Table of Contents

[***1. Spring Boot Initial Steps:*** 1](#_Toc526431446)

[***2. Spring Boot Initializers:*** 3](#_Toc526431447)

[***3.*** ***Working of Spring Boot:*** 5](#_Toc526431448)

[***4.*** ***Why Container less? :*** 6](#_Toc526431449)

[***5.*** ***A Sample Restful Web App:*** 6](#_Toc526431450)

[***6.*** ***Integrating UI Client:*** 7](#_Toc526431451)

[***7.*** ***Server Side code – Rest Controller:*** 8](#_Toc526431452)

[***8.*** ***Overriding Default behavior - Application Properties and Environmental Configurations:*** 9](#_Toc526431453)

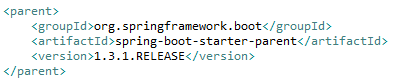
[***8.1.*** ***application.properties*** 9](#_Toc526431454)

[***8.2.*** ***Environmental Configurations:*** 9](#_Toc526431455)

### 1. Spring Boot Initial Steps:

1. Create a new maven Project with maven archetype (maven-quickstart 1.1)
2. In POM.xml

**Define Parent as**

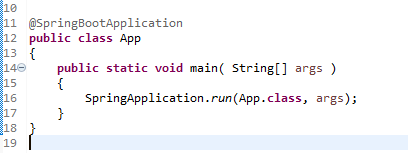


**And add new dependency**

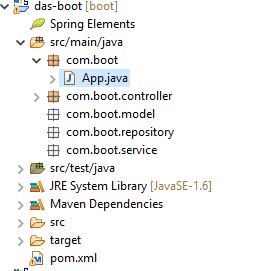


We do not have to enter version here because the version is obtained from parent.

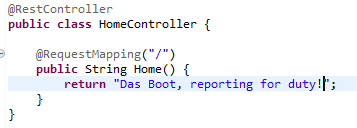
1. Then in Java Application write the following code. This is the starter class.



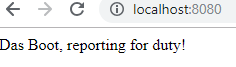
1. Create 4 packages to separate the concerns



1. Write a Rest controller in the controller package.



1. Run the app as a java app and type <https://localhost:8080>



### 2. Spring Boot Initializers:

1. Web Initializer

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

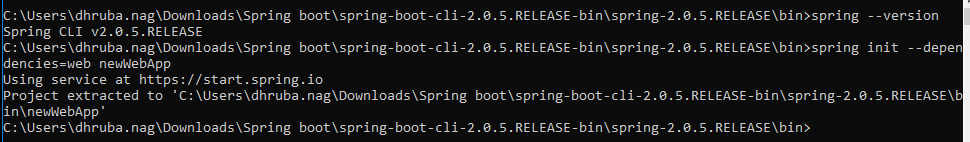
Creates a maven project with all necessary components and downloads as a zip file.

1. Spring Boot CLI(command line)

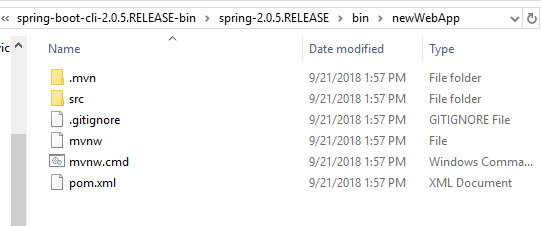
<https://repo.spring.io/release/org/springframework/boot/spring-boot-cli/2.0.5.RELEASE/spring-boot-cli-2.0.5.RELEASE-bin.zip>

Download the CLI

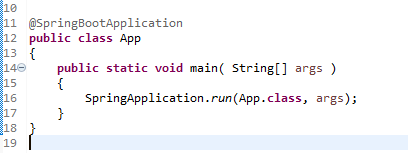
Navigate to the bin folder through command line and run the following commands.



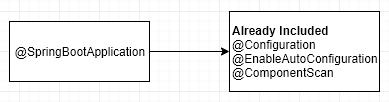
A new project will be created at the location mentioned.



### Working of Spring Boot:



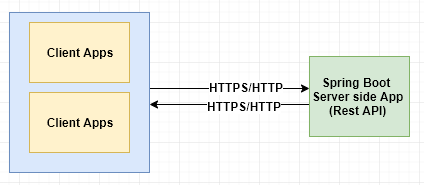
|  |  |  |
| --- | --- | --- |
| Index | Components | Functions |
| 1 | Public static void main | Starting point of a Spring Boot Application just like older java applications |
| 2 | @SpringBootApplication | A convenience annotation that wraps annotations commonly used with Spring Boot |
| 2.1 | @Configuration | Spring Configuration on startup. This will be used to configure the spring context by Spring Boot |
| 2.2 | @EnableAutoConfiguration | Auto Configures other compatible frameworks on class Path |
| 2.3 | @ComponentScan | Scans directory and subdirectory of package containing this class. So this class should be at the top. |
| 3 | SpringApplication.run(className.class, args); | Starts spring, creates spring context, applies annotations and configurations, places any spring components in spring context and sets up embedded container |



### Why Container less? :

|  |  |
| --- | --- |
| **With Containers** | **Without Containers** |
| Pre Set up and Configuration. The container and the server needs to be setup and configured to run the application | Only needs Java. Can be run in any host which has Java. The container is embedded inside the application. We just have a jar file to run. |
| Deployment Descriptors – web.xml  This is needed to tell the container how to deploy and serve up the application | No deployment descriptors are needed since the container is embedded inside the application. The spring boot property files can be used to configure setup of application. |
| All environment settings need to be provided as external settings. e.g. JNDI configurations set up in server | All environment settings are internal |

### A Sample Restful Web App:



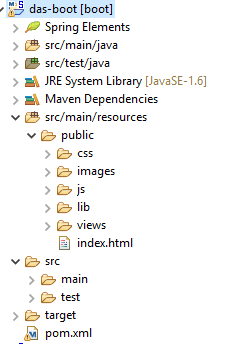
### Integrating UI Client:

Default Static Locations: /static, /public, /resources

We can put static files for UI layer (angular-JS files and HTML) here under

Src>main>resources> (static/public/resources)

e.g.:



After src/main/resources/ (static/public/resources) is created, run a maven update to include these folder and its contents in project class path.

### Server Side code – Rest Controller:

**Base URL:**



|  |
| --- |
| **Endpoints** |
| GET /api/v1/shipwrecks (list) |
|  |
| POST /api/v1/shipwrecks (add) |
|  |
| GET /api/v1/shipwrecks/ {id} (view) |
|  |
| PUT /api/v1/shipwrecks/ {id} (update) |
|  |
| DELETE /api/v1/shipwrecks/ {id} (delete) |
|  |

|  |
| --- |
| **Role of Spring Boot** |
| Enables Auto configuration.  It tells spring mvc to set up view resolvers like content negotiating view resolvers.  It sets up Jackson – Json library to handle views for application/json types |
| It sets up spring mvc to serve static content from class path root / (static/public/resources) folder |
| It sets up spring mvc http message converters so that it can convert JSON objects to java and vice versa. |
| The basic string encoding is set to UTF 8 out of the box with Spring boot |
| It leaves a way to override default behavior by means of programmable hooks. |

### Overriding Default behavior - Application Properties and Environmental Configurations:

Src/main/resources/application.properties

The application properties can be standard java properties or YAML format (for YAML we need to add Snake-YAML dependency)

Environmental Configuration:

application-{profile}.properties e.g. : application-dev.properties

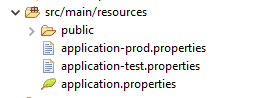
This overrides the properties defined in application.properties based on environment.

### application.properties

|  |  |
| --- | --- |
| Property | Function |
| logging.level.org.springframework.web=debug | This helps us to set logging level without log4j configuration |
| server.port=8181 | The port the application is running on. If nothing is mentioned, default is 8080. |

### Environmental Configurations:

Step 1:

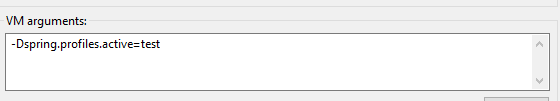


Test has server.port = 9090

Prod has server.port = 80

Step 2:

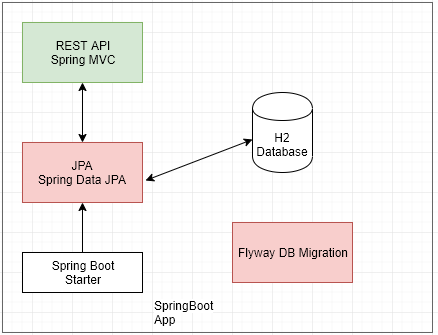
Run Congurations>Run as>Arguments>VM Arguments



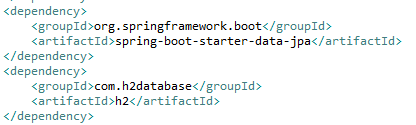
Common Spring Boot Application properties:

<https://docs.spring.io/spring-boot/docs/current/reference/html/common-application-properties.html>

### Spring Boot Data Integration



1. In pom.xml add the following dependencies.

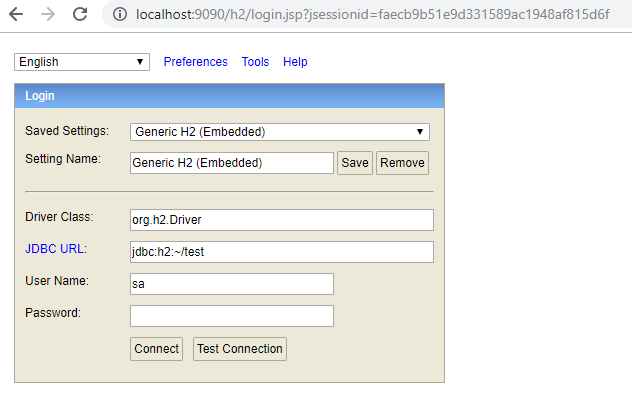


1. In application.properties file add the below configuration



1. Access H2 database with the following URL and default creds

<http://localhost:9090/h2>

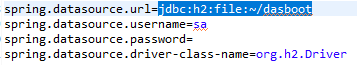


Database configuration and pooling libraries need to be defined in application.properties.

By default spring boot starter data jpa uses tomcat-jdbc as default pooling strategy.

commons-dbcp, commons-dbcp2 can also be integrated.

1. Add the data source properties in the application.properties file.



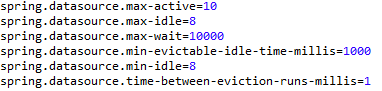
1. Connect to H2 through browser and change the DB URL



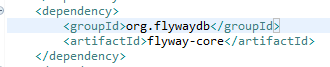
1. Create some test data



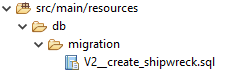
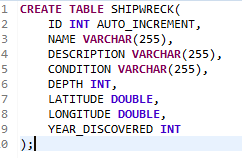
1. Pooling properties



1. Flyway Integration
2. Add POM dependency



1. Create Migration Scripts under src>main>resources

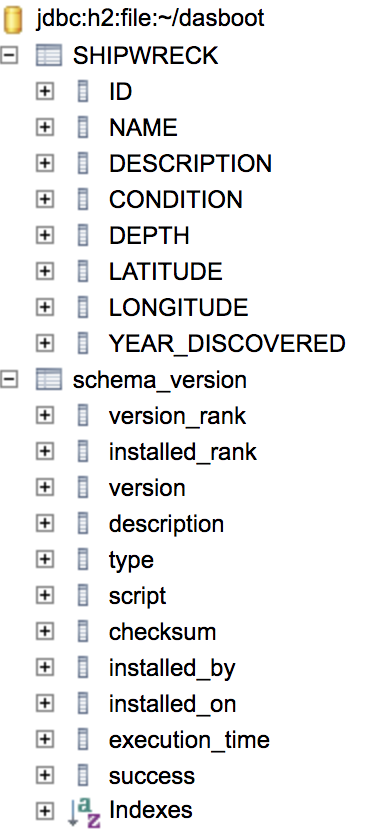
 

This will create a table in database next time application is fired.

1. Add properties to disable spring JPA behaviour and generate metadata for flyway.

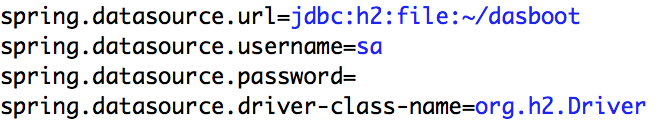


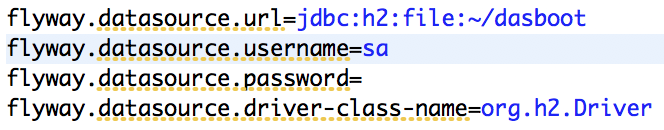
1. Run application

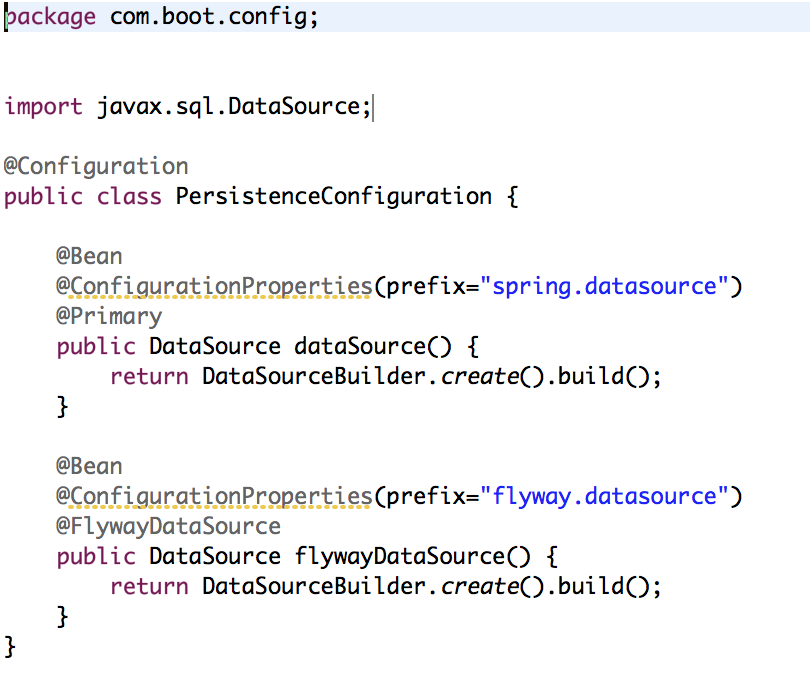


1. Configuring Data Sources.

The datasources defined in the property files need to be configured. Earlier it was done using xml. Now Java configuration can be used.







The datasources can be defines as primary and secondary.

@FlywayDataSource is used to mark a datasource to be used by Flyway.