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 building a Web project.

Do all your work in a new directory called `project` your personal class homework repository.
These instructions are also captured here (`https://goo.gl/8A8EVL`).

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 Java 8 throughout the semester.[1]

If you need to install Java, 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)
```

[1]If you have Java 9 installed, these instructions will still work. You may have to do some extra work when you set up STS later on.

macOS/Linux

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
```

Verify environment variable `JAVA_HOME` points to the Java installation folder:

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

If `JAVA_HOME` is not set, set it. 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:

```
> 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.

Windows

On Windows, Java should install in `Program Files\Java`. Verify environment variable `JAVA_HOME` points to the Java installation folder:

```
> echo %JAVA_HOME%
C:\Program Files\Java\jdk1.8.0_60
```

You set up environment variables in the `Environment Variables` Control Panel:

1. Select Start, select Control Panel. Double click System, and select the Advanced tab (or ask Cortana for the Environment Variables Control Panel).
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. Unzip and move the files
   macOS/Linux: `/usr/local/apache/maven/`
   Windows: `\Program Files\apache\maven`
3. Create environment variables `M2_HOME` and `MAVEN_HOME` using the same methods described earlier for setting `PATH`. Set the value to the directory where you put Maven. The latest Maven documentation only mentions `M2_HOME`, but older tools might to 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 will tell you the version of Maven, where it sits, a repeat of the Java information, and then information about your machine.

Create a Spring Boot Web Maven Project

Spring Boot[2] is a pre-configured Spring[3] framework that allows creating Spring-based Java applications with minimal effort. Spring Boot bootstraps a Spring application based on a heavily pre-configured 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%.

You will define applications using the Spring Tool Suite (STS)[4], which is plugs into Eclipse[5] and is optimized for working with Spring Boot projects.

[2] `https://projects.spring.io/spring-boot/`
[3] `https://spring.io`
[4] `https://spring.io/tools`
[5] `https://www.eclipse.org`

Install Spring Tool Suite (STS)

1. Download STS from `https://spring.io/tools/sts/all`. Pick the version right for your machine.
2. Unzip it[6] and then . . .

   macOS: drag the STS application icon to the Applications folder.

   Windows: copy the `sts.xxx.release` folder to `Program Files\spring`.
3. Run STS and accept the default configuration.

Create a Spring Starter Project

Assuming STS has started, which kicks off Eclipse, 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: cs4500
- Semester: Spring 2018
- Last name: Annunziato

use the following configuration values:

- Name and Artifact: cs4500-spring2018-annunziato
- Group and Package: edu.northeastern.cs4500
- Packaging: War

You may need to click `Next` to get to the project dependencies choices in the next step.

Now add Web (under Web), JPA (under SQL), and Apache Derby (under SQL) as a project dependencies and click finish. Find the path of the project folder by right clicking on the project, selecting Properties and then Resource on the left, and then Location on the right. Make a note of the path since we'll need it in a later step. Click Apply and Close.

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, then click Next, and name the file: `index.html`. Use the following content for the new `index.html` file:

[6] In Windows, if you get a file path too long error (error code 0x80004005) when you extract the files, try unzipping the files using a tool such as 7-Zip (see 7-zip.org).

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

Test Web Application Locally

The next step is to start up your simple web app using Spring Boot.

In STS (eclipse), navigate to your application's entry point (e.g. `src/main/java/Cs4550Spring2018/AnnunziatoApplication.java`) right click, select `Run As` and then choose `Java Application`. This will start a Tomcat server on your machine that listens on port 8080.

You should see in the STS (eclipse) console something that looks like:

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::      (v1.5.9.RELEASE)
2018-01-12 15:14:16.298  INFO 8576 --- [           main]
 [LOTS OF MESSAGES]...
2018-01-12 15:22:36.824  INFO 8576 --- [nio-8080-exec-1]
o.s.web.servlet.DispatcherServlet     : FrameworkServlet
'dispatcherServlet': initialization completed in 42 ms
```

To test it, open a browser and go to `localhost:8080`. You should see `Hello World!`

Create a Spring Boot REST Controller

In later assignments, we will need to access data and functions from the server. Since we will be doing web apps, you will work with the server by making REST requests over HTTP. A common way to do this is to map URL patterns and HTTP methods to server side resources, usually called REST Web service end-points. Said more directly, the server receives an HTTP request and a framework handles how dispatch that request to some object that actually does the work. Here, we will use Spring as the framework, which uses a Model-View-Controller design pattern.

Now you will walk through an example of doing this.

Create a Simple REST Controller

What we are going to do is treat a simple Java String literal as a resource available and expose it as a REST-ful web service. In other words, clients will be using HTTP to get to our server and access this resource.

The first step is to go into STS, and under `src/main/java`, create a new class called `HelloController` in package `edu.neu.cs4500.controllers.hello`. Use the code below as an example.

```
package edu.neu.cs4500.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␣World!";
        }

}
```

This code 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 World!"` as an HTTP response.

Create a Simple REST JSON Controller

We can make this more interesting and expose much more complicated data structures using JSON. For now, we'll start small and create a `HelloObject` that contains just a String message property and expose a modest data model.

Add a simple POJO (Plain Old Java Object) called `HelloObject` in package `edu.neu.cs4500.controllers.hello` (the one you just created in the previous step). 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.cs4500.controllers.hello;

/**
 * Simple example of CRU services on an object.
 */
public class HelloObject {

  /*
  * Read function
  */
  public String getMessage() {
    return message;
  }
  /*
  * Update function
  */
  public void setMessage(String message) {
    this.message = message;
  }

  /*
  * Create function within an initialized value
  */
  public HelloObject(String message) {
    this.message = message;
  }

  /*
  * Create function
  */
  public HelloObject() {
  }

  private String message;
}
```

To expose the our objectâĂŹs services to the web, you need to add an additional `@RequestMapping` directive that exposes the `HelloObject` to an HTTP request mapped to `/api/hello/object`. Change `HelloController.java` to the following.

```
package edu.northeastern.cs4500.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 World!";
        }

        @RequestMapping("/api/hello/object")
        public HelloObject sayHelloObject() {
                HelloObject obj = new HelloObject("Hello World!");
                return obj;
        }

}
```

This code configures the server to listen for an incoming HTTP request for a URL (of pattern) `/api/hello/object`, and when such a message arrives, execute the `sayHelloObject()` method. Here, the method returns an instance of `HelloObject` with the message property set to `"Hello World!"`. Since the method returns an instance, instead of a literal, the object is formatted as XML or JSON. By default, Spring boot request controllers uses JSON.

Recompile, repackage, and restart the application.

Open your browser to `http://localhost:8080/api/hello/string` and you should see the string: `Hello World!`

Now, point your browser to `http://localhost:8080/api/hello/object` and you should see the following JSON data:

```
{
    "message":"Hello World!"
}
```

Again, note the JSON attribute "message" is the same as `HelloObject.message`. The JSON value `"Hello World!"` is the value with which we initialized the `HelloObject` instance. The project will involve much more interesting and complex data models.

Customize Your Application

Change the `index.html` and `HelloController.java` to use your name instead of just `"Hello World!"`. In other words, if your name is *Alice Wonderland*, then the URLs should return `"Hello Alice Wonderland!"`. Test your URLs and verify that they return the following content.

URL	Expected Return
`http://localhost:8080`	`Hello Alice Wonderland!`
`http://localhost:8080/api/hello/string`	`Hello Alice Wonderland!`
`http://localhost:8080/api/hello/object`	`{ "message":"Hello Alice Wonderland!" }`

Deploy a Spring Boot Web Application to AWS

Now that we have tested the Web application running locally, we'll deploy it to the cloud. You will now set a Tomcat server running on Amazon Web Services (AWS).

First, open a browser to `https://aws.amazon.com/console/` and log in using your AWS credentials.

Create an Elastic Beanstalk service instance to deploy your Web application. 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., cs4500-spring2018-annunziato, and click create.

2. Now you will create an environment (the tomcat server). If you arenâĂŹt brought directly to a choice between web server and worker, look to the left, click on Environments and then Create one now on the right. In the Choose an environment tier dialog choose Web server 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, cs4500 -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.

5. Copy the url for your application.

Test Your Deployed Application

Navigate to your Website, e.g.,

```
http://cs4500-spring2018-annunziato.us-west-2.elasticbeanstalk.com
```

and you should see `Hello Alice Wonderland!` 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.,

```
http://cs4500-spring2018-annunziato.us-west-2.elasticbeanstalk.com/api/hello/string
```

and verify the server responds with the string `Hello Alice Wonderland!` Finally, replace string with object in the previous URL, e.g, point your browser to

```
http://cs4500-spring2018-annunziato.us-west-2.elasticbeanstalk.com/api/hello/object
```

and verify the server responds with the following JSON data

```
{
    "message":"Hello World!"
}
```

Deliverables

As a deliverable, add a section called *AWS Hello World Endpoints* to your `README.md` file[7] that contains the three links to your AWS web site: one to `index.html`, and two to the RESTful endpoints. Your README.md should look like:

AWS Hello World Endpoints

```
Link to the index page
Link to the string REST endpoint
Link to the object REST endpoint
```

Use the following markdown as a guide:

[7]in the directory where you cloned your repository

```
## AWS Hello World Endpoints
[Link to the index
page](http://cs4500-spring2018-annunziato.us-west-2.elasticbeanstalk.com/)
[Link to the string REST
endpoint](http://cs4500-spring2018-annunziato.us-west-2.elasticbeanstalk.com/api/hello/string)
[Link to the object REST
endpoint](http://cs4500-spring2018-annunziato.us-west-2.elasticbeanstalk.com/api/hello/object)
```

Add, commit and push all the work completed under the `project` directory, to your personal class homework repository. Use the following message for your commit: "Setting up the development environment."