Setup Development Environment

Introduction

For this course we will be using the Java programming language to implement Spring Boot based Web applications hosted on Amazon Web Services (AWS). This assignment will walk you through setting up the local development environment and deploying a simple Hello World Web application on AWS. We will be expanding this application as the semester progresses. This document is also available online.

Learning Objectives

By the end of this assignment you should be able to

- Configure a local Java development environment
- Create a simple Spring Boot Web application
- Configure a remote Tomcat server running on AWS
- Deploy a Spring Boot Web application to AWS

Setup Java JDK 8

We will be using the latest version of Java throughout the semester. Navigate to Oracle's Java Development Kit (JDK) 8 download website

http://www.oracle.com/technetwork/java/javase/downloads/index.html

Download and install the JDK for your particular operating system. Once the JDK has installed, verify you have the right version of Java installed. From your terminal or console, type the following

```
> java -version
java version "1.8.0_77"
Java(TM) SE Runtime Environment (build 1.8.0_77-b03)
```

On macOS you can find out where Java is installed by typing the following on the terminal:

```
> which java
/Library/Java/JavaVirtualMachines/jdk1.8.0_77.jdk/Contents/Home//bin/java
```

On Windows, Java should install in `Program Files/Java`

Verify environment variable JAVA_HOME points to the Java installation folder. From your terminal or console, type the following

```
> echo $JAVA_HOME
/Library/Java/JavaVirtualMachines/jdk1.8.0_77.jdk/Contents/Home/
```

The above path might be different in your particular operating system. If JAVA_HOME is not set, set it for your particular operating system and add Java's bin directory to the PATH environment variable. On macOS, or other Unix like OSs, add JAVA_HOME as an environment variable in your ~/.bash_profile. The .bash_profile is a hidden configuration file in your home directory (~). If you have a new machine it might not yet exist, so you might need to create the file (touch). From your terminal, type the following

```
> cd ~
> touch .bash_profile
> edit .bash_profile
```

This will open .bash_profile in your system text editor. Add the following line at the end your file

```
export JAVA_HOME=/my/path/to/jdk/jdk1.8.0_77.jdk/Contents/Home/
export PATH=$JAVA_HOME/bin:$PATH
```

Where the path /my/path/to/jdk/ will be different for your particular machine.

On Windows, set up environment variables in the *Environment Variables Control Panel*

1. Select *Start*, select *Control Panel*. Double click *System*, and select the *Advanced* tab
2. Click *Environment Variables*
3. In the *Edit System Variable* (or *New System Variable*) window, specify the value of the JAVA_HOME environment variable
4. Edit the PATH environment variable and add %JAVA_HOME%\bin

Setup Maven

Maven is a Java dependency package manager. It simplifies managing the lifecycle of projects such as downloading libraries, compiling, running automated tests, and packaging projects for deployment. Download maven from

http://maven.apache.org/download.cgi

1. Download the latest ZIP file version, e.g., apache-maven-3.5.2-bin.zip.
2. On macOS, unzip to /urs/local/apache/maven/. On Windows, unzip to /Program Files/apache/maven/.

3. Create environment variables `M2_HOME` and `MAVEN_HOME` to refer to the paths where you unzipped maven. Latest Maven documentation only mentions `M2_HOME`, but older tools might still use the older environment variable `MAVEN_HOME`.
4. Add `M2_HOME/bin` and `MAVEN_HOME/bin` to your `PATH` environment variable so you can execute maven from the terminal or console.
5. In a new terminal or console verify maven is installed by typing the following

```
> mvn –version
Apache Maven 3.5.2 (r01de14724cdef16f02da; 2013-02-19 08:51:28-0500)
Maven home: /usr/local/apache-maven-3.0.5
Java version: 1.8.0_77, vendor: Oracle Corporation
Java home: /my/path/to/jdk/jdk1.8.0_77.jdk/Contents/Home/jre
Default locale: en_US, platform encoding: UTF-8
OS name: "mac os x", version: "10.12.6", family: "mac"
```

Your output might vary slightly

Create a Spring Boot Web Maven Project

Spring Boot is a preconfigured Spring framework that allows creating Spring based Java applications with minimal effort. Spring Boot bootstraps a Spring application based on a heavily preconfigured set of default assumptions, hence the name Spring Boot. The assumptions Spring Boot makes address 80% of the needs of typical applications. Additional configuration would be needed when exploring the other 20%.

Setup Spring Tool Suite (STS)

Spring Tool Suite (STS) is an *Eclipse* based IDE that has been optimized for working with Spring Boot projects. Download STS for your operating system from

https://spring.io/tools/sts/all

unzip it, run the installer, and follow the installation instructions. On macOS, drag the *STS* application icon to the *Applications* folder. On Windows, unzip to `Program Files/spring`. Run STS and accept the default configuration.

Create a Spring Starter Project

Using STS, create a Spring Boot project. From the File menu, select New and then Spring Starter Project. Configure the project based on the name of the course, the semester term, and your last name. For instance, consider the following course, semester and last name

- Course: cs5200
- Semester: Spring 2018

- Last name: Annunziato

use the following configuration values:

- Name and Artifact: cs5200-spring2018-annunziato
- Group and Package: edu.northeastern.cs5200
- Packaging: War

Add *Web*, *JPA*, and *Apache Derby* as a project dependencies and click finish. Find the path of the project folder by right clicking on the project, selecting Properties and Resources on the left, and then Location on the right. Make a note of the path since we'll need it in a later step.

Create an HTML Index File

Once the project is created we'll add a simple `index.html` in the `webapp` folder by right clicking the `webapp` folder, selecting *New*, *Other*, *Web*, *HTML File*, *Next*, and type the name of the file: `index.html`. Use the following content for the new `index.html` file, replacing my name with YOUR NAME:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
        <h1>Hello Jose Annunziato!</h1>
</body>
</html>
```

Test Web Application Locally

Spring Boot allows to easily run a Web application locally by running a boilerplate class file that bootstraps the Spring framework and serves as the entry point of the server application. In STS, navigate to your application entry point, e.g., `src/main/java/Cs5200Spring2018AnnunziatoApplication.java` and run it as a Java Application. To run it, right click on the class, select *Run As*, and then *Java Application*. This will start a local embedded Tomcat server that listens at port 8080. To test it, open a browser and point it to `localhost:8080`. Verify the browser displays `Hello Jose Annunziato!`, but instead of my name, YOUR NAME displays.

Create a Spring Boot REST Controller

In later assignments we will need to access data from the server using HTTP requests. A common method of accessing data from a server is REST (Representational state transfer) which maps URL patterns and HTTP methods

to server side resources. The following sections walk you through exposing different types of data by mapping HTTP request URL patterns to executable methods on the server, usually referred to as *REST Web service end points*.

Create a Simple REST Controller

In this section we expose a simple Java String literal as a resource available through an HTTP server request. In STS, under src/main/java, create a new class called HelloController in package edu.northeastern.cs5200.controllers.hello. Use the code below as an example, but instead of my name, use YOUR NAME.

```
package edu.northeastern.cs5200.controllers.hello;

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

@RestController
public class HelloController {
	@RequestMapping("/api/hello/string")
	public String sayHello() {
		return "Hello Jose Annunziato!";
	}
}
```

The code above configures the server to listen for an incoming HTTP request with the URL pattern /api/hello/string and maps the request to execute the sayHello() method. The method returns the string "Hello Jose Annunziato!" as an HTTP response, but instead of my name, YOUR NAME displays.

Create a Simple REST JSON Controller

A more interesting use case is to expose structured data such as Java objects of arbitrary complexity. The Java objects can implement a data model that fulfills a particular business need. As a simple example, we'll create a HelloObject that contains just a String message property and expose the trivial data model as a REST HTTP request. Create a simple POJO (Plain Old Java Object) called HelloObject in package edu.northeastern.cs5200 .controllers.hello to test REST access to data on the server. Use the following code as a guide. Note the property is private and is only accessible through public setters and getters. Also note the required default constructor HelloObject().

```
package edu.northeastern.cs5200.controllers.hello;

public class HelloObject {
	private String message;
	public String getMessage() {
		return message;
	}
```

```
	public void setMessage(String message) {
		this.message = message;
	}
	public HelloObject(String message) {
		this.message = message;
	}
	public HelloObject() {
	}
}
```

To make the above data accessible through an HTTP request (to expose), add an additional @RequestMapping that exposes the HelloObject to an HTTP request mapped to /api/hello/object. Add the bolded code below, but instead of my name, use YOUR NAME.

```
package edu.northeastern.cs5200.controllers.hello;

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

@RestController
public class HelloController {
	@RequestMapping("/api/hello/string")
	public String sayHello() {
		return "Hello Jose Annunziato!";
	}
	@RequestMapping("/api/hello/object")
	public HelloObject sayHelloObject() {
		HelloObject obj = new HelloObject("Hello Jose Annunziato!");
		return obj;
	}
}
```

The code above configures the server to listen for an incoming HTTP request with the URL pattern /api/hello/object and maps the request to execute the sayHelloObject() method. The method returns an instance of HelloObject with the message property set to "Hello Jose Annunziato!". Since the method returns an object instance, instead of a litteral, the object is formatted in an HTTP friendly format such as XML (Extensible Markup Language) or JSON (JavaScript Object Notation). Spring boot request controllers format returned objects as JSON by default. Recompile, repackage, and restart the application. Point your browser to

http://localhost:8080/api/hello/string

and verify the server responds with the string Hello Jose Annunziato!, but instead of my name, YOUR NAME displays. Then point your browser to

http://localhost:8080/api/hello/object

and verify the server responds with the following JSON data, but instead of my name, YOUR NAME displays

```
{
   "message":"Hello Jose Annunziato!"
}
```

Note that the JSON attribute "message" is the same as the HelloObject's property "message", and the value "Hello Jose Annunziato!" is the value we used to initialize the HelloObject instance. In later assignments we will be exploring much more interesting and complex data models.

Deploy a Spring Boot Web Application to AWS

Now that we have tested the Web application running locally, we'll deploy it to a remote Tomcat server running on Amazon Web Services (AWS). Navigate and login to your AWS console

https://aws.amazon.com/console/

Create an Elastic Beanstalk service instance to deploy your Web application. Elastic Beastalk can host any number of applications built on Java, .NET, PHP, Node.js, Python, Ruby, Go, and Docker. We'll be deploying the Java Web application we built on Spring Boot in the previous sections. Follow the steps below to deploy your Java Web application on a remote Tomcat server hosted on an AWS Elastic Beanstalk service.

1. Once you've logged in to your AWS console, expand *All services* and select *Elastic Beanstalk*. Click *Create New Application* on the top right and name the application the same way you named it for the project, e.g., cs5200-spring2018-annunziato, and click create.
2. On the left, click on *Environments* and then *Create one now* on the right to create a runtime environment to host the Tomcat server. In the *Choose an environment tier* dialog choose *Web server environment* and then *Select*.
3. In the *Create a new environment* screen choose a public domain for your Web application. Try the same name as the name of the application. Click on *Check availability* to see if the domain is not already taken. In the unlikely event that the domain is already taken, add a counter to the domain name to distinguish it, e.g, cs5200 -spring2018-annunziato-**1**. Scroll down and choose *Tomcat* for the *Platform*.
4. Down further select *Upload your code* and click on *Upload*. Browse to the location of the WAR file in the target folder in the location of your project. You made a note of the location of your project in an earlier step. It can be retrieved by right clicking the project, selecting *Properties*, then *Resources*, then *Location* on the right. Upload the WAR file, click create environment, and wait for the application to deploy. After a few minutes your application should have deployed.

Test Your Deployed Application

Navigate to your Website, e.g.,

http://cs5200-spring2018-annunziato.us-west-2.elasticbeanstalk.com/

and verify that Hello Jose Annunziato! displays from the root `index.html` file as shown below, but instead of my name, YOUR NAME displays.

Hello Jose Annunziato!

Then navigate to the REST controllers that returned a string and an object. Add `/api/hello /string` to the URL used in the previous step, e.g., navigate to

http://cs5200-spring2018-annunziato.us-west-2.elasticbeanstalk.com/api/hello/string

and verify the server responds with the string `Hello Jose Annunziato!`, but instead of my name, YOUR NAME displays. Finally, replace string with object in the previous URL, e.g, point your browser to

http://cs5200-spring2018-annunziato.us-west-2.elasticbeanstalk.com/api/hello/object

and verify the server responds with the following JSON data, but instead of my name, YOUR NAME displays.

```
{
   "message":"Hello Jose Annunziato!"
}
```

Submit all AWS URLs above as deliverables for this assignment, e.g.,

http://cs5200-spring2018-annunziato.us-west-2.elasticbeanstalk.com/
http://cs5200-spring2018-annunziato.us-west-2.elasticbeanstalk.com/api/hello/string
http://cs5200-spring2018-annunziato.us-west-2.elasticbeanstalk.com/api/hello/object

Commit and Push Your Code to GitHub.com

Create an account at github.com and push your code to a repository named the same as your project, e.g., cs5200-spring2018-annunziato.

1. Navigate and login at github.com

2. Click on Repositories and click on New to create a new repository
3. Type the name of the repository in the *Repository name* field, e.g., cs5200-spring2018-annunziato, and click on *Create repository*. Several commands are shown to initialize, add, commit, and push your code.
4. On your machine, use the terminal to navigate to your project folder and type the commands github suggests, e.g.,

```
git init
git add .
git commit -m "first commit"
git remote add origin https://github.com/jannunzi/cs5200-spring2018-annunziato.git
git push -u origin master
```

5. Where my jannunzi above is my github username and cs5200-spring2018-annunziato is my repository. Use YOUR usename and your repository instead
6. Submit the URL to your repository as a deliverable for this assignment, e.g.,

   https://github.com/jannunzi/cs5200-spring2018-annunziato.git

Deliverables

Submit all AWS URLs, e.g.,

http://cs5200-spring2018-annunziato.us-west-2.elasticbeanstalk.com/
http://cs5200-spring2018-annunziato.us-west-2.elasticbeanstalk.com/api/hello/string
http://cs5200-spring2018-annunziato.us-west-2.elasticbeanstalk.com/api/hello/object

Submit the URL to your github.com repository as a deliverable for this assignment, e.g.,

https://github.com/jannunzi/cs5200-spring2018-annunziato.git