Spring and SpringBoot by Ken Kousin

https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/

What is spring?

* Provides Lot of services, like transaction, Persistence, Security
* In olden days, these were part of the application server
* The services are present, only need to be configured as per the need
* Defines code in terms of interfaces
  + We can change from one implementation to another
* Spring is metadata driven
  + XML – Old Style
  + Annotation – better – Java 1.5, Spring 2.0
  + JavaConfig – preffered
  + All are still supported
* Programmatic vs Declrative Programming
  + Programatically - Getting an instance will be to actually create an instance and call its methids
  + Declarative – Provide Anotations, and Spring will handle the specific behavior.
  + Spring Supports both programmatical and declarative but recommended to be latter
* 15 years anniversary of 1.0

ApplicatonContext

* Interface within the spring
* Accesses all the beans that the spring manages
* Collection of managed beans
* Lightweight spring container

Dependency injection

Beans that the spring manages, we do not have to manually instantiate it

Spring will handle the lifecycle of such beans

Spring Boot

Spring issue

* Apps were getting bigger and complivated
* Easy way to create and configure a spring app
* Makes building a spring app faster
* Version management
* Version conflict management
  + Add starter to the build file and spring manages to fetch all the correct dependencies
* Add a suitable starter dependency to the config file and spring configures the default beans

Spring Initializer

Java 8 is a minimum requirement

* Java 11 is current LTS
* Spring 5 (requires Java 8)
* Springboot 2.1.2
* Go to Start.spring.io
  + Build
  + Language
  + Spring boot version – this is the only version you specify
    - Choose stable version over snapshot or release candidates
  + Group-- artifact – name
  + Packaging
  + Java released major release every 6 months in march and September
  + Java 8 (EOL) 1
  + 11 are LTS versions
  + Packaging – jar and war,
    - Previously war and then the war was deployed onto the web server
    - With cloud native development jar is more powerful
      * Jar has a server bundled inside
      * Jar is an executable jar even it is a web app
      * Single jar is a deployable artifact
  + Dependencies
    - Many dependencies available in various categories
      * Spring Web
      * Thymeleaf
      * Devtools
        + Live reloads
        + Applivation restarts
      * Actuator
        + Monitor and manage your applivation
    - Explorer
      * Shows the source code before downloading
    - \*Spring web Services – For SOAP web servces
  + Download
  + Unzip
  + Import in STS
* Let the Project Build
* Overview of pom.xml
* Overview of Maven Dependencies folder

=====================TIME – 1 HOUR==================================

Walkthrough of the Application class

@SpringBootApplication

* @SpringBootTest
  + Junit5

dependency>

<groupId>org.junit.platform</groupId>

<artifactId>junit-platform-launcher</artifactId>

</dependency>

Run the Application class

SpringMVC helps to build web apps and rest web services

* In MVC Architecture, Controller work is to receive a request and connect it to a method
  + Create a Controller class
  + @RestController
    - Has reference to @Controller -> which then has @component
    - @SprinBootApplication has reference to @ComponentScan
    - Hence makes this a Managed Bean

@RestController

public class Controller {

@GetMapping("/hello")

public String method(String name) {

return "Hello";

}

}

* + - Maps the “method” to the Handler

@GetMapping("/hello")

public String method(@RequestParam(required=false, defaultValue="World") String name) {

return "Hello"+name;

}

=============================================================

@GetMapping("/hello")

public String method(@RequestParam(required=false, defaultValue="World") String name,

Model model, HttpSession session) {

model.addAttribute("user", name);

return "hello";

}

* + http://localhost:8089/hello?name=Anic
  + Spring has a Servlet called DispatcherServlet
  + Dispatcher Servlet consults Handler Mapping to say which handler to map to the URL
  + If we have thymeleaf dependency, when we return “hello” it is expocting ViewResolver to find a view name “hello” under recources/templates hello.html
* Server.port=8081
* Rename the application.properties to applivation.yml
* Change the server port and test again
* Create an index.html under resources/static with a <h2> <a href=”/hello”>hello<h2>
  + Test the navigation
  + Static folder is for htlm css js
  + Templates is for dymanic
* Execute from the Jar directly
  + Java -jar path
  + Mvn springboot:run (Check)
* Spring MVC uses server side rendering using Thymeleaf, hence this is a webapp. Viw is actually rendered on the server and displayed
* Now a days, especially with Mobile apps we have with React and Angular, BE is mostly RESTFul endpoint whch sends JSON data that the FE can use to present

===========================2 Hours=============================================

JUNIT

**public** **class** **HelloControllerUnitTest** {

***@Test***

**public** **void** **sayHello**() {

**Controller** **controller** = **new** Controller();

Model **model** = **new** BindingAwareModelMap();

**String** **result** = controller.method("Hello", model);

*assertEquals*("Hello", result);

*assertEquals*("Hello",model.asMap().get("user"));

}

* When one assertion fails, next on does not execute
* @Autowired – Springs Dependency Management
* Already done above as part of @Component annotation
  + Anythng that is in application Context can be autowired
  + All the @Autowired annotated objects are instantiated and put in application context on startup
* ======================================================

@WebMvcTest(HelloController.class) or AutoConfigureMockMvc – For legacy Spring

public class HelloControllerMockMVCTest {

@Autowired

private MockMvc mvc;

@Test

public void testHelloWithoutName() throws Exception {

mvc.perform(get("/hello").accept(MediaType.TEXT\_PLAIN))

.andExpect(status().isOk())

.andExpect(view().name("hello"))

.andExpect(model().attribute("user", is("World")));

}

@Test

public void testHelloWithName() throws Exception {

mvc.perform(get("/hello").param("name", "Dolly").accept(MediaType.TEXT\_PLAIN))

.andExpect(status().isOk())

.andExpect(view().name("hello"))

.andExpect(model().attribute("user", is("Dolly")));

}

}

========================================

}

Creating a restful Endpoint

***@GetMapping***("/greetings/{requestGreeting}")

**public** **Greetings** **greeting**(***@PathVariable*** **String** requestGreeting) {

greetings.setGreeting(requestGreeting);

**return** greetings;

}

Write Functional Test Cases

@Test

public void greetWithoutName(@Autowired TestRestTemplate template) {

ResponseEntity<Greeting> entity = template.getForEntity("/rest", Greeting.class);

assertEquals(HttpStatus.OK, entity.getStatusCode());

assertEquals(MediaType.APPLICATION\_JSON, entity.getHeaders().getContentType());

Greeting response = entity.getBody();

if (response != null) {

assertEquals("Hello, World!", response.getMessage());

}

}

@Test

public void greetWithName(@Autowired TestRestTemplate template) {

Greeting response = template.getForObject("/rest?name=Dolly", Greeting.class);

assertEquals("Hello, Dolly!", response.getMessage());

}

**==================================Day 2============================**

Spring boot provides automatic configuration

* @Bean given at a method level
* The class with a bean method will be annotated with @configuration
* Also provides automatic config for JSON processing
* Spring adds dependencies on request
  + Annotate field or setter or constructor
  + @Autowired – autowiring by type(Class or Interface), singleton instance by default
  + @Resource – Java EE, autowiring by bean name and then by type
* **@Qualifier – to distinguish between two beans of same type**

SpringBoot packages the application as a Jar file that can be deployed on any cloud provided and horizontally scaled

Jar need not be deployed the tomcat, instead tomcat is embedded in every jar

Micronaut is designed specifically for microservices and has discovery, failover embedded in it.

Makes it cloud independent.

Spring was originally designed for monoliths – 15 years ago

**Extra - Calling REST Services with RestTemplate**

**Sync Call**

@ Component – Spring Bean

@Controller, @Service @Repository @Configuration– Specialization of Component, so when we do component scan all these beans will be detected and added automatically.

**Calling rest service with reactive WebClient**

[**http://www.icndb.com/api/**](http://www.icndb.com/api/)

**Data Connectivity – Part2 - 03:15:00**

Spring JdbcTemplate

DAO Interface, Class implements the interface, add methods that use the JdbcTemplate

Jdbc template will contain the actual SQL

**Spring JPA**

@Entity

@Table(name=”Officers”)

@Id @GeneratedValue

For DAO

EntityManager

**Spring Data JPA**

public interface OfficerRepository extends JpaRepository<Officer, Integer>{ }

**HAL Explorer**

implementation 'org.springframework.boot:spring-boot-starter-data-rest'

implementation 'org.springframework.data:spring-data-rest-hal-explorer'