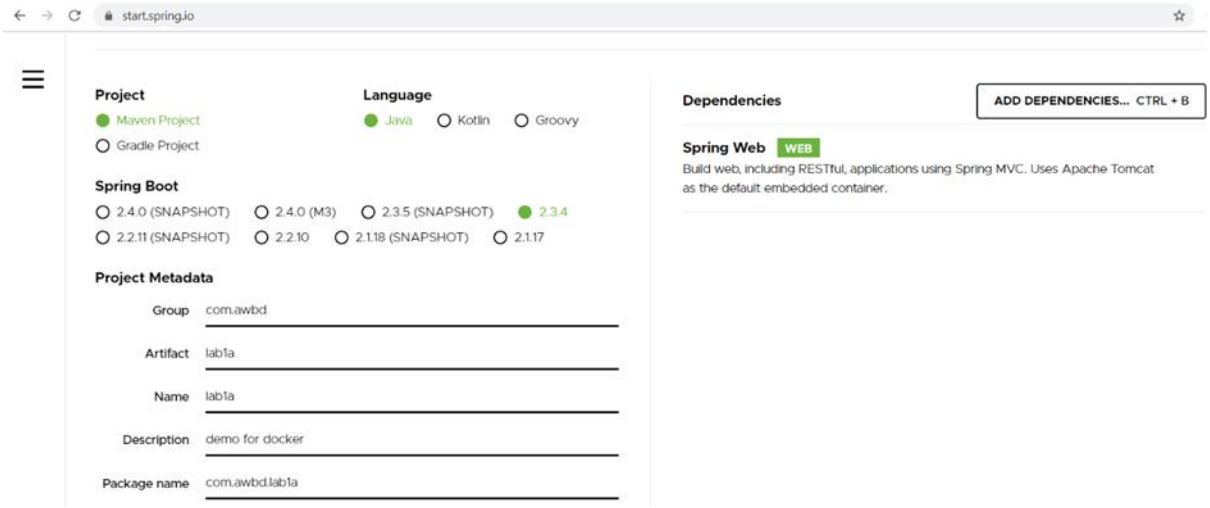
Practice 2: Spring Boot Introduction, Tools

1. Use **Spring initializr** to generate a maven project with a simple Spring Boot application.
https://start.spring.io/



2. Open the project in IntelliJ IDE: File – New Project from Existing Sources. Check java.version in pom.xml file.

```
<properties>
      <java.version>11</java.version>
</properties>
```

3. Add new java class, Lab2Application.

```
package com.awbd.lab2;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication
@RestController
public class Lab2Application {

   @RequestMapping("/home")
   public String home() {
      return "Hello World!!!";
   }

   public static void main(String[] args) {

      SpringApplication.run(Lab2Application.class, args);
   }

}
```

@SpringBootApplication shortcut for adding:

@Configuration

Info

allows to register beans in the context or import other configuration classes.

@EnableAutoConfiguration

automatically configure Spring application based on jar dependencies. For example, if HSQLDB is on project classpath, and you have not manually configured any database connection beans, then Spring Boot auto-configures an in-memory database.

@ComponentScan

enable @Component scan on the package where the application is located

[1][2]

@RestController shortcut for:

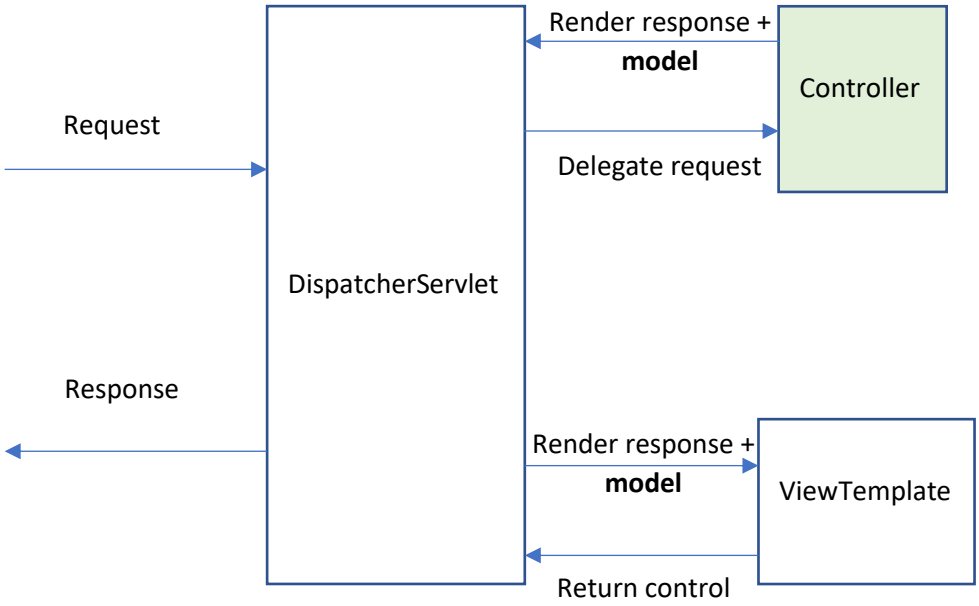
@Controller

More „readable" specialization of @Component. There are two other important specializations of @Component: @Service and @Repository.
DispatcherServlet will scan classes annotated with @Controller, for **@RequestMapping** methods.

@ResponseBody

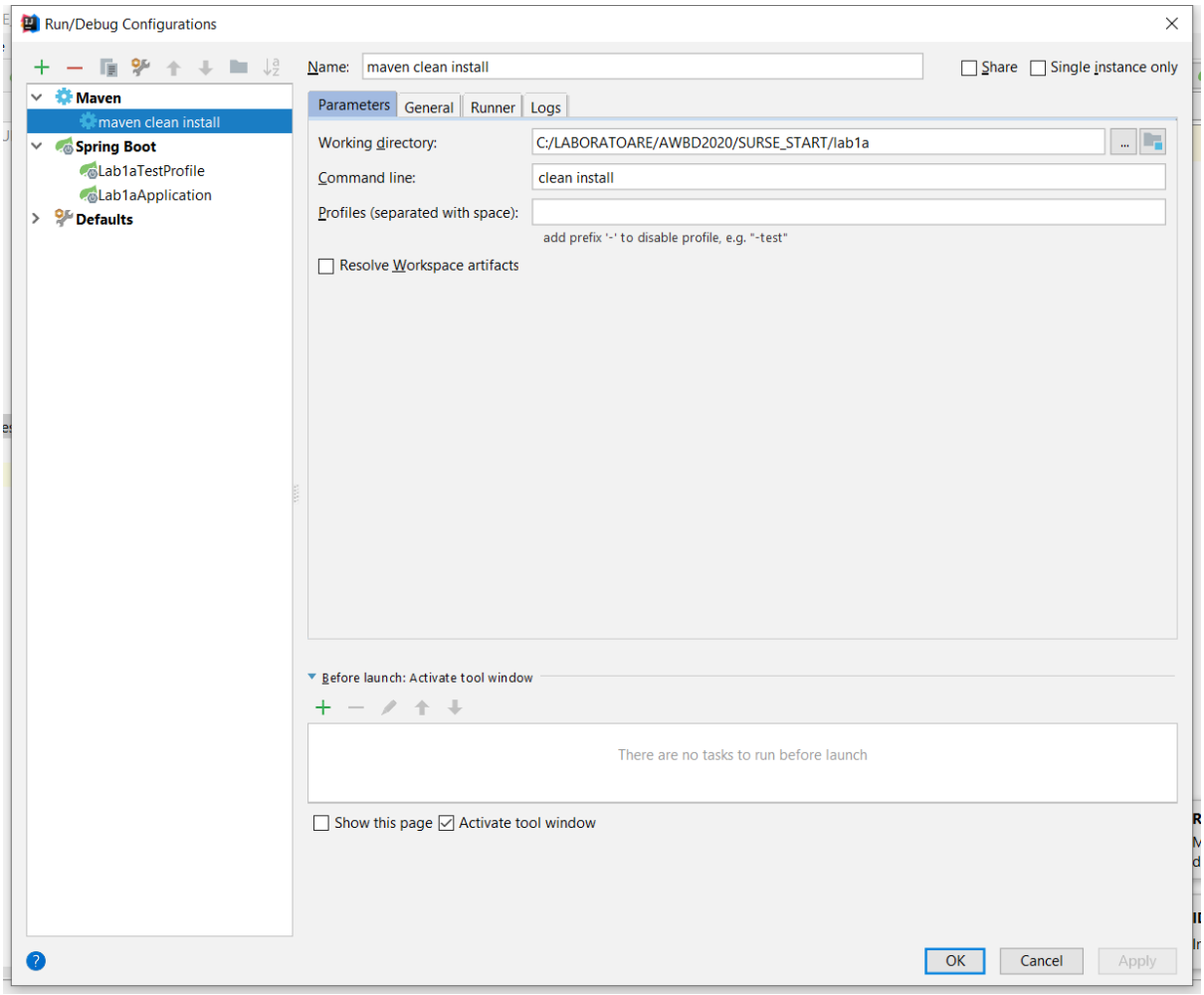
Using @RestController eliminate the need to annotate with @ResponseBody every request handling method. The @ResponseBody annotation tells a controller that the object returned by the method is automatically serialized into JSON and passed back into the *HttpResponse* object.

[3] [4] [5]



4. Run the application. You may add Maven Configuration or run the default configuration. Test in browser: http://localhost:8080/home.

Check local repository **.m2** (C:\Users\username\.m2\repository), you should find the archive: Lab2-0.0.1-SNAPSHOT.jar



5. The default port for the application is 8080. Add a properties file, named application-**test**.properties, in src/main/resources. Add in application-test.properties

```
server.port=8081
```

A Lab2TestProfile run configuration with VM options:

-Dserver.port=8081 -Dspring.profiles.active=test

Test in browser: http://localhost:8081/home. When no suffix is added in application.properties file name, the application runs with the profile **default**.

Info

Maven

Optimize build, test, deploy
Manage dependencies, plug-ins, libraries.
Automatically assures consistency between project's modules versions, keeps modules up to date.

Similar tools npm (for node projects) composer (for php projects), groovy etc.

Maven projects are defined in **POM** files -- „Project Object Model"

POM files

Info

Minimal configuration includes groupId, artifactId, version si modelVersion.

```
<modelVersion>4.0.0</modelVersion>
<groupId>com.awbd</groupId>
<artifactId>lab2</artifactId>
<version>0.0.1-SNAPSHOT</version>
```

POM configurations inherit "super pom" configuration or parent configuration. For instance, default value for packaging is jar:

```
<packaging>jar</packaging>
```

For parent POM value for packaging is pom.
Parent pom may include <modules>, <plugins>, <dependencies> etc.

[6] For each project in the <dependencies> sections we must provide group, id, version and

scope = compile | provided | test etc.

To check that Maven is installed [7] execute in cmd

>> mvn -v

build lifecycle and **phases**:

Default lifecycle phases:

validate	check that all necessary information is available
compile	
test	run unit tests
package	package in distribution format, .jar, .war etc.
verify	run integration tests
install	install the package into the local repository
deploy	copies the final package to the remote repository

Other lifecycles: **clean** (handles project cleaning), **site** (handles project documentation)
[8]

Examples

```
mvn install
```

mvn install will execute validate, compile, test, package, verify and install, all phases preceding install and install.

```
mvn clean install
```

Traverse every subproject and executes clean, then executes install.

goals maven may also execute specific tasks (goals) using plug-ins
Example

```
mvn jar:jar
```

pack into jar
Docker -- toolkit for container management.

Info

- Platform for developing, shipping and running applications.
- Separates applications from infrastructure.
- Run on physical or virtual machines, in a data center, on cloud providers etc.

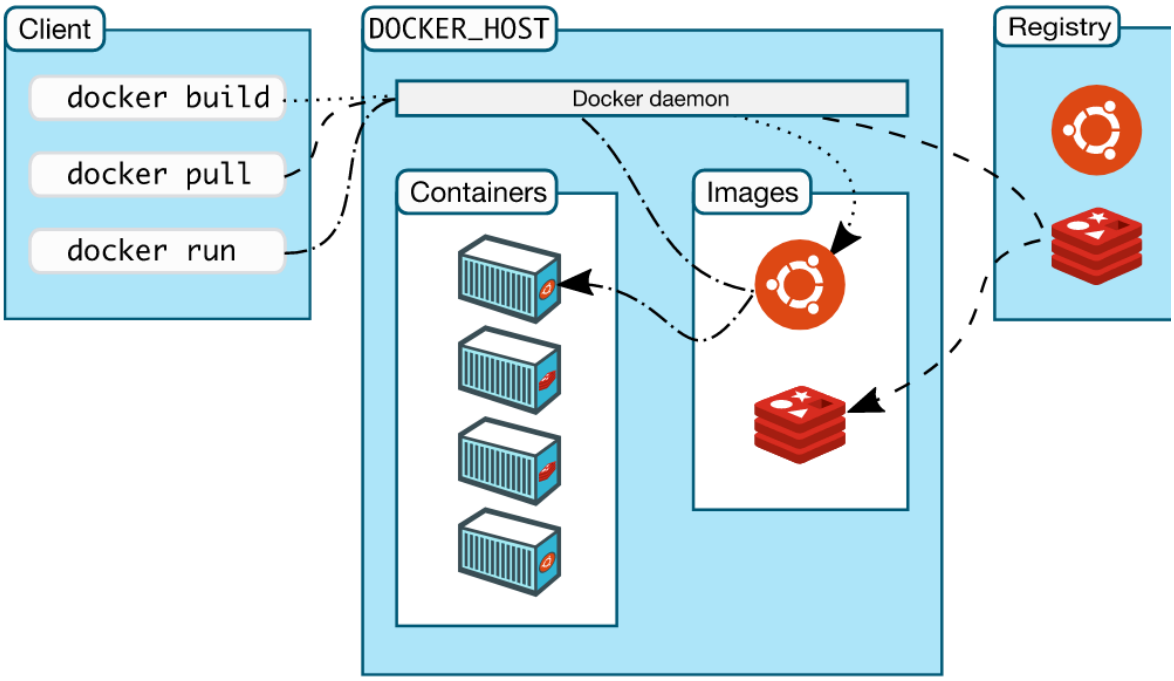
- Runs application in isolated environment, in *containers*.
- Develop, test, deploy using containers.
- CI/CD continuous integration, continuous delivery.

Docker components:

- Server or daemon process, `dockerd` command.
- REST API interfaces to daemon.
- Command line interface, CLI client `docker` command.

Docker objects: [11]

- Images: read-only template with instructions to create a container. Images are published in a *docker registry*. To build an image a *Dockerfile* is created, with instructions for each layer of the image. Rebuilding an image affects only those layers changed in the *Dockerfile*.
- Container: runnable instances of an image. By default, containers can connect to external networks using the host machine's network connection.
- networks, volumes etc.



6. Create a docker file for the project.

```
FROM openjdk:11-oracle
ARG JAR_FILE=target/*.jar
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java","-jar","/app.jar"]
```

7. Run in PowerShell (in the directory where a docker file is present) docker build and create a docker image *laborator1*. Check all the available images using docker images.

```
>> docker build -t laborator2 .

>> docker images
```

8. Instantiate the image, running a container named lab1. List all containers.
Test in browser: http://localhost:8080/home.

```
>> docker run --name lab2 -p 8080:8080 laborator2

>> docker container ls

>> docker stop lab2
```

9. Instantiate the image with the profile test on port 8081.
Test in browser: http://localhost:8081/home.

```
>> docker run -e "SPRING_PROFILES_ACTIVE=test" --name lab2_test -p 8081:8081
laborator2

>> docker stop lab2

>> docker rm lab2_test
```

B

[1] https://docs.spring.io/spring-boot/docs/2.0.x/reference/html/using-boot-using-springbootapplication-annotation.html

[2] https://docs.spring.io/spring-boot/docs/2.0.x/reference/html/using-boot-auto-configuration.html

[3] https://docs.spring.io/spring-framework/docs/3.0.0.M4/spring-framework-reference/html/ch15s02.html

[4] https://www.baeldung.com/spring-controller-vs-restcontroller

[5] https://www.baeldung.com/spring-request-response-body

[6] https://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html

[7] https://maven.apache.org/download.cgi

[8] https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html

[9] https://docs.docker.com/installation/#installation

[10] https://docs.docker.com/

[11] https://docs.docker.com/get-started/overview/

[12] https://spring.io/guides/gs/spring-boot-docker/