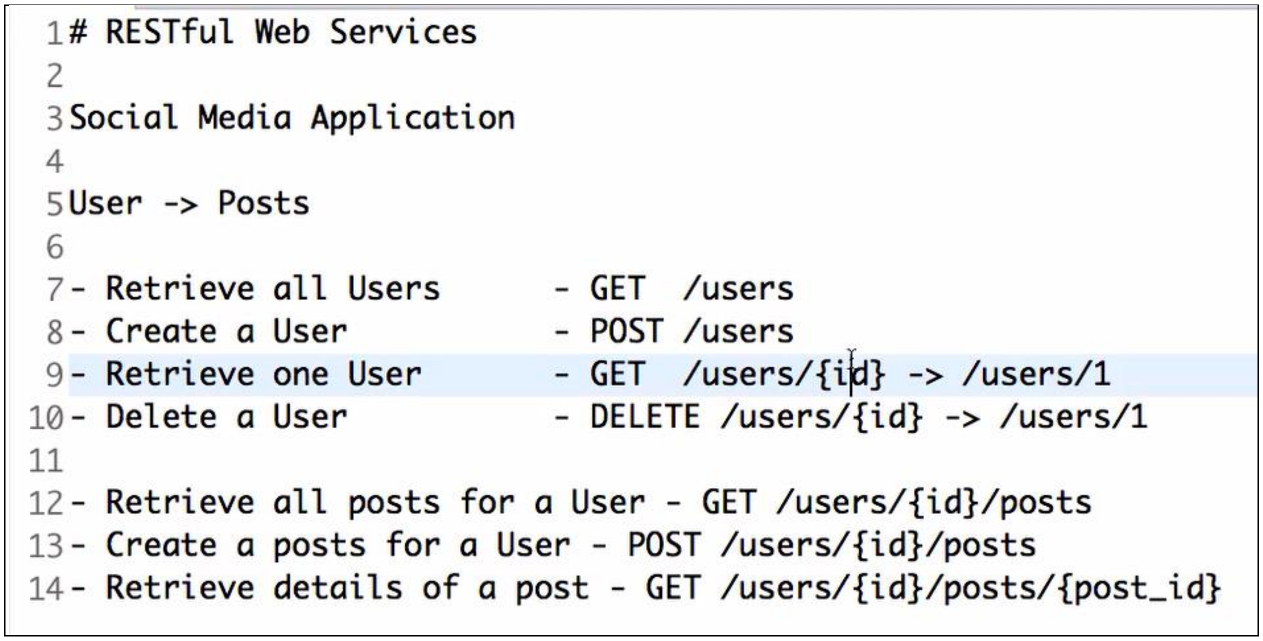
RESTful Web Services with Spring Boot

2. Step 01 - Initializing a RESTful Services Project with Spring Boot

Dependency added: Web, JPA, DevTools, H2

4. Step 02 - Understanding the RESTful Services we would create in this course



```
# RESTful Web Services

Social Media Application

User -> Posts

- Retrieve all Users      - GET  /users
- Create a User           - POST /users
- Retrieve one User       - GET  /users/{id} -> /users/1
- Delete a User           - DELETE /users/{id} -> /users/1

- Retrieve all posts for a User - GET /users/{id}/posts
- Create a posts for a User - POST /users/{id}/posts
- Retrieve details of a post - GET /users/{id}/posts/{post_id}
```

5. Step 03 - Creating a Hello World Service

```
// To tell spring boot that this will be listening to the rest requests
@RestController
public class HelloWorldController {

	// @GetMapping(path = "/hello-world")
	@RequestMapping(method = RequestMethod.GET, path = "/hello-world")
	public String helloWorld() {
		return "Hello World";
	}
}
```

---**Step 04 - Enhancing the Hello World Service to return a Bean**

HelloWorldBean.java

```java
package com.personal.kunj.springbootrestfulservice;

public class HelloWorldBean {

	private String message;

	public HelloWorldBean(String message) {
		this.message = message;
	}

	public void setMessage(String message) {
		this.message = message;
	}

	public String getMessage() {
		return message;
	}

	@Override
	public String toString() {
		return "HelloWorldBean [message=" + message + "]";
	}

}
```

Note: // getMessage() is required otherwise we will get the below error.

```
{
  "timestamp": "2018-08-25T09:25:23.693+0000",
  "status": 500,
  "error": "Internal Server Error",
  "message": "No converter found for return value of type: class
com.personal.kunj.springbootrestfulservice.HelloWorldBean",
  "path": "/hello-world-bean"
}
```

HelloWorldController.java

```java
package com.personal.kunj.springbootrestfulservice;

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

@RestController
public class HelloWorldController {
	// @GetMapping(path = "/hello-world")
	@RequestMapping(method = RequestMethod.GET, path = "/hello-world")
	public String helloWorld() {
		return "Hello World";
	}

	@RequestMapping(method = RequestMethod.GET, path = "/hello-world-bean")
	public HelloWorldBean helloWorldBean() {
		return new HelloWorldBean("Hello World Bean");
	}
}
```

Step 05 - Quick Review of Spring Boot Auto Configuration and Dispatcher Servlet

spring-boot-starter-web has dependency on spring-mvc framework therefore we get org.springframework.web.servlet.DispatcherServlet class in our classpath.

Object to JSON conversion is being done by Spring boot.

DispatcherServlet is handling all the requests. Anything after root [localhost:8080/…]

Step 06 - Enhancing the Hello World Service with a Path Variable

```
@GetMapping(path = "/hello-world/path-var/{name}")
	public HelloWorldBean helloWorldWithPath(@PathVariable("name") String myName) {
		return new HelloWorldBean(String.format("Hello World, %s", myName));
	}
```

----

9. Step 07 - Creating User Bean and User Service

User.java

```
package com.personal.kunj.springbootrestfulservice.user;

import java.util.Date;

public class User {

	private Integer id;
	private String name;

	private Date birthDate;

	public User(Integer id, String name, Date birthDate) {
		super();
		this.id = id;
		this.name = name;
		this.birthDate = birthDate;
	}

	public Integer getId() {
		return id;
	}

	public void setId(Integer id) {
		this.id = id;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public Date getBirthDate() {
		return birthDate;
	}

	public void setBirthDate(Date birthDate) {
		this.birthDate = birthDate;
	}

	@Override
	public String toString() {
		return String.format("User [id=%s, name=%s, birthDate=%s]", id, name,
birthDate);
	}

}
```

UserDaoService.java

```java
package com.personal.kunj.springbootrestfulservice.user;

import java.util.ArrayList;
import java.util.Date;
import java.util.List;

import org.springframework.stereotype.Component;

@Component
public class UserDaoService {

	private static List<User> users = new ArrayList<>();

	private static int usersCount = 3;

	static {
		users.add(new User(1, "Adam", new Date()));
		users.add(new User(2, "Eve", new Date()));
		users.add(new User(3, "Jack", new Date()));
	}

	public List<User> findAll() {
		return users;
	}

	public User save(User user) {
		if (user.getId() == null) {
			user.setId(++usersCount);
		}
		users.add(user);
		return user;
	}

	public User findOne(int id) {
		for (User user : users) {
			if (user.getId() == id) {
				return user;
			}
		}
		return null;
	}

}
```

UserResource.java

```java
package com.personal.kunj.springbootrestfulservice.user;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class UserResource {

	@Autowired
	private UserDaoService service;

	@GetMapping("/users")
	public List<User> retrieveAllUsers() {
		return service.findAll();
	}

	@GetMapping("/users/{id}")
	public User retrieveUser(@PathVariable int id) {
		return service.findOne(id);
	}
}
```

-- Step 09 - Implementing POST Method to create User Resource

User.java

```java
package com.personal.kunj.springbootrestfulservice.user;

import java.util.Date;

public class User {

	private Integer id;

	private String name;

	private Date birthDate;
	// Must have for REST to convert json data to a java object
	protected User() {

	}

	public User(Integer id, String name, Date birthDate) {
		super();
		this.id = id;
		this.name = name;
		this.birthDate = birthDate;
	}

	public Integer getId() {
		return id;
	}

	public void setId(Integer id) {
		this.id = id;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public Date getBirthDate() {
		return birthDate;
	}

	public void setBirthDate(Date birthDate) {
		this.birthDate = birthDate;
	}

	@Override
	public String toString() {
		return String.format("User [id=%s, name=%s, birthDate=%s]", id, name,
birthDate);
	}

}
```

UserResource.java

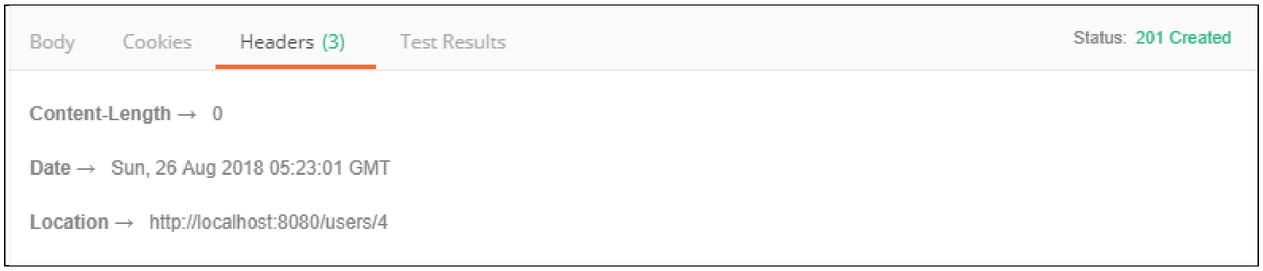
```
package com.personal.kunj.springbootrestfulservice.user;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class UserResource {

	@Autowired
	private UserDaoService service;

	@GetMapping("/users")
	public List<User> retrieveAllUsers() {
		return service.findAll();
	}

	@GetMapping("/users/{id}")
	public User retrieveUser(@PathVariable int id) {
		return service.findOne(id);
	}

	// Input --> Details of the new user
	// output --> CREATED (status) and URI of the created resource
	@PostMapping(path = "/users")
	public void createUser(@RequestBody User user) {
		User newUser = service.save(user);
	}

}
```

Step 10 - Enhancing POST Method to return correct HTTP Status Code and Location

UserResource.java

```
package com.personal.kunj.springbootrestfulservice.user;
```

```
import java.net.URI;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.servlet.support.ServletUriComponentsBuilder;

@RestController
public class UserResource {

	@Autowired
	private UserDaoService service;

	@GetMapping("/users")
	public List<User> retrieveAllUsers() {
		return service.findAll();
	}

	@GetMapping("/users/{id}")
	public User retrieveUser(@PathVariable int id) {
		return service.findOne(id);
	}

	// Input --> Details of the new user
	// output --> CREATED (status) and URI of the created resource (Location header
	// will have the uri of the new resource)
	@PostMapping(path = "/users")
	public ResponseEntity<Object> createUser(@RequestBody User user) {
		User newUser = service.save(user);
		URI location =
ServletUriComponentsBuilder.fromCurrentRequest().path("/{id}").buildAndExpand(newUser.get
Id())
					.toUri();
		return ResponseEntity.created(location).build();
	}

}
```



```
Content-Length → 0
Date → Sun, 26 Aug 2018 05:23:01 GMT
Location → http://localhost:8080/users/4
```

Step 11 - Implementing Exception Handling - 404 Resource Not Found

UserResource.java

```
@GetMapping("/users/{id}")
	public User retrieveUser(@PathVariable int id) {
		User user = service.findOne(id);
		if (user == null) {
			throw new UserNotFoundException("id-" + id);
		}
		return user;
	}
```

UserNotFoundException.java

```
package com.personal.kunj.springbootrestfulservice.user;

import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ResponseStatus;

@ResponseStatus(HttpStatus.NOT_FOUND)
public class UserNotFoundException extends RuntimeException {

	private static final long serialVersionUID = 1L;

	public UserNotFoundException(String message) {
		super(message);
	}
}
```

Step 12 - Implementing Generic Exception Handling for all Resources

ResponseEntityExceptionHandler → A convenient base class for @ControllerAdvice classes that wish to provide centralized exception handling across all @RequestMapping methods through @ExceptionHandler methods.

ExceptionResponse.java

```
package com.personal.kunj.springbootrestfulservice.exception;

import java.util.Date;

public class ExceptionResponse {

	private Date timestamp;
	private String message;
	private String details;

	public ExceptionResponse(Date timestamp, String message, String details) {
		super();
		this.timestamp = timestamp;
		this.message = message;
		this.details = details;
	}

	public Date getTimestamp() {
		return timestamp;
	}

	public String getMessage() {
		return message;
	}

	public String getDetails() {
		return details;
	}

}
```

CustomResponseEntityExceptionHandler.java

```
package com.personal.kunj.springbootrestfulservice.exception;

import java.util.Date;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.context.request.WebRequest;
import
org.springframework.web.servlet.mvc.method.annotation.ResponseEntityExceptionHandler;

import com.personal.kunj.springbootrestfulservice.user.UserNotFoundException;

// To share this exception across controllers
@ControllerAdvice
// As it is providing response
@RestController
public class CustomResponseEntityExceptionHandler extends ResponseEntityExceptionHandler
{

	@ExceptionHandler(Exception.class) // To handle all the exceptions
	public final ResponseEntity<Object> handleAllExceptions(Exception ex, WebRequest
request) {
		// We want to return our exception response back
		ExceptionResponse exceptionResponse = new ExceptionResponse(new Date(),
ex.getMessage(),
				request.getDescription(false));
		return new ResponseEntity<Object>(exceptionResponse,
HttpStatus.INTERNAL_SERVER_ERROR);
	}

	@ExceptionHandler(UserNotFoundException.class) // To handle all the exceptions
	public final ResponseEntity<Object>
handleUserNotFoundException(UserNotFoundException ex, WebRequest request) {
		// We want to return our exception response back
		ExceptionResponse exceptionResponse = new ExceptionResponse(new Date(),
ex.getMessage(),
				request.getDescription(false));
		return new ResponseEntity<Object>(exceptionResponse,
HttpStatus.NOT_FOUND);
	}
}
```

Step 13 - Exercise User Post Resource and Exception Handling

Bol Bachchan

Step 14 - Implementing DELETE Method to delete a User Resource

UserResource.java

```
@DeleteMapping("/users/{id}")
	public void deleteUser(@PathVariable int id) {
		User user = service.deleteById(id);

		if (user == null)
			throw new UserNotFoundException("id-" + id);
	}
```

UserDaoService.java

```
public User deleteById(int id) {
		Iterator<User> iterator = users.iterator();
		while (iterator.hasNext()) {
			User user = iterator.next();
			if (user.getId() == id) {
				iterator.remove();
				return user;
			}
		}
		return null;
	}
```

Step 15 - Implementing Validations for RESTful Services

CustomResponseEntityExceptionHandler.java

```
@Override
	protected ResponseEntity<Object>
handleMethodArgumentNotValid(MethodArgumentNotValidException ex,
			HttpHeaders headers, HttpStatus status, WebRequest request) {
		ExceptionResponse exceptionResponse = new ExceptionResponse(new Date(),
"Validation Failed",
				ex.getBindingResult().toString());
		return new ResponseEntity<Object>(exceptionResponse,
HttpStatus.BAD_REQUEST);
	}
```

User.java

```
private Integer id;
	@Size(min = 2, message = "Name should have at least 2 characters")
	private String name;
	@Past
	private Date birthDate;
```

UserResource.java

```
@PostMapping(path = "/users")
	public ResponseEntity<Object> createUser(@Valid @RequestBody User user) {
```

Step 16 - Implementing HATEOAS for RESTful Services

pom.xml

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

UserResource.java

```
@GetMapping("/users/{id}")
	public Resource<User> retrieveUser(@PathVariable int id) {
		User user = service.findOne(id);
		if (user == null) {
			throw new UserNotFoundException("id-" + id);
		}
		// HATEOAS
		// Creating resource around the user
		Resource<User> resource = new Resource<User>(user);
		/*
		 * Now add links to the resource. But before this, get the links for
retrieveAllUsers(). We are getting the links for retrieveAllUsers() bcz we do not want to hard
code the path to "/users". ControllerLinkBuilder class helps us in creating links from methods.
		 */
		ControllerLinkBuilder linkTo = ControllerLinkBuilder.linkTo(this.getClass(),
retrieveAllUsers());
		resource.add(linkTo.withRel("all-users"));
		return resource;
	}
```

O/P:

```
	{
   "id": 1,
   "name": "Adam",
   "birthDate": "2018-08-26T11:54:35.334+0000",
  "_links": {
    "all-users": {
      "href": "http://localhost:8080"
    }
  }
}
```

Step 17 - Overview of Advanced RESTful Service Features

--

Step 18 - Internationalization for RESTful Services

Internationalization(i18n) → Customizing your services for your different people around the world

Internationalization :

## Configuration to be done:

- ➔ LocaleResolver
- ➔ → Default Locale – locale.US (if a user does not ask for the customization, locale.US will be shown as the default locale)
- ➔ ResourceBundleMessageSource (We will store here the List of properties which will be internationalized). ResourceBundleMessageSource is a spring concept for handling properties.

## Usage

- ➔ Autowire MessageSource
- ➔ @RequestHeader(value="Accept-Language", required=false) Locale locale
- ➔ messageSource.getMessage("helloWorld.message", null,locale)

Messages.properties

```
good.morning.message=Good Morning
```

Messages_fr.properties

```
good.morning.message=Bonjour
```

HelloWorldController.java

```
@RequestMapping(method = RequestMethod.GET, path = "/hello-world-internationalized")
	public String helloWorldnternationalized(@RequestHeader(name = "Accept-Language",
required = false) Locale locale) {
		return messageSource.getMessage("good.morning.message", null, locale);
	}
```

SpringbootRestfulServiceApplication.java

```
package com.personal.kunj.springbootrestfulservice;

import java.util.Locale;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.context.support.ResourceBundleMessageSource;
import org.springframework.web.servlet.LocaleResolver;
import org.springframework.web.servlet.i18n.AcceptHeaderLocaleResolver;

@SpringBootApplication
public class SpringbootRestfulServiceApplication {

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

	@Bean
	public LocaleResolver localeResolver() {
		SessionLocaleResolver localeResolver = new SessionLocaleResolver();
		localeResolver.setDefaultLocale(Locale.US);
		return localeResolver;
	}

	@Bean
	public ResourceBundleMessageSource messageSource() {
		ResourceBundleMessageSource resourceBundleMessageSource = new
ResourceBundleMessageSource();
		resourceBundleMessageSource.setBasename("messages");
		return resourceBundleMessageSource;
	}
}
```

Step 18 Part 2 - Internationalization for RESTful Services

An Alternative to the above approach:

Application.properties

```
spring.messages.basename=messages
```

SpringbootRestfulServiceApplication.java

```
package com.personal.kunj.springbootrestfulservice;
```

```
import java.util.Locale;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.web.servlet.LocaleResolver;
import org.springframework.web.servlet.i18n.AcceptHeaderLocaleResolver;

@SpringBootApplication
public class SpringbootRestfulServiceApplication {

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

	@Bean
	public LocaleResolver localeResolver() {
		// After using AcceptHeaderLocaleResolver, we will not need to configure locale
		// as request parameter/header in every controller method
		AcceptHeaderLocaleResolver localeResolver = new
AcceptHeaderLocaleResolver();
		localeResolver.setDefaultLocale(Locale.US);
		return localeResolver;
	}

	/*
	 * You can replace this code with property "spring.messages.basename=messages"
	 * in application.properties
	 */
	/*
	 * @Bean public ResourceBundleMessageSource messageSource() {
	 * ResourceBundleMessageSource resourceBundleMessageSource = new
	 * ResourceBundleMessageSource();
	 * resourceBundleMessageSource.setBasename("messages"); return
	 * resourceBundleMessageSource; }
	 */
}
```

HelloWorldController.java

```
@RequestMapping(method = RequestMethod.GET, path = "/hello-
world-internationalized")
	public String helloWorldnternationalized() {
		return 
messageSource.getMessage("good.morning.message", null, 
LocaleContextHolder.getLocale());
	}
```

Step 19 - Content Negotiation - Implementing Support for XML

```xml
		<dependency>
	<groupId>com.fasterxml.jackson.dataformat</groupId>
		<artifactId>jackson-dataformat-xml</artifactId>
		</dependency>
```

Step 20 - Configuring Auto Generation of Swagger Documentation

Swagger is documentation format for rest web services.

Add dependencies:

```xml
<dependency>
			<groupId>io.springfox</groupId>
			<artifactId>springfox-swagger2</artifactId>
			<version>2.7.0</version>
		</dependency>
		<dependency>
			<groupId>io.springfox</groupId>
			<artifactId>springfox-swagger2</artifactId>
			<version>2.7.0</version>
		</dependency>
```

Configure Swagger:

SwaggerConfig.java

```java
package com.personal.kunj.springbootrestfulservice;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import springfox.documentation.spi.DocumentationType;
import springfox.documentation.spring.web.plugins.Docket;
import 
springfox.documentation.swagger2.annotations.EnableSwagger2;

// configuration
@Configuration
```

```
// Enable Swagger
@EnableSwagger2
public class SwaggerConfig {
      // Bean - Docket
      @Bean
      public Docket api() {
            return new Docket(DocumentationType.SWAGGER_2);
            /*
             * return new
             *
Docket(DocumentationType.SWAGGER_2).select().apis(RequestHandler
Selectors.any
             * ()) .paths(PathSelectors.any()).build();
             */
      }

}
```

http://localhost:8080/v2/api-docs → documentation URL
http://localhost:8080/swagger-ui.html

Step 21 - Introduction to Swagger Documentation Format

Step 22 - Enhancing Swagger Documentation with Custom Annotations

SwaggerConfig.java

```
package com.personal.kunj.springbootrestfulservice;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashSet;
import java.util.Set;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import springfox.documentation.service.ApiInfo;
import springfox.documentation.service.Contact;
import springfox.documentation.service.VendorExtension;
import springfox.documentation.spi.DocumentationType;
import springfox.documentation.spring.web.plugins.Docket;
import
springfox.documentation.swagger2.annotations.EnableSwagger2;
```

```
// configuration
@Configuration
// Enable Swagger
@EnableSwagger2
public class SwaggerConfig {

     public static final Contact DEFAULT_CONTACT = new
Contact("Kunj Biahri", "www.kunjbihari.com",
               "kunj.bihari@abc.com");
     public static final ApiInfo DEFAULT_API_INFO = new
ApiInfo("Kunj's API Title", "Kunj's API Description", "1.0",
               "urn:tos", DEFAULT_CONTACT, "Apache 2.0",
"http://www.apache.org/licenses/LICENSE-2.0",
               new ArrayList<VendorExtension>());
     private static final Set<String>
DEFAULT_PRODUCES_AND_CONSUMES = new HashSet<String>(
               Arrays.asList("application/json",
"application/xml"));

     // Bean - Docket
     @Bean
     public Docket api() {
          return new
Docket(DocumentationType.SWAGGER_2).apiInfo(DEFAULT_API_INFO).pr
oduces(DEFAULT_PRODUCES_AND_CONSUMES)
                    .consumes(DEFAULT_PRODUCES_AND_CONSUMES);

          /*
           * return new
           *
Docket(DocumentationType.SWAGGER_2).select().apis(RequestHandler
Selectors.any
           * ()) .paths(PathSelectors.any()).build();
           */
     }
}
```

User.java

```
package com.personal.kunj.springbootrestfulservice.user;

import java.util.Date;
import javax.validation.constraints.Past;
import javax.validation.constraints.Size;
import io.swagger.annotations.ApiModel;
import io.swagger.annotations.ApiModelProperty;
```

```
// Providing more info about the user in swagger documentation
@ApiModel(description = "All details about the user")
public class User {

	private Integer id;
	@Size(min = 2, message = "Name should have at least 2 characters")
	// To show the notes in the swagger documentation
	@ApiModelProperty(notes = "Name should have at least 2 characters")
	private String name;

	@Past
	@ApiModelProperty(notes = "Bithdate should be in the past")
	private Date birthDate;

	// Must have for REST to convert json data to a java object
	protected User() {

	}

	public User(Integer id, String name, Date birthDate) {
		super();
		this.id = id;
		this.name = name;
		this.birthDate = birthDate;
	}

	public Integer getId() {
		return id;
	}

	public void setId(Integer id) {
		this.id = id;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public Date getBirthDate() {
		return birthDate;
	}

	public void setBirthDate(Date birthDate) {
		this.birthDate = birthDate;
	}

	@Override
	public String toString() {
		return String.format("User [id=%s, name=%s, birthDate=%s]", id, name,
birthDate);
	}

}
```

Step 23 - Monitoring APIs with Spring Boot Actuator

pom.xml

```
<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-
actuator</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.data</groupId>
			<artifactId>spring-data-rest-hal-
browser</artifactId>
		</dependency>
```

Application.properties

```
## Enabling exposure over HTTP of all the management end points
(actuator specific config)
management.endpoints.web.exposure.include=*

Actuator URL: http://localhost:8080/actuator
HAL Browser URl: localhost:8080/browser/index.html
```

Step 24 - Implementing Static Filtering for RESTful Service

If we want to ignore fields based on scenarios, we have to go foe dynamic filtering.

SomeBean.java

```
package com.personal.kunj.springbootrestfulservice.filtering;

import com.fasterxml.jackson.annotation.JsonIgnore;
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;

// Other way to ignore fields in the response
@JsonIgnoreProperties(value = { "field1" })
public class SomeBean {

	private String field1;

	private String field2;

	// Let's say this field is secure and we do not want to
pass this field in the
	// response
	@JsonIgnore
	private String field3;

	public SomeBean(String field1, String field2, String
field3) {
		super();
		this.field1 = field1;
		this.field2 = field2;
		this.field3 = field3;
	}

	public String getField1() {
		return field1;
	}

	public void setField1(String field1) {
		this.field1 = field1;
	}

	public String getField2() {
		return field2;
	}

	public void setField2(String field2) {
		this.field2 = field2;
	}

	public String getField3() {
		return field3;
	}
	public void setField3(String field3) {
		this.field3 = field3;
	}

}
```

Step 25 - Implementing Dynamic Filtering for RESTful Service

In dynamic filtering we have to start the filtering right there where we are retrieving the values (Unlike static filtering where we are doing the filtering at the bean).

SomeBean.java

```
@JsonFilter("SomeBeanFilter")
public class SomeBean {

// fields and methods
}
```

FilteringController.java

```
package com.personal.kunj.springbootrestfulservice.filtering;

import
org.springframework.http.converter.json.MappingJacksonValue;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

import com.fasterxml.jackson.databind.ser.FilterProvider;
import
com.fasterxml.jackson.databind.ser.impl.SimpleBeanPropertyFilter
;
import
com.fasterxml.jackson.databind.ser.impl.SimpleFilterProvider;

@RestController
public class FilteringController {

	// We only want to send field1,field2 in the response
	@GetMapping("/filtering")
	public MappingJacksonValue retrieveSomeBean() {
		SomeBean someBean = new SomeBean("value1", "value2",
"value3");
		// Filter out all the fields in the response except
field1 and field2
```

```
		SimpleBeanPropertyFilter filter =
SimpleBeanPropertyFilter.filterOutAllExcept("field1", "field2");
		FilterProvider filters = new
SimpleFilterProvider().addFilter("SomeBeanFilter", filter);
		MappingJacksonValue mapping = new
MappingJacksonValue(someBean);
		mapping.setFilters(filters);
		return mapping;
	}
}
```

Step 26 - Versioning RESTful Services - Basic Approach with URIs

Create a package versioning.
Create 2 classes PersonV1 and PersonV2.
PersonV1 → Wants a name to be returned as a String (one name).
PersonV2 → Wants a name to be displayed as first and last name.

How do we solve this problem? For the same API we need to have 2 versions, one giving the combined name back and other giving the name as first and last name. How do we create 2 versions of the same service?

```
package com.personal.kunj.springbootrestfulservice.versioning;

public class PersonV1 {

	private String name;

	public PersonV1() {
		super();
	}

	public PersonV1(String name) {
		super();
		this.name = name;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}
}
```

```java
package com.personal.kunj.springbootrestfulservice.versioning;

public class PersonV2 {

	private Name name;

	public PersonV2() {
		super();
	}

	public PersonV2(Name name) {
		super();
		this.name = name;
	}

	public Name getName() {
		return name;
	}

	public void setName(Name name) {
		this.name = name;
	}

}
```

```java
package com.personal.kunj.springbootrestfulservice.versioning;

public class Name {
	private String firstName;
	private String lastName;

	public Name() {
	}

	public Name(String firstName, String lastName) {
		super();
		this.firstName = firstName;
		this.lastName = lastName;
	}

	public String getFirstName() {
		return firstName;
	}

	public void setFirstName(String firstName) {
		this.firstName = firstName;
	}

	public String getLastName() {
		return lastName;
	}

	public void setLastName(String lastName) {
		this.lastName = lastName;
	}

}
```

```
package com.personal.kunj.springbootrestfulservice.versioning;

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

@RestController
public class PersonVersioningController {

	@GetMapping("v1/person")
	public PersonV1 personV1() {
		return new PersonV1("Bob Charlie");
	}

	@GetMapping("v2/person")
	public PersonV2 personV2() {
		return new PersonV2(new Name("Bob", "Charlie"));
	}

}
```

----

30. Step 27 - Versioning RESTful Services - Header and Content Negotiation Approach

(1) Doing versioning using a request parameter

```
@GetMapping(value = "/person/param", params = "version=1")
	public PersonV1 paramV1() {
		return new PersonV1("Bob Charlie");
	}

	@GetMapping(value = "/person/param", params = "version=2")
	public PersonV2 paramV2() {
		return new PersonV2(new Name("Bob", "Charlie"));
```

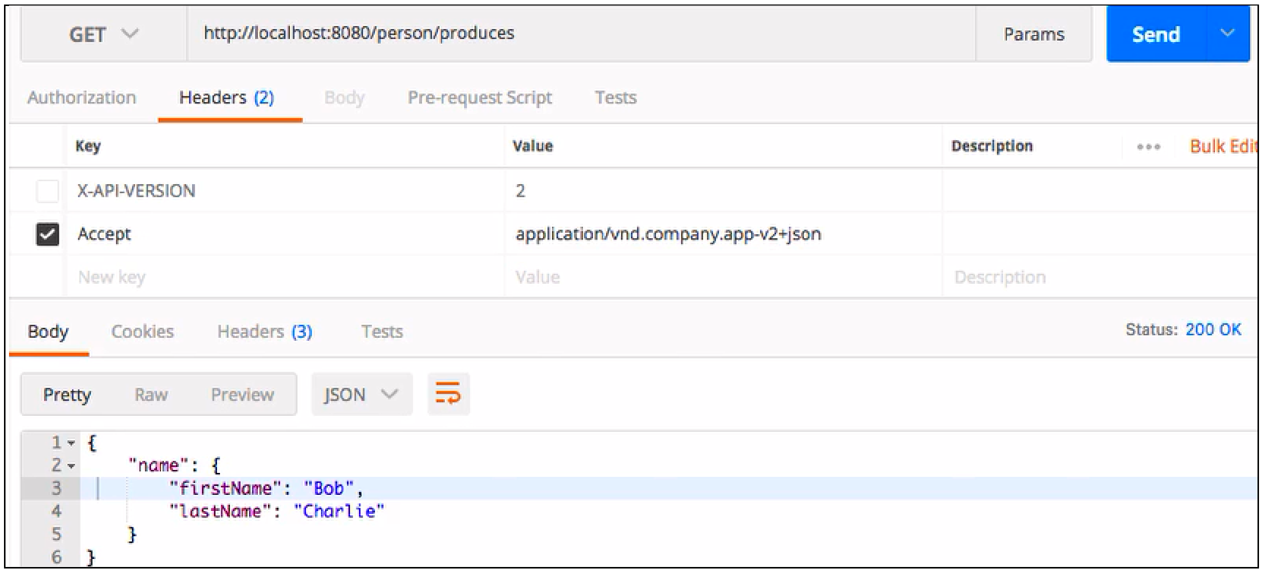
```
      }
URL:    http://localhost:8080/person/param?version=1
```

(2) Doing versioning using a request header

```
      @GetMapping(value = "/person/header", headers = "X-API-
VERSION=1")
      public PersonV1 headerV1() {
            return new PersonV1("Bob Charlie");
      }
      @GetMapping(value = "/person/header", headers = "X-API-
VERSION=2")
      public PersonV2 headerV2() {
            return new PersonV2(new Name("Bob", "Charlie"));
      }
```

(3) Content negotiation or Accept versioning

```
@GetMapping(value = "/person/produces", produces = "application/vnd.company.app-
v1+json")
        public PersonV1 producesV1() {
                return new PersonV1("Bob Charlie");
        }
        @GetMapping(value = "/person/produces", produces = "application/vnd.company.app-
v2+json")
        public PersonV2 producesV2() {
                return new PersonV2(new Name("Bob", "Charlie"));
        }
```

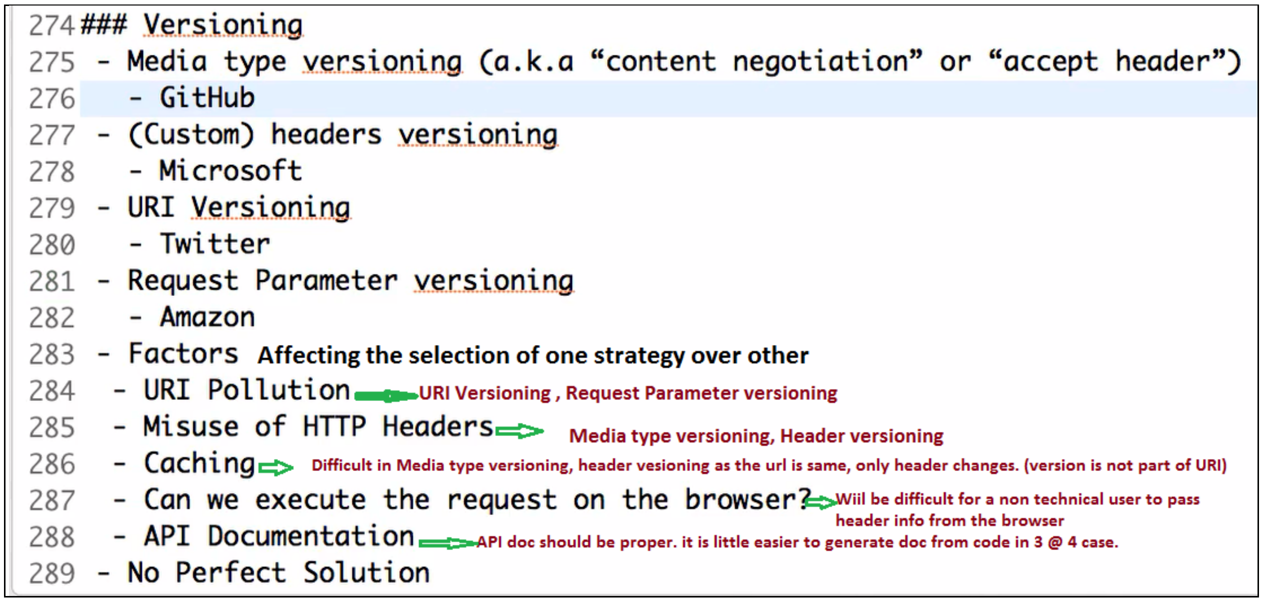


GET http://localhost:8080/person/produces

	Key	Value	Description
☐	X-API-VERSION	2	
✓	Accept	application/vnd.company.app-v2+json	
	New key	Value	Description

Status: 200 OK

```
{
    "name": {
        "firstName": "Bob",
        "lastName": "Charlie"
    }
}
```



```
### Versioning
 - Media type versioning (a.k.a "content negotiation" or "accept header")
   - GitHub
 - (Custom) headers versioning
   - Microsoft
 - URI Versioning
   - Twitter
 - Request Parameter versioning
   - Amazon
 - Factors  Affecting the selection of one strategy over other
  - URI Pollution ⟹ URI Versioning , Request Parameter versioning
  - Misuse of HTTP Headers ⟹ Media type versioning, Header versioning
  - Caching ⟹ Difficult in Media type versioning, header vesioning as the url is same, only header changes. (version is not part of URI)
  - Can we execute the request on the browser? ⟹ Wiil be difficult for a non technical user to pass header info from the browser
  - API Documentation ⟹ API doc should be proper. it is little easier to generate doc from code in 3 @ 4 case.
 - No Perfect Solution
```

Step 28 - Implementing Basic Authentication with Spring Security

pom.xml

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

```
Default username : user
Default password : Take from console
```

Note: If you do not like the default credentials, you can configure them as below in application.properties.

```
spring.security.user.name=username
spring.security.user.password=password
```

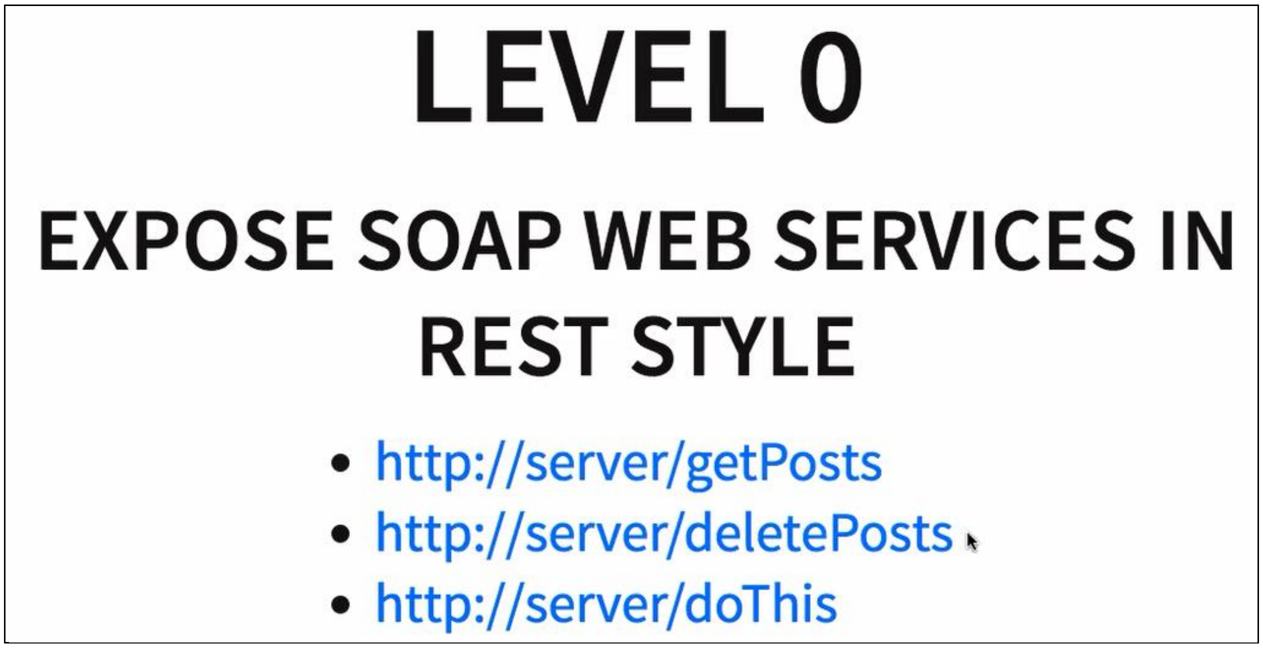
Step 29 - Overview of Connecting RESTful Service to JPA

Step 30 - Creating User Entity and some test data

Step 31 - Updating GET methods on User Resource to use JPA

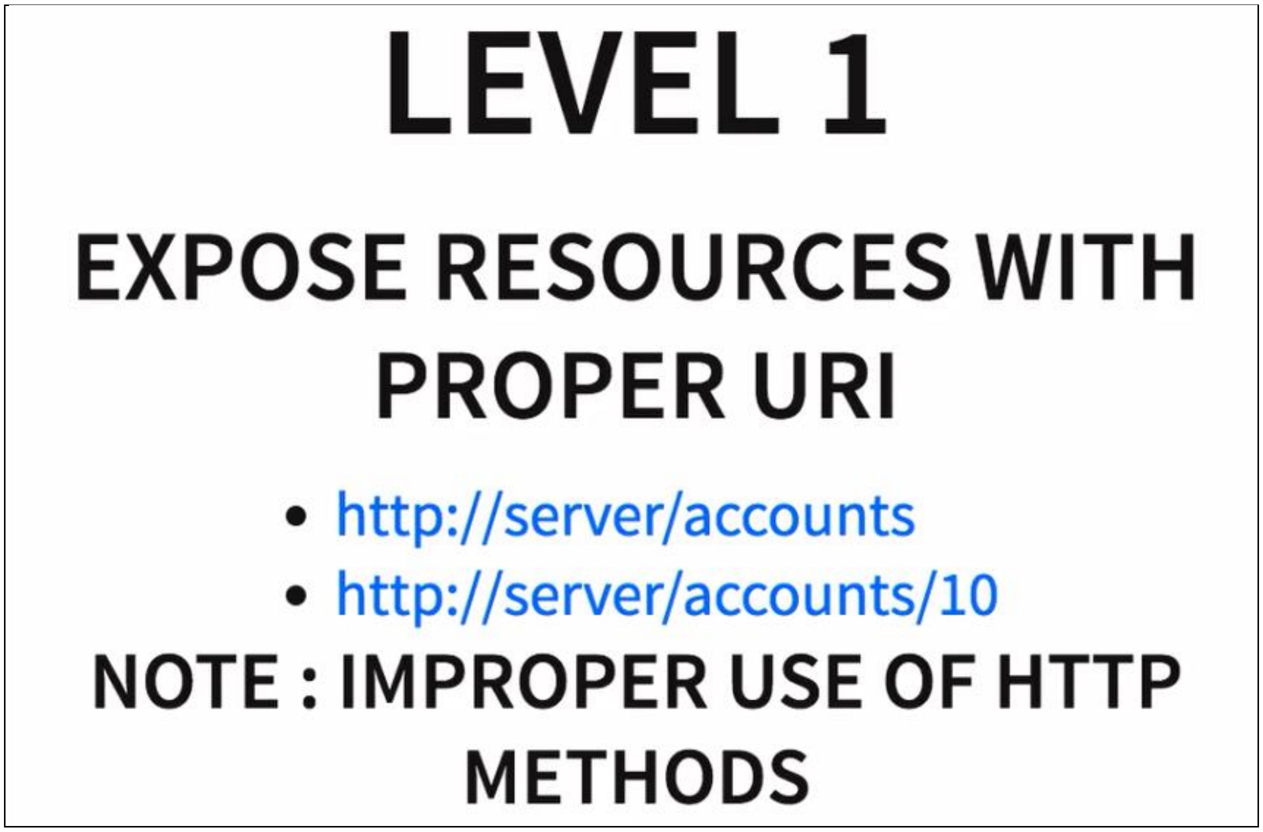
Step 32 - Updating POST and DELETE methods on User Resource to use JPA

Step 36 - Richardson Maturity Model


LEVEL 0

EXPOSE SOAP WEB SERVICES IN REST STYLE

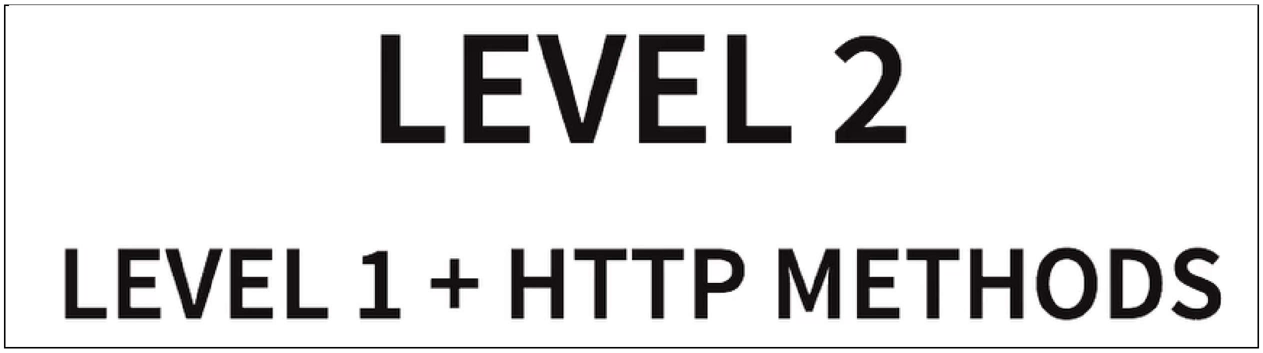
- http://server/getPosts
- http://server/deletePosts
- http://server/doThis


LEVEL 1

EXPOSE RESOURCES WITH PROPER URI

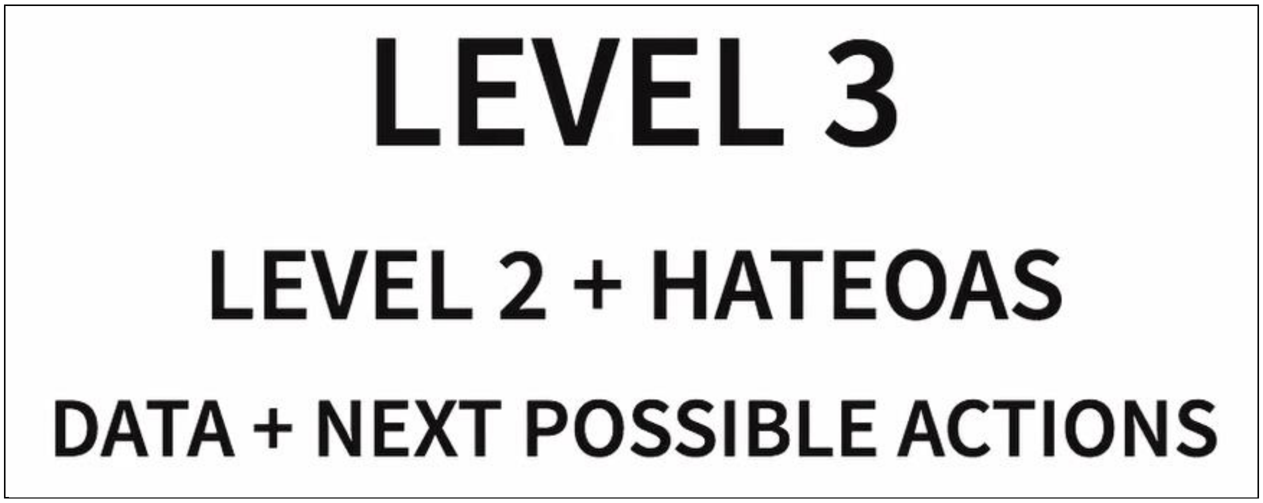
- http://server/accounts
- http://server/accounts/10

NOTE : IMPROPER USE OF HTTP METHODS



LEVEL 2

LEVEL 1 + HTTP METHODS



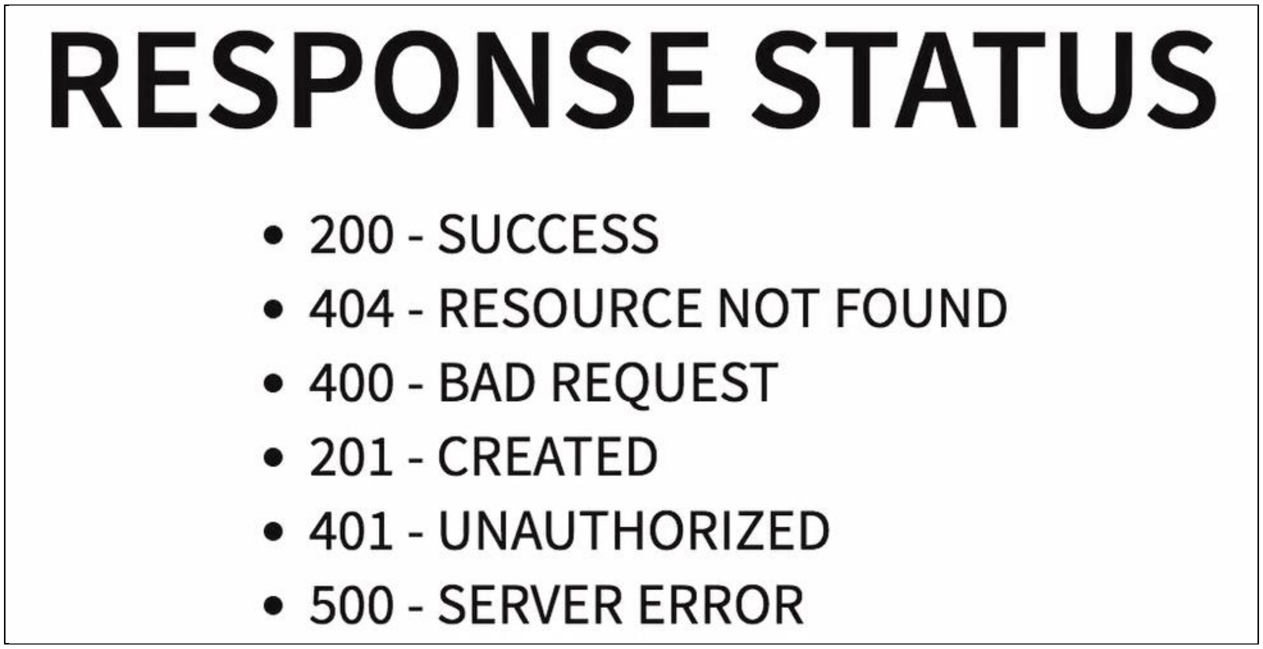
LEVEL 3

LEVEL 2 + HATEOAS

DATA + NEXT POSSIBLE ACTIONS

Step 37 - RESTful Web Services - Best Practices
CONSUMER FIRST
GREAT DOCUMENTATION
MAKE BEST USE OF HTTP
MAKE THE BEST USE OF REQUEST METHODS

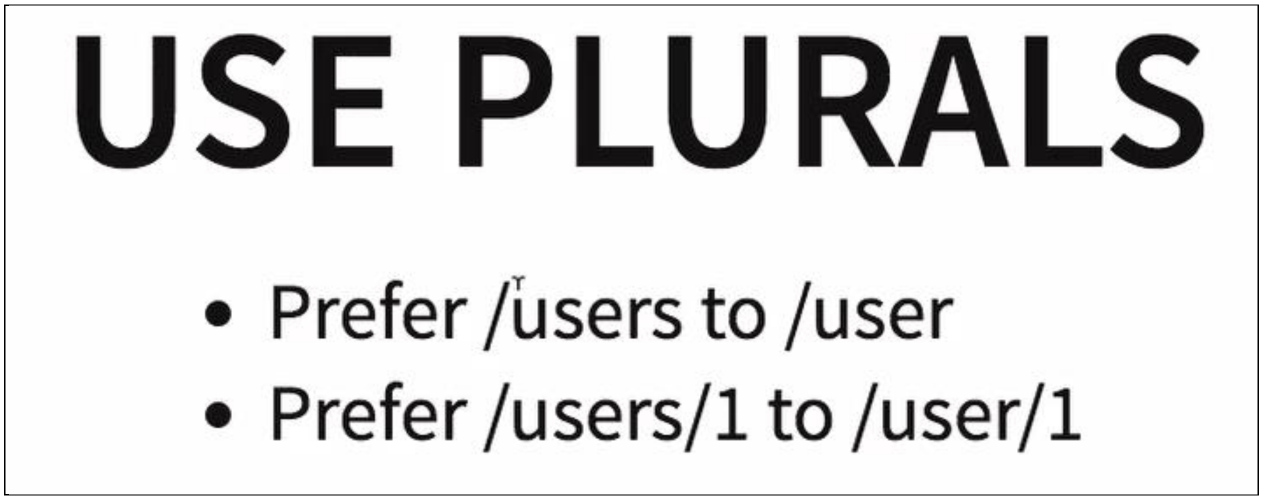
- . GET
- .POST
- .PUT
- .DELETE



RESPONSE STATUS

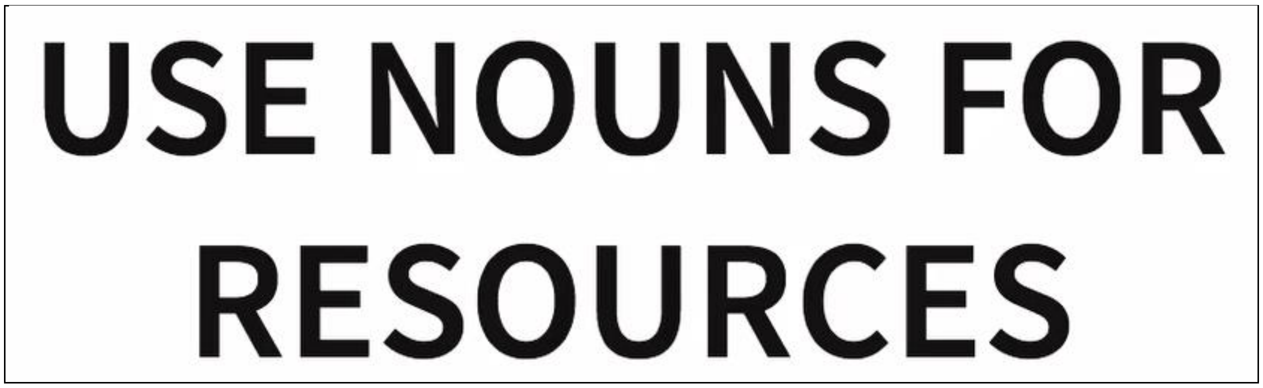
- 200 - SUCCESS
- 404 - RESOURCE NOT FOUND
- 400 - BAD REQUEST
- 201 - CREATED
- 401 - UNAUTHORIZED
- 500 - SERVER ERROR

NO SECURE INFO IN THE URI

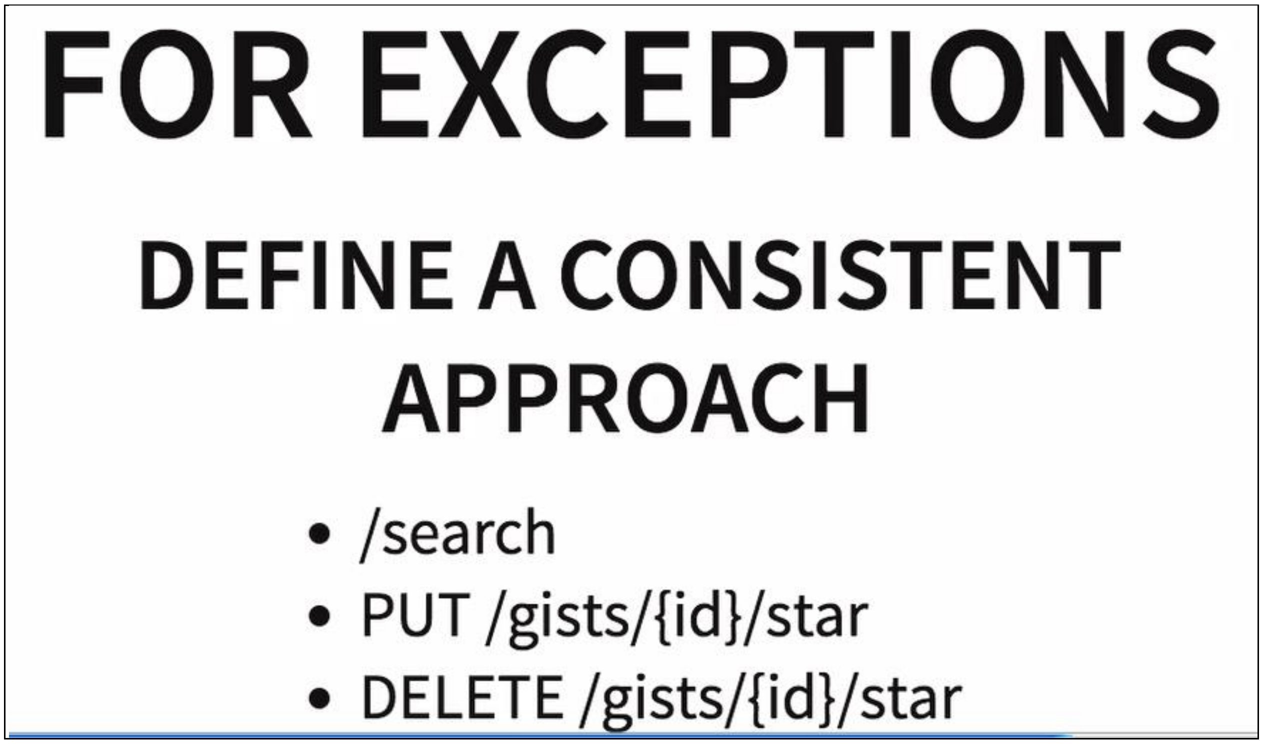


USE PLURALS

- Prefer /users to /user
- Prefer /users/1 to /user/1



USE NOUNS FOR RESOURCES



FOR EXCEPTIONS

DEFINE A CONSISTENT APPROACH

- /search
- PUT /gists/{id}/star
- DELETE /gists/{id}/star

Examples of the above: If there is a search link on the web page. Define a consistent approach.