- Let’s create a restController:

1. Configure the project at spring initializer
2. Download the zip file
3. Unzip the file
4. Import the project in IDE
5. Run the application

- WhitelabelErrorPage

We have not added any real code to our project that’s why we are getting this page.

6. Create a package -> rest
7. Create a class FunRestController
8. Add @RestController Annotation on class
9. Create a method which will return a string (“Hello Team!!”)
10. Annotate it with @GetMapping(“/”)
11. Run the main application

```java
package com.flynaut.springboot.demo.primaryApp.rest;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

import java.time.LocalDate;

@RestController
public class FunRestController {

    //expose endpoint - "/hello" that returns "Hello Team"
    @GetMapping("/")
    public String sayHello(){
        return "Hello Team!!";
    }
    //This method will handle GET request at "hello" endpoint

    @GetMapping("/date")
    public LocalDate date(){
        LocalDate localDate = LocalDate.now();
        return localDate;
    }
    // return yesterdays date
}
```

URL – Uniform Resource Locator

http://localhost:8080

http://www.abc.com:8080/college

http : Application Layer Protocol (Hypertext Transfer Protocol)

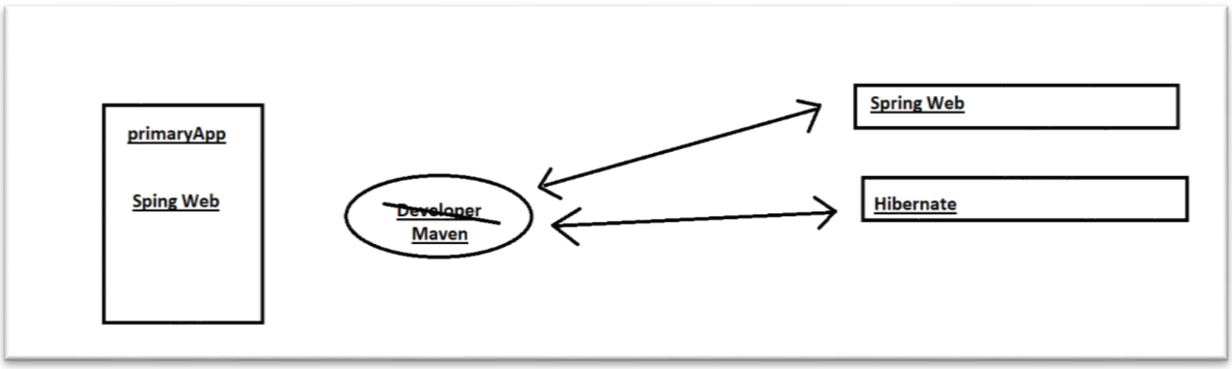
www.abc.com : DNS qualified hostname/IP address (used to resolve the host problem)

8080: TCP port(used to identify the port)

/college: URI (Uniform resource identifier) or path

MAVEN:

- Project build tool
- Used for build management and dependencies



- Tell maven the projects we are going to work on(dependencies)
- Go out and download the jar files for us

- Maven Project Structure

Maven uses standard directory structure

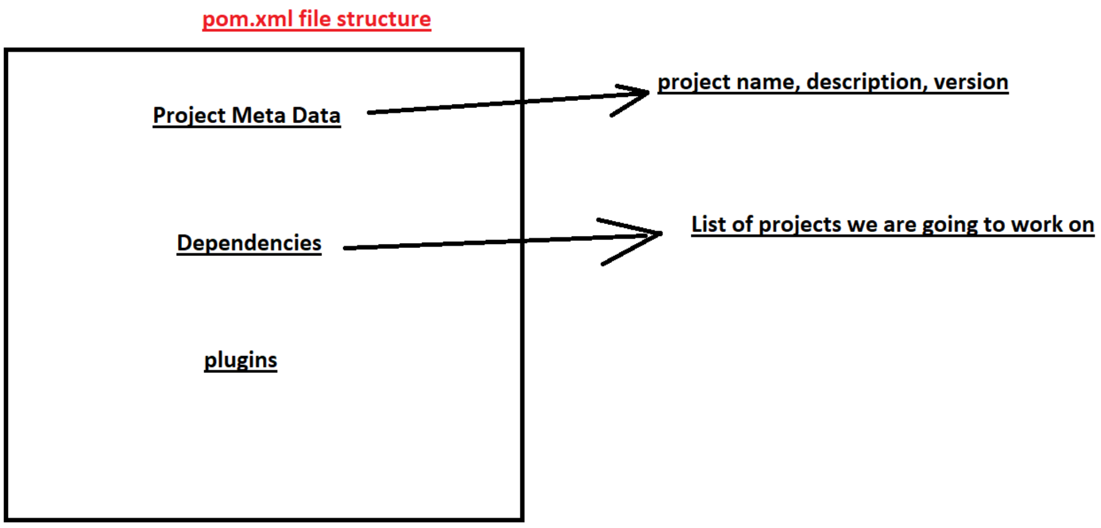
Directory	Description
src/main/java	Our source code
Src/main/resources	Properties/ configuration files used by our app
src/test	Unit testing code and properties
target	The destination directory for compiled code(Automatically created by maven)
pom.xml	Maven Configuration File

POM.xml

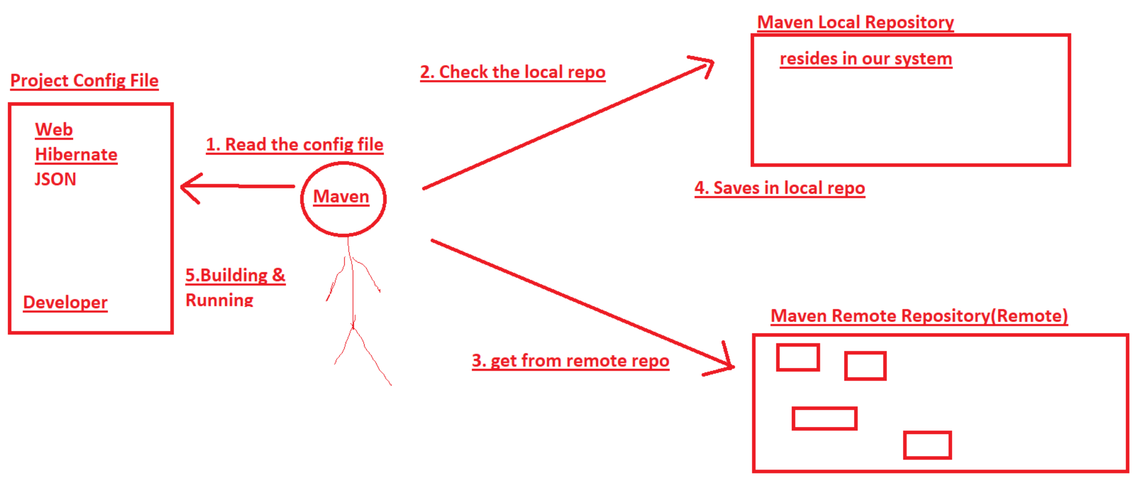
Project Object Model File

- Configuration File for our project
- Shopping List
- Located at root of project

POM file Structure


pom.xml file structure
Project Meta Data
project name, description, version
Dependencies
List of projects we are going to work on
plugins

Maven flow:



Maven Central – Remote Repository

Project Coordinates:

- To uniquely identify a project
- Similar to GPS coordinates for home- longitude/latitude
- Precise address of our home(city, street, home no.)

```
<groupId>org.springframework.boot</groupId>         <City>
<artifactId>spring-boot-starter-parent</artifactId> <Street>
<version>3.4.2</version>                            <Home No.>
```

GroupId: Name of company, group, organization

ArtifactId: Name of our project : primaryApp

Version

If we want to manually add the dependencies, we need GAV

Maven Wrapper Files:

mvnw

mvnw.cmd/ mvnw.sh

- mvnw allows us to run a maven project
- No need to have maven installed in our path

Mvnw.cmd = for windows

Mvnw.sh = for mac/linux

Application.properties file

By default, SB loads properties from this file

- Created by spring initializer
- It is empty at beginning

We can add properties in this file:

Server.port=7070

To add our own custom properties

Task ->

Why Starters?

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```