Back-End Server Development: MVC With Spring STS

In this experiment, a brief introduction to back-end development, MVC design pattern and Spring Tool Suit with Maven are introduced to make the task of developing a back-end support for any kind of software easy.

Back-End Vs Front-End

In software engineering, the terms "front end" and "back end" are distinctions which refer to the separation of concerns between a presentation layer and a data access layer respectively. The front end is an interface between the user and the back end. The front and back ends may be distributed amongst one or more systems.

In software architecture there may be many layers between the hardware and end user. Each can be spoken of as having a front end and a back end. The front is an abstraction, simplifying the underlying component by providing a user-friendly interface.

In software design, for example, the model-view-controller architecture provides front and back ends for the database, the user, and the data processing components. The separation of software systems into front and back ends simplifies development and separates maintenance. A rule of thumb is that the front (or "client") side is any component manipulated by the user. The server-side (or "back end") code resides on the server. The confusion arises when one must make front-end edits to server-side files. Most HTML designers, for instance, don't need to be on the server when they are developing the HTML; conversely, the server-side engineers are, by definition, never on anything but a server. It takes both to ultimately make a functioning, interactive website.

For major computer subsystems, a graphical file manager is a front end to the computer's file system, and a shell interfaces with the operating system. The front end faces the user, and the back end launches the programs of the operating system in response.

Model View Controller (MVC) Design Pattern

MVC is a software architectural pattern for implementing user interfaces. It divides a given software application into three interconnected parts, so as to separate internal representations of information from the ways that information is presented to or accepted from the user.

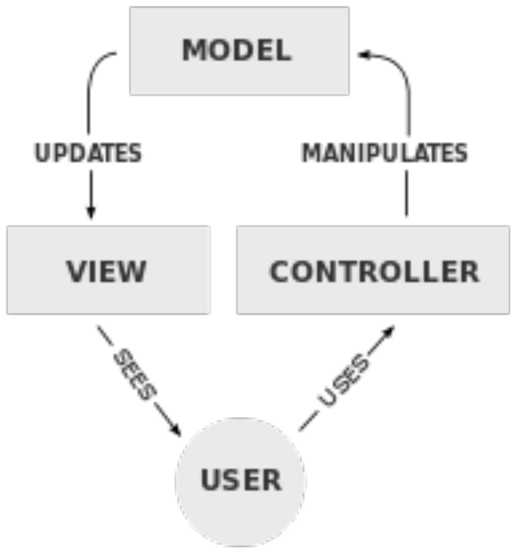
Components:

The central component of MVC, the *model*, captures the behavior of the application in terms of its problem domain, independent of the user interface. The model directly manages the data, logic and rules of the application. A *view* can be any output representation of information, such as a chart or a diagram; multiple views of the same information are possible, such as a bar chart for management and a tabular view for accountants. The third part, the *controller*, accepts input and converts it to commands for the model or view.

How do these components interact?

In addition to dividing the application into three kinds of components, the model–view–controller design defines the interactions between them:

- A **controller** can send commands to the model to update the model's state (e.g., editing a document). It can also send commands to its associated view to change the view's presentation of the model (e.g., by scrolling through a document).
- A **model** notifies its associated views and controllers when there has been a change in its state. This notification allows the views to produce updated output, and the controllers to change the available set of commands. In some cases an MVC implementation might instead be "passive," so that other components must poll the model for updates rather than being notified.
- A **view** requests information from the model that it uses to generate an output representation to the user.



<u>Spring Framework</u>

Spring is the most popular application development framework for enterprise Java. Millions of developers around the world use Spring Framework to create high performing, easily testable, reusable code. Spring framework is an open source Java platform and it was initially written by Rod Johnson and was first released under the Apache 2.0 license in June 2003. Spring is lightweight when it comes to size and transparency. The basic version of spring framework is around 2MB. The core features of the Spring Framework can be used in developing any Java application, but there are extensions for building web applications on top of the Java EE platform. Spring framework targets to make J2EE development easier to use and promote good programming practice by enabling a POJO-based programming model.

Ideally speaking, a POJO is a Java object not bound by any restriction other than those forced by the Java Language Specification. I.e., a POJO **should not** have to

1. Extend prespecified classes, as in
   `public class Foo extends javax.servlet.http.HttpServlet { ...`

2. Implement prespecified interfaces, as in
   `public class Bar implements javax.ejb.EntityBean { ...`

3. Contain prespecified annotations, as in
   `@javax.persistence.Entity public class Baz { ...`

Before talking about the benefits of using spring framework, you should understand the terms Inversion of Control and Dependency Injection. The Inversion of Control (IoC) and Dependency Injection (DI) patterns are all about removing dependencies from your code. For example, say your application has a text editor component and you want to provide spell checking. Your standard code would look something like this:

```
public class TextEditor {
        private SpellChecker checker;
        public TextEditor()    {
                checker = new SpellChecker();
        }
}
```

What we've done here is create a dependency between the TextEditor and the SpellChecker. In an IoC scenario we would instead do something like this:

```
public class TextEditor {
        private ISpellChecker checker;
        public TextEditor(ISpellChecker checker)    {
                this.checker = checker;
        }
}
```

Now, the client creating the TextEditor class has the control over which SpellChecker implementation to use. We're injecting the TextEditor with the dependency.

This is just a simple example, but for more details please refer to the reference[1]

1 http://dotnetslackers.com/articles/designpatterns/InversionOfControlAndDependencyInjectionWithCastleWindsorContainerPart1.aspx

Benefits of Spring Framework:

Following is the list of few of the great benefits of using Spring Framework:

- Spring enables developers to develop enterprise-class applications using POJOs. The benefit of using only POJOs is that you do not need an EJB container product such as an application server but you have the option of using only a robust servlet container such as Tomcat or some commercial product.
- Spring is organized in a modular fashion. Even though the number of packages and classes are substantial, you have to worry only about ones you need and ignore the rest.
- Spring does not reinvent the wheel instead, it truly makes use of some of the existing technologies like several ORM frameworks[2], logging frameworks, JEE, Quartz and JDK timers, other view technologies.
- Testing an application written with Spring is simple because environment-dependent code is moved into this framework. Furthermore, by using JavaBean-style POJOs, it becomes easier to use dependency injection for injecting test data.
- Spring's web framework is a well-designed web MVC framework, which provides a great alternative to web frameworks such as Struts or other over engineered or less popular web frameworks.
- Spring provides a convenient API to translate technology-specific exceptions (thrown by JDBC, Hibernate, or JDO, for example) into consistent, unchecked exceptions.
- Lightweight IoC containers tend to be lightweight, especially when compared to EJB containers, for example. This is beneficial for developing and deploying applications on computers with limited memory and CPU resources.
- Spring provides a consistent transaction management interface that can scale down to a local transaction (using a single database, for example) and scale up to global transactions (using JTA, for example).

Spring STS Tool Suit

The Spring Tool Suite is an Eclipse-based development environment that is customized for developing Spring applications. Please download the latest version of this tool suit from the link [3] and install it on your laptop.

For more and more details about the spring framework, you are strongly required to read the tutorial in the link [4]

[2] http://en.wikipedia.org/wiki/Object-relational_mapping

[3] https://spring.io/tools/sts/all

[4] http://www.tutorialspoint.com/spring/spring_tutorial.pdf

Maven Dependency Management Tool

Maven, a Yiddish word meaning *accumulator of knowledge*, was originally started as an attempt to simplify the build processes in the Jakarta Turbine project. There were several projects each with their own Ant build files that were all slightly different and JARs were checked into CVS. Maven wanted a standard way to build the projects, a clear definition of what the project consisted of, an easy way to publish project information and a way to share JARs across several projects.

The result is a tool that can now be used for building and managing any Java-based project. Maven's primary goal is to allow a developer to comprehend the complete state of a development effort in the shortest period of time. In order to attain this goal there are several areas of concern that Maven attempts to deal with:

- Making the build process easy
- Providing a uniform build system
- Providing quality project information
- Providing guidelines for best practices development
- Allowing transparent migration to new features

You are required to install maven tool on your laptops, here is a detailed link of how to install maven[5].

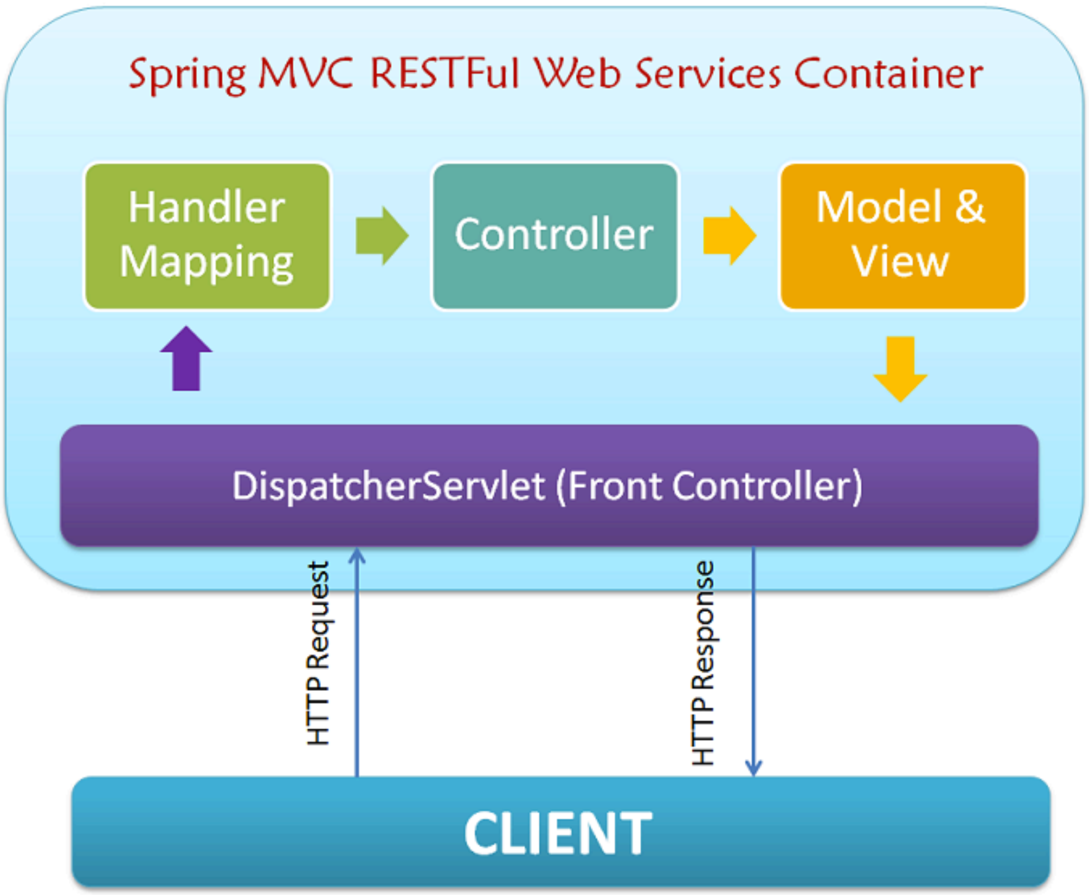
Rest Controllers

REST (Representational State Transfer) is an architectural style with which Web Services can be designed that serves resources based on the request from client. A Web Service is a unit of managed code that can be invoked using HTTP requests. Let me put it simple for those who are new to Web Service. You develop the core functionality of your application, deploy it in a server and expose to the network. Once it is exposed, it can be accessed using URI's through HTTP requests from a variety of client applications. Instead of repeating the same functionality in multiple client (web, desktop and mobile) applications, you write it once and access it in all the applications.

Spring MVC supports REST from version 3.0. It is easier to build restful web services with spring with it's annotation based MVC Framework. In this manual, I am going to explain how to build a simple RESTFul web service using Spring MVC 4.0 that would return plain text, later on it will return json object.

[5] http://maven.apache.org/guides/getting-started/maven-in-five-minutes.html

Now, take a look at the architecture diagram below to understand how spring mvc restful web service handles requests from client.


Spring MVC RESTFul Web Services Container
Handler Mapping
Controller
Model & View
DispatcherServlet (Front Controller)
HTTP Request
HTTP Response
CLIENT

The request process flow is as follows:
1. Client application issues request to web service in the form of URI's.
Example: http://localhost:8080/FirstApp/testMyApp

2. All HTTP Requests are intercepted by DispatcherServlet (Front End Controller). - This is defined in the web.xml file.

3. DispatcherServlet looks for Handler Mappings. Spring MVC supports three different ways of mapping request URI's to controllers : annotation, name conventions and explicit mappings. Handler Mappings section defined in the application context file, tells DispatcherServlet which strategy to use to find controllers based on the incoming request.

4. Requests are now processed by the Controller and response is returned to DispatcherServlet. DispatcherServlet looks for View Resolver section in the application context file. For RESTFul web services, where your web controller returns ModelAndView object or view names, 'ContentNegotiatingResolver' is used to find the

correct data representation format.

5. There is also an alternate option to the above step. Instead of forwarding ModelAndView object from Controller, you can directly return data from Controller using @ResponseBody annotation as depicted below in the procedure.

JSON Data Representation

JSON, or JavaScript Object Notation, is a minimal, readable format for structuring data. It is used primarily to transmit data between a server and web application, as an alternative to XML.

The following JSON example defines an employee object:

```
{"employees":[
       {"firstName":"John",
       "lastName":"Doe"
       }
]}
```

The following JSON example defines an employees object, with an array of 3 employee records:

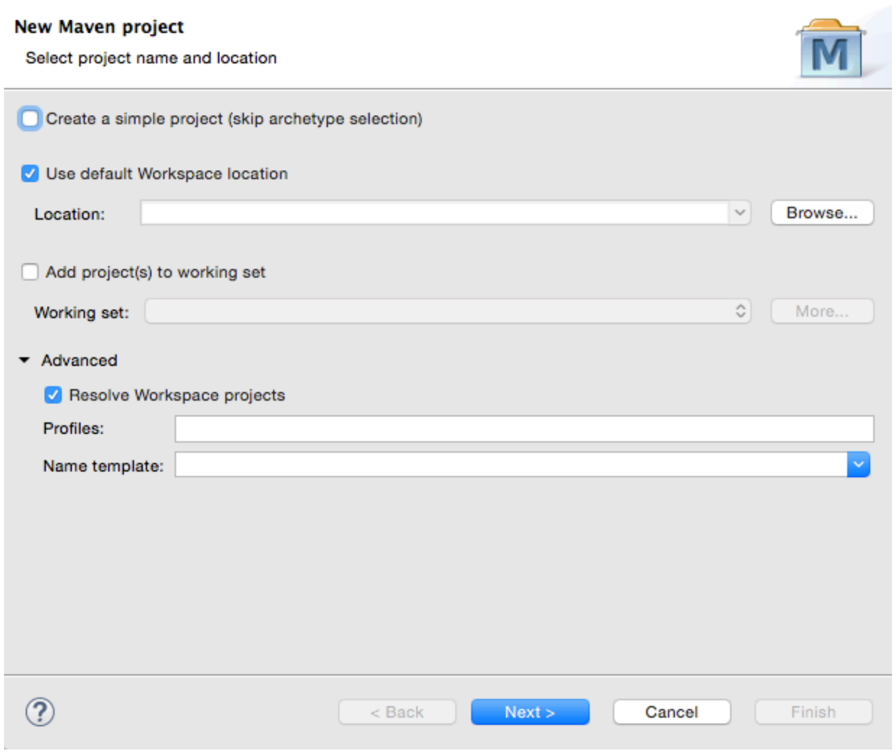
```
{"employees":[
         {"firstName":"John", "lastName":"Doe"},
         {"firstName":"Anna", "lastName":"Smith"},
        {"firstName":"Peter", "lastName":"Jones"}
]}
```

Prepare yourself well for the lab, search online for more JSON examples to get familiar with it.

Procedure: Handling simple HTTP requests

1. Create a new Maven Project from File-> New-> Maven Project

2. In the “Select project name and location” page of the wizard, make sure that “Create a simple project (skip archetype selection)” option is unchecked, hit “Next” to continue with default values.



New Maven project

Select project name and location

- [ ] Create a simple project (skip archetype selection)
- [x] Use default Workspace location

Location:

- [ ] Add project(s) to working set

Working set:

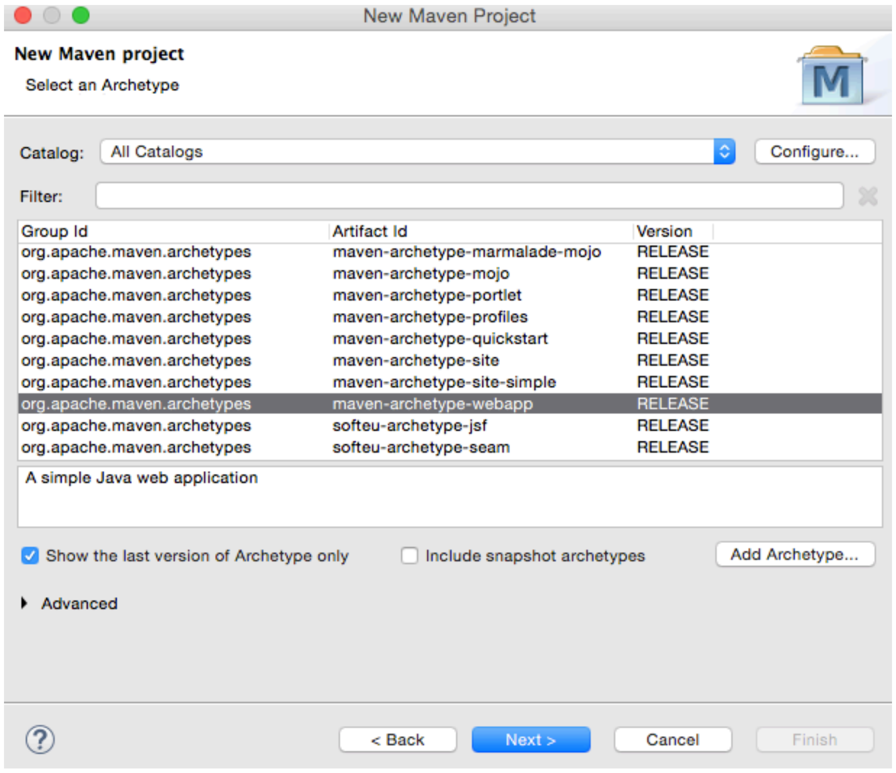
Advanced

- [x] Resolve Workspace projects

Profiles:

Name template:

3. Here the maven archetype for creating a web application must be selected. Choose the "org.apache.maven.archetypes" from the group id colum, choose the “Archetype artifact Id” variable to "maven-archetype-webapp" then press next.


New Maven Project

New Maven project

Select an Archetype

Catalog: All Catalogs

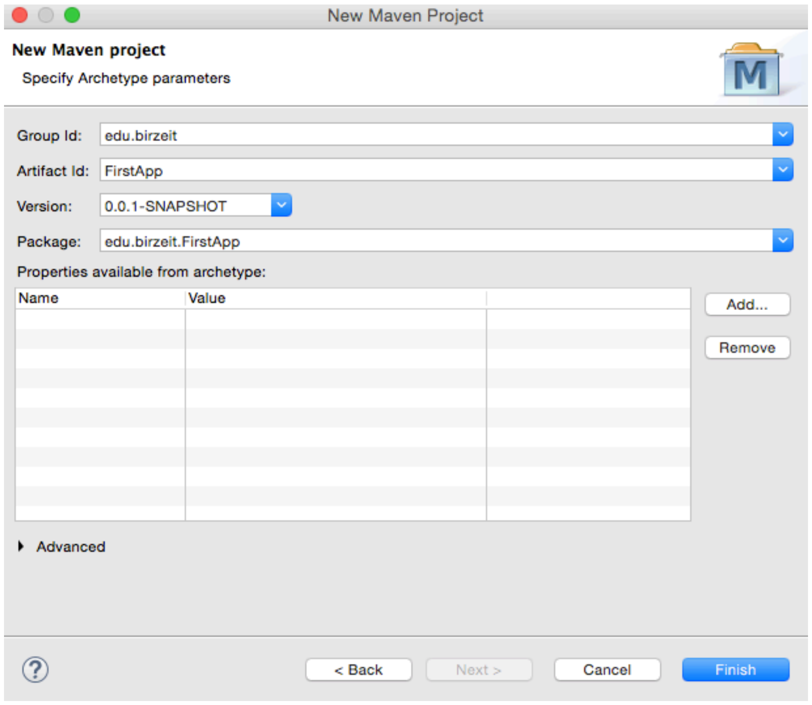
Filter:

Group Id	Artifact Id	Version
org.apache.maven.archetypes	maven-archetype-marmalade-mojo	RELEASE
org.apache.maven.archetypes	maven-archetype-mojo	RELEASE
org.apache.maven.archetypes	maven-archetype-portlet	RELEASE
org.apache.maven.archetypes	maven-archetype-profiles	RELEASE
org.apache.maven.archetypes	maven-archetype-quickstart	RELEASE
org.apache.maven.archetypes	maven-archetype-site	RELEASE
org.apache.maven.archetypes	maven-archetype-site-simple	RELEASE
org.apache.maven.archetypes	maven-archetype-webapp	RELEASE
org.apache.maven.archetypes	softeu-archetype-jsf	RELEASE
org.apache.maven.archetypes	softeu-archetype-seam	RELEASE

A simple Java web application

[✓] Show the last version of Archetype only
[ ] Include snapshot archetypes

4. In the following window fill in the group id and artifact id for your project, group id is the identifier of your project where the artifact id is the name of your project. Then Press Finish.


New Maven Project

New Maven project

Specify Archetype parameters

Group Id: edu.birzeit

Artifact Id: FirstApp

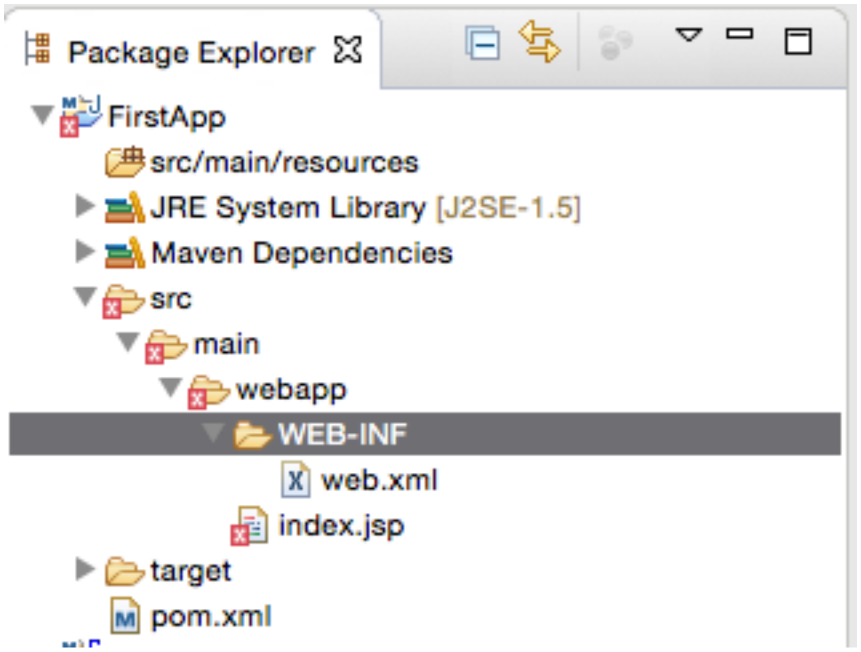
Version: 0.0.1-SNAPSHOT

Package: edu.birzeit.FirstApp

Properties available from archetype:

Name	Value

5. This is what you should be seeing in the project explorer:



6. Add Spring-MVC dependencies

Add the dependencies in Maven's pom.xml file, by editing it at the "Pom.xml" page of the POM editor. The dependency needed for MVC is the spring-webmvc package, as shown below:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>edu.birzeit</groupId>
  <artifactId>FirstApp</artifactId>
  <packaging>war</packaging>
  <version>0.0.1-SNAPSHOT</version>
  <name>FirstApp Maven Webapp </name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>

    <dependency>
                <groupId>org.springframework</groupId>
				<artifactId>spring-core</artifactId>
				<version>${spring.version}</version>
			</dependency>
			<dependency>
				<groupId>org.springframework</groupId>
				<artifactId>spring-webmvc</artifactId>
				<version>${spring.version}</version>
			</dependency>
  </dependencies>
  <build>
    <finalName>FirstApp</finalName>
  </build>

	<properties>
		<spring.version>4.0.2.RELEASE</spring.version>
	</properties>

</project>
```

Save the pom file, you will notice that the maven tool downloads the spring framework needed jars and adds it to your project.

7. Open /WEB-INF/web.xml file and add below configuration:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
	<servlet>
		<servlet-name>rest</servlet-name>
		<servlet-class>
			org.springframework.web.servlet.DispatcherServlet
		</servlet-class>
		<load-on-startup>1</load-on-startup>
	</servlet>

	<servlet-mapping>
		<servlet-name>rest</servlet-name>
		<url-pattern>/*</url-pattern>
	</servlet-mapping>

	  </web-app>
```

Note that in the above code, we have named Spring Dispatcher servlet class as "rest" and the url pattern is given as "/*" which means any url with the root of this web application will call DispatcherServlet. So what's next? DispatcherServlet will look for configuration files following this naming convention - [servlet-name]-servlet.xml. In this example, we have named dispatcher servlet class as "rest" and hence it will look for file named 'rest-servlet.xml'.

8. Now create an xml file under WEB-INF folder and name it 'rest-servlet.xml'. Copy the below in it

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:p="http://www.springframework.org/schema/p"
       xsi:schemaLocation="
        http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-4.0.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-4.0.xsd
        http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd">
       <context:component-scan base-package="edu.birzeit.controllers" />
       <mvc:annotation-driven />

</beans>
```

With ComponentScan tag, Spring auto scans all elements in the provided base package and all its child packages for Controller servlet. Also, we have used <mvc:annotation-driven> tag instead of ViewResolver, with which we can directly send response data from the controller.

9. Create the Controller

If you don't have a src/main/java source folder then create one then create a package called edu.birzeit.controllers then create a new class called HelloWorldController and add this code to it.

```java
package edu.birzeit.controllers;

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

@RestController
@RequestMapping("/")
public class HelloWorldController {
       @RequestMapping(value = "/testMyApp", method = RequestMethod.GET)
       public String test() {
              String result = "Awesome, it works :) " ;
              return result;
       }

}
```

'@RequestMapping' annotation is used for defining incoming request urls for class and method levels. In this case all url's in the format <root-url>/ will be routed to this Controller class where root-url = "https://localhosy:8080/FirstApp". With *@RequestMapping* annotation, we can only define generic uri mappings.

I explained that while using <mvc:annotation-config> tag instead of view resolver, we use *'@ResponseBody'* annotation to return response data directly from controller. But in the above code, I have not used*'@ResponseBody'*. This is because, in Spring MVC 4.0, they have introduced *'@RestController'* such that we need not use *'@ResponseBody'* tag in each and every method. *'@RestController'* will handle all of that at the type level.

10. Go a head and run the project on the embedded server of the STS tool suit, or start the server and request the url http://localhost:8080/FirstApp/testMyApp on any browser you have. What do you see?

11. What about passing parameters in the url you request? Let's pass some parameters and see how to handle it.

Add the following method to the controller:

```
@RequestMapping(value = "/testWithParams", method = RequestMethod.GET)
    public String handlePassedParams(
                    @RequestParam(value = "name", defaultValue = "no-name") String name,
                    @RequestParam(value = "age", defaultValue = "23") int id) {
            if (name == null) {
                    name = "don't send me null parameters again!! ";
            }
            String result = " Hello, " + name + " your age is: " + id;
            return result;
    }
```

As you can see, when you want to handle parameters in some request url, you simply define @RequestParam with the key for that parameter and the default value to consider if the parameter is not sent.
Go ahead and request this url and see the results:
http://localhost:8080/FirstApp/testWithParams?name=Mus'ab&age=25

Task1: Add a new method to the controller that can handle a request with three integer parameters and return them in ascending order.

All what you did above is just an introduction to how to create a rest service, nowadays, almost no one use the url it self to send parameters in most applications. What is really used to send and receive data is the JSON Objects. Below you will learn how to receive a json object and parse it and how to send json object when the client asks for it.

Handling HTTP Requests with JSON objects

Before continuing, you need to know that HTTP request with json objects can be in the form of GET or POST, in GET requests, a client is asking the server for some data, while in the POST request, the client is sending some data to the server. We will handle these two cases. Moreover, in order to handle these requests correctly, JSON objects must be transformed into java objects and vise-versa, this conversion can be done manually by developer or one can simply use library such as JACKSON to make the conversion.

1. Add the JACKSON dependency in the pom file by adding the following code under the dependencies tag:

    ```
    <dependency>
       <groupId>com.fasterxml.jackson.core</groupId>
       <artifactId>jackson-databind</artifactId>
       <version>2.5.0</version>
    </dependency>
    ```

    This few lines load the JSON binding library when maven tool build the project and adds it to jars of your project.

2. Create a new package and name it: edu.birzeit.bo, in this package create a class called Student and add the following code inside it.

```
package edu.birzeit.bo;

import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonProperty;

public class Student {

	private String name;
	private String address;
	private String major;


	@JsonCreator
    public Student(@JsonProperty("name") String name,@JsonProperty("city")
String address,@JsonProperty("Major") String major) {
         this.name = name;
         this.address = address;
         this.major = major;
    }
	public String getName() {
		return name;
```

```
	}
	public void setName(String name) {
		this.name = name;
	}
	public String getAddress() {
		return address;
	}
	public void setAddress(String address) {
		this.address = address;
	}
	public String getMajor() {
		return major;
	}
	public void setMajor(String major) {
		this.major = major;
	}
}
```

Note that annotations are indicate that when we return an student object or a list of students from a rest service, JSON object is generated to represent this or these objects, if you didn't understand this, feel free to ask your teaching assistant.

3. Create a new controller and call it StudentsController and add this code to it:

```
package edu.birzeit.controllers;

import java.util.ArrayList;

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

import com.fasterxml.jackson.databind.ObjectMapper;

import edu.birzeit.bo.Student;

@RestController
@RequestMapping("/")
public class StudentsController {


	@RequestMapping(value = "/getAllStudents", method = RequestMethod.GET)
	public ArrayList<Student> getAllStudents() {
		ArrayList<Student> list = new ArrayList<Student>();

		Student s = new Student("John","London","CSE");
		list.add(s);

		s = new Student("Mike","Dubai","CS");
```

```
            list.add(s);

            s = new Student("Steve","LA","Marketing");
            list.add(s);

            return list;
        }
}
```

4. Run the application on the server and notice the output you see when requesting the following url: http://localhost:8080/FirstApp/getAllStudents

Note that the returned data from this request is statically generated in the controller for the sake of simplicity, in all applications, the list of objects to be returned is retrieved from another service or from a database, you should be familiar with databases and how to connect to a database from java.

TASK2: Create a new method in the Students controller that can accept a parameter "major" and returns as a JSON object all students that study that sent major.

Notes:

- Please keep your work for next lab, it's very important to finish all tasks of this experiment.
- In next lab, you will learn how to post JSON objects to a controller and convert this json object to Java objects, do what the controller is asked to and return the result back to the client.
- In the next lab you will learn how to integrate an android app with such a back-end service, how to post json objects to it and how to receive the reply in the form of json objects as well.