**Approach – 1 On how to read the properties with the help of @value annotation**

So v1 indicates version 1 – In side this version 1 microservices inside our section 6 we are going to follow the basic approaches provided by the spring boot framework. Once we discuss all the basic approaches provided by the spring boot framework then will look for the advanced approaches inside the next version which is v2.

So, inside this v1-springboot folder we are copying all the microservices that we have in the section4. In the section6 we are going to follow a google jib approach.

Step 1 – inside the application.yml file we can define any properties along with their values which we can read inside our java code

build:  
 version: "1.0.0"

using this property read the value inside our java code and also build a small rest API which can our client applications can invoke and understand what is the build version right now our microservice is using.

Step 2 – Go to the controller class **AccountsController** -> Here, first we read the property define inside our application.yml file -> for that we create a new java field inside our controller class.

private String buildVersion;

This will have null value right now to inject the property value during the start up of the spring boot application into this java filed we need to mention the annotation is **@value** to this annotation we need to pass the property key name by following the spring expression language @value(“${build.version:1.0.0}”) so with this changes the build version java field will have the value that we define inside the yml file.

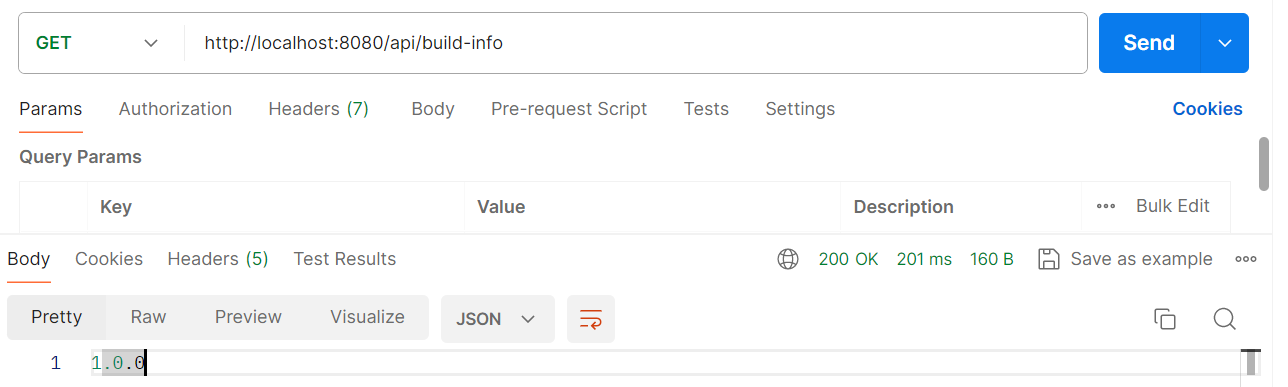
Going to build a small REST API service to send the build version information to our client for that -> we have to go to the end of controller class.

@Operation(  
 summary = "Get Build Information",  
 description = "Get Build Information that is deployed into accounts microservice"  
 )  
 @ApiResponses({  
 @ApiResponse(  
 responseCode = "200",  
 description = "Http Status OK"  
 ),  
 @ApiResponse(  
 responseCode = "500",  
 description = "Http Status Internal Server Error",  
 content = @Content(  
 schema = @Schema(implementation = ErrorResponseDto.class)  
 )  
 )  
 }  
 )  
 @GetMapping("/build-info")  
 public ResponseEntity<String> getBuildInfo() {  
 return ResponseEntity  
 .*status*(HttpStatus.*OK*)  
 .body(buildVersion);  
 }  
}

Step 3 – We can remove the @AllArgsConstructor instead we have to create a manual constructor

private final IAccountsService iAccountsService;  
public AccountsController(IAccountsService iAccountsService){  
 this.iAccountsService=iAccountsService;  
}// run the accounts microservice with the debug mode.

Step 4 – using this Api endpoint <http://localhost:8080/api/build-info> Get method to get the version response from the postman.



This is the version mention in the application.yml file. So, this way we can read the properties inside the application.properties file or application.yml file or inside any other file with the help of @value annotation and this allows to read all our properties from our java code itself.

This approach is sufficient for microservice? This is not going to work if we have 100s of microservices and if they have 100 different properties we can’t crate 100s different fields inside our microservices? This is not a good solution.

Under the same time in this approach in the AccountsController while injecting java filed we are hardcoding the property name like build.version so, creating a java filed and mentioning the hardcoded property value for every microservice is not going to be feasible option. That’s why this approach is only recommended we have only one or two property, we have many properties this approach will not going to work.