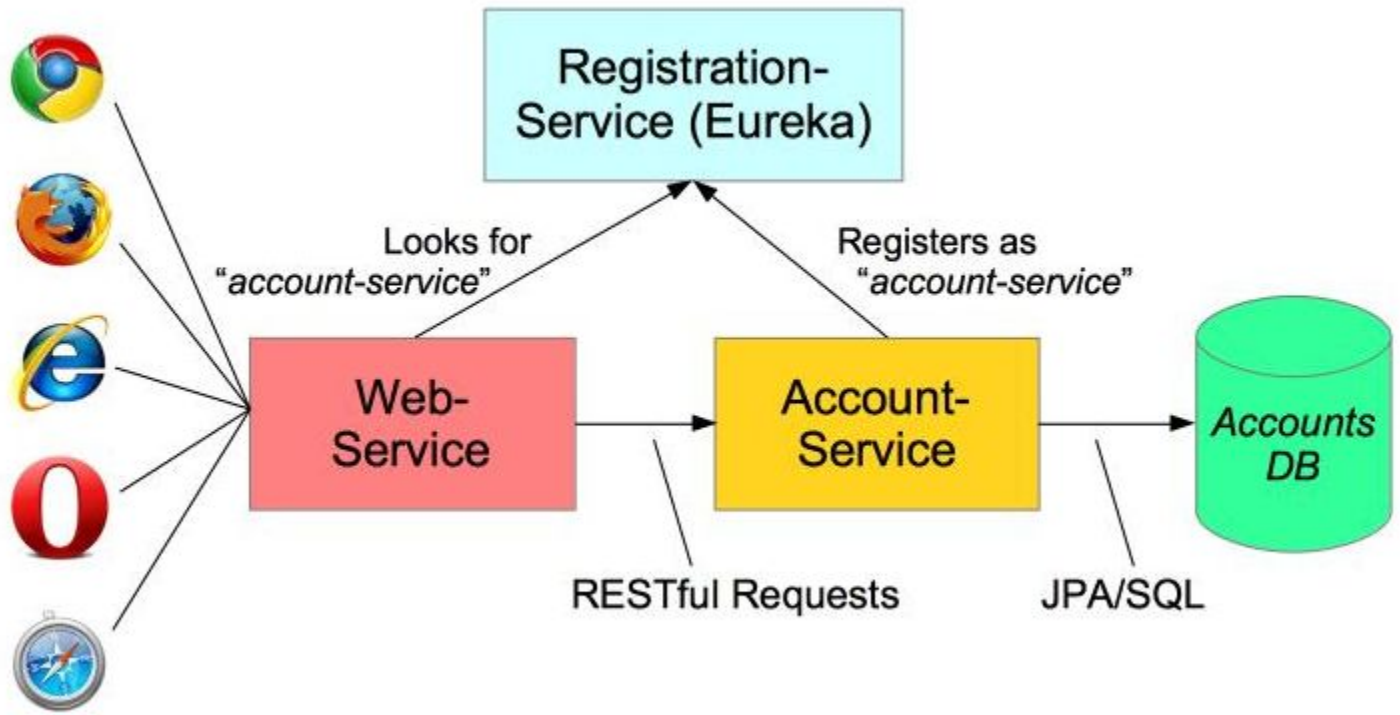
Introduction to Springboot

Java Spring Boot (Spring Boot) is a tool that makes developing web application and microservices with Spring Framework faster and easier

Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run".

Architectural style



Code Deep Dive

```xml
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.5.4</version>
</parent>
```

The spring-boot-starter-parent project is a special starter project – that provides default configurations for our application and a complete dependency tree to quickly build our Spring Boot project.

It also provides default configuration for Maven plugins such as maven-failsafe-plugin, maven-jar-plugin, maven-surefire-plugin, maven-war-plugin.

Beyond that, it also inherits dependency management from spring-boot-dependencies which is the parent to the spring-boot-starter-parent.

Create Your first project

```xml
<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 https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.5.4</version>
    </parent>
    <groupId>com.training</groupId>
    <artifactId>first-springboot</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>first-springboot</name>
    <description>Demo project for Spring Boot</description>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <java.version>11</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
```

Property Overrides

Running java version

```
<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <java.version>11</java.version>
</properties>
```

Managing Dependencies

Once, we've declared the starter parent in our project, we can pull any dependency from the parent by just declaring it in our dependencies tag.

Also, we don't need to define versions of the dependencies, Maven will download jar files based on the version defined for starter parent in the parent tag.

For example, if we're building a web project, we can add spring-boot-starter-web directly, and we don't need to specify the version:

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

The Spring Boot Maven Plugin provides Spring Boot support in Apache Maven. It allows you to package executable jar or war archives, run Spring Boot applications, generate build information and start your Spring Boot application prior to running integration tests.

```
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
```

```
<project>
    <properties>
        <app.profiles>local,dev</app.profiles>
    </properties>
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                    <profiles>${app.profiles}</profiles>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

Build

mvn clean install

```
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by com.google.inject.internal.cglib.core.$ReflectUtils$1 (file:/usr/share/maven/lib/guice.jar) to method java.lang.ClassLoader.defineClass(java.lang.String,byte[],int,int,java.security.
tectionDomain)
WARNING: Please consider reporting this to the maintainers of com.google.inject.internal.cglib.core.$ReflectUtils$1
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
[INFO] Scanning for projects...
```

```
13:02:32.794 [main] DEBUG org.springframework.test.context.support.TestPropertySourceUtils - Adding inlined properties to environment: {spring.jmx.enabled=false, org.springframework.boot.test.context.SpringBootTestContextBo
otstrapper=true}

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.5.4)

2021-08-29 13:02:32.980  INFO 1046911 --- [           main] com.training.demo.DemoApplicationTests   : Starting DemoApplicationTests using Java 11.0.10 on WCAR-9L9Z7C3 with PID 1046911 (started by aa100418 in /home/aa100418
/gitrama/tcs_microservice_training/Day1/first-springboot)
2021-08-29 13:02:32.983  INFO 1046911 --- [           main] com.training.demo.DemoApplicationTests   : No active profile set, falling back to default profiles: default
2021-08-29 13:02:33.302  INFO 1046911 --- [           main] com.training.demo.DemoApplicationTests   : Started DemoApplicationTests in 0.506 seconds (JVM running for 1.134)
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.974 s - in com.training.demo.DemoApplicationTests
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO]
[INFO] --- maven-jar-plugin:3.2.0:jar (default-jar) @ first-springboot ---
```

```java
package com.training.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class DemoApplication {
  public static void main(String[] args) { SpringApplication.run(DemoApplication.class, args); }

}
```

Many Spring Boot developers like their apps to use auto-configuration, component scan and be able to define extra configuration on their "application class". A single @SpringBootApplication annotation can be used to enable those three features, that is:

@EnableAutoConfiguration: enable Spring Boot's auto-configuration mechanism

@ComponentScan: enable @Component scan on the package where the application is located (see the best practices)

@Configuration: allow to register extra beans in the context or import additional configuration classes

The @SpringBootApplication annotation is equivalent to using @Configuration, @EnableAutoConfiguration, and @ComponentScan with their default attributes, as shown in the following example:

```java
import org.springframework.boot.SpringApplication;
import org.springframework.context.annotation.ComponentScan
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Import;

@Configuration
@EnableAutoConfiguration
@Import({ MyConfig.class, MyAnotherConfig.class })
public class Application {

        public static void main(String[] args) {
                        SpringApplication.run(Application.class, args);
        }

}
```

Run

mvn spring-boot:run

```
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by com.google.inject.internal.cglib.core.$ReflectUtils$1 (file:/usr/share/maven/lib/guice.jar) to method java.lang.Cla
tectionDomain)
WARNING: Please consider reporting this to the maintainers of com.google.inject.internal.cglib.core.$ReflectUtils$1
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
[INFO] Scanning for projects...
[INFO]
[INFO] -------------------< com.training:first-springboot >--------------------
[INFO] Building first-springboot 0.0.1-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] >>> spring-boot-maven-plugin:2.5.4:run (default-cli) > test-compile @ first-springboot >>>
[INFO]
[INFO] --- maven-resources-plugin:3.2.0:resources (default-resources) @ first-springboot ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Using 'UTF-8' encoding to copy filtered properties files.
[INFO] Copying 1 resource
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.8.1:compile (default-compile) @ first-springboot ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-resources-plugin:3.2.0:testResources (default-testResources) @ first-springboot ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Using 'UTF-8' encoding to copy filtered properties files.
[INFO] skip non existing resourceDirectory /home/aa100418/gitrama/tcs_microservice_training/Day1/first-springboot/src/test/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.8.1:testCompile (default-testCompile) @ first-springboot ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] <<< spring-boot-maven-plugin:2.5.4:run (default-cli) < test-compile @ first-springboot <<<
[INFO]
[INFO]
[INFO] --- spring-boot-maven-plugin:2.5.4:run (default-cli) @ first-springboot ---
[INFO] Attaching agents: []
```

Not starting as a web container?

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.5.4)

2021-08-29 13:06:27.223  INFO 1049010 --- [           main] com.training.demo.DemoApplication        : Starting DemoApplication using Java 11.0.10 on WCAR-9L9Z7C3 with PID 1049010 (
aining/Day1/first-springboot/target/classes started by aa100418 in /home/aa100418/gitrama/tcs_microservice_training/Day1/first-springboot)
2021-08-29 13:06:27.224  INFO 1049010 --- [           main] com.training.demo.DemoApplication        : No active profile set, falling back to default profiles: default
2021-08-29 13:06:27.477  INFO 1049010 --- [           main] com.training.demo.DemoApplication        : Started DemoApplication in 0.461 seconds (JVM running for 0.681)
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  1.501 s
[INFO] Finished at: 2021-08-29T13:06:27+02:00
```

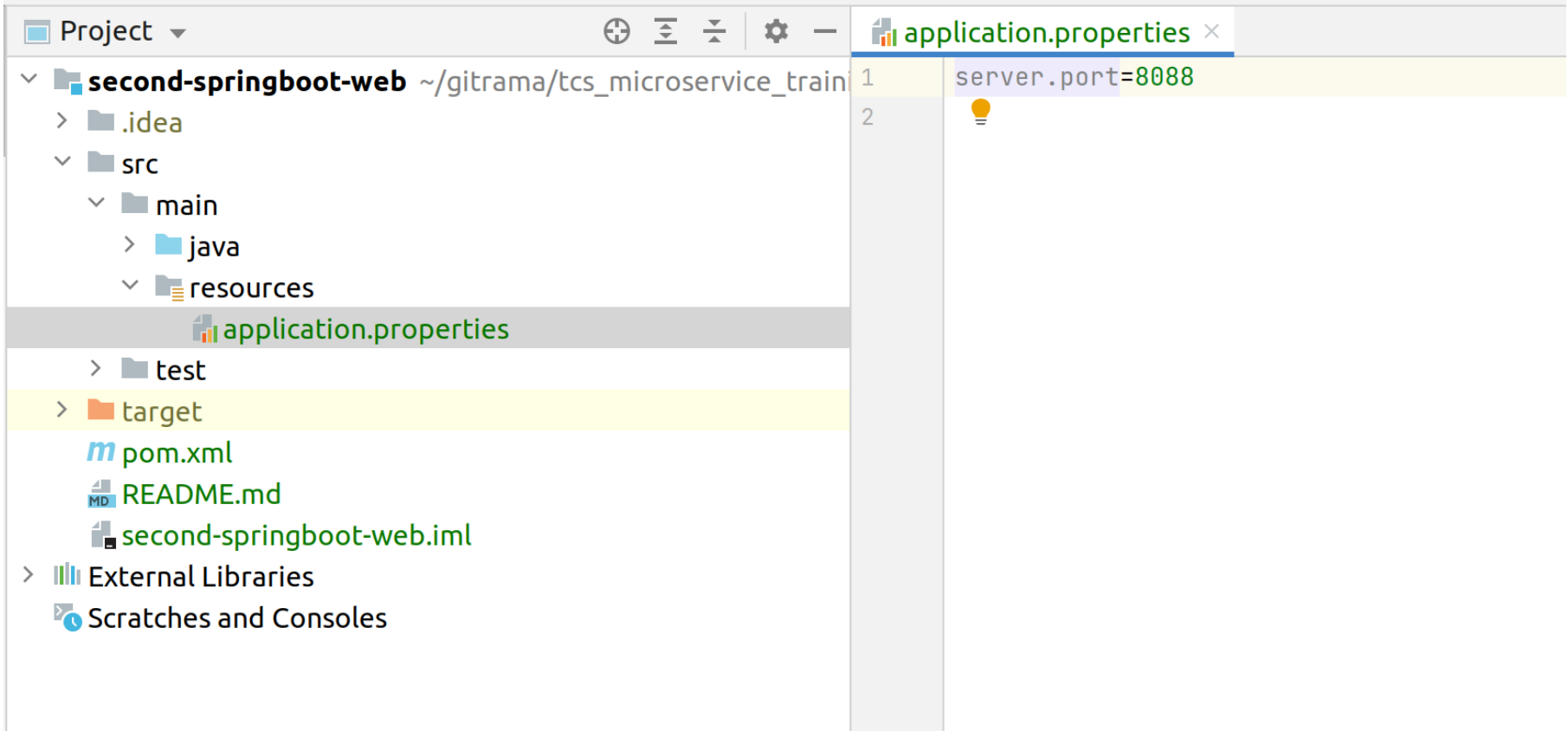
Starting as web container

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

```
[INFO] Attaching agents: []

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.5.4)

2021-08-29 13:37:24.159  INFO 1067616 --- [           main] com.training.demo.DemoApplication        : Starting DemoApplication using Java 11.0.10 on WCAR-9L9Z7C3 with PID 1067616 (/home/aa100418/gitrama/tcs_
aining/Day1/first-springboot/target/classes started by aa100418 in /home/aa100418/gitrama/tcs_microservice_training/Day1/first-springboot)
2021-08-29 13:37:24.160  INFO 1067616 --- [           main] com.training.demo.DemoApplication        : No active profile set, falling back to default profiles: default

                                                    2021-08-29 13:37:24.607  INFO 1067616 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2021-08-29 13:37:24.607  INFO 1067616 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.52]
2021-08-29 13:37:24.651  INFO 1067616 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2021-08-29 13:37:24.651  INFO 1067616 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 467 ms
                                  2021-08-29 13:37:24.845  INFO 1067616 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
2021-08-29 13:37:24.852  INFO 1067616 --- [           main] com.training.demo.DemoApplication        : Started DemoApplication in 0.918 seconds (JVM running for 1.144)
```

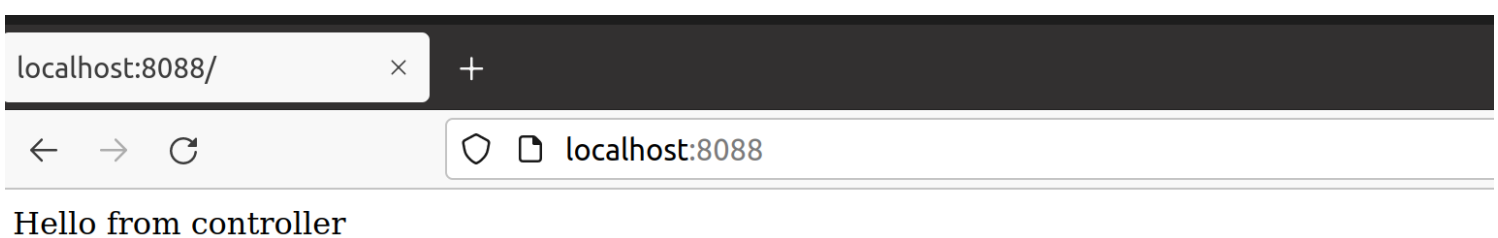
Application Properties:

https://docs.spring.io/spring-boot/docs/current/reference/html/application-properties.html


Project
second-springboot-web ~/gitrama/tcs_microservice_train
.idea
src
main
java
resources
application.properties
test
target
pom.xml
README.md
second-springboot-web.iml
External Libraries
Scratches and Consoles
application.properties
1 server.port=8088
2

Web Component

```java
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class AppController {
   /* @RequestMapping("/")
    public String getMessage() {
        return "Hello from controller";
    }*/

    @GetMapping("/")
    public String getMessage() { return "Hello from controller"; }
}
```



Lab1: Create a simple spring boot project with rest controller

Rest controller:

Spring 4.0 introduced the @RestController annotation in order to simplify the creation of RESTful web services. It's a convenient annotation that combines @Controller and @ResponseBody, which eliminates the need to annotate every request handling method of the controller class with the @ResponseBody annotation.

```
@RestController
public class AppController {
    @GetMapping("/get")
    public String getDetail() { return "get employee "; }

    @PostMapping("/post")
    Employee postEmployee(@RequestBody Employee newEmployee) { return newEmployee; }

    @PutMapping("/put/{name}")
    String putEmployee(@RequestBody Employee newEmployee, @PathVariable String name) { return newEmployee.toString() + ":Updated with name:" + name; }

    @DeleteMapping("/delete/{name}")
    String deleteEmployee(@PathVariable String name) { return name; }

    @GetMapping("/path/{name}")
    public String getPathVar(@PathVariable("name") String name) { return "Path Variable:" + name; }

    @GetMapping("/request")
    public String getRequestparam(@RequestParam(name = "name", required = true, defaultValue = "rama") String name) { return "Request Param:" + name; }

    @GetMapping("/request/params")
    public String getRequestparams(@RequestParam List<String> id) { return "Request Param:" + id; }
```

curl -X GET http://localhost:8088/get
curl -X POST http://localhost:8088/post -d '{"name": "rama", "role": "developer"}' -H 'Content-type:application/json'
curl -X PUT http://localhost:8088/put/shanker -H 'content-type: application/json' -d '{"name": "Rama", "role": "developer"}'
curl -X DELETE http://localhost:8088/delete/rama
curl -X GET http://localhost:8088/path/rama curl -X GET 'http://localhost:8088/request?name=test1'

Annotation Magic

@SpringBootApplication

One of the most basic and helpful annotations, is @SpringBootApplication. It's syntactic sugar for combining other annotations that we'll look at in just a moment. @SpringBootApplication is @Configuration, @EnableAutoConfiguration and @ComponentScan annotations combined, configured with their default attributes.

@Configuration and @ComponentScan

The @Configuration and @ComponentScan annotations that we described above make Spring create and configure the beans and components of your application. It's a great way to decouple the actual business logic code from wiring the app together.

@EnableAutoConfiguration

Now the @EnableAutoConfiguration annotation is even better. It makes Spring guess the configuration based on the JAR files available on the classpath. It can figure out what libraries you use and preconfigure their components without you lifting a finger. It is how all the spring-boot-starter libraries work. Meaning it's a major lifesaver both when you're just starting to work with a library as well as when you know and trust the default config to be reasonable.

Important Spring MVC Web Annotations

The following annotations make Spring configure your app to be a web application, capable of serving the HTTP response.
@Controller - marks the class as a web controller, capable of handling the HTTP requests. Spring will look at the methods of the class marked with the @Controller annotation and establish the routing table to know which methods serve which endpoints.

@ResponseBody - The @ResponseBody is a utility annotation that makes Spring bind a method's return value to the HTTP response body. When building a JSON endpoint, this is an amazing way to magically convert your objects into JSON for easier consumption.

@RestController - Then there's the @RestController annotation, a convenience syntax for @Controller and @ResponseBody together. This means that all the action methods in the marked class will return the JSON response.

@RequestMapping(method = RequestMethod.GET, value = "/path") - The @RequestMapping(method = RequestMethod.GET, value = "/path") annotation specifies a method in the controller that should be responsible for serving the HTTP request to the given path. Spring will work the implementation details of how it's done. You simply specify the path value on the annotation and Spring will route the requests into the correct action methods.

@RequestParam(value="name", defaultValue="World") - Naturally, the methods handling the requests might take parameters. To help you with binding the HTTP parameters into the action method arguments, you can use the @RequestParam(value="name", defaultValue="World") annotation. Spring will parse the request parameters and put the appropriate ones into your method arguments.

@PathVariable("placeholderName") - Another common way to provide information to the backend is to encode it in the URL. Then you can use the @PathVariable("placeholderName") annotation to bring the values from the URL to the method arguments.

@Configuration - used to mark a class as a source of the bean definitions. Beans are the components of the system that you want to wire together. A method marked with the @Bean annotation is a bean producer. Spring will handle the life cycle of the beans for you, and it will use these methods to create the beans.

@ComponentScan -use to make sure that Spring knows about your configuration classes and can initialize the beans correctly. It makes Spring scan the packages configured with it for the @Configuration classes.

@Import - If you need even more precise control of the configuration classes, you can always use @import to load additional configuration. This one works even when you specify the beans in an XML file like it's 1999.

@Component - Another way to declare a bean is to mark a class with a @Component annotation. Doing this turns the class into a Spring bean at the auto-scan time.

@Service - Mark a specialization of a @Component. It tells Spring that it's safe to manage them with more freedom than regular components. Remember, services have no encapsulated state.

@Autowired - To wire the application parts together, use the @Autowired on the fields, constructors, or methods in a component. Spring's dependency injection mechanism wires appropriate beans into the class members marked with @Autowired.

@Bean - A method-level annotation to specify a returned bean to be managed by Spring context. The returned bean has the same name as the factory method.

@Lookup - tells Spring to return an instance of the method's return type when we invoke it.

@Primary - gives higher preference to a bean when there are multiple beans of the same type.

@Required - shows that the setter method must be configured to be dependency-injected with a value at configuration time. Use

@Required on setter methods to mark dependencies populated through XML. Otherwise, a BeanInitializationException is thrown.

@Value - used to assign values into fields in Spring-managed beans. It's compatible with the constructor, setter, and field injection.

@DependsOn - makes Spring initialize other beans before the annotated one. Usually, this behavior is automatic, based on the explicit dependencies between beans. The @DependsOn annotation may be used on any class directly or indirectly annotated with

@Component or on methods annotated with @Bean.

@Lazy - makes beans to initialize lazily. @Lazy annotation may be used on any class directly or indirectly annotated with

@Component or on methods annotated with @Bean.

@Scope - used to define the scope of a @Component class or a @Bean definition and can be either singleton, prototype, request, session, globalSession, or custom scope.

@Profile - adds beans to the application only when that profile is active.

Lab2: Create a spring boot project and try with different rest controller annotation

Input:
Auto message communication :

Message: to,from,content,corelationid(uuid)

Operation:
Get
Post
Put
Delete

Run and test through curl and postman

Content-Negotiation:

REST stands for REpresentational State Transfer.

Key abstraction in REST is a Resource.

A resource can have multiple representations
XML
HTML
JSON
When a resource is requested, we provide the representation of the resource.

When a resource is requested, we provide the representation of the resource.
When a consumer sends a request, it can specify two HTTP Headers related to Content Negotiation
Accept and
Content-Type
Content-Type indicates the content type of the body of the request.
Accept indicates the expected content type of the response.

For example, if a consumer sends a request to http://localhost:8080/students/10001 with
Accept header as ‘application/xml’, we need to provide the xml representation of the resource.

```
<Student>
    <name>Shanker</name>
    <roll>1re106</roll>
</Student>
```

```
If a consumer sends a request with Accept header as ‘application/json’, we need to provide the JSON
representation of the resource.
{
  "id": 10001,
  "name": "Ranga",
  "passportNumber": "E1234567"
}
```

Store Resource:

```
curl -X POST \
  http://localhost:8080/students/ \
  -H 'cache-control: no-cache' \
  -H 'content-type: application/xml' \
  -d '<Student>
    <name>Shanker</name>
    <roll>1re106</roll>
</Student>'
```

Get Resource

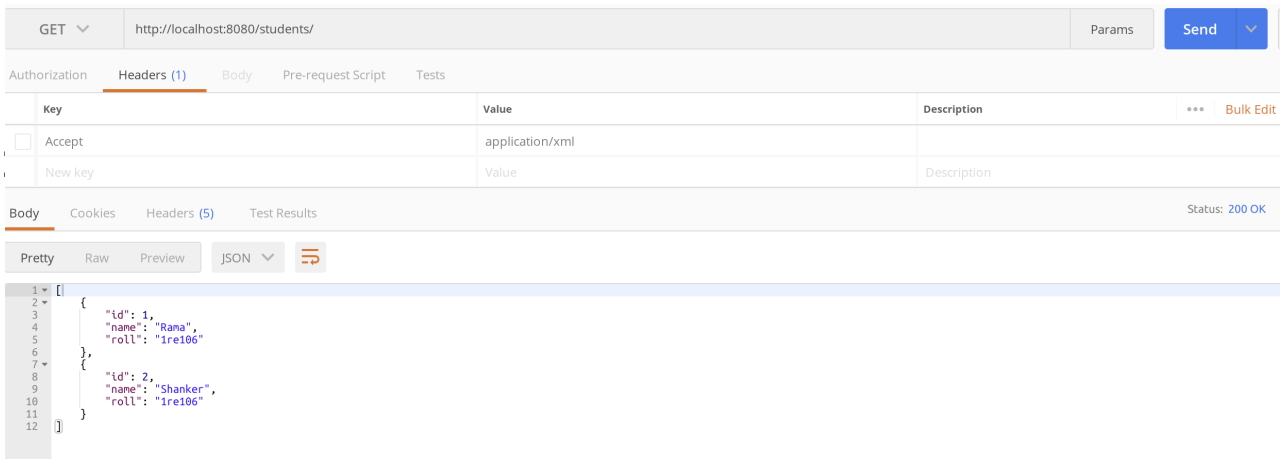
```
curl -X GET \
  http://localhost:8080/students/ \
  -H 'accept: application/xml' \
  -H 'cache-control: no-cache' \
  -H 'postman-token: cee55e76-0535-3619-20f9-d39d0aa7aa70'
```

Update content:

```
curl -X PUT \
  http://localhost:8080/students/1 \
  -H 'cache-control: no-cache' \
  -H 'content-type: application/xml' \
  -d '<Student>
    <name>Rama</name>
    <roll>1re106</roll>
</Student>'
```

Remove header:

Default its support json



THANKS !