JDK, JRE, JVM:



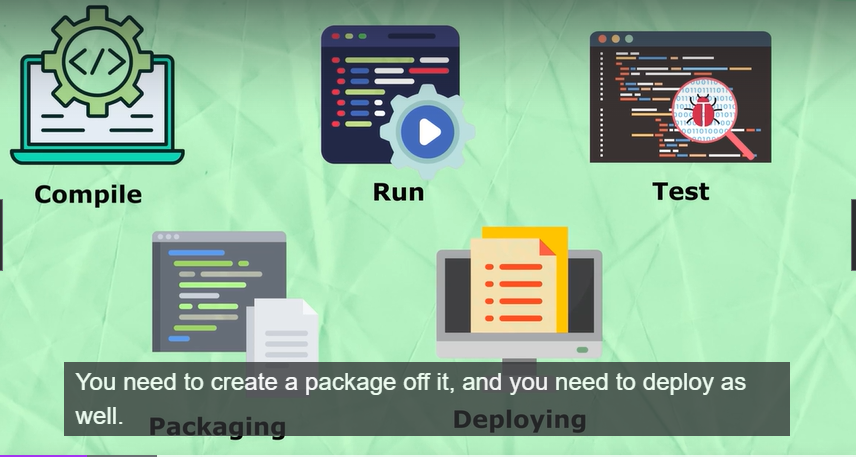
JDK – contains all the necessary tools required for developing a java based application.

JRE – Aids in running/executing the java application.

JVM – aids in running the bytecode (interpreter)zint

Maven:

* It is a product management tool.



POM – Project Object Model

The pom. xml file contains information of project and its configuration information like dependencies.

Maven lifecycle:  
- when we click on clean it will delete all the files

* Package it will install new files

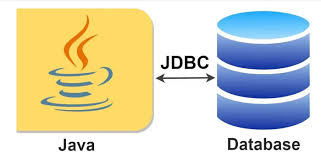
All the dependencies that you need will be present in the maven repository (https://mvnrepository.com/)  
  
all the dependencies should be placed inside the dependencies tag in POM.xml

Maven Archetype:

* It’s a template for your POM file

JDBC (Java Database Connectivity)  
- it is a part of JDK

* It gives you an API to connect with the database



Steps (JDBC Steps):

1. Import packages
2. Load driver
3. Register driver
4. Create connection
5. Create statement
6. Execute statement
7. Close connection

To get the driver for postgres jdbc just to a search postgres jdbc driver

Or you can also use maven repository

After downloading it we must add the jar file in the library section of project structure under files in the IDE

Spring:

* Spring boot works on spring framework

What is spring framework:

* We can consider spring to be an ecosystem
* It is used to develop enterprise application

IoC – (Inversion of Control)

* Simply does mean we are inverting the control
* As a programmer our focus should be on the business logic and not the object creation
* So we’re inverting the control of object creation and the flow to spring
* In spring this is called IoC Container.
* So now the object creation is handled by the IoC and the objects are present in the IoC Container
* Someone has to inject those object into the application
* This is the place were the Dependency Injection (DI) comes into the action

Spring boot is built on top of spring

Configuring a project through spring is a very tedious task

Therefore, the existence of spring boot framework came into picture.

Start.spring.io – spring initializer

Beans;  
- any object created by spring is called beans

Wiring:

Autowiring helps in DI when the class doesn’t have the ApplicationContext

Exploring Spring:

* While creating a new project go for quick start in archetype since it gives you the basic structure
* Beans are objects that are managed by spring framework

While creating the xml file (spring101)

* The bean tag should have the id that we can use later to refer to that bean
* The class that we use should contain the full patch ie. It should also include the package
* For bean configuration don’t by heart it, you can always refer to the documentation by googling spring bean configuration

Scopes:

Singleton and prototype

We have to change in the xml

Setter Injection:

* We can use the value attribute for primitive types like int
* But for an another object we have to use the ref attribute (for more information refer to Spring101 xml file)
* Getter and setter methods are required for injection

Constructor injection:

If we want to inject the values through a parameterized constructor we can use constructor-arg instead of property tag in the xml file

Lazy init bean:

When we initialize the application context (refer to Spring101) all the objects would be created. Sometimes there would be a use case were we don’t want to create all the objects at once.

Therefore in the bean definition we just add lazy-init at set it to true. So from here onwards the object will be created only if it is required.

Inner bean:

(refer to spring101)

Alien bean uses a ref to use laptop bean

But what if we want only alien to use laptop and no other class should have access to laptop bean

In that case we just remove the ref attribute and add the entire bean definition inside the property tag.

Spring Boot:

Annotations:

@Value – defaulting a value

@Component - tells the Spring framework that this class should be treated as a Spring bean, allowing the framework to automatically detect, instantiate, and manage it throughout the application.

@Autowired - automatically inject dependencies into a Spring-managed component (DI)

@Primary

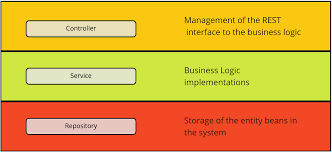
@Qualifier - the @Qualifier annotation is used alongside @Autowired to resolve ambiguity when multiple beans of the same type are available in the Spring application context. It helps specify exactly which bean to inject when there is more than one candidate.

@Scope – defines whether the bean is a singleton or prototype

Server:

Server has multiple layers

* Controller – handles the incoming and outgoing request and response from and to the client
* Service – Service takes care of all the computations of the server
* Repository – handles the process of getting the data from the database



Application.properties is a file that you provide all the necessary details required configurations for application setup. As soon as the application start this is the place were spring boot will look into

Spring JDBC:

Refer to SpringJDBC

Spring web:

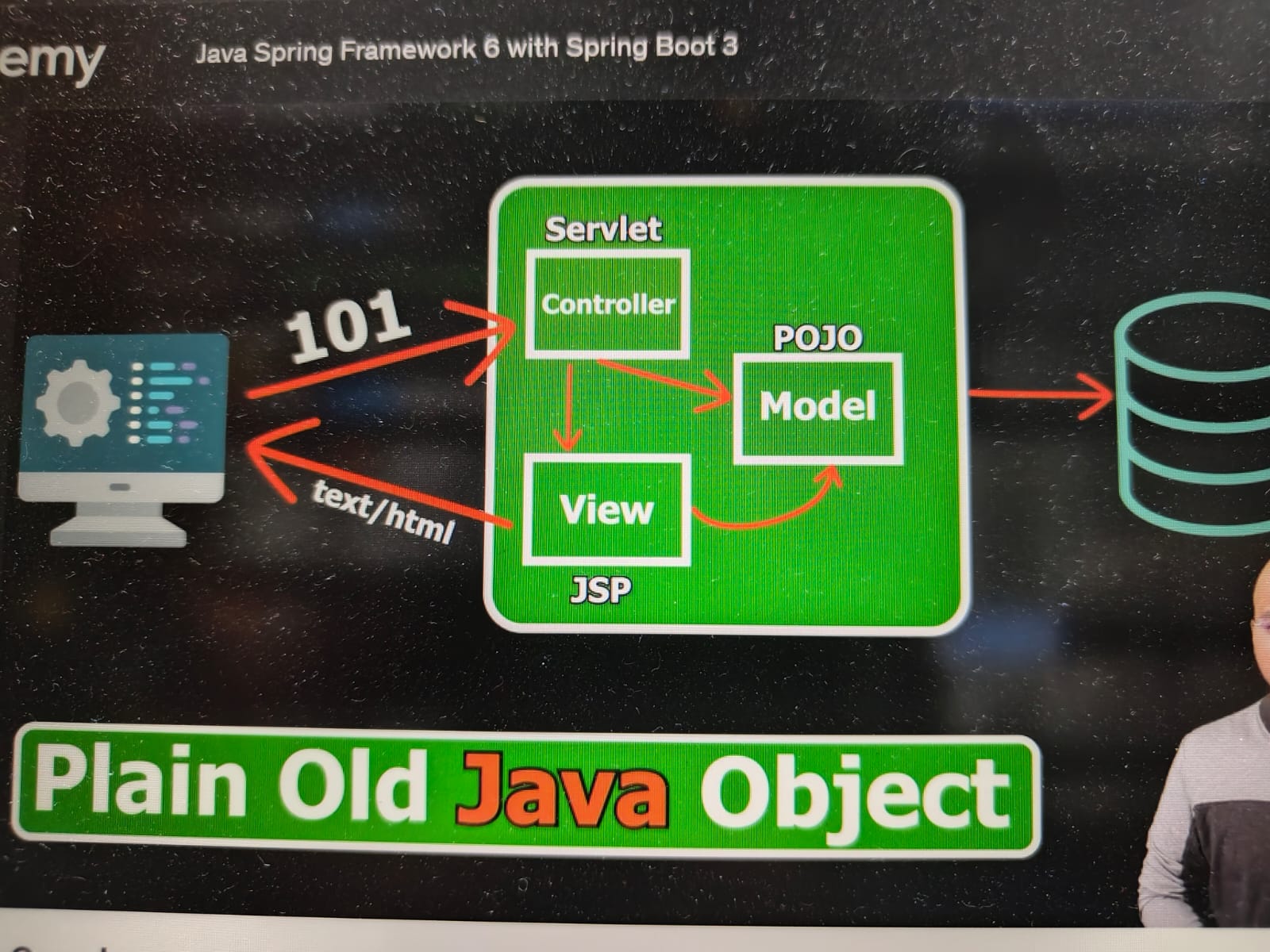
Servlets – server components that accepts the request, process it and sends back the response

We need a web container to run servlets

JSP (Jakarta web pages):

Basically, a html page that contains java inside it

MVC:



Spring Web:

Refer springweb101

Spring web dependency uses embedded tomcat inside it

By default tomcat will have the port as 8080

Spring will look for the webapp folder for the homepage

Request mapping:

We use @RequestMapping annotation

By default springboot doesn’t support jsp pages

We have to use an another module called tomcat jasper

Make sure you use the same version as the tomcat server (check in the external libraries folder)

If we want to write a java code in a jsp page, we just have to encapsulate the code with <% %>

Dispatcher Servlet:

In spring we have a dispatcher servlet which is the first layer of the servlet that redirects the incoming request to the desired controller (like ‘add’)

DTO – Data Transfer Object

These are the objects that we use to transfer data between layers (repo, service and controller)

REST (Representational State Transfer):  
Postman – it is a rest client

State – the current snapshot of the database

The request should have noun instead of actions

HTTP methods:

GET – to fetch data (read data)

POST – create data

PUT – update data

DELETE – delete data

Jackson-core library deals with converting java objects to JSON

ORM and JPA:

ORM – Object Relational Mapping

We will be connecting the object world to the relational world

ORM tools is the one that we use for the mapping ex. Hibernate

JPA:

Java Persistence API

While mapping the objects to the table using ORM tools like Hibernate we also have a possibility to switch to a different ORM tool. In that case we must switch the entire code so that it supports the new ORM tool. To overcome this issue, we use JPA.

Note:

This line in the application.properties tells the jpa to create a table if there isn’t any table

spring.jpa.hibernate.ddl-auto=update

Microservices:

Monolithic architecture – were we have all the services in one place

Cloud Native and Cloud Ready:

Cloud ready – we have an existing application that runs on on premise. When we need to deploy the same application onto the cloud, we have to make some changes to the application

Cloud native – these are the applications that we develop to be solely deployed in the cloud

Inorder to build a cloud native application we have some rules and those are called 12-factor App

<https://www.geeksforgeeks.org/what-is-twelve-factor-app/>

API Gateway(reverse proxy server):

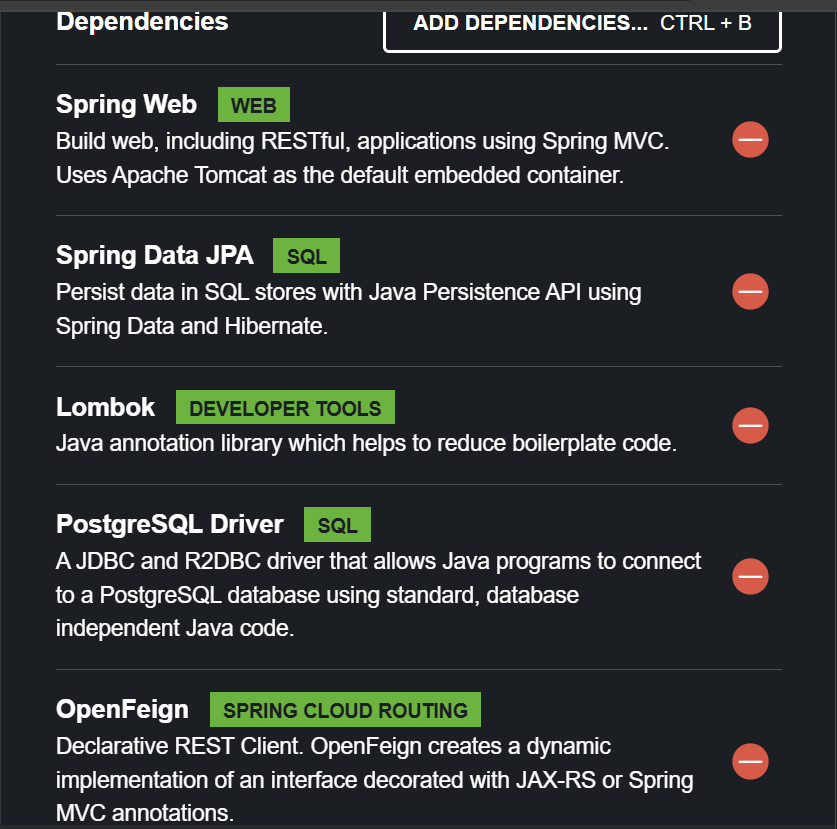
This acts as a gateway between the client and the microservices. This properly redirects the client requests to the desired microservice.

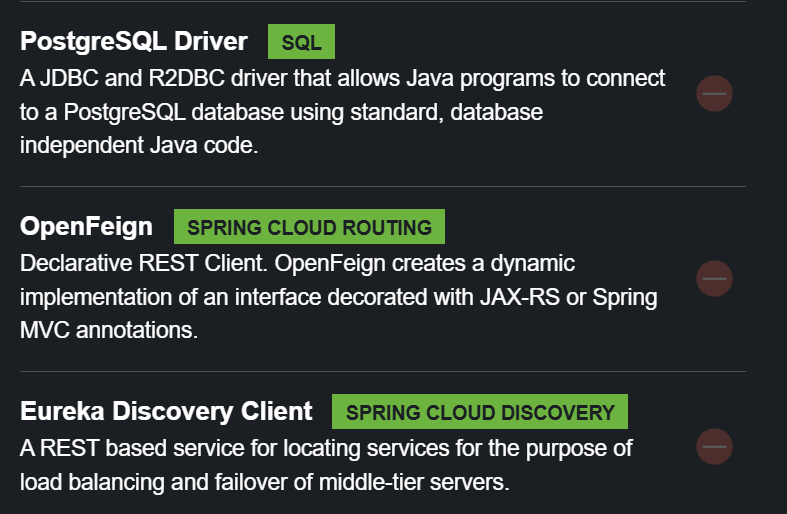
Load balancers: (Refer QuizApp)  
In this example we have quiz service and question service. Lets say quiz service is scaled more than the question service. Now to properly redirect the requests between services we need a load balancer.

Registry:

Every service can know where the other service is present.

Dependencies:





Note:

To run multiple instance go to the configurations of the main file

Modify option > Add VM option

Add -Dserver.port=portnumber

Then click apply

Note:

For every microservice to be discoverable by the other microservice, they must register themselves. This is called as service registry.

While registering the microservice to the service registry we have to provide a good name for the service. This is done in the application.properties file