**Spring**

**Spring Boot Actuator**: is an application which you can use to monitor your application spring boot. Actuator is basically like a manager console; it helps you to manage your application.

Monitor and Manage Spring Boot Application using REST/JMX Actuator endpoints. (health, metrics)

**Dependency injection:** We use Spring Framework to instantiate beans and wire dependencies. Spring injects dependencies with @Component, @Autowired. Springs finds the beans and autowired.

**Inversion of Control:** Spring finds the beans and wire them, then injects the dependencies in the classes.

Inversion of Control is a principle in software engineering which transfers the control of objects or portions of a program to a container or framework. We most often use it in the context of object-oriented programming.

**@Component**: Spring should manage the bean. Generic component.

**@Autowired:** Spring should find the matching bean and wire the dependency in.

**@Repository:** encapsulating storage, retrieval, and search behavior typically from a relational database. DTO, usado en data layer. Obtener datos de la BD

**@Service:** Business Service Facade. Usado en la business layer, transaction management

**@Controller:** Controller in MVC pattern. Se usa en web, en el patron mvc. Web layer

**@SpringBootApplication:** this tells this spring boot application is going to create a spring context.

JPA defines the mapping from your Java object to a table. JPA provides a lot of useful annotations where you can define the relationship between your classes and tables. We create the mapping, and the API creates the queries using annotations.

@Entity: becomes the bean into an entity that can be manage by JPA.

@Table(name = “Todo”): add when the name of the class is different to the name of the table

@GeneratedValue(strategy = GenerationType.IDENTITY): autogenerates the value of the column in the table.

REST: Representational State Transfer. REST is a style architecture for distributed hypermedia systems.

@GetMapping: shortcut of @RequestMapping

**HATEOAS :** Hypermedia as The Engine of Application State

* Example
  + When requested for details of a facebook post, we return
    - Link for actions to like, unlike or comment on the post
* HATEOAS is an extra level upon REST.
* It is used to present information about a REST API to a client without the need to bring up the API documentation.
* It includes links un a returned response and client can use those API links to further communicate with the server.
* Simplify the client by making the API discoverable.
* Self: creates a link for the object “\_links”:{“self:”}

**Swagger:** OpenAPI Specification (formerly called the Swagger Specification)

The specification creates the RESTful contract for API, detailing all of its resources and operations in a human and machine readable format for easy development, discovery, and integration.

Internationalization(i18n): is a process that makes our application adaptable to different languages without making any changes to our source code.

SessionLocaleResolver: The LocaleResolver interface has implementations that determine the current locale based on the session, cookies, the Accept-Language header, or a fixed value.

In our example, we've used the session-based resolver SessionLocaleResolver and set a default locale with the value US.

**Spring Bean Scopes**

There are five types of spring bean scopes:

* singleton - only one instance of the spring bean will be created for the spring container. This is the default spring bean scope. While using this scope, make sure bean doesn’t have shared instance variables otherwise it might lead to data inconsistency issues. Lives in the application context and dies with the application.
* prototype – A new instance will be created every time the bean is requested from the spring container. Lives in the application context but is live cycle does not depend of the application.
* request – This is same as prototype scope, however it’s meant to be used for web applications. A new instance of the bean will be created for each HTTP request. Lives in the request context and dies when the request finalizes.
* session – A new bean will be created for each HTTP session by the container. Lives in the session context and dies with the session.
* global-session – This is used to create global session beans for Portlet applications. Lives in the session context and dies with the session.
* Application - A new instance will be created every time the application initializes and dies with the application. Lives in the application context.
* Websocket – this bean is created once for every Websocket connection. Context connection.

MySQL

DDL (Data Definition Language) :

* A syntax
* A set of statements that allow the user to define or modify data structures and objects, such as tables.
* DDL (Data Definition Language) es un conjunto de comandos SQL para definir el esquema de la base de datos. Se trata simplemente de descripciones del esquema de la base de datos y se utiliza para crear y modificar la estructura de los objetos de la base de datos. Algunos ejemplos de DDL son CREATE, ALTER y DROP.
* DML (Data Manipulation Language (DML) es un subconjunto de SQL (Structured Query Language) que se utiliza para gestionar y manipular datos en una base de datos relacional. Los comandos de DML se utilizan para insertar, actualizar, eliminar y recuperar datos de una base de datos.

¿Qué es el DDL y DML?

Sentencias DDL son aquellas utilizadas para la creación de una base de datos y todos sus componentes: tablas, índices, relaciones, disparadores (triggers), procedimientos almacenados, etc. sentencias DML son aquellas utilizadas para insertar, borrar, modificar y consultar los datos de una base de datos.

DCL (Data Control Language): es un lenguaje proporcionado por el sistema de gestión de base de datos que incluye una serie de comandos SQL que permiten al administrador controlar el acceso a los datos contenidos en la base de datos.

Algunos ejemplos de comandos incluidos en el DCL son los siguientes:

GRANT: Permite dar permisos a uno o varios usuarios o roles para realizar tareas determinadas.

REVOKE: Permite eliminar permisos que previamente se han concedido con GRANT.

TLC (Transaction Control Language DCL): El Lenguaje de control de transacción es, o TCL, son comandos de SQL (COMMIT, ROLLBACK, etc.) que permiten manejar transacciones en una base de datos relacional, por lo que es importante primeramente aclarar el concepto de transacción.

**Indexes:** In MySQL, the Indexes can be formed by using one or many columns, which provides the base for both speedy casual lookups and proficient assembling of access to records. Indexes are essentially used in the server database tables to swiftly find the table rows having specific values of the table column. If there is an absence of an index in the table, then MySQL needs to scan the entire table to locate and check the appropriate table rows.

* **Unique** is a type of MySQL Index which specifies that all values of the tables columns, when implemented, have to be distinct. There can be no duplicate values in the column which is indexed existing in a single column unique index.
* **Primary Key** is a type of Unique Index which specifies that the column mentioned as Primary key must contain no NULL value.
* In this type of Index named as **simple, regular or normal**, the specified column values do not require to be unique and as well as can be NULL also. They are supplemented basically to aid the database searching process for records quicker.
* **Full-Text:** This type of index is implemented for full-text searches as the name itself implies it. Occasionally

**JOINS**:

* Inner join: Este tipo de unión te ayuda a combinar varias tablas, y te devuelve únicamente los datos que estén disponibles en todas las tablas a la vez.
* Left join: devuelve todos los resultados que coincidan en la primera tabla, con los datos que tenga de la segunda. En el caso de que falte algún dato, devolverá un valor null en lugar del dato, pero seguiremos teniendo el valor de la primera tabla.
* RIGHT JOIN: pasa exactamente lo mismo que con el anterior, pero con la diferencia de que devuelve todos los datos de la tabla con la que se relaciona la anterior.
* OUTER JOIN consiste en recuperar TODOS los datos que haya en ambas tablas, tanto los que tienen contenido en ambos extremos, como los que no.

**Behavior Driven Development**: BDD es una estrategia de desarrollo dirigido por comportamiento, empatizando con el usuario final de tus desarrollos.

**WireMock** es un simulador de APIS REST HTTP, también podemos decir un REST API Mock Server.

A **message-driven bean** is an enterprise bean that allows Java EE applications to process messages asynchronously. This type of bean normally acts as a JMS message listener, which is similar to an event listener but receives JMS messages instead of events. The messages can be sent by any Java EE component (an application client, another enterprise bean, or a web component) or by a JMS application or system that does not use Java EE technology. Message-driven beans can process JMS messages or other kinds of messages.

Apache **Avro**™ es un sistema para serializar datos en un formato binario compacto. Una estructura de datos Avro se define en un esquema Avro, que está escrito en formato JSON. Escrito en binario, buena estructura de datos ordenados. Se integra bien con hadoop.

You can use **ksqlDB** to build event streaming applications from Apache Kafka® topics by using only SQL statements and queries. ksqlDB is built on Kafka Streams, so a ksqlDB application communicates with a Kafka cluster like any other Kafka Streams application.

SQL Engine, REST API, Built-in CLI

**N-layer application** is partition application logic into specific layers. The N-layer architecture pattern is a mature architecture and simply refers to applications separate various logical layers into separate physical tiers.

The most common layers include:

• User interface

• Business logic

• Data access

**Clean architecture** is a software design philosophy that separates the elements of a design into ring levels. An important goal of clean architecture is to provide developers with a way to organize code in such a way that it encapsulates the business logic but keeps it separate from the delivery mechanism.

**Clean architecture** is a software design philosophy that separates the elements of a design into ring levels. An important goal of clean architecture is to provide developers with a way to organize code in such a way that it encapsulates the business logic but keeps it separate from the delivery mechanism.

**Kubernetes** automatiza las tareas operativas de la administración de contenedores e incluye comandos integrados para implementar aplicaciones, actualizarlas, escalarlas a fin de que se ajusten a tus necesidades, supervisarlas y mucho más. Todo esto facilita la administración de las aplicaciones.

**Docker** es una plataforma de contenedorización y un tiempo de ejecución de contenedores, mientras que **Kubernetes** es una plataforma para ejecutar y gestionar contenedores a partir de numerosos tiempos de ejecución de contenedores.

**Docker** es un sistema operativo (o runtime) para contenedores. El motor de Docker se instala en cada servidor en el que desee ejecutar contenedores y proporciona un conjunto sencillo de comandos que puede utilizar para crear, iniciar o detener contenedores.

**Decomposition patterns**. These patterns help to break down a monolithic application into smaller, more manageable microservices. In detail, we will discuss three decomposition patterns: microservices decomposition by business capability, microservices decomposition by subdomain, and microservices decomposition by transaction.

**ACID** is an acronym that stands for atomicity, consistency, isolation, and durability. Together, these ACID properties ensure that a set of database operations (grouped together in a transaction) leave the database in a valid state even in the event of unexpected errors.

Why are ACID principles important?

The acronym ACID stands for atomicity, consistency, isolation, and durability. These describe the set of properties of database transactions that guarantee data integrity despite errors, system failures, power failures, or other issues.

The **SAGA** (or Saga) pattern is a Microservice design pattern for managing data consistency in distributed systems. It provides a way to handle long-lived transactions that are composed of multiple steps, where each step is a separate database operation.

The main idea is to capture all the steps of the transaction in a database, so that in case of failure, the system can roll back the transaction to its initial state.

**JAVA**

Shallow Copy: When we do a copy of some entity to create two or more than two entities such that changes in one entity are reflected in the other entities as well, then we can say we have done a shallow copy. In shallow copy, new memory allocation never happens for the other entities, and the only reference is copied to the other entities.

Deep Copy: When we do a copy of some entity to create two or more than two entities such that changes in one entity are not reflected in the other entities, then we can say we have done a deep copy. In the deep copy, a new memory allocation happens for the other entities, and reference is not copied to the other entities. Each entity has its own independent reference.

**POJO** stands for Plain Old Java Object. It is an ordinary Java object, not bound by any special restriction other than those forced by the Java Language Specification and not requiring any classpath. POJOs are used for increasing the readability and re-usability of a program. POJO is an object which encapsulates Business Logic.