* **What is Spring?**
  + Spring has a container, called spring application context that creates and manages application components called beans.
  + Those components are wired together inside the spring application context to make a complete application
    - Like bricks, cement, nails etc are wired together to make a house.
  + Wiring beans together is based on a pattern called dependency injection.
* **Configuration Class:**

@Configuration

public class ServiceConfiguration

{

@Bean

public InventoryService inventoryService() {

return new InventoryService();

}

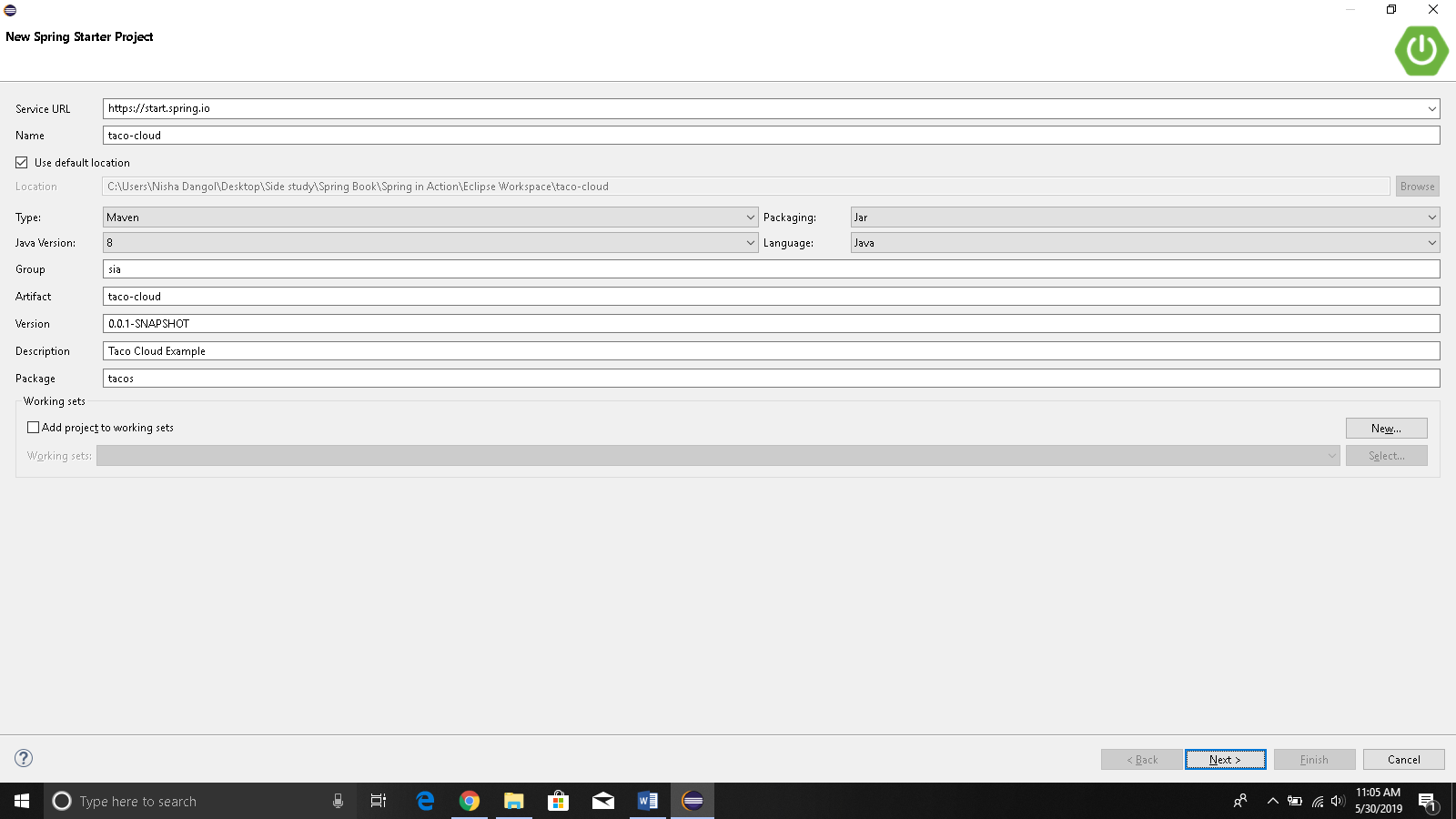
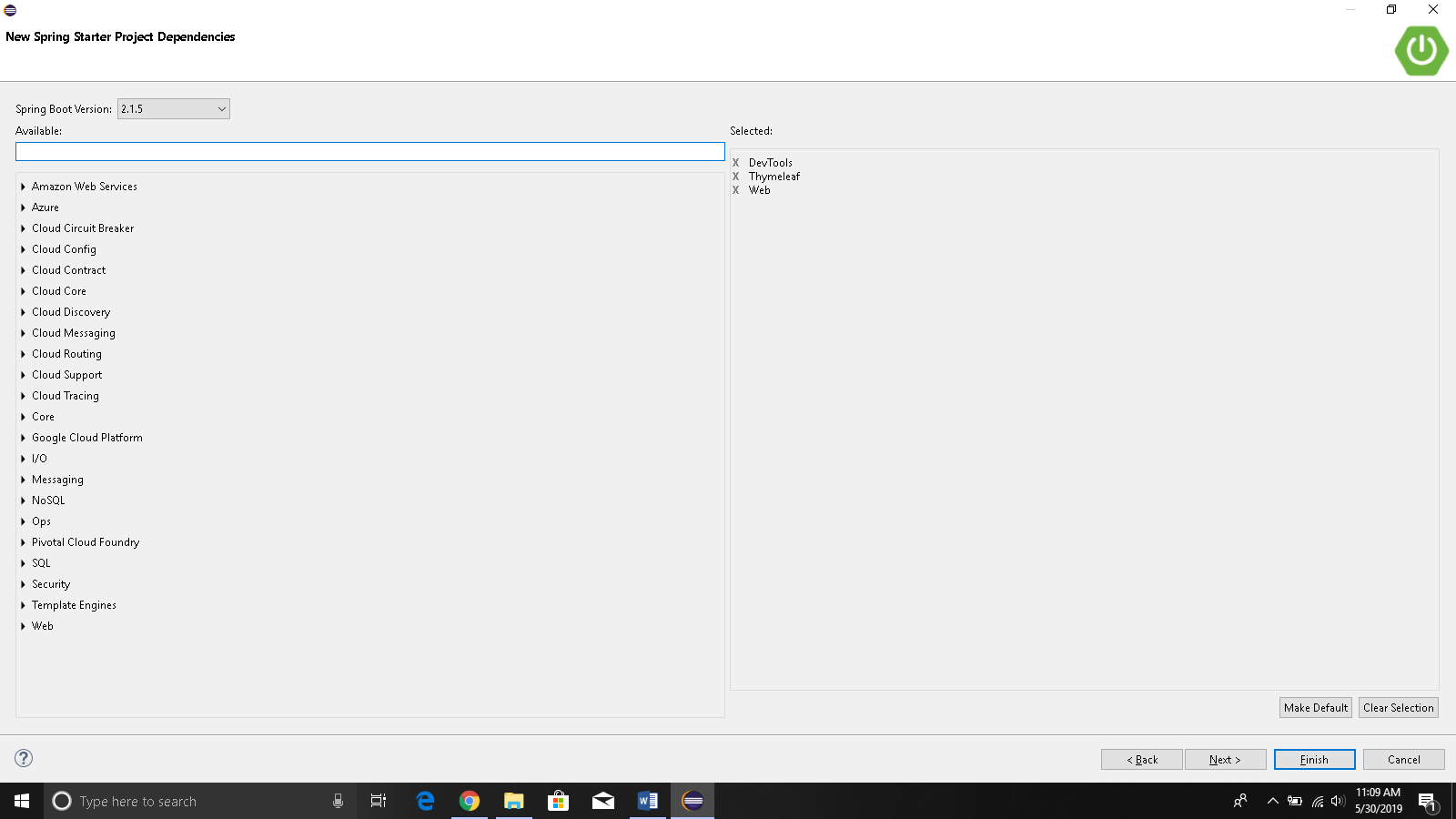
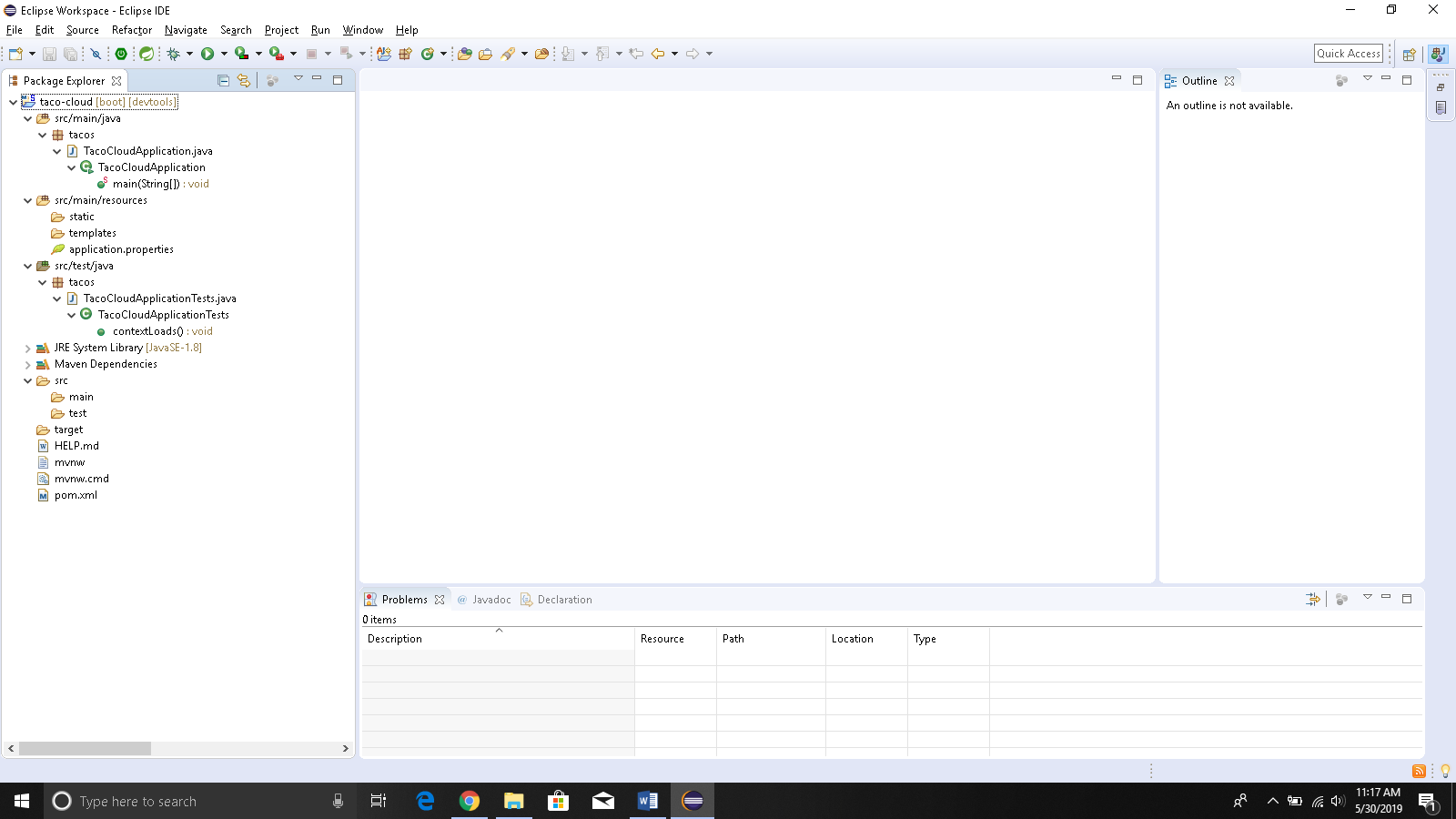
@Bean

public ProductService productService() {

return new ProductService(inventoryService());

}

}

* The @configuration annotation indicates this is a configuration class that will provide beans to the Spring application context (Container).
* The configuration’s class methods are annotated with @Bean, indicating that objects they return should be added as beans in the application context.
* Their respective bean ids will be the same as the names of methods that define them.
* **SPRING STARTER PROJECT**
  + Download Spring tool suite for eclipse
  + Create new spring starter project and put the following information on a dialogue box:
  + 
    - Jar because all java cloud platforms can run executable JAR file but not WAR files but traditional java application server requires a WAR files.
  + After this you will choose the dependencies as follows;
  + 
  + Click finish to download the project from the initializer.
  + In Eclipse, you will find the following project:
  + 
  + These are the details about the files that we got:
    - ϒ mvnw and mvnw.cmd—These are Maven wrapper scripts. You can use these scripts to build your project even if you don’t have Maven installed on your machine.
    - ϒ pom.xml—This is the Maven build specification. We’ll look deeper into this in a moment.
    - ϒ TacoCloudApplication.java—This is the Spring Boot main class that bootstraps the project. We’ll take a closer look at this class in a moment.
    - ϒ application.properties—This file is initially empty, but offers a place where you can specify configuration properties. We’ll tinker with this file a little in this chapter, but I’ll postpone a detailed explanation of configuration properties to chapter 5. Figure 1.6 The initial Spring project structure as shown in Spring Tool Suite 12 CHAPTER 1 Getting started with Spring
    - ϒ static—This folder is where you can place any static content (images, stylesheets, JavaScript, and so forth) that you want to serve to the browser. It’s initially empty.
    - ϒ templates—This folder is where you’ll place template files that will be used to render content to the browser. It’s initially empty, but you’ll add a Thymeleaf template soon.
    - ϒ TacoCloudApplicationTests.java—This is a simple test class that ensures that the Spring application context loads successfully. You’ll add more tests to the mix as you develop the application.
  + **@SpringBootApplication** is a composite application that combines three other annotations:
    - **@SpringBootConfiguration**—Designates this class as a configuration class. Although there’s not much configuration in the class yet, you can add Javabased Spring Framework configuration to this class if you need to. This annotation is, in fact, a specialized form of the @Configuration annotation.
    - **@EnableAutoConfiguration**—Enables Spring Boot automatic configuration. We’ll talk more about autoconfiguration later. For now, know that this annotation tells Spring Boot to automatically configure any components that it thinks you’ll need.
    - **@ComponentScan**—Enables component scanning. This lets you declare other classes with annotations like @Component, @Controller, @Service, and others, to have Spring automatically discover them and register them as components in the Spring application context.
  + **@RunWith** is a JUnit annotation, providing a test runner that guides JUnit in running a test
* **WRITING A SPRING APPLICATION**
  + HANDLING WEB REQUEST

package tacos;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.GetMapping;

//Identifies this class as a component for component scanning

@Controller

public class HomeController {

//if an http get request is received for the root path /, then this method will handle the //request

@GetMapping("/")

public String home() { //returns the view name

return "home";

}

}

* DEFINING THE VIEW
  + Create src/main/resources/templates/home.html
  + Save a taco image in src/main/resources/static/images/taco.jpg

<!DOCTYPE html>

<html xmlns = "http://www.w3.org/1999/xhtml"

xmlns: th = "http://www.thymeleaf.org">

<head>

<title>Taco Cloud</title>

</head>

<body>

<h1>Welcome to Taco</h1>

<img th:src="@{/images/taco.png}"/>

</body>

</html>

* TESTING THE CONTROLLER:
  + Create src/test/java/tacos/HomeControllerTest.java
  + Remove junit: configure build path🡪library🡪junit4
    - Gave me error if I didn’t remove junit for some reason.

package tacos;

import static org.hamcrest.Matchers.*containsString*;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.*get*;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.*content*;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.*status*;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.*view*;

import org.junit.Test;

import org.junit.runner.RunWith;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;

import org.springframework.test.context.junit4.SpringRunner;

import org.springframework.test.web.servlet.MockMvc;

/\*jUnit annotation that provides a test runner that guides Junit in running the test\*/

*@RunWith*(SpringRunner.class)

/\*This is a special test annotation provided by Spring Boot that arranges for the test to run in the context of a Spring MVC application.\*/

*@WebMvcTest*(HomeController.class)

public class HomeControllerTest {

*@Autowired*

private MockMvc mockMvc;

*@Test*

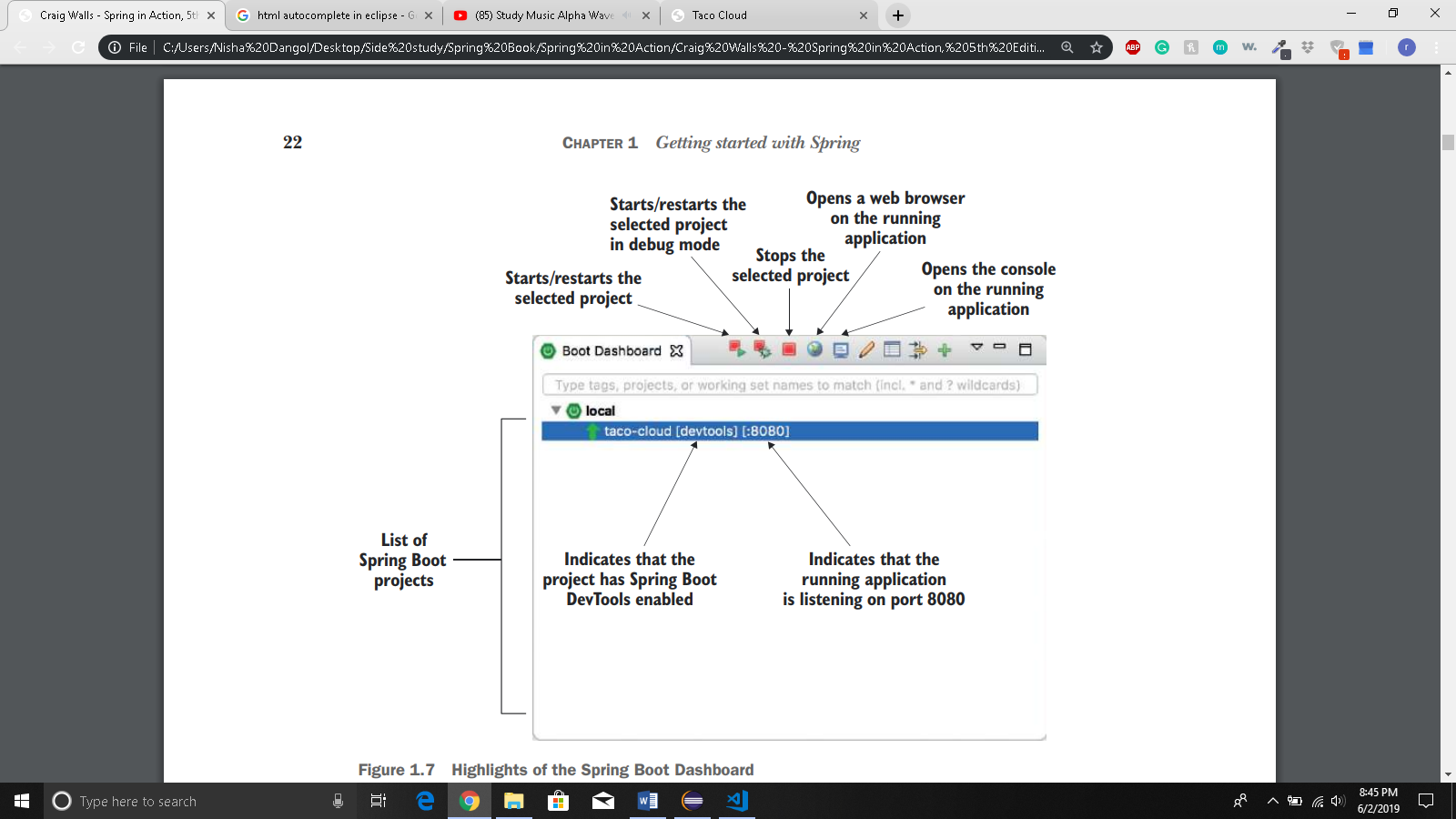
// defines the test you want to perform against the homepage.

public void testHomePage() throws Exception {

//performs an HTTP GET request for / (the root path)

mockMvc.perform(*get*("/")).andExpect(*status*().isOk()).andExpect(*view*().name("home"))

.andExpect(*content*().string(*containsString*("Welcome to...")));

* BUILDING AND RUNNING THE APPLICATION
  + Click on boot dashboard icon below the main menu
  + Select taco-cloud from the boot dashboard tab window.
  + Click on start button on top right i.e red square and green triangle
  + Click on the globe button to see the webpage you’ve created
* 
* 1.3.5 GETTING TO KNOW SPRING BOOT DEVTOOLS
  + Provides spring developers with handy development-time tools like:
    - Automatic Application restart
      * Has two class loaders, one for your code that changes frequently and the other for the dependencies that rarely changes.
      * When you change your code it automatically restarts the spring application
      * However, if you make changes in dependencies you have to manually restart the application
    - Automatic Browser refresh and Template cache disable
      * In cached templates, if we make changes to the template and refresh the browser, we cannot see the effects of those changes until you restart the application.
      * Devtools disables the template caching
      * Devtools also has livereload server along with your application that automatically reloads your browser but the browser needs to have livereload plugin installed.
    - Built in H2 console
      * Devtools automatically enables h2 console that you can access through web browser
        + Localhost:8080/h2-console