SPRING BOOT RESTful WEB SERVICE

Instructor:

Table Content

Learning Goals

- **After the session, attendees will be able to:**

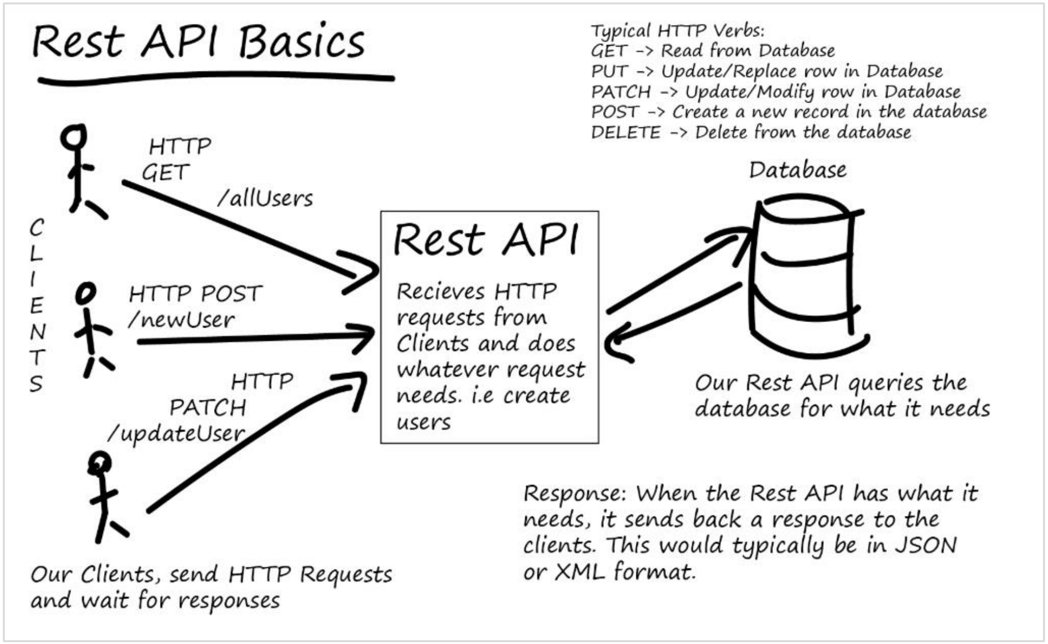
Know how to write a RESTful API web service with Spring Boot.

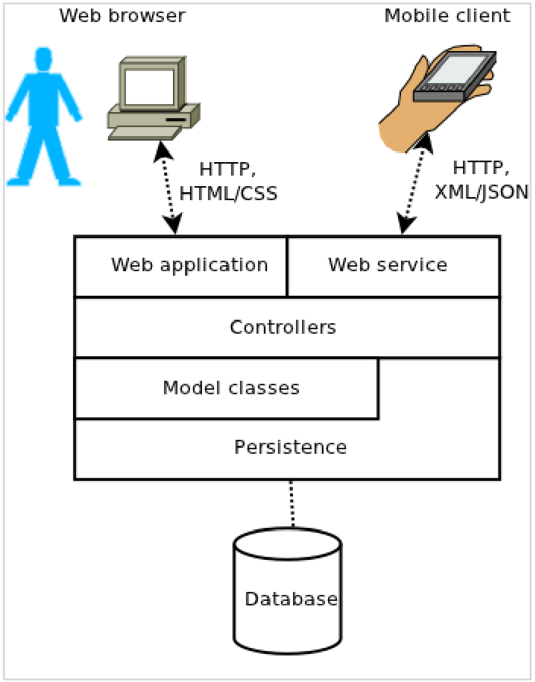
Section 1

INTRODUCTION

Introduction

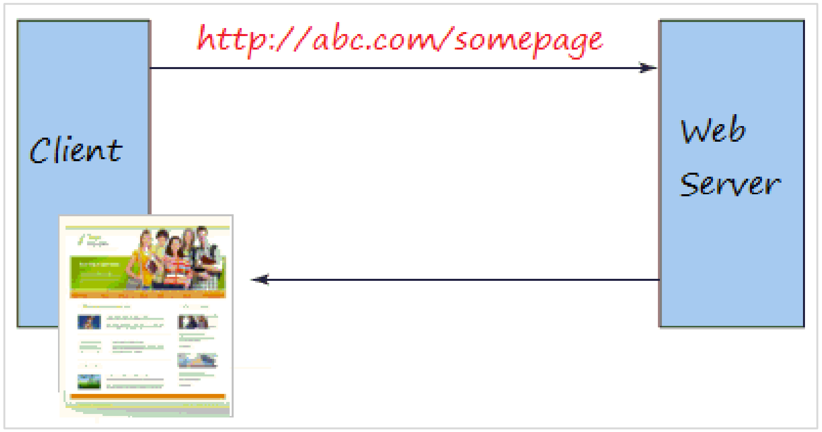
- **REST** is the acronym for **REpresentational State Transfer**.
- REST is an **architectural style** for developing applications that can be accessed over the network.
- REST architectural style was brought in light by Roy Fielding in his doctoral thesis in 2000.

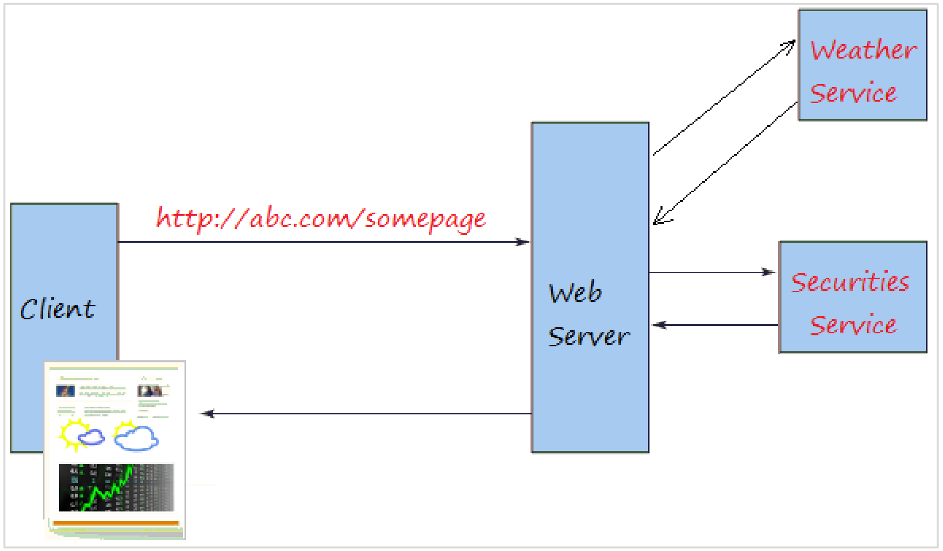

Rest API Basics
Typical HTTP Verbs:
GET -> Read from Database
PUT -> Update/Replace row in Database
PATCH -> Update/Modify row in Database
POST -> Create a new record in the database
DELETE -> Delete from the database
CLIENTS
HTTP GET /allUsers
HTTP POST /newUser
HTTP PATCH /updateUser
Rest API
Recieves HTTP requests from Clients and does whatever request needs. i.e create users
Database
Our Rest API queries the database for what it needs
Response: When the Rest API has what it needs, it sends back a response to the clients. This would typically be in JSON or XML format.
Our Clients, send HTTP Requests and wait for responses



Introduction

- RESTful web services try to define services using the different concepts that are already present in HTTP. The main goal of RESTful web services is to make web services **more effective.**
- We can build REST services with both XML and JSON. JSON is more popular format with REST.
- It can be accessed through a **Uniform Resource Identifier (URI)**

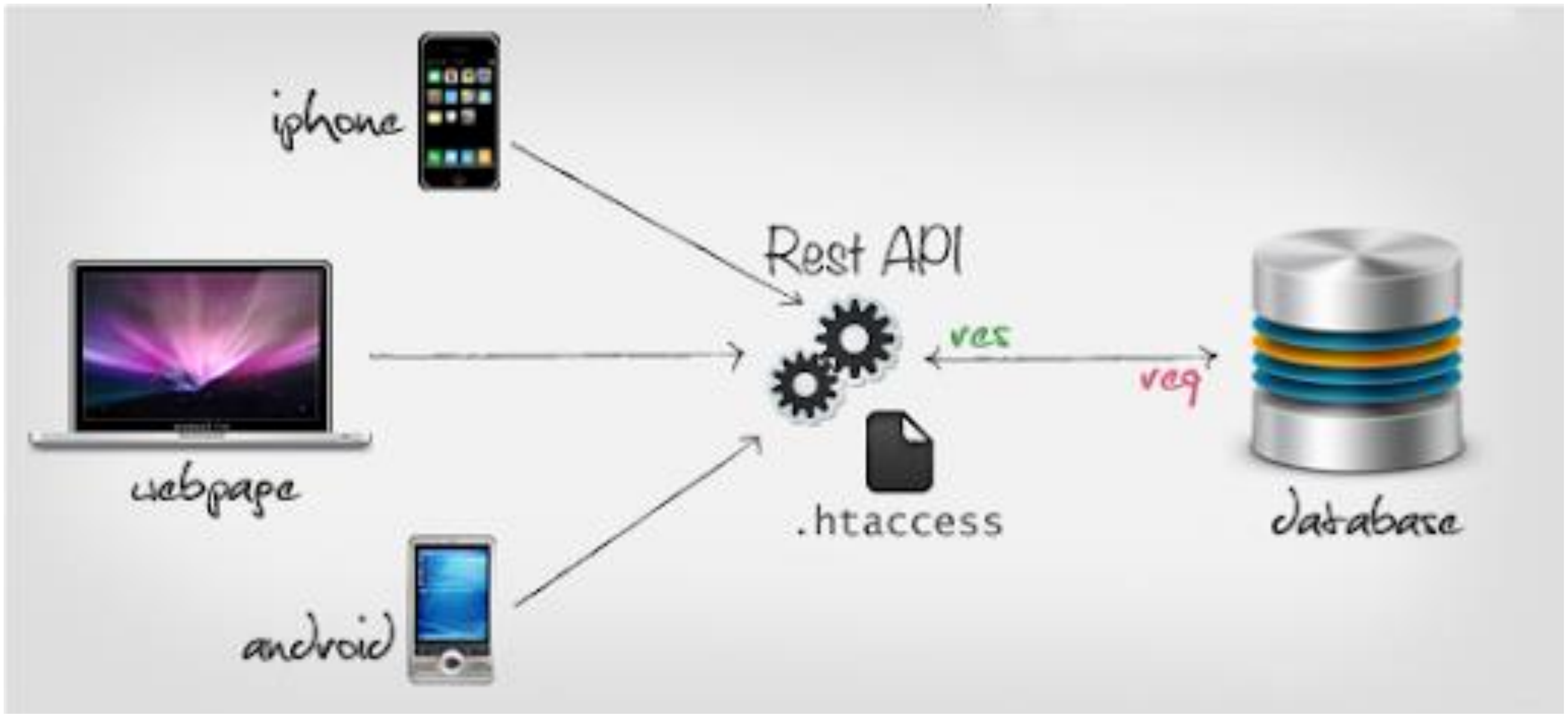




Introduction

❖ Need of REST API

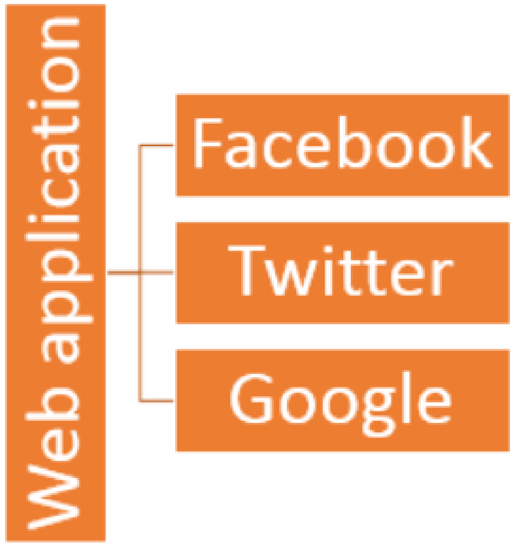
- ✓ Sharing data between two or more systems has always been a fundamental requirement of software development.



Introduction

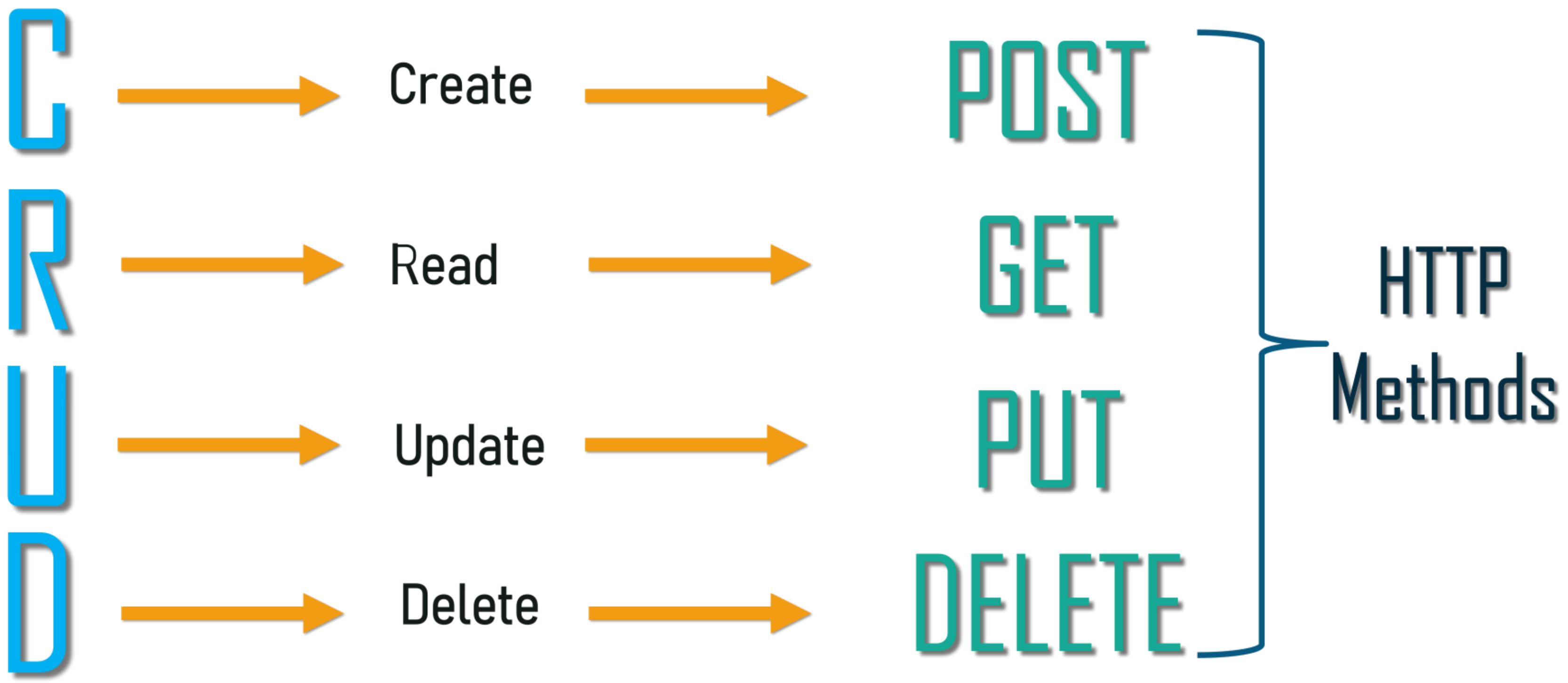
❖ Why Restful

- ✓ Heterogeneous[không đồng nhất] **languages** and **environments**:
  - It enables web applications that are built on various programming languages to communicate with each other.
  - With the help of Restful services, these web applications can reside on different environments, some could be on Windows, and others could be on Linux.
- ✓ **Example**:



Introduction

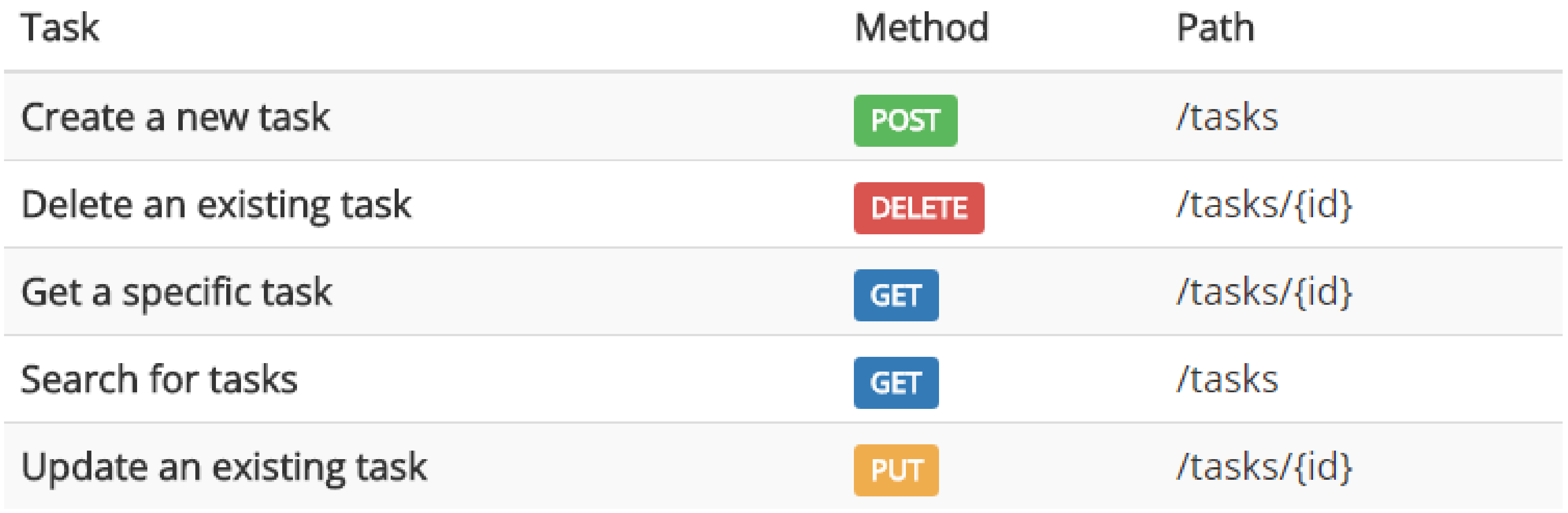
- Methods of REST API
  - ✓ All of us working with the technology of the web, do CRUD operations



Introduction

❖ Methods of REST API

✓ Example



Task	Method	Path
Create a new task	POST	/tasks
Delete an existing task	DELETE	/tasks/{id}
Get a specific task	GET	/tasks/{id}
Search for tasks	GET	/tasks
Update an existing task	PUT	/tasks/{id}

Introduction

- For example, if we want to perform the following actions in the social media application, we get the corresponding results.
  - ✓ **POST /users:** It creates a user.
  - ✓ **GET /users/{id}:** It retrieves the detail of a user.
  - ✓ **GET /users:** It retrieves the detail of all users.
  - ✓ **DELETE /users:** It deletes all users.
  - ✓ **DELETE /users/{id}:** It deletes a user.
  - ✓ **GET /users/{id}/posts/post_id:** It retrieve the detail of a specific post.
  - ✓ **POST / users/{id}/ posts:** It creates a post of the user.

Introduction

- HTTP also defines the following standard status code:
  - ✓ **404:** RESOURCE NOT FOUND
  - ✓ **200:** SUCCESS
  - ✓ **201:** CREATED
  - ✓ **401:** UNAUTHORIZED
  - ✓ **500:** SERVER ERROR
- RESTful Service Constraints
  - ✓ There must be a service producer and service consumer.
  - ✓ The service is stateless.
  - ✓ The service result must be cacheable.
  - ✓ The interface is uniform and exposing resources.
  - ✓ The service should assume a layered architecture.

Restful Web Services and SOAP

- SOAP is a **protocol** whereas REST is an **architectural style**.
- SOAP server and client applications are tightly coupled and bind with the WSDL **contract** whereas there **is no contract in REST** web services and client.
- Learning curve is easy for REST when compared to SOAP web services.
- REST web services request and response types can be XML, JSON, text etc. whereas SOAP works with XML only.
- JAX-RS is the Java API for REST web services whereas JAX-WS is the Java API for SOAP web services.

REST API Implementations

- There are two major implementations of JAX-RS API.
  - ✓ **Jersey**: Jersey is the reference implementation provided by Sun. For using Jersey as our JAX-RS implementation, all we need to configure its servlet in web.xml and add required dependencies. Note that JAX-RS API is part of JDK not Jersey, so we have to add its dependency jars in our application.
  - ✓ **RESTEasy**: RESTEasy is the JBoss project that provides JAX-RS implementation.

JSON format

- JSON Data - A Name and a Value

```
var myJSON = '{"name":"John", "age":31, "city":"New York"}';
```

- Arrays in JSON Objects

```
{
"name":"John",
"age":30,
"cars":[ "Ford", "BMW", "Fiat" ]
}
```

- Nested Arrays in JSON Objects

```
myObj = {
  "name":"John",
  "age":30,
  "cars": [
    { "name":"Ford", "models":[ "Fiesta", "Focus", "Mustang" ] },
    { "name":"BMW", "models":[ "320", "X3", "X5" ] },
    { "name":"Fiat", "models":[ "500", "Panda" ] }
  ]
 }
```

Section 2

INITIALIZING A RESTFUL WEB SERVICE PROJECT

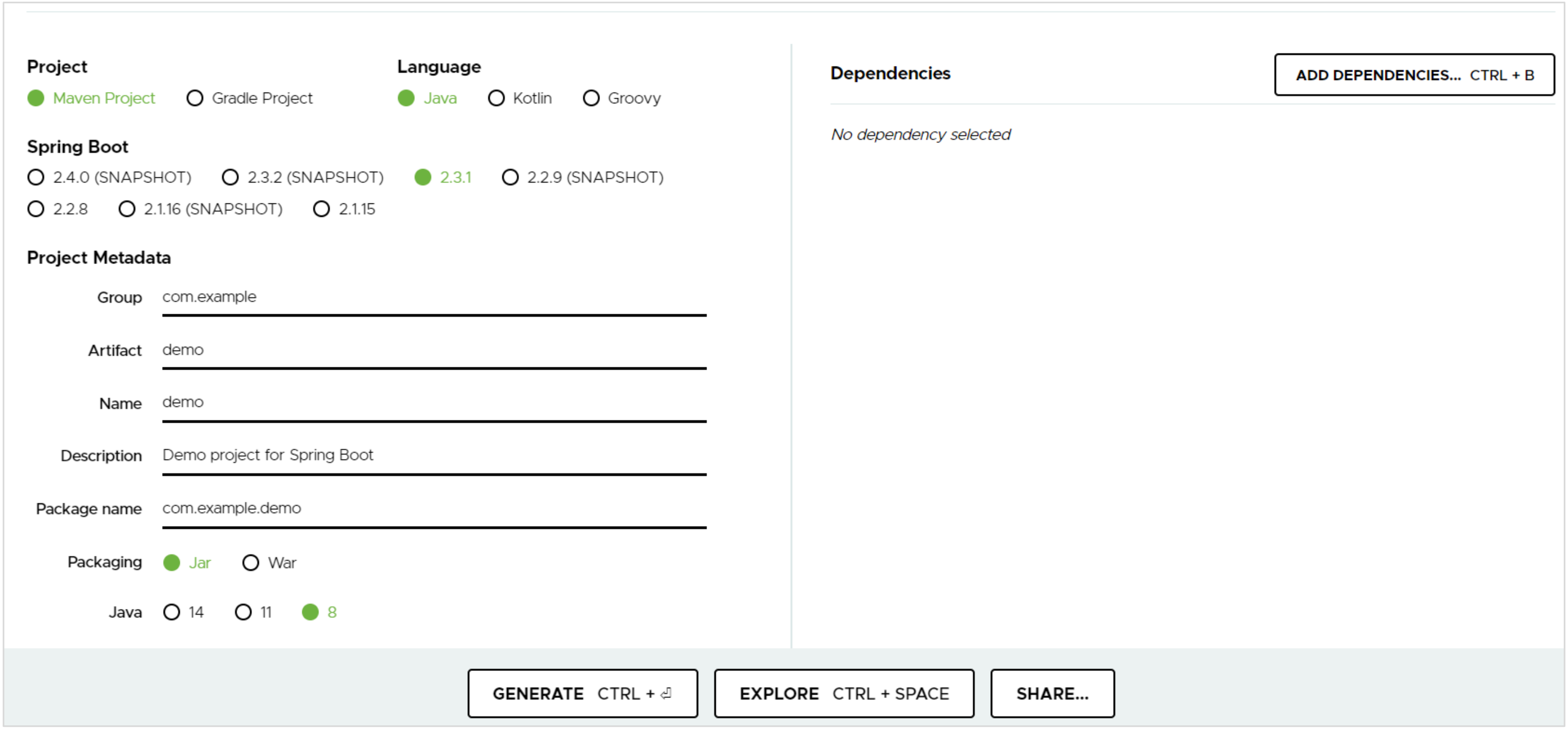
Creating a RESTful Web Service

- **The steps to create a RESTful Web Service with Spring Boot:**

1	Create the Spring Boot Project
2	Define Database configurations
3	Create an Entity Class
4	Create JPA Data Repository layer
5	Create Rest Controllers and map API requests
6	Build and run the Project

Create the Spring Boot Project

- First, go to https://start.spring.io/ and create a project with below settings



Project
[✓ Maven Project] Gradle Project

Language
[✓ Java] Kotlin Groovy

Spring Boot
2.4.0 (SNAPSHOT) 2.3.2 (SNAPSHOT) [✓ 2.3.1] 2.2.9 (SNAPSHOT)
2.2.8 2.1.16 (SNAPSHOT) 2.1.15

Project Metadata

Group	com.example
Artifact	demo
Name	demo
Description	Demo project for Spring Boot
Package name	com.example.demo
Packaging	[✓ Jar] War
Java	14 11 [✓ 8]

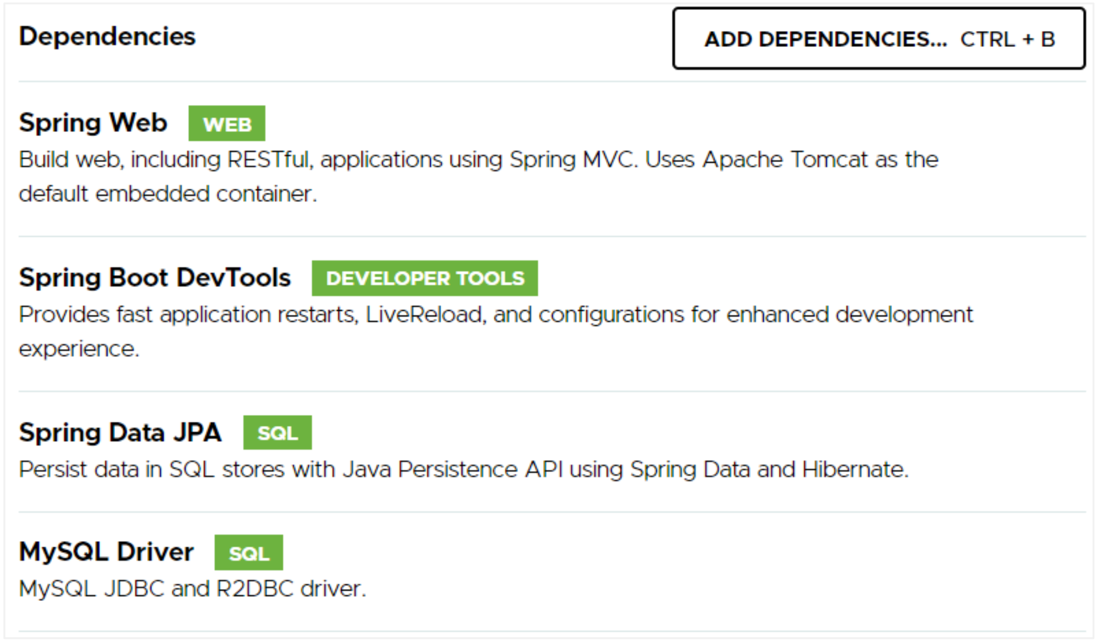
Dependencies

No dependency selected

Create the Spring Boot Project

Dependencies

- ✓ **Web**: Full-stack web development with Tomcat
- ✓ **DevTools**: Spring Boot Development Tools
- ✓ **JPA**: Java Persistence API including spring-data-JPA, spring-orm, and Hibernate
- ✓ **MySQL/SQL Server**: MySQL JDBC driver/MS SQL Server Driver SQL



Dependencies

Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Spring Boot DevTools DEVELOPER TOOLS
Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

Spring Data JPA SQL
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

MySQL Driver SQL
MySQL JDBC and R2DBC driver.

Define Database Configurations

- Next, create the database name in database server and define connection properties **application.properties**:
- **MySQL Server:**

```
## Database Properties
spring.datasource.url = jdbc:mysql://localhost:3306/db?useSSL=false
spring.datasource.username = root
spring.datasource.password = root

## Hibernate Properties
# The SQL dialect makes Hibernate generate better
# SQL for the chosen database
spring.jpa.properties.hibernate.dialect =
org.hibernate.dialect.MySQL5InnoDBDialect

# Hibernate ddl auto (create, create-drop, validate, update)
spring.jpa.hibernate.ddl-auto = update
```

Define Database Configurations

- **application.properties MS SQL Server:**

```
spring.datasource.url=jdbc:sqlserver://localhost;
                                 databaseName=HumanResourceDB
spring.datasource.username=sa
spring.datasource.password=12345678
spring.datasource.driverClassName=com.microsoft.sqlserver.jdbc
                                             .SQLServerDriver
spring.jpa.show-sql=true
spring.jpa.hibernate.dialect=org.hibernate.dialect.SQLServer2012Dialect
spring.jpa.hibernate.ddl-auto =update
```

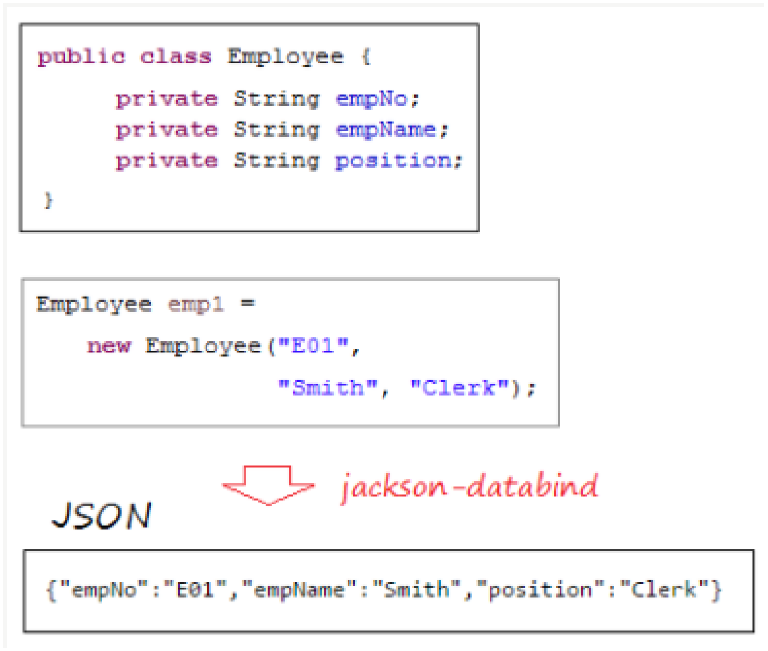
Create Entity Class

- The **@Entity** annotation specifies that the class is an entity and is mapped to a database table.

```java
@Entity
@Table(name = "USERS")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "USER_ID")
    private long userId;

     @Column(name = "USERNAME", unique = true)
    private String username;

    @Column(name = "PASSWORD")
     private String password;

     // getter and setter methods
}
```

JSON ←→ Java

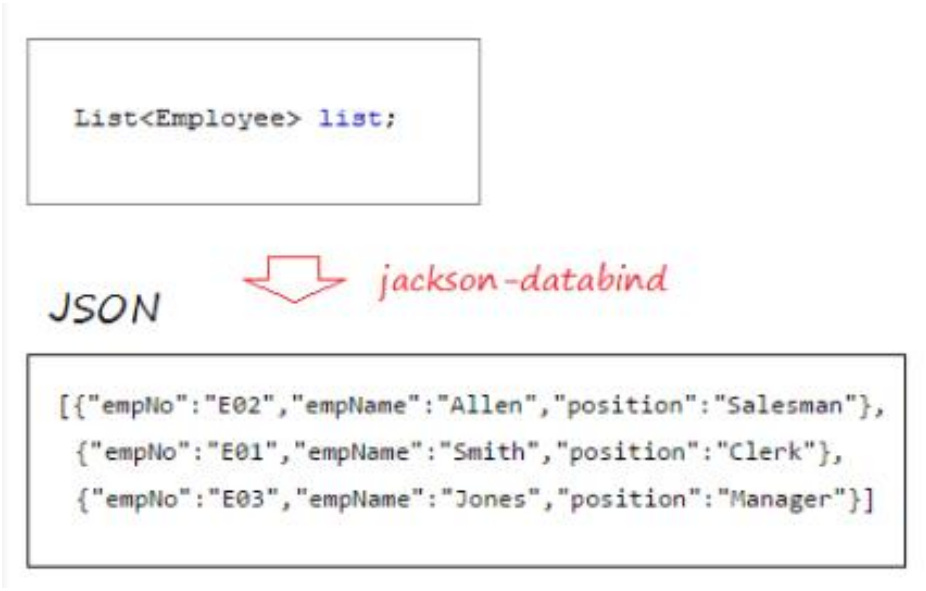
- How to convert the **JSON** object into a **Java** one and vice versa:
  - ✓ The **spring-boot-starter-web** has built in **jackson-databind**, which helps to convert **JSON** into **Java object** and vice versa.



```
public class Employee {
      private String empNo;
      private String empName;
      private String position;
}
```

```
Employee emp1 =
    new Employee("E01",
               "Smith", "Clerk");
```

```
{"empNo":"E01","empName":"Smith","position":"Clerk"}
```



```
List<Employee> list;
```

```
[{"empNo":"E02","empName":"Allen","position":"Salesman"},
 {"empNo":"E01","empName":"Smith","position":"Clerk"},
 {"empNo":"E03","empName":"Jones","position":"Manager"}]
```

Create JPA Data Repository Layer

- **@Repository** annotation indicates that an annotated class is a repository, which is an abstraction of data access and storage.
- **Example**

```
public interface JobRepository extends
                                JpaRepository<Jobs, String> {

}
```

Restful Web Services Annotations

- **@RestController:**
  - ✓ The *@RestController* annotation was introduced in Spring 4.0 to simplify the creation of RESTful web services.
  - ✓ **It's a convenience annotation that combines *@Controller* and *@ResponseBody*** – which eliminates the need to annotate every request handling method of the controller class with the *@ResponseBody* annotation
- **@Path**: used to specify the relative path of class and methods. We can get the URI of a webservice by scanning the Path annotation value.
- **@GET**, **@PUT**, **@POST**, **@DELETE** and **@HEAD**: used to specify the HTTP request type for a method.
- **@Produces**, **@Consumes**: used to specify the request and response types.
- **@PathParam**: used to bind the method parameter to path value by parsing it.

Create Rest Controllers

- **@RestController** annotation marks the class as web controller, capable of handling the requests
- **Example**

```java
@RestController
@RequestMapping("/api/v1/user")
public class UserController {
    @Autowired
    private UserService userService;

    /**
     * Create user user.
     *
     * @param user the user
     * @return the user
     */
    @PostMapping("/add")
    public User create(@Valid @RequestBody User user) {
        return userService.save(user);
    }
```

Create Rest Controllers

- **@PostMapping** annotation marks the POST method
- **Example**

```
/**
* Create user user.
* @param user the user
* @return the user
*/
  @PostMapping("/add")
  public User create (@Valid @RequestBody User user) {
      return userService.save(user);
  }
```

Create Rest Controllers

- **@PutMapping** annotation marks the PUT method
- **Example**

```
@PutMapping("/update/{id}")
    public ResponseEntity<User> update(@PathVariable("id") long userId,
            @RequestBody User userDetail) {

        userService.findById(userId).orElseThrow(
                () -> new ResourceNotFoundException("User not found: "
                        + userId, "404"));

        final User updatedUser = userService.save(userDetail);

        return ResponseEntity.ok(updatedUser);

    }
```

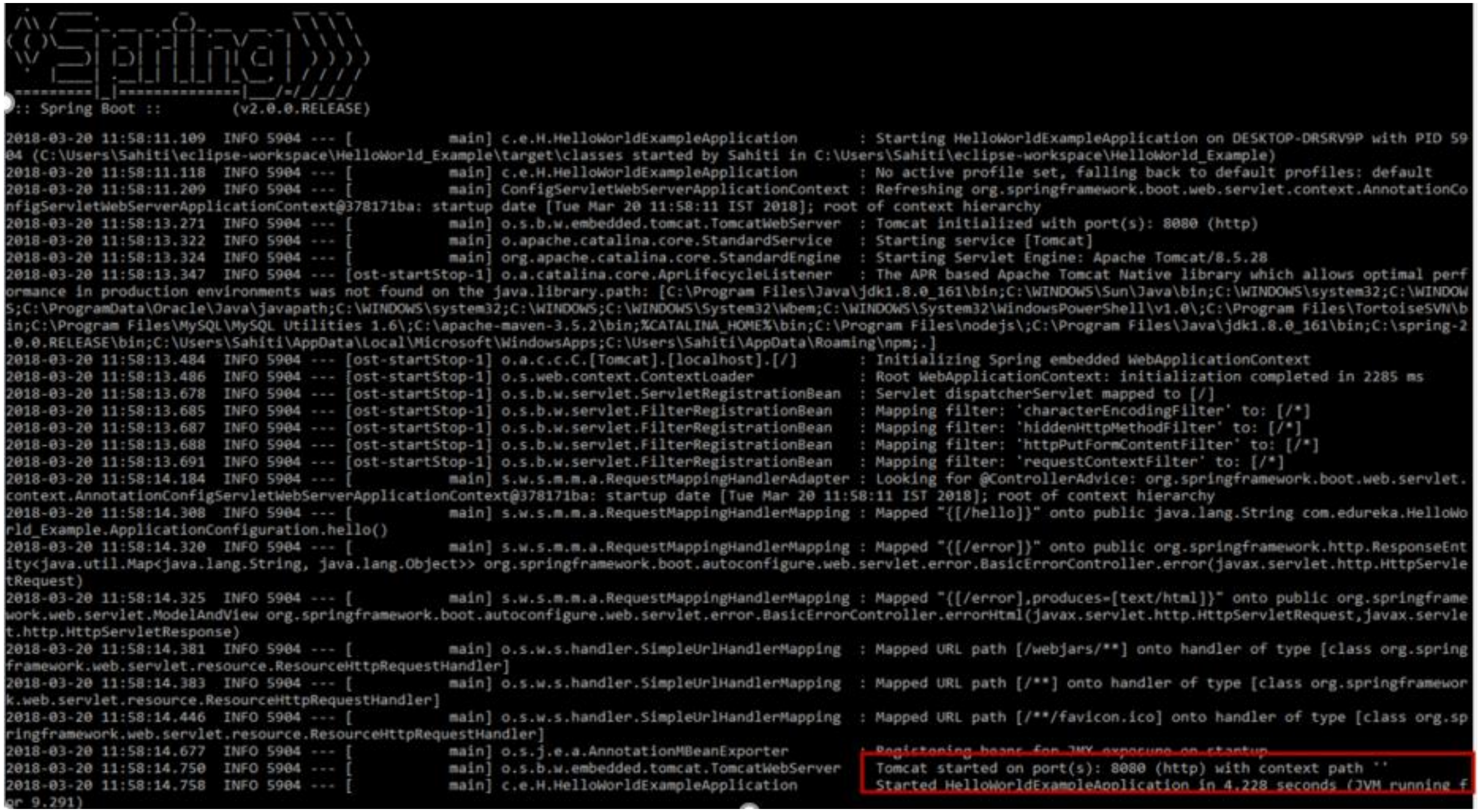
Create Rest Controllers

- **@DeleteMapping** annotation marks the DELETE method
- **Example**

```
@DeleteMapping("/delete/{id}")
public Map<String, Boolean> delete(@PathVariable(value = "id")
                                    long userId) throws Exception {
        User user = userService.findById(userId)
                .orElseThrow(() -> new ResourceNotFoundException(
                        "User not found: " + userId, "404"));

        userService.delete(user);

        Map<String, Boolean> response = new HashMap<>();
        response.put("deleted", Boolean.TRUE);

        return response;

    }
```

Build and Run the Project

- Right click on project -> Run As -> Run on Server



```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.0.0.RELEASE)

2018-03-20 11:58:11.109  INFO 5904 --- [           main] c.e.H.HelloWorldExampleApplication       : Starting HelloWorldExampleApplication on DESKTOP-DRSRV9P with PID 59
04 (C:\Users\Sahiti\eclipse-workspace\HelloWorld_Example\target\classes started by Sahiti in C:\Users\Sahiti\eclipse-workspace\HelloWorld_Example)
2018-03-20 11:58:11.118  INFO 5904 --- [           main] c.e.H.HelloWorldExampleApplication       : No active profile set, falling back to default profiles: default
2018-03-20 11:58:11.209  INFO 5904 --- [           main] ConfigServletWebServerApplicationContext : Refreshing org.springframework.boot.web.servlet.context.AnnotationCo
nfigServletWebServerApplicationContext@378171ba: startup date [Tue Mar 20 11:58:11 IST 2018]; root of context hierarchy
2018-03-20 11:58:13.271  INFO 5904 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
2018-03-20 11:58:13.322  INFO 5904 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2018-03-20 11:58:13.324  INFO 5904 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet Engine: Apache Tomcat/8.5.28
2018-03-20 11:58:13.347  INFO 5904 --- [ost-startStop-1] o.a.catalina.core.AprLifecycleListener   : The APR based Apache Tomcat Native library which allows optimal perf
ormance in production environments was not found on the java.library.path: [C:\Program Files\Java\jdk1.8.0_161\bin;C:\WINDOWS\Sun\Java\bin;C:\WINDOWS\system32;C:\WINDOW
S;C:\ProgramData\Oracle\Java\javapath;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\WINDOWS\System32\WindowsPowerShell\v1.0\;C:\Program Files\TortoiseSVN\b
in;C:\Program Files\MySQL\MySQL Utilities 1.6\;C:\apache-maven-3.5.2\bin;%CATALINA_HOME%\bin;C:\Program Files\nodejs\;C:\Program Files\Java\jdk1.8.0_161\bin;C:\spring-2
.0.0.RELEASE\bin;C:\Users\Sahiti\AppData\Local\Microsoft\WindowsApps;C:\Users\Sahiti\AppData\Roaming\npm;.]
2018-03-20 11:58:13.484  INFO 5904 --- [ost-startStop-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2018-03-20 11:58:13.486  INFO 5904 --- [ost-startStop-1] o.s.web.context.ContextLoader            : Root WebApplicationContext: initialization completed in 2285 ms
2018-03-20 11:58:13.678  INFO 5904 --- [ost-startStop-1] o.s.b.w.servlet.ServletRegistrationBean  : Servlet dispatcherServlet mapped to [/]
2018-03-20 11:58:13.685  INFO 5904 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter: 'characterEncodingFilter' to: [/*]
2018-03-20 11:58:13.687  INFO 5904 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter: 'hiddenHttpMethodFilter' to: [/*]
2018-03-20 11:58:13.688  INFO 5904 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter: 'httpPutFormContentFilter' to: [/*]
2018-03-20 11:58:13.691  INFO 5904 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean   : Mapping filter: 'requestContextFilter' to: [/*]
2018-03-20 11:58:14.184  INFO 5904 --- [           main] s.w.s.m.m.a.RequestMappingHandlerAdapter : Looking for @ControllerAdvice: org.springframework.boot.web.servlet.
context.AnnotationConfigServletWebServerApplicationContext@378171ba: startup date [Tue Mar 20 11:58:11 IST 2018]; root of context hierarchy
2018-03-20 11:58:14.308  INFO 5904 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/hello]}" onto public java.lang.String com.edureka.HelloWo
rld_Example.ApplicationConfiguration.hello()
2018-03-20 11:58:14.320  INFO 5904 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/error]}" onto public org.springframework.http.ResponseEnt
ity<java.util.Map<java.lang.String, java.lang.Object>> org.springframework.boot.autoconfigure.web.servlet.error.BasicErrorController.error(javax.servlet.http.HttpServle
tRequest)
2018-03-20 11:58:14.325  INFO 5904 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/error],produces=[text/html]}" onto public org.springframe
work.web.servlet.ModelAndView org.springframework.boot.autoconfigure.web.servlet.error.BasicErrorController.errorHtml(javax.servlet.http.HttpServletRequest,javax.servle
t.http.HttpServletResponse)
2018-03-20 11:58:14.381  INFO 5904 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path [/webjars/**] onto handler of type [class org.spring
framework.web.servlet.resource.ResourceHttpRequestHandler]
2018-03-20 11:58:14.383  INFO 5904 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path [/**] onto handler of type [class org.springframewor
k.web.servlet.resource.ResourceHttpRequestHandler]
2018-03-20 11:58:14.446  INFO 5904 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Mapped URL path [/**/favicon.ico] onto handler of type [class org.sp
ringframework.web.servlet.resource.ResourceHttpRequestHandler]
2018-03-20 11:58:14.677  INFO 5904 --- [           main] o.s.j.e.a.AnnotationMBeanExporter        : Registering beans for JMX exposure on startup
2018-03-20 11:58:14.750  INFO 5904 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
2018-03-20 11:58:14.758  INFO 5904 --- [           main] c.e.H.HelloWorldExampleApplication       : Started HelloWorldExampleApplication in 4.228 seconds (JVM running f
or 9.291)
```

Test

- **Use Postman App to get services:**

Section 3

@REQUESTBODY AND @RESPONSEBODY

@RequestBody

- **The *@RequestBody* annotation maps the *HttpRequest* body to a transfer or domain object, enabling automatic deserialization** of the inbound *HttpRequest* body onto a Java object.

```
@RestController
@RequestMapping("/api/v1/user")
public class UserRestController {

    @Autowired
    private UserService userService;

    /**
     * The method to insert a new user into User table in DB.
     */
    @PostMapping("/add")
    public User create(@Valid @RequestBody User user) {
        return userService.save(user);
    }
}
```

@RequestBody

- Spring **automatically deserializes** the JSON into a Java type, assuming an appropriate one is specified.
- By default, **the type we annotate with the *@RequestBody* annotation must correspond to the JSON sent from our client-side controller:**

```java
public class User {

    private long userId;

    private String username;

    private String password;

    // setter and getter methods
}
```

- Here, the object we use to represent the *HttpRequest* body maps to our *User* object.

@ResponseBody

- The *@ResponseBody* annotation tells a controller that the object returned is automatically serialized into JSON and passed back into the *HttpResponse* object.
- Suppose we have a custom ***Response*** object:

```
public class ResponseTransfer {
    private String text;

    // standard getters/setters
}
```

@ResponseBody

- Next, the associated controller can be implemented:

```
@RestController
@RequestMapping("/api/v1/user")
public class UserRestController {

    @Autowired
    ExampleService exampleService;

    @PostMapping("/response")
    @ResponseBody
    public ResponseTransfer postResponseController(
            @RequestBody User user) {
        return new ResponseTransfer("Thanks For Posting!!!");
    }
}
```

- In the developer console of our browser or using a tool like Postman, we can see the following response:

```
{"text":"Thanks For Posting!!!"}
```

Setting the Content Type

- When we use the *@ResponseBody* annotation, we're still able to explicitly set the content type that our method returns.
- **We can use the *@RequestMapping*'s *produces* attribute.** Note that annotations like *@PostMapping*, *@GetMapping*, etc. define aliases for that parameter.

```
@PostMapping(value = "/content",
                produces = MediaType.APPLICATION_JSON_VALUE)
@ResponseBody
public ResponseTransfer postResponseJsonContent(
                        @RequestBody LoginForm loginForm) {
    return new ResponseTransfer("JSON Content!");
}
```

- We used the *MediaType.APPLICATION_JSON_VALUE* constant. Alternatively, we can use *application/json* directly:

```
produces = { MediaType.APPLICATION_JSON_VALUE, MediaType.APPLICATION_XML_VALUE }
produces = { "application/json" , "application/xml" }
```

Section 4

RESPONSEENTITY
TO MANIPULATE THE HTTP RESPONSE

ResponseEntity

- *ResponseEntity* **represents the whole HTTP response: status code, headers, and body**. As a result, we can use it to fully configure the HTTP response.
- *ResponseEntity* is a generic type. Consequently, we can use any type as the response body:

```
@GetMapping("/hello")
public ResponseEntity<String> hello() {
        return new ResponseEntity<>("Hello World!",
                                        HttpStatus.OK);
}
```

ResponseEntity

- Since we specify the response status programmatically, we can return with different status codes for different scenarios:

```
@GetMapping("/age")
public ResponseEntity<String> age(
      @RequestParam("yearOfBirth") int yearOfBirth) {

    if (isInFuture(yearOfBirth)) {
        return new ResponseEntity<>(
                "Year of birth cannot be in the future",
                HttpStatus.BAD_REQUEST);
    }

    return new ResponseEntity<>(
        "Your age is " + calculateAge(yearOfBirth),
        HttpStatus.OK);
    }
```

ResponseEntity

- We can set HTTP headers:

```
@GetMapping("/customHeader")
public ResponseEntity<String> customHeader() {

    HttpHeaders headers = new HttpHeaders();
    headers.add("Custom-Header", "foo");

    return new ResponseEntity<>(
          "Custom header set", headers, HttpStatus.OK);
}
```

ResponseEntity

- *ResponseEntity* **provides two nested builder interfaces**: *HeadersBuilder* and its subinterface, *BodyBuilder*.
- Therefore, we can access their capabilities through the static methods of *ResponseEntity*.

```
@GetMapping("/hello")
public ResponseEntity<String> hello() {
        return ResponseEntity.ok("Hello World!");
}
```

- For the most popular HTTP status codes we get static methods:

BodyBuilder **accepted**();
BodyBuilder **badRequest**();
BodyBuilder **created**(java.net.URI location);
HeadersBuilder<?> **noContent**();
HeadersBuilder<?> **notFound**();
BodyBuilder **ok**();

ResponseEntity

- We can use the *BodyBuilder status(HttpStatus status)* and the *BodyBuilder status(int status)* methods to set any HTTP status.
- With *ResponseEntity<T> BodyBuilder.body(T body)* we can set the HTTP response body:

```java
@GetMapping("/age")
ResponseEntity<String> age(@RequestParam("yearOfBirth") int yearOfBirth) {
        if (isInFuture(yearOfBirth)) {
            return ResponseEntity.badRequest()
                .body("Year of birth cannot be in the future");
        }

        return ResponseEntity.status(HttpStatus.OK)
            .body("Your age is " + calculateAge(yearOfBirth));
}
```

Thank you