Graphical user interface, text, application

Description automatically generated

**Problem:**

version compatibility

Need to deploy by its own

**Spring boot features**

Graphical user interface, text, application

Description automatically generated

1. **Auto configuration :--- that’s takes xml and annotation based configuration away**

Means no need to configure xml based or annotation based

For orm no need to configure data source

**2-- version compatibility problem will get resolved due to spring boot starter parent**

All types of version will be available in spring-boot-starter-parent

\

Graphical user interface, text, application

Description automatically generated

1. **Embedded servlet container:-----**

Graphical user interface, text, application

Description automatically generated

by default tomcat but we can switch to jetty and undertow also

1. **Spring boot actuators:--- give health check**

Table

Description automatically generated with low confidence

Autoconfig report:----will display everything configured automatically

Mappings :- will give all http request endpoint of application

Info:- will give info about application

Health :- display what is up in application

Metrics:- will show different metrics of application collected at different point and time

Now spring boot will use free for our application

First Project

**Spring boot starters in pom**

**Note:---**

Parent project only having the version info i.e **spring-boot-starter-parent**

Will add all dependency due to **spring-boot-starter** in pom.xml

The libraries are pull what is listed in dependency section but version of those libraries is pulled by parent

**Version incompatibility problem resolved**

**@SpringBootApplication**

Is a top level annotation which contain several other annotation

Graphical user interface, text, application, timeline

Description automatically generated

**@SpringBootConfiguration :-**

this class can be have several spring bean which will be available at runtime in application

**@EnableAutoConfiguration**

This annotation tells spring boot to automatically configure the spring application

Based on the dependencies it sees on the class path and do any configuration if it is required

Graphical user interface, application

Description automatically generated

Example :--

it will automatically configure the data source for in memory database

Spring web :--- spring boot automatically create dispatcher servlet

**@ComponentScan**

This annotation tells spring Scan through the classes and see which classes have respective annotation like @Component , @Service, @Repository and create bean of those annotation

By default it scan packages where it lives and also it will scan child or sub packages of that

**@SpringBootTest**

This tell spring boot go and search class that marked with @SpringBootApplication and creates a container which will contain all the beans

Do dependency injection

Diagram

Description automatically generated

@Repository

public class PaymentDaoImpl implements PaymentDao {

}

**@Repository** :- Object of this class be created and wherever it is required should be injected

**Note:--**

all stereotype annotation tell a bean of this class can be created and injected

**Autowired :-- bean will automatically wired at run time**

Means it will check that class which is marked by @Repository it means bean will created and which will wired in the variable which is marked by autowired

Autowiring feature of spring framework enables you to inject the object dependency implicitly. It internally uses setter or constructor injection. Autowiring can't be used to inject primitive and string values. It works with reference only.

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Spring will search PaymentDaoImpl which is marked @Repository , it will create bean and due to autowired it will wired into dao

Testing

If the dependency injection is working

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

**Tested dao being injected into service**

Graphical user interface, text, application

Description automatically generated

Spring data JPA

Create a repository per entity which will extends jpa repository interface

Using this we can perform crud operation

**NOTE:-**

In pom :- include spring-boot-starter-data-jpa

Will use in memory :- h2 d/b and

spring boot will automatically configure everything and will create table

1. If we want to use mysql then we need to define d/b url,user name and pw

In application.properties

1. By default jpa use hibernate orm

Pom.xml

Graphical user interface, text

Description automatically generated

Entity

Graphical user interface, application

Description automatically generated

Repos

Graphical user interface, text, application

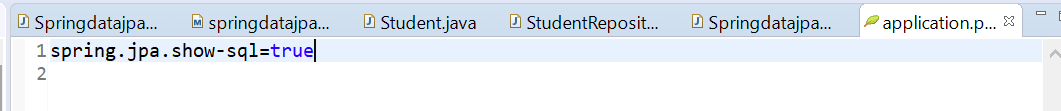
Description automatically generated

**Test the application**

Graphical user interface, text, application

Description automatically generated

**Requirement :---** to see all sql query created by hibernate behind the scene



It means orm tools will display sql query created behind the scene

A picture containing diagram

Description automatically generated

Drop table, create table if record not there then it will insert but if record is there then it will update

Create Rest CRUD api

Rest intro

1. Create product

Graphical user interface, text, application

Description automatically generated

1. Get

Graphical user interface, text, application

Description automatically generated

1. Put

Graphical user interface, text, application

Description automatically generated

1. Delete

Graphical user interface, text, application

Description automatically generated

**@GetMapping :--** **@PostMapping :-- @PutMapping :-- @DeleteMapping :--**

bind java method to incoming http method

Pom.xml

Graphical user interface, text, application

Description automatically generated

Product.java

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

**Configure the data source**

Graphical user interface, text, application

Description automatically generated

**Spring-boot-auto-configuration**

It will auto configure everything and at run time spring will look into it and enable all configuration

Simply put, the Spring Boot autoconfiguration represents **a way to automatically configure a Spring application based**

**on the dependencies that are present on the classpath**. This can make development faster and easier by

eliminating the need for defining certain beans that are included in the auto-configuration classes

Graphical user interface, text, application

Description automatically generated

Spring boot will do this for us

**Note**

**Jackson :-** will do serialization and deserialization of json object

**ContextPath**

By default spring boot will not add any contextpath -----localhost:8080/**products/**

**Adding context path:--**

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

**// output**

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated

**Retrieve a single product:-**

Graphical user interface, text, email

Description automatically generated

**Update :**  in case of update we need to send a id

Graphical user interface, text, application, email

Description automatically generated

Text

Description automatically generated with medium confidence

**Changing the embedded server**

**By default spring boot uses embedded server---tomcat**

**First step**

Exclude tomcat dependency from web dependency

**Second step :-**

include other dependency

Consume a rest service using RestTemplate

Get ---getForObject------------

expect a url and type of java object expecting back. It will hit a url and get a json response

And it will convert the response into a java object and hands it to over application

Post --- postForObject

**Create java restful client using RestTemplate**

Rest Template is used **to create applications that consume RESTful Web Services**. You can use the exchange() method to consume the web services for all HTTP methods

**Requirement :-** Retrieve a single product using RestTemplate

**Note :--**

when json come back , response will automatically converted or deserialized into product object

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

**Http post using RestTemplate**

**Requirement :---** how to create a product using RestTemplate

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generated with medium confidence

**Http put using RestTemplate**

**Requirement :-** update a product

Initially

Text

Description automatically generated

Graphical user interface, text, application

Description automatically generated with medium confidence

Output

Graphical user interface, text

Description automatically generated

Profiles

Spring boot profiles like configuration(db,rest url, queues and topics) could be different across dev,QA,staging

And production that is where profiles comes into pictures allow us to use different configuration across different

Environment

**Note**

We need to activate the profile when we run the application

**Extracted out url from restTemplate**

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated

**Create and activate a profile**

**Note :-**

above url configure in application.properties will be different across dev, test and staging env

**Step 1 :-** To use spring boot profile we need to create multiple configuration file

Graphical user interface, text, application

Description automatically generated

**Step 2:-** to activate a particular profile

If I want to use dev profile

Graphical user interface, text, application

Description automatically generated

Test will get failed because dev server not exists

Graphical user interface, text, application

Description automatically generated

Note :-

this is not a good way of changing profile manually since if testing team doing test they will not go to code

And change it and there is a way to change it dynamically

Logging

Graphical user interface, text, application

Description automatically generated

Log to file

Note :--

By default all logs go to console.

by using this will get all console log in file

Graphical user interface, application

Description automatically generated

Change log level

Logging.level.root=error

We can do logging at application level also

Logging.level.com.bharath.springweb.controller.ProductController = error

Health Check and metrics

**Health check :**--- whether our application is up or down. If down then why it is down

**Application configuration** :---- we should have complete access of configuration that application is using

At runtime

**Application metrics :--** the exact health of application what kind of memory is using

**Key application events :---** application receiving or sending a message to a queue or it saving it in a d/b

Note:--

Spring boot provides all these metrics free to us

For this we need to enable spring boot actuators for our project

Once we configure it spring boot exposes several rest points

**By default health and info information exposed out**

**Enabling health checks**

**Need to add :-** spring-boot-starter-actuator to our project

Graphical user interface, text, application

Description automatically generated

**Expose Health details**

**Note:-**

Spring boot provides various properties for each of the end points that exposes out under actuator

Graphical user interface

Description automatically generated with medium confidence

Text

Description automatically generated with low confidence

**Add Build info**

Want to add build info information to the project like version etc

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

**Expose Other endpoints**

**To expose all other endpoints we need to include in application.properties**

Graphical user interface, text, application, email

Description automatically generated

Star represent include all the endpoints

**This time we are able to see many other metrics**

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Note:--- all the name but no value. We can copy the name and paste in url and hot we will get value

Graphical user interface, text, application

Description automatically generated

**Note:**

Metrics are very useful when we are doing performance tuning information

**Expose other endpoints**

**Note**

All health information is coming from spring but if we want to customize it

We need to implement certain interface for this :---custom health indicator

Graphical user interface, text, application

Description automatically generated

**Custom info added**

Diagram

Description automatically generated

**Spring Security**

**Enabled:----** need to add

**Spring-boot-starter-security**

**Note**

Automatically basic authentication will be enabled

By default :--- username --user

Password :--- will be there in console

**Thymeleaf**

Instead of using jsp will use template

Thymeleaf will create .html pages which will have both static and dynamic code

**Thymeleaf in action**

Create the project

Create the controller

Create the template

**Thymeleaf** is a modern server-side Java template engine for both web and standalone environments.

*From <*[*https://www.thymeleaf.org/*](https://www.thymeleaf.org/)*>*

Thymeleaf is **a Java library**. It is an XML/XHTML/HTML5 template engine that is able to apply a set of transformations to template files in order to display data and/or text produced by your applications

Graphical user interface, text, application

Description automatically generated

**Note:**

This controller forward the request to templates

Template method return hello and template engine look under template by name hello.html

And it will send the html to the web browser

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, chat or text message

Description automatically generated

**sending data to template**

Dynamic data passing

Graphical user interface, text, application

Description automatically generated

Note:--

In this template we are passing modelAndView object

We can render the message inside data.html

**Create a template**

Graphical user interface, text, application

Description automatically generated

**Note:**

To use thymeleaf syntax we need to include name space of thymeleaf

Graphical user interface, application

Description automatically generated

**Disable cache**

**Note:**

When thymeleaf render html entire file come into cache

Graphical user interface, text, application

Description automatically generated

We change the message but still it will not reflect in browser although it is static message

To reflect this we need to turn off thymeleaf caching

Graphical user interface, text, application, Word

Description automatically generated

Graphical user interface, text, application

Description automatically generated

**Sending Object Data**

Text, application

Description automatically generated

Text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, chat or text message

Description automatically generated

**Rendering Multiple Records**

**Loop list of object using thymeleaf**

A picture containing application

Description automatically generated

Graphical user interface, text

Description automatically generated with medium confidence

Graphical user interface, text, application, chat or text message

Description automatically generated

**Create a html form**

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generated

Graphical user interface, application

Description automatically generated

**Process form data**

Graphical user interface, text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

**Database Caching**

**Storing the data of an object in temporary location**

**Note:-**

Cache will be refresh every time object is updated

When client read the same exact data the orm tools need not need to go D/b.

It will read it from caching

It improves the performance of our application

**Steps to enable caching**

Spring uses hazlecast, eh cache. Here will use hazlecast

Hazlecast will stored object and maintain it

**Steps:**

**Add dependencies**  spring boot and hazlecast

**Create cache configuration** config

**Enable and use caching** @EnableCaching

**Evict cache** when should be the cache will get clean because overtime the cache size will get increased and application

Can get crashed

For evict cache hazlecast uses some policy like

**LRU--** least recently used means which object use less will get removed

From cache after some time

**LFU,NONE,Random**

**Add Maven dependencies**

Graphical user interface, text, application

Description automatically generated

**These three dependency are required**

First one :---to enable cache

Second:-- to support hazelcast jar

Third:---it will support into spring also

**Configuring caching**

Graphical user interface, text, application

Description automatically generated

**Note:-**

We are returning a bean of type hazelcast config

**Enable and use caching**

**Step 1-**

Graphical user interface, text, application, email

Description automatically generated

**Note:--**

**We need to implement Serializable for caching to work**

**Step -2**

Graphical user interface, text, application, email

Description automatically generated

**Note:--** internally hazlecast serialize our models

**Step 3-**

Graphical user interface, application

Description automatically generated

**@Cacheable(provide cache name)**

Now test the application caching is enabled

Spring Batch

Note:

Job we create and each job comprised of multiple task

Task 1- copying data

Task 2 - transferring files

Task 3- generate reports

These job can be scheduled at a certain time

Step

ItemReader -----------read from d/b or message queue

ItemProcessor-----------apply some business login

ItemWriter----------------write to d/b or message queue

Job all these details are stored in d/b using jobRepository

**Job launcher:**--- is responsible for taking the job and executing it

Job:---multiple step:---multiple task---each step responsible for IR,IP,IW

**Batching API**

**A job** is made of one or more step

And to create a step we need to create IR,IP,IW

Job listener :---before job and after job

Batch config will have all job and step

Diagram

Description automatically generated

Once we have IR,IP,IW will use stepBuilderFactory to create a step to which we pass R,P,W once we step in place

Will use a JobBuilderFactry and pass a step to get a job and will configure all these in java based configuration file

And to launch this will use jobLauncher

**Create Project**

Graphical user interface, application

Description automatically generated

**Create Reader**

Responsible for reading data and return some data

Graphical user interface, text, application

Description automatically generated

**Implement Processor Writer and listener**

**Processor**

Graphical user interface, text, application

Description automatically generated

**Writer**

Graphical user interface, text, application

Description automatically generated

**Listener**

Graphical user interface, text, application, email

Description automatically generated

**Configure the beans**

Graphical user interface, text, application

Description automatically generated

**configure the step**

Graphical user interface, text, application

Description automatically generated

**Configure the jobs**

**Note:**

**Incrementor** will provide new id for each job

**RunidIncrementor ;** that will automatically increment job id

Start will take step as argument

Text

Description automatically generated

.build :--- responsible for creating a job using all these information

**Write a test**

Graphical user interface, text, application

Description automatically generated

**Batching in action**

Graphical user interface, text, application, email

Description automatically generated

This tells we want spring batch support

**Need to put this ( not required)**

Graphical user interface, text, application, Word

Description automatically generated

Output

Text

Description automatically generated

**Configuring Chunk size**

**Change chunk size as 2**

Text

Description automatically generated

**//output**

Graphical user interface, text

Description automatically generated

**Chunk size 3**

Text

Description automatically generated with medium confidence

**CSV to Database**

**Requirement :-** how to copy data from csv file to d/b using spring batch application

**LineMapper** :--- knows to read one line at a time

**LineTokenizer** :---convert into comma seperated

**FieldSetMapper** :--- convert into model object

**JDBCBatchItemReader** :--- responsible for taking the object and writing into d/b

Chart, bubble chart

Description automatically generated

Create the project and model