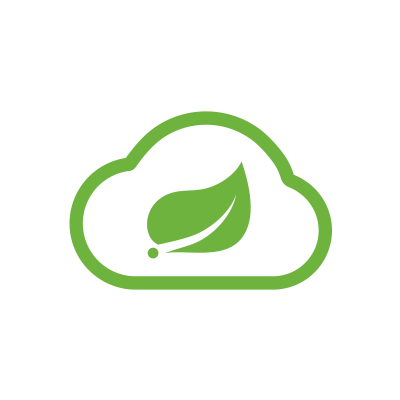
# Micro Services Spring Cloud

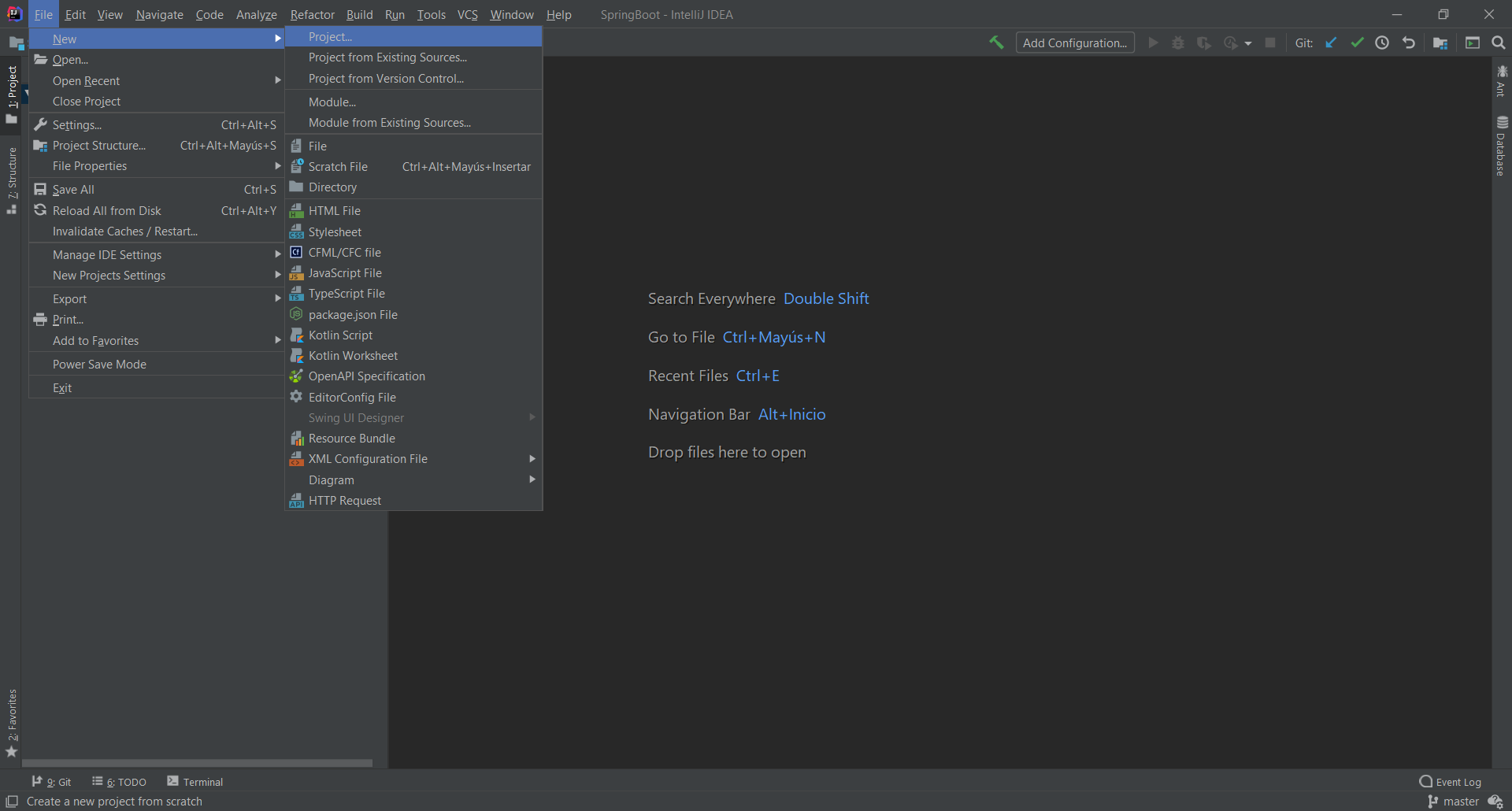


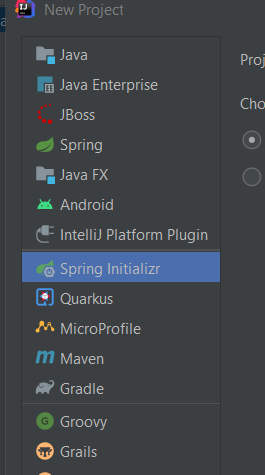
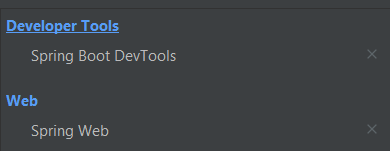
# Getting Started

There are many ways to start a project in spring, one is with initializr of <https://start.spring.io/>

Where you select your dependencies,

Another way its with the IDE, in this case Intellij IDE



 🡺Dependencies 

Then click on next 🡺 change name, click on next 🡺 select the dependencies. Click on next and finish🡺 Ready.

## @Rest Controller:

Documentation: <https://spring.io/guides/gs/rest-service/>

In Spring’s approach to building RESTful web services, HTTP requests are handled by a controller. These components are identified by the [@RestController](https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/bind/annotation/RestController.html) annotation,

The Rest controller will be the one that handle your HTTP request and the class can support multiple annotations in order to create a CRUD.

The annotation [@RestController](https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/bind/annotation/RestController.html) will allow the class to receive HTTP request but has to have a annotations on a function to do a specific function:

@RequestMapping will be the one that put the endpoint to the controller http://localhost:8080/**users**

@GetMapping is the one that will handle the request GET (to obtain objects or a response)

@PostMapping is the one that will handle the request POST (to create a new object)

@PutMapping is the one that will handle the request PUT (to edit a object already created)

@DeleteMapping is the one that will handle the request Delete (to delete a object already created)

Example:

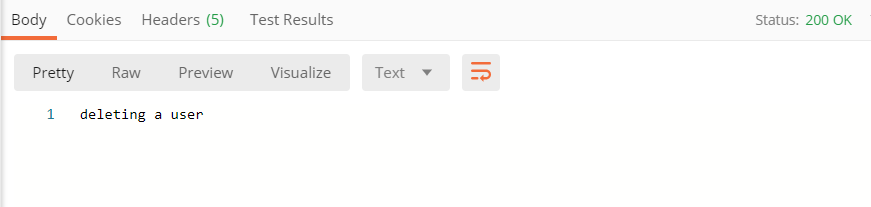
@RestController  
@RequestMapping("/users")  
public class Controller {  
  
 @GetMapping  
 public String getUser(){  
 return "getting user";  
 }  
  
 @PostMapping  
 public String postUser(){  
 return "user Created";  
 }  
  
 @PutMapping  
 public String editAUser(){  
 return "editing a user";  
 }  
  
 @DeleteMapping  
 public String deleteAUser(){  
 return "deleting a user";  
 }  
  
  
}

When we run the code, tomcat will deploy it to a local server by default the port is set in 8080

Using postman, we will send the request GET, PUT, POST, DELETE



Change the verb into , put, post ,delete



Status code:

<https://www.restapitutorial.com/httpstatuscodes.html>

Response of the http request

Rfttt

## @PathVariable

Path variable is an annotation that will allow take a value in the URL in order to user it in the function, for example when we want to obtain a specific user:

The annotation GetMapping will take the path (“/{**VARIABLE**}”) from the URL and will pass it to the function as an argument under the annotation @PathVariable dataType VARIABLE

@GetMapping(path = "/{VARIABLE}")  
public String getUser(@PathVariable String VARIABLE){  
 return "getting user: "+ VARIABLE;

@GetMapping(path = "/{userId}")  
public String getUser(@PathVariable String userId){  
 return "getting user: "+ userId;

PostMan:



## @RequestParam

Request params is an annotation that will take the parameters from the URL directly on the arguments of the function

Function(@RequestParams(value=”StringVariable”) dataType StringVariable, @RequestParams(value=”StringVariable2”) dataType StringVariable2)

@GetMapping()  
public String getUsers(@RequestParam(value = "page")int page,@RequestParam(value = "limit")int limit ){  
 return "getting users of page: "+page+" and limit of: "+ limit+" users";  
}



### Optional parameters:

To make optional parameters and void a error for Null Point Exception we will user **defaulValue=”value” AFTER value=”variable”**

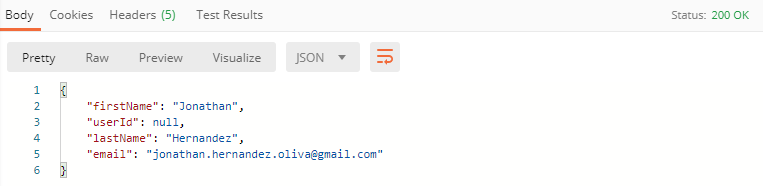
@GetMapping()  
public String getUsers(@RequestParam(value = "page", defaultValue = "1")int page,@RequestParam(value = "limit", defaultValue = "50")int limit ){  
 return "getting users of page: "+page+" and limit of: "+ limit+" users";  
}



## Returning a Java Object as Returning value.

In this example, the controller will return a object in JSON

@GetMapping(path = "/{userId}")  
public UserRest getUserModel(@PathVariable String userId){  
 UserRest user1 = new UserRest();  
 user1.setFirstName("Jonathan");  
 user1.setLastName("Hernandez");  
 user1.setEmail("jonathan.hernandez.oliva@gmail.com");  
 return user1;  
}



## Set Response Status Code:

As a best practice we need to return a reponse code status and a object, sometimes only status code its ok, for this reason we need to return a **new ResponseEntity<Class>(Object, HttpStatus.Status)**

Example:

@GetMapping(path = "/{userId}")  
public ResponseEntity<UserRest> getUserModel(@PathVariable String userId){  
 UserRest user1 = new UserRest();  
 user1.setFirstName("Jonathan");  
 user1.setLastName("Hernandez");  
 user1.setEmail("jonathan.hernandez.oliva@gmail.com");  
 return new ResponseEntity<UserRest>(user1,HttpStatus.*OK*);  
}

## Post Request:

For this example we will créate another Class called “UserPostRequest” and will be a copy of UserRest, it means the same parameters will be seted. In the function we will return the same as the provious but **in the parameters we will pass @RequestBody Class Object**

@PostMapping  
public ResponseEntity<UserRest> postUser(@RequestBody UserPostRequest userPost){  
 UserRest user1 = new UserRest();  
 user1.setFirstName(userPost.getFirstName());  
 user1.setLastName(userPost.getLastName());  
 user1.setEmail(userPost.getEmail());  
 return new ResponseEntity<UserRest>(user1,HttpStatus.*OK*);  
}

@RequestBody Class Object recieve the object and we will map the object with the UserRest in order to return UserRest as response

In Postman we will set it like this:

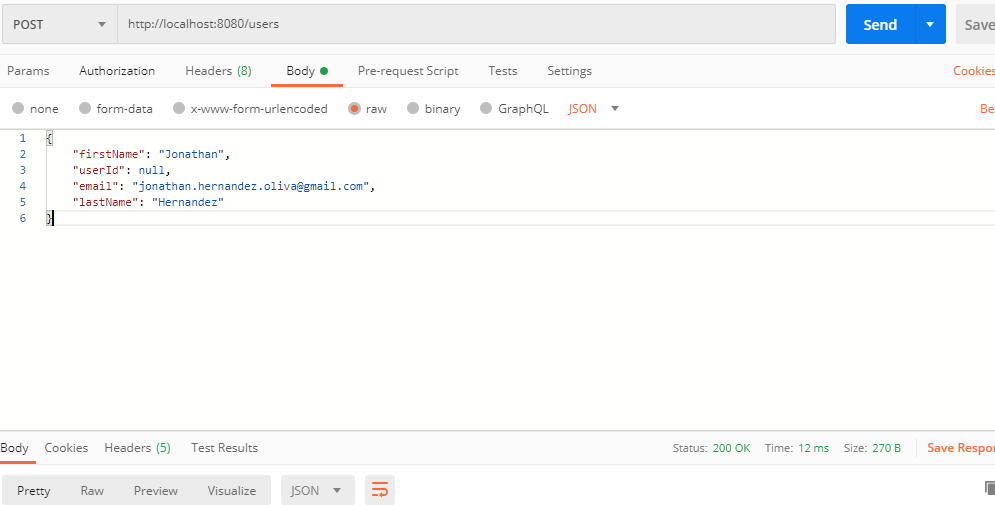
Post Verb

Body in JSON format

URL without params

Status code

Response



## Validating Http POST Request Body

In newer versions you may need this dependency to make validations

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

We will use hibernate validater,

The annotation @valid from javax.valid will be the one that validate that all the requierements of the class are true, those validations comes from another annotations like @NotNull, @Email, @size etc.

Example:

@PostMapping  
public ResponseEntity<UserRest> postUser(@Valid @RequestBody UserPostRequest userPost){  
 UserRest user1 = new UserRest();  
 user1.setFirstName(userPost.getFirstName());  
 user1.setLastName(userPost.getLastName());  
 user1.setEmail(userPost.getEmail());  
 return new ResponseEntity<UserRest>(user1,HttpStatus.*OK*);  
}

The @Valid has to be placed in the beggining of the parameters of the function.

public class UserPostRequest {

@NotNull(message = "first name cannot be empty")  
 private String firstName;  
 @NotNull(message = "last name cannot be empty")  
 private String LastName;  
 @Email  
 @NotNull(message = "e-mail cannot be empty")  
 private String Email;  
 @NotNull(message = "password cannot be null")  
 @Size(max = 16, min = 6, message = "password must be greater than 6 characters and greater than 16 characters")  
 private String password;  
 //GETTERS AND SETTERS

## Store Users Temporary.

In order to user putMapping we need to persist the Object without using database,

For this we need to créate a HashMap object

Map<String,UserRest> users;

Map<String,UserRest> users;

Its a class that implements the class Map

This object accepts 2 objects, one will be the key(could be String) and the second will be the value, when we Access to the value, will be throught the key.

To créate a Map object we will use the method put()

**Example:**

Map<String ,UserRest> users;  
 //to post a new user  
  
 @PostMapping  
 public ResponseEntity<UserRest> postUser(@Valid @RequestBody UserPostRequest userPost){  
 UserRest user1 = new UserRest();  
 user1.setFirstName(userPost.getFirstName());  
 user1.setLastName(userPost.getLastName());  
 user1.setEmail(userPost.getEmail());  
 String userId = UUID.*randomUUID*().toString(); // create a UUID  
 user1.setUserId(userId);  
  
 if(users== null ){ //if users is null initialice the object  
 users= new HashMap<>();  
 users.put(userId,user1); // and create a new object  
 }  
 return new ResponseEntity<UserRest>(user1,HttpStatus.*OK*);  
 }  
//to get a specific user through the Id  
 @GetMapping(path = "/{userId}")  
 public ResponseEntity<UserRest> getUserModel( @PathVariable String userId){  
 if(users.containsKey(userId)){  
 return new ResponseEntity<>(users.get(userId),HttpStatus.*OK*);  
 }else{  
 return new ResponseEntity<>(HttpStatus.*NO\_CONTENT*);  
 }  
  
 }

## PUT:

For the put, we will look for the Id, and we will recieve a json so we will recieve the pathVariable and the RequestBody

When we recieve the pathVarible (userId) we will find it throught the Map id and create a new object

with this line: UserRest userEdit = users.get(userId);

after this, we will edit the object with the payload that we recieved :

@PutMapping(path = "/{userId}")  
public ResponseEntity<UserRest> editAUser(@PathVariable String userId,@RequestBody UserPostRequest userPost ){  
 UserRest userEdit = users.get(userId);  
 userEdit.setFirstName(userPost.getFirstName());  
 userEdit.setLastName(userPost.getLastName());  
   
 return new ResponseEntity<UserRest>(userEdit,HttpStatus.*OK*);  
}