* Spring Boot:-

Spring Boot is a powerful framework for building Java applications with ease and speed. It is built on top of the Spring framework, leveraging its capabilities while providing a simplified approach to building and deploying applications.

* Features of Spring Boot:-

1. Auto-Configuration: Spring Boot automatically configures your application based on the dependencies you have added to your project. This significantly reduces the amount of boilerplate configuration code you need to write.

2. Starter Dependencies: Spring Boot provides a collection of "starter" dependencies that encapsulate sets of libraries and configurations for various use cases. These starters make it easy to add functionality to your application by simply including the relevant starter dependency in your project.

3. Embedded Servers: Spring Boot includes support for embedded servers like Tomcat, Jetty, and Undertow. This means you can package your application as a self-contained JAR file that includes the server, making it easy to deploy and run your application without needing to install and configure a separate server.

4. Externalized Configuration: Spring Boot allows you to externalize your configuration, which means you can configure your application using properties files, YAML files, environment variables, or command-line arguments. This makes it easy to configure your application differently for different environments without modifying your code.

5. Spring Boot DevTools: Spring Boot DevTools provides a set of tools to streamline the development process. This includes automatic application restarts, live reload for static resources, and remote debugging support.

* @SpringBootApplication:-

• The @SpringBootApplication annotation is a key annotation in Spring Boot applications. It combines several other annotations to provide a shortcut for configuring Spring-based applications. Here's what it does:

1. @Configuration: This annotation indicates that the class is a Spring configuration class. It allows the class to define beans and other components that will be managed by the Spring container.

2. @EnableAutoConfiguration: This annotation enables Spring Boot's auto-configuration mechanism. Auto-configuration need for automatically configures the Spring application based on its classpath and the dependencies present. It greatly reduces the manual configuration by providing sensible defaults and configurations.

3. @ComponentScan: This annotation tells Spring to scan the specified package and its sub-packages for Spring components, such as @Component, @Service, @Repository, and @Controller. It registers these components with the Spring container, making them available for dependency injection and other Spring features.

* application.properties:-

• The application.properties file is a configuration file commonly used in Spring Boot applications to externalize configuration settings.

• The application.properties file allows developers to externalize configuration settings from their code. Instead of hardcoding configuration values directly into the application code, developers can specify them in this properties file.

• The application.properties file follows a simple key-value pair format.

• Spring Boot provides a wide range of configuration properties that can be specified in the application.properties file. These properties cover various aspects of the application, including server configuration, database configuration, logging, security, and more.

• In a Spring Boot application, the application.properties file is read and processed automatically by the Spring Environment.

• By default, Spring Boot looks for the application.properties file in the src/main/resources directory of your project. If found, it automatically loads the properties from this file.

* CRUD Repository:-

• It is an interface present in org.springframework.data.repository package.

• CrudRepository is an interface provided by Spring Data, a part of the larger Spring Framework ecosystem, designed to simplify data access and manipulation in Spring-based applications.

• It provides CRUD (Create, Read, Update, Delete) operations for working with entities in a database.

1. CRUD Operations:

=> CrudRepository defines methods for performing CRUD operations on entities.
=> These operations include:

save(entity): Saves or updates the given entity.

findById(id): Retrieves an entity by its primary key.

findAll(): Retrieves all entities.

delete(entity): Deletes the given entity.

deleteById(id): Deletes an entity by its primary key.

count(): Returns the number of entities.

existsById(id): Checks if an entity with the given primary key exists.

2. Type Parameters:

=> CrudRepository is a generic interface that takes two type parameters:

The entity type: Specifies the type of entity managed by the repository.

The ID type: Specifies the type of the entity's primary key.

3. Custom Query Methods:

=> In addition to the predefined CRUD methods, CrudRepository supports the creation of custom query methods. By following a specific naming convention, developers can define method signatures in the repository interface, and Spring Data will automatically generate the corresponding SQL queries at runtime.

4. No Implementation Required:

=> CrudRepository is an interface, so it doesn't provide any implementation. Instead, Spring Data generates the implementation dynamically at runtime based on the entity type and the methods defined in the repository interface. This makes it easy to create data access layers without writing boilerplate code.

5. Extensible:

=> CrudRepository is part of a hierarchy of repository interfaces provided by Spring Data. Developers can extend CrudRepository with additional custom methods or use more specialized interfaces like PagingAndSortingRepository or JpaRepository for specific use cases.

* JPA Repository:-

• It is an interface present in org.springframework.data.jpa.repository package.

• JpaRepository is an interface provided by Spring Data JPA, a part of the Spring Framework ecosystem, which simplifies data access and manipulation in Spring-based applications using the Java Persistence API (JPA).

• JpaRepository extends the CrudRepository interface. Therefore, it inherits all the CRUD (Create, Read, Update, Delete) operations provided by CrudRepository.

• JpaRepository integrates with the JPA EntityManager, providing methods for flushing changes to the database, refreshing entities, and accessing the underlying JPA EntityManager.

• Overall, JpaRepository provides an extended set of features on top of CrudRepository, making it a powerful tool for working with JPA entities in Spring applications. It offers additional querying capabilities, support for pagination and sorting, and integration with the JPA EntityManager, while still providing the basic CRUD operations inherited from CrudRepository.

* Web-Application & Web-Service:-

• A web application and a web service are both web-based technologies, but they serve different purposes and have distinct characteristics:

Web Application:

1. User Interaction:

=> A web application is a software application that is accessed via a web browser and provides interactive functionality to users.

=> It typically consists of a user interface (UI) that allows users to perform various tasks, such as inputting data, viewing information, and interacting with the application's features.

2. Client-Side Rendering:

=> In a web application, much of the application logic and processing occurs on the client-side (i.e., within the web user browser). This means that the browser receives HTML, CSS, and JavaScript files from the server and renders the UI and handles interactions without requiring frequent communication with the server.

3. Stateful:

=> Web applications are often stateful, meaning they maintain information about the current user session and application state. This allows users to interact with the application over time, with the application remembering their preferences, settings, and activities.

Examples:

e-commerce websites, social media platforms, online banking systems.

Web Service:

1. Data Exchange:

=> A web service is a software system designed to facilitate communication and data exchange between different applications over the internet.

=> It provides a standardized way for applications to interact with each other, regardless of the programming languages or platforms they are built on.

2. Machine-to-Machine Communication:

=> Web services primarily focus on machine-to-machine communication, where one application requests data or services from another application using standard protocols such as HTTP, SOAP, or REST.

3. Stateless:

=> Web services are typically stateless, meaning each request from a client to a web service is independent and self-contained. The web service does not maintain information about the client's previous requests or state.

Examples:

APIs that provide access to data or functionality, such as weather data APIs, payment processing APIs, and location-based services APIs.

• A web application is a user-facing software application accessed via a web browser, providing interactive functionality and often maintaining user session state. In contrast, a web service is a backend system that facilitates communication and data exchange between different applications over the internet, primarily focusing on machine-to-machine interaction and providing access to data or functionality via standardized protocols.

* RESTful Service or RESTful API:-

• It stands for Representational State Transfer.

• A RESTful Web Service, also known as a RESTful API, is an architectural style for designing networked applications.

• It defines a set of rules,guidelines and conventions for building web services that follows the principle of REST.

• It allows applications to communicate with each other over the internet using standard HTTP methods (GET,POST,PUT,DELETE)

to perform various operations on resourses.

• REST APIs are widely used for building scalable and stateless web services.

• Resources are represented in a format such as JSON or XML. Clients interact with these representations by making URL

requests.

• Statelessness - Each request from a client to a server must contain all the information needed to understand and fullfill

the request. The server does not store information about the previous requests.

* JSON (JavaScript Object Notation) :-

• JSON is a lightweight data interchange format that is easy for humans to read and write and easy for machines to parse and generate.

• It is widely used as a data format for exchanging data between a server and a web browser or between different parts of an application.

* Characteristics and syntax JSON

1. Human-Readable Format:

=> JSON is designed to be easy to read and write for both humans and machines.

=> It uses a lightweight, text-based format that resembles JavaScript object literal syntax, making it familiar to programmers who are comfortable with JavaScript.

2. Data Structure:

=> JSON represents data as key-value pairs enclosed in curly braces {}.

=> Each key is followed by a colon : and its corresponding value. Multiple key-value pairs are separated by commas ,.

3. Data Types:

=> JSON supports several data types, including:

Strings: Enclosed in double quotes "".

Numbers: Integer or floating-point numbers.

Booleans: true or false.

Arrays: Ordered lists of values enclosed in square brackets [].

Objects: Unordered collections of key-value pairs enclosed in curly braces {}.

Example:

Here's an example of a JSON object representing information about a person:

json

Copy code

```
{
   "name"      : "John Doe",
   "age"      : 30,
   "isEmployed": true,
   "languages" : ["English", "Spanish", "French"],
   "address"   : {
                    "city": "New York",
               "country": "USA"
                     }
}
```

4. Widely Supported:

=> JSON is supported by virtually all programming languages and platforms.

=> Most modern programming languages provide built-in support for parsing and generating JSON data, making it easy to work with JSON in a wide range of applications.

5. Used for Data Exchange:

=> JSON is commonly used as a data interchange format in web development, particularly for exchanging data between a client (e.g., web browser, mobile app) and a server. It is often used in web APIs to send and receive structured data over HTTP.

* @ResponseBody:-

• It is an annotation present in org.springframework.web.bind.annotation package.

• The @ResponseBody annotation in Spring boot indicates that the return value of a method should be serialized directly to the HTTP response body. It is commonly used in RESTful web services to return data in JSON or XML.

• In simple terms, the @ResponseBody annotation in Spring boot tells Spring to directly mreturn the method's return value as the body of the HTTP response, without rendering a template or view. It's handy for building RESTful APIs because it allows you to easily send data, typically in JSON or XML format, back to the client.

* @RestController:-

• It is an annotation present in org.springframework.web.bind.annotation package.

• @RestController is an annotation that combines @Controller and @ResponseBody.

• It is used to indicate that a particular class is a a controller where methods return data to be directly written into the response body as opposed to rendering a view.

• This annotation is commonly used in buliding RESTful web services where the focus is on returning data(usually in JSON format) rather than rendering HTML views.

* @RequestBody:-

• It is an annotation present in org.springframework.web.bind.annotation package.

• This annotation is used to extract data from the body of an HTTP request.

• The @RequestBody annotation is used in Spring Boot applications to bind the body of an HTTP request to a method parameter in a controller class.

• This is commonly used to receive complex data structures like JSON or XML in POST or PUT requests.

* @RequestParam:-

• It is an annotation present in org.springframework.web.bind.annotation package.

• This annotation is used to extract data from the query parameters of a URL.

• It binds the value of a query parameter to a method parameter in a controller class.

• ex:-

example.com/api/user?id=123

=>@RequestParam("id") would extract the value "123" from the "id" parameter.

* @PathVariable:-

• It is an annotation present in org.springframework.web.bind.annotation package.

• The @PathVariable annotation is used to extract values from the URI template of a request mapping and bind them to method parameters in a controller handler method.

• Path variables represent dynamic parts of the URL and are typically specified within curly braces {} in the URI template. For example, in the URL /users/{id}, id is a path variable that can take different values.

* Optional Class:-

• It is a class present in java.util package.

• It is designed to address the issue of handling ptentially nullable values without explicitly checking for null, thus reducing the risk of NullPointerException.

• It provides methods for dealing with the presence or absense of a value, allowing you to perform operations such as mapping, filtering and chaining.

* ResponseEntity:-

• It is a class present org.springframework.http package.

• ResponseEntity is a class in the Spring framework which is used to represent the entire HTTP response.

• It allows you to control the HTTP response status, headers, and body.

• With ResponseEntity, you can customize the response instead of just returning the body.

* @Query:-

• It is an annotation present in org.springframework.data.jpa.repository package.

• In Spring Boot, the @Query annotation is used in conjunction with Spring Data repositories to define custom JPQL (Java Persistence Query Language) or native SQL queries to be executed by the underlying persistence framework (JPA or Hibernate).

* Exception Handling:-

• Exception handling in a Spring Boot application involves capturing, processing, and responding to exceptions that occur during the execution of the application.

• Proper exception handling ensures that the application remains stable and resilient, providing meaningful error responses to clients when issues arise.

* Exception handling in a Spring Boot application:-

1. Global Exception Handling:

• Spring Boot provides mechanisms for global exception handling, allowing you to define how exceptions are handled across the entire application.

• One common approach is to use @ControllerAdvice along with @ExceptionHandler to define global exception handling logic in a centralized location.

2. @ControllerAdvice:

• It is an annotation present in org.springframework.web.bind.annotation package.

• In Spring Boot, @ControllerAdvice is used to define global exception handlers for Spring Boot controllers.

• It allows you to centralize exception handling logic in a single class that can be shared across multiple controllers.

• With @ControllerAdvice, you can handle exceptions thrown by controller methods, providing customized error responses or performing additional processing.

• Steps to use @ControllerAdvice in a Spring Boot application:

(i). Create a Controller Advice Class:

=> Start by creating a class annotated with @ControllerAdvice. This class will contain methods to handle exceptions thrown by controllers in your application.

(ii). Define Exception Handling Methods:

=> Inside the @ControllerAdvice class, define methods annotated with @ExceptionHandler to handle specific types of exceptions. You can create multiple methods to handle different types of exceptions or group related exceptions together.

(iii).Customize Error Responses:

=> Within each @ExceptionHandler method, customize the error response based on the type of exception. You can return different HTTP status codes, error messages, or additional details in the response body.

(iv). Apply Global Exception Handling:

=> Spring Boot automatically detects and applies the global exception handling defined in classes annotated with @ControllerAdvice across your application.

=> When an exception is thrown by a controller method, Spring Boot invokes the appropriate @ExceptionHandler method in the @ControllerAdvice class to handle the exception.

3. @ExceptionHandler:

=> It is an annotation present in org.springframework.web.bind.annotation package.

=> @ExceptionHandler is used to define methods that handle specific types of exceptions.

=> When an exception of the specified type is thrown during the execution of a controller method, the corresponding @ExceptionHandler method is invoked based on the method parameter to process the exception and return an appropriate error response.