma en el dominio de la aplicación, evitando mezclar idiomas en el código fuente.

**MapStruct**

Para aplicar el patrón DataMapper, utilizamos MapStruct, una herramienta que simplifica el proceso de mapeo entre objetos.

**¿Cómo crear clases de dominio?**

Las clases de dominio son esenciales, ya que encapsulan la lógica del negocio. Al crear estas clases, le damos estructura al proyecto para manejar objetos que representan entidades del negocio, como productos y categorías.

**¿Cómo definir interfaces de repositorios?**

Definimos interfaces con métodos que cualquier repositorio debe implementar al trabajar con productos. Esto facilita la gestión y el acceso a datos de productos dentro de una base de datos sin acoplarse a una tecnología específica.

**¿Qué anotaciones se deben utilizar en un mapeador?**

MapStruct nos ofrece una sintaxis clara y precisa para definir nuestras conversiones:

@**Mapper**: Esta anotación debe añadirse a cada interfaz para indicar que se trata de un mapeador MapStruct.

**componentModel = "spring":** Se establece como parte de @Mapper, permitiendo integrar MapStruct con Spring. Indica que nuestro mapeador debe comportarse como un componente Spring.

Definir estas anotaciones garantiza que MapStruct gestione correctamente la creación de instancias de mapeadores y su integración con el ciclo de vida de Spring.

**¿Qué beneficios ofrece MapStruct en el desarrollo?**

MapStruct simplifica la configuración de mapeadores, lo que reduce el tiempo de desarrollo e incrementa la legibilidad, mantenibilidad y precisión de las conversiones de datos. Incentiva a los desarrolladores a estructurar su código de manera modular y reutilizable.

**¿Qué es la inyección de dependencias y por qué es importante?**

La inyección de dependencias es un principio clave en el desarrollo de software moderno que busca desacoplar las clases de las implementaciones que utilizan, promoviendo así el principio de inversión de control. Al pasar las dependencias a la clase en vez de crearlas internamente, se reduce el acoplamiento y se incrementa la flexibilidad y facilidad de mantenimiento del código.

**¿Cómo implementa Spring la inyección de dependencias?**

Spring, un marco de trabajo popular para el desarrollo de aplicaciones en Java, implementa la inyección de dependencias utilizando su contenedor de inversión de control. Este contenedor toma el control de la creación y gestión de instancias de los objetos necesarios para el funcionamiento de la aplicación.

**Autowire en Spring**: Spring facilita la inyección de dependencias mediante la anotación @**Autowired**. Al usar esta anotación, se indica que el control de la creación de instancias será delegado al contenedor de Spring. Esto hace transparente el proceso de inyección de dependencias para el desarrollador, quien no tendrá que preocuparse por la creación manual de objetos.

**¿Por qué es crucial utilizar correctamente los componentes de Spring?**

Un aspecto esencial al inyectar dependencias en Spring es asegurarse de que las clases o atributos que se desean inyectar sean componentes reconocidos por el framework. En otras palabras, deben ser beans de Spring.

Ejemplo de uso: Consideremos un repositorio que extiende de CrudRepository. Al extenderlo, automáticamente se marca como un componente de Spring, permitiendo así ser inyectado. Del mismo modo, si se utiliza una clase decorada con anotaciones específicas de otra biblioteca, como MapStruct, es importante verificar que el componente o su modelo sea compatible para ser gestionado por Spring.

**¿Cómo ayuda este enfoque a mejorar el desarrollo?**

La inyección de dependencias automatizada elimina la necesidad de crear objetos manualmente, lo que puede ser propenso a errores y difícil de mantener. Al confiar en el framework de Spring para gestionar estas instancias, los desarrolladores pueden concentrarse en la lógica de negocio de sus aplicaciones, asegurando un código más limpio y eficiente.

**Recomendaciones para el uso efectivo**

**Verificar componentes**: Asegúrate de que las clases a inyectar son parte del ecosistema de Spring para evitar problemas.

**Utilizar anotaciones adecuadas:** Aprovecha las anotaciones que ofrece Spring, como @Component, @Service, @Repository, para que el contenedor pueda identificar y gestionar tus beans.

**Evitar el acoplamiento excesivo:** La inyección de dependencias promueve la creación de un código modular y más fácil de mantener, crucial para proyectos de gran escala.

**¿Cómo crear un servicio de dominio en Spring Boot?**

Construir un servicio de dominio es una parte crucial en el desarrollo de aplicaciones, dado que actúa como un intermediario vital entre el controlador de nuestra API y el repositorio. Este enfoque permite mantener una separación clara entre las responsabilidades, mejorando así el mantenimiento y la escalabilidad de tu aplicación

**¿Qué es un servicio en Spring Boot?**

En el contexto de Spring Boot, un servicio es una clase que contiene la lógica de negocio específica de la aplicación. Generalmente, estas clases están decoradas con la anotación @Service, que se importa desde el paquete org.springframework.stereotype. El uso de @Service no sólo proporciona una diferenciación semántica, también facilita la gestión de la inyección de dependencia.

**¿Cómo se inyecta el repositorio en un servicio?**

En Spring Boot, se utiliza la anotación @Autowired para inyectar dependencias automáticamente. Cuando definimos el repositorio como un atributo de tipo interfaz en el servicio, Spring Boot se encarga de crear y asignar la implementación real del mismo.

**¿Qué consideraciones adicionales tener al crear servicios?**

**Uso de Optional y Map:** Optional es una excelente herramienta para manejar resultados que pueden ser nulos, promoviendo un código más seguro y limpio al evitar las excepciones de tipo NPE (Null Pointer Exception).

**Anotaciones y Gestión de Dependencias:** Las anotaciones en Spring como @Service y @Autowired facilitan la gestión de dependencias y el establecimiento de clases dentro del contexto de Spring.

**Semántica y Buenas Prácticas:** Es importante seguir las convenciones y buenas prácticas al usar anotaciones para facilitar el mantenimiento y comprensión del código.

**¿Qué anotaciones usar para un controlador REST?**

En el ecosistema de Spring, las anotaciones son esenciales para configurar y operar tus componentes. Para un controlador REST, las más relevantes son:

@**RestController**: Indica que la clase va a manejar peticiones REST.

@**RequestMapping**: Define la ruta que las distintas peticiones van a seguir.

**¿Cómo verificar que el controlador está funcionando?**

Una vez que el controlador esté configurado, el siguiente paso es lanzar la aplicación y verificar que se ejecute correctamente:

Ejecutar el proyecto: Usa el comando Run para compilar y lanzar la aplicación de Spring Boot.

Solucionar problemas comunes: Si encuentras el error "el puerto 8090 ya está en uso", significa que otro proceso está usando este puerto. Soluciona cerrando el proceso conflictivo.

**¿Qué hacer si no se encuentra un recurso al acceder a la API?**

Si al ingresar a una ruta específica en la API aparece un mensaje de error indicando que no encontró ningún recurso, es probable que los métodos del controlador no estén bien configurados para manejar las peticiones. La solución consiste en asegurarse de que cada método está correctamente "decorado" con anotaciones adecuadas

**¿Cómo exponer una API utilizando Spring?**

El mundo del desarrollo de software es vasto, y una herramienta muy valiosa en la construcción de aplicaciones es la capacidad de exponer APIs de manera eficiente. Spring Framework nos facilita esta tarea proporcionándonos varias anotaciones que nos permiten publicar nuestros servicios rápidamente y con menos complicaciones. En esta clase exploraremos cómo usar las anotaciones @GetMapping, @PostMapping y @DeleteMapping.

**¿Cómo usar GetMapping para obtener información?**

Para obtener información de nuestro servidor, utilizamos la anotación @GetMapping. Esta anotación nos permite vincular un método a una solicitud HTTP GET. Por ejemplo, si queremos publicar un servicio que recupere una lista completa de productos, simplemente añadimos @GetMapping al método correspondiente, especificando la ruta de acceso adecuada

**¿Cómo recuperar un producto específico por su ID?**

Recuperar un producto por su ID es igualmente sencillo, pero requiere un enfoque ligeramente distinto: debemos incluir una variable de ruta en la anotación @GetMapping. Usamos {} para indicar parámetros de ruta dinámicos y @PathVariable para mapearlos al método pertinente

**¿Cómo guardar y actualizar información con PostMapping?**

Para enviar información nueva al servidor, @PostMapping es la anotación que debemos utilizar, indicando que la información se envía en el cuerpo de la solicitud

**¿Cómo eliminar registros con DeleteMapping?**

Cuando necesitamos eliminar un recurso, @DeleteMapping es la solución. Como en operaciones de obtención de datos, utilizamos la variable de ruta para especificar qué recurso debe eliminarse:

**¿Cómo mejorar los endpoints con ResponseEntity en Spring?**

Hablar de un controlador API efectivo es hablar de la gestión adecuada de las respuestas HTTP. Implementar ResponseEntity en Spring no solo mejora la legibilidad del código, sino también la claridad y robustez del API en cuestión.

**¿Qué es ResponseEntity en Spring?**

ResponseEntity es una clase en Spring que permite gestionar de una manera más controlada las respuestas de los endpoints de una API. Este control es fundamental para definir el cuerpo y el código de estado de una respuesta HTTP, dando mayor claridad a los usuarios sobre cómo fue manejada su petición.

Control de Respuestas: Utilizando ResponseEntity, puedes indicar exactamente qué se está devolviendo y con qué código de estado (como 200 OK, 201 Created o 404 Not Found).

Manejo de Errores: Te permite definir qué hacer cuando una operación no resulta como se esperaba, mejorando la comunicación con el cliente.

**Ejemplo de Prueba:**

Producto no existente: Consulta un producto que no existe, ahora debería responder con 404 Not Found.

Creación de Producto: Crea un producto y verifica que la respuesta ahora incluye 201 Created, reflejando que el recurso fue exitosamente creado.

**Consejos para una implementación exitosa**

Comprende el esquema de tus datos: Antes de implementar los mapeadores, asegúrate de entender cómo están estructuradas tus entidades y sus relaciones.

Verifica los nombres de atributos: Los errores comunes en el mapeo suelen derivarse de líos con los nombres de los atributos. Revisa minuciosamente cuál es el nombre correcto de cada atributo en tus fuentes y destinos.

Pruebas exhaustivas: Aunque los mapeadores estén configurados, es esencial validar su funcionalidad mediante pruebas para asegurar la integración correcta de los datos.

**¿Cómo desplegar una aplicación Java utilizando el terminal?**

Desplegar una aplicación Java en un entorno de producción puede parecer una tarea intimidante al principio. Sin embargo, con las instrucciones correctas, puede ser un proceso sencillo y directo. Aquí te mostraré cómo hacerlo utilizando una aplicación autocontenida y el terminal de tu computadora.

**¿Qué comandos debo utilizar para desplegar mi aplicación?**

Para desplegar tu aplicación desde el terminal, seguimos los siguientes pasos:

Generar el artefacto final:

Usamos el comando BootJard para empaquetar la aplicación.

Accedemos al IDE, vamos al apartado de Gradle y buscamos la tarea BootJard dentro del folder build.

Cambiar la versión de nuestra aplicación:

Editamos el archivo build.gradle para modificar la versión a 1.0 en vez de 0.0.1.

Decidir los parámetros de despliegue:

Ajustar la memoria asignada usando -Xmx2048m.

Cambiar el puerto utilizando -Dserver.port.

Ejecutar el comando desde la terminal:

Nos ubicamos dentro de la carpeta del proyecto.

Ejecutamos el siguiente comando reemplazando nombre y ubicación por los valores correctos:

java -jar market/nombre.jar

**¿Cómo resolver conflictos comunes durante el despliegue?**

Durante el proceso de despliegue, podrían surgir inconvenientes. A continuación, te muestro cómo abordar algunos problemas comunes:

**Puerto en uso:** Si el puerto está ocupado, debemos detener cualquier servicio que lo esté utilizando (en Intellij IDEA, debemos parar el servicio en ejecución).

Seleccione el perfil activo adecuado:

Si necesitas cambiar el perfil activo a producción, cancela el proceso actual y vuelve a ejecutar el comando java -jar, pero adicionando:

-Dspring.profiles.active=prod

**¿Cuál es la importancia de seleccionar el perfil correcto?**

Los perfiles de aplicación son vitales para modificar la configuración dependiendo del entorno (desarrollo, prueba o producción). Al utilizar -Dspring.profiles.active=prod, instruimos a nuestra aplicación para que utilice las configuraciones definidas para producción en application-prod.properties.

**¿Qué requerimientos técnicos se deben cumplir?**

Versión de Java Compatible:

Asegúrate de desplegar tu archivo JAR en un entorno que use la misma versión de Java que empleaste durante su desarrollo.

**Comando para ejecutar el .jar**

java -jar build\\libs\market-1.0.jar

Esto estando en la ubicacion general del proyecto (en este caso market)

Y para seleccionar un perfil de produccion u otro perfil para que no se despliegue con un perfil de dev:

java -jar -Dspring.profiles.active=prd build\libs\market-1.0.jar

Esto estando en la ubicacion general del proyecto (en este caso market)

**IMPORTANTE**

#cada variable que hay, hay que cambiarla del siguiente modo ${}, esto para configurarlo con variables destinadas al despliegue, pero este tema toca que

#investigarlo por cuenta de una ya que los cursos que hay estan desactualizados, y encontrar un servicio de nubes gratuito para poder hacer la prueba de despliegue

#con la base de datos de postgresql y con spring, y mas adelante realizar el front y realizar las debidas conexiones y desplegarlos