**Spring Boot**

It is a framework that is built on the top of Spring Framework. It provides an easier and faster way of application development.

It is a spring module that helps to rapidly develop the application [ RAD ].

Spring Boot = Spring Framework – xml configuration + Embedded Server

Spring Boot is also used to develop microservices.

Spring Boot can be developed by using

1. Java
2. Kotlin
3. Groovy
4. Spring Initializer
5. STS - Spring Tool Suites 4 for Eclipse
6. Eclipse +STS plugged in.
7. IntelliJ Ultimate
8. STS with VS Code

**Spring Initializer**

<https://start.spring.io/>

**Spring Boot JPA**

Interface CrudRepository<Entity, ID>

This contains all abstract method to perform CRUD operations.

<https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/repository/CrudRepository.html>

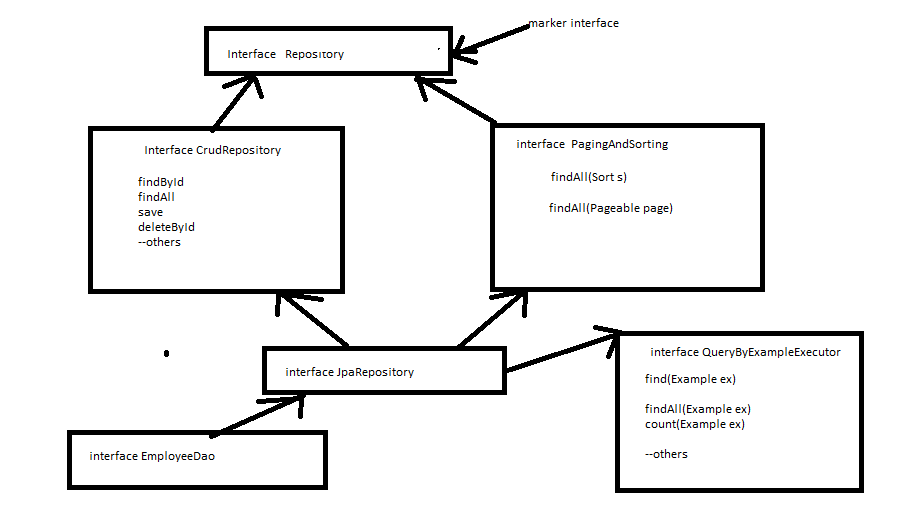
findById(ID id)

findAll()

deleteById(ID id)

count()

Spring Boot runtime will supply the implementation for all abstract methods from your repository.



**Spring REST**

Web services are into practices to expose the business functionalities that can be easily integrated with other applications [ enterprise app or UI app ] developed in any languages.

This is towards making an interoperable application.

Web Services are two types

1. XML Web Service / SOAP Web Service
2. RESTful Web Service / REST Service [ REpresentational STateful Web Service ]

Our preferred service is REST Service and JSON data as preferred mode of data transportation

REST has become the de-facto standard for building web services on the web because they are easy to build and consume.

* Spring Web is the dependency to be added in pom.xml

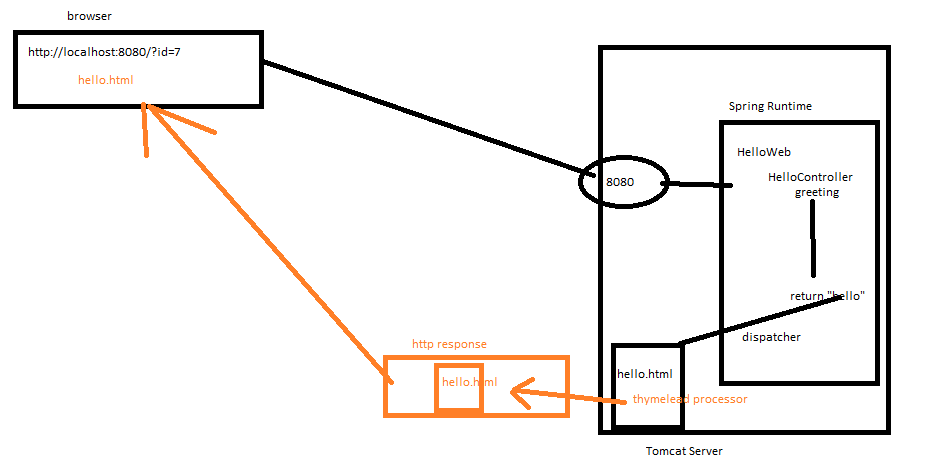
<dependency>

<groupId>org.springframework.boot</groupId>

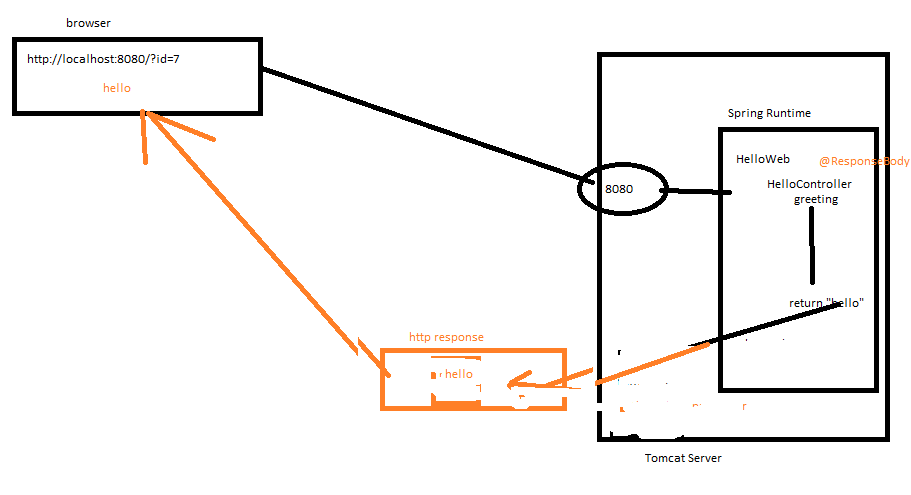
<artifactId>spring-boot-starter-web</artifactId>

</dependency>

* Use @Controller @ResponseBody combination on the rest class that returns data.



@RestController = @Controller @ResponseBody



Stereotype

1. @Component
2. @Repository
3. @Service
4. @Configuration
5. @Controller
6. @RestController

**Primitive**

byte, short, int , long default initialization value is 0

float, double 0.0

boolean false

char ‘\u’

**Reference**

Class, Enum, & Interface null

\*\*\*\*\* It gets initialized with default value for instance & static field only. Not for Local Variable.

**Transaction Properties “**ACID”

A : Atomicity - A transaction must be treated as a single unit of operation.

C : Consistency - This represents the consistency of the referential integrity

I : Isolation - There might be many transaction processing with the same data at the sometime. Each transaction should be isolated from others.

D : Durability : Once a transaction has completed, the result of this transaction have to be permanent.

* **Begin Transaction**
  + **Op1**
  + **Op2**
* **Commit Or Rollback**

Since all the transaction on database are initiated through backend programming , the transaction needs to started and end by backend logic. There are two ways to manage them.

1. Programmatic
2. Declarative

Spring AOP can also be use to manage transaction.

**Spring Transaction**

**Programmatic Approach**

org.springframwork.transaction.PlatformTransactionManager

* TransactionStatus getTransaction(TransactionDefinition definition)
* commit()
* rollback();

TransactionStatus

* createSavePoint()

TransactionDefinition

static final int

[**PROPAGATION\_MANDATORY**](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/transaction/TransactionDefinition.html#PROPAGATION_MANDATORY)

Support a current transaction; throw an exception if no current transaction exists.

static final int

[**PROPAGATION\_NESTED**](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/transaction/TransactionDefinition.html#PROPAGATION_NESTED)

Execute within a nested transaction if a current transaction exists, behaving like [PROPAGATION\_REQUIRED](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/transaction/TransactionDefinition.html#PROPAGATION_REQUIRED) otherwise.

static final int

[**PROPAGATION\_NEVER**](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/transaction/TransactionDefinition.html#PROPAGATION_NEVER)

Do not support a current transaction; throw an exception if a current transaction exists.

static final int

[**PROPAGATION\_NOT\_SUPPORTED**](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/transaction/TransactionDefinition.html#PROPAGATION_NOT_SUPPORTED)

Do not support a current transaction; rather always execute non-transactionally.

static final int

[**PROPAGATION\_REQUIRED**](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/transaction/TransactionDefinition.html#PROPAGATION_REQUIRED)

Support a current transaction; create a new one if none exists.

static final int

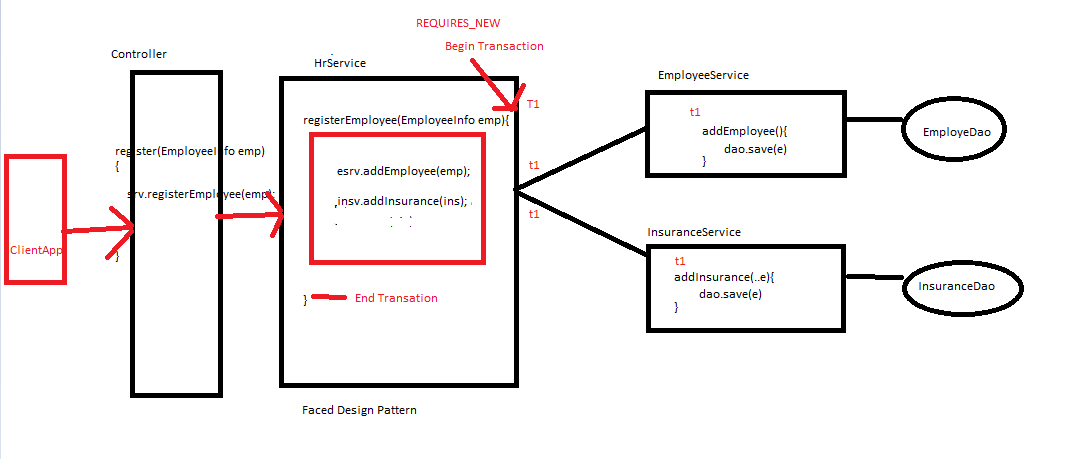
[**PROPAGATION\_REQUIRES\_NEW**](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/transaction/TransactionDefinition.html#PROPAGATION_REQUIRES_NEW)

Create a new transaction, suspending the current transaction if one exists.

static final int

[**PROPAGATION\_SUPPORTS**](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/transaction/TransactionDefinition.html#PROPAGATION_SUPPORTS)

Support a current transaction; execute non-transactionally if none exists.



**Spring Transaction HelloWebService Example**