Spring MVC - @RequestParam Annotation

Last Updated : 07 Aug, 2025

The @RequestParam annotation is one of the most commonly used annotations in Spring MVC for handling HTTP request parameters. @RequestParam annotation enables Spring to extract input data that may be passed as a query, form data or any arbitrary custom data.

Key features of @RequestParam annotation

- **Query Parameter Extraction:** Easily extract query parameters from URLs.
- **Default Values:** Provide default values for optional parameters.
- **Multiple Values:** Handle multiple values for a single parameter.

Now, we will see how we can use @RequestParam when building RESTful APIs for a web-based application.

Let us suppose we are implementing a sample feature for the **Geeksforgeeks** web application, where users will be able to post and retrieve article topics available to write articles on. Here, we will be using hashmaps as our database for simplicity. We use a static block to load the default entry into our hashmap/DB.

Example:

```
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

@RestController
public class ArticleController {
    static int ID = 1;

    public static Map<Integer, String> articleTopics = new HashMap<>();

    static {
        articleTopics.put(0, "GFG");
    }
}
```

Simple GET Mapping using @RequestParam

Let's say we have an endpoint /api/v1/article which takes a query parameter articleId where users will be able to get the article topic given the article id. If the given articleId is not present, we return "Article not accepted" as a response with 400 Bad Request as status.

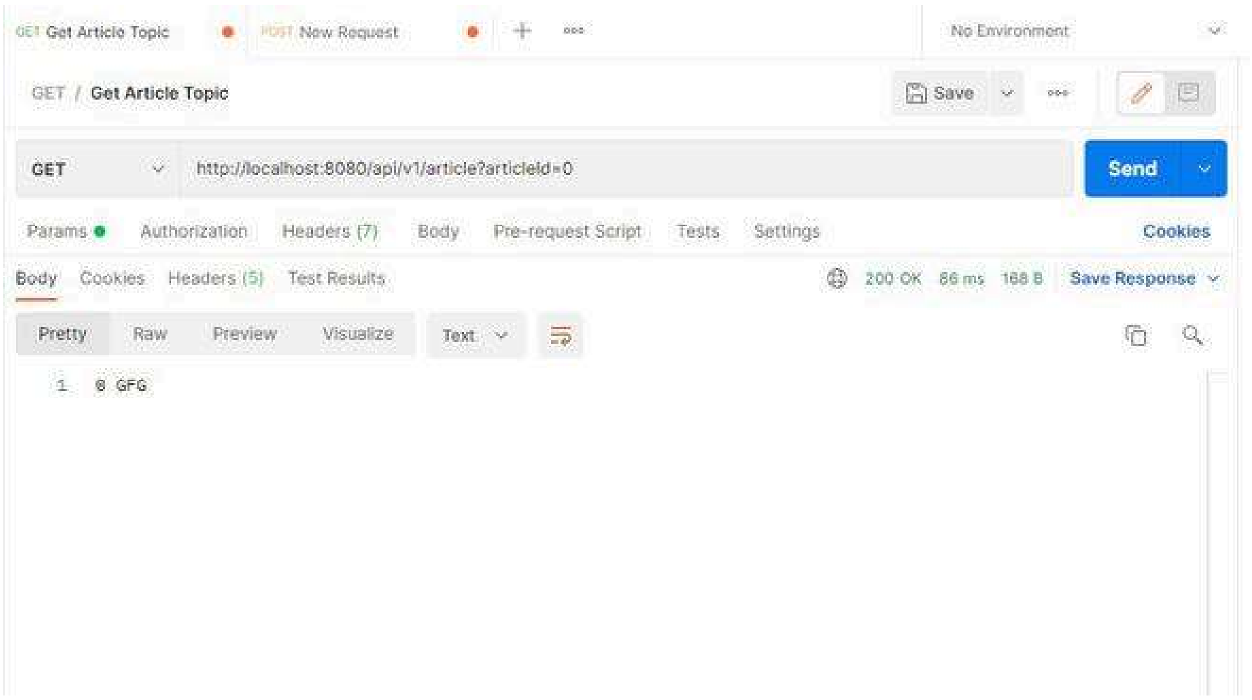
Example:

```
@GetMapping("/api/v1/article")

public ResponseEntity<String> getArticleTopic(@RequestParam Integer
articleId)
{
    if (articleTopics.containsKey(articleId)){
        return ResponseEntity.ok(articleId + " " +
articleTopics.get(articleId));
    }

    return ResponseEntity.badRequest().body(
               "Article does not exist");
}
```

Note:

- Static blocks executes automatically when the class is loaded in memory.
- Annotation @GetMapping is used for mapping HTTP GET requests for specific handler methods.
- ResponseEntity represents an HTTP response which includes headers, body and status.

Testing the APIs with Postman: When we tried to get the article already present in our database, we get the required response body with '200 OK' as status code.

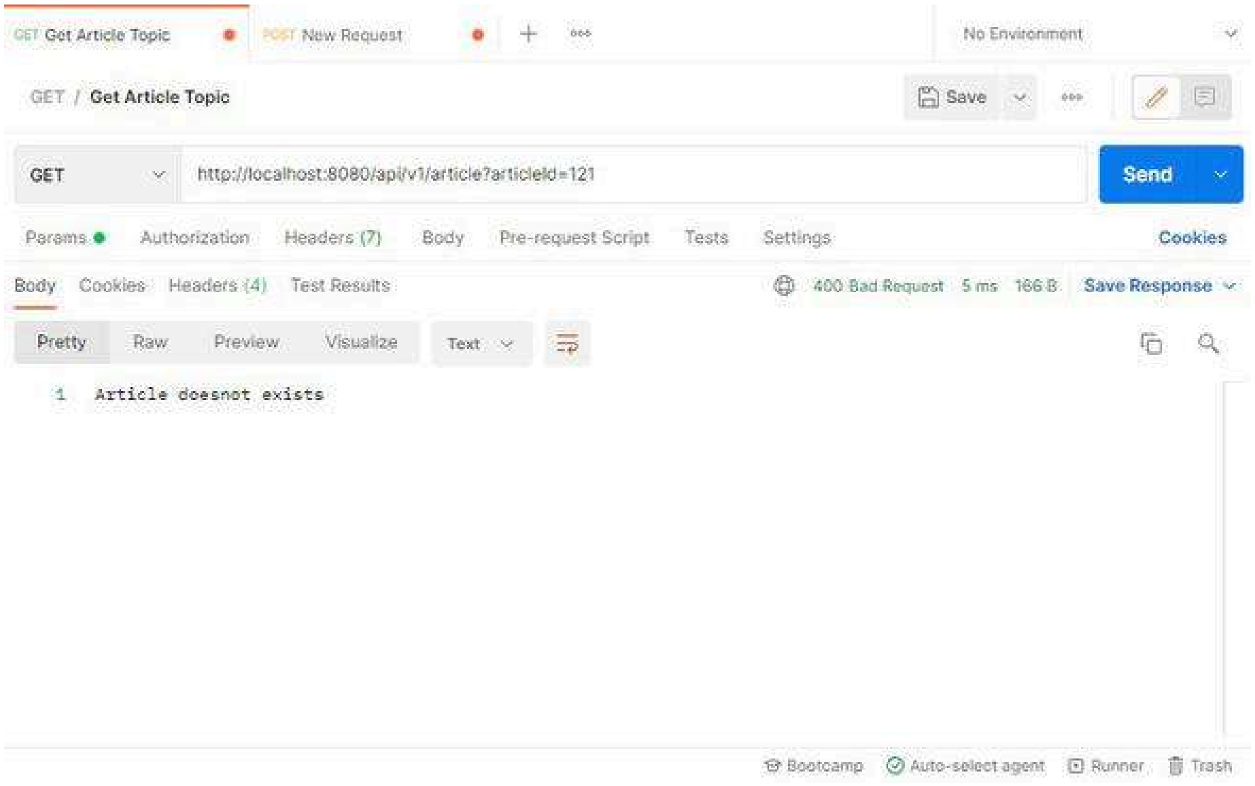


```
http://localhost:8080/api/v1/article?articleId=0
```

200 OK 86 ms 168 B

```
0 GFG
```

When we tried to get the article not present in our database, we get the "Article does not exist" response body with 400 BAD REQUEST as status code.



```
http://localhost:8080/api/v1/article?articleId=121
```

400 Bad Request 5 ms 166 B

```
Article doesnot exists
```

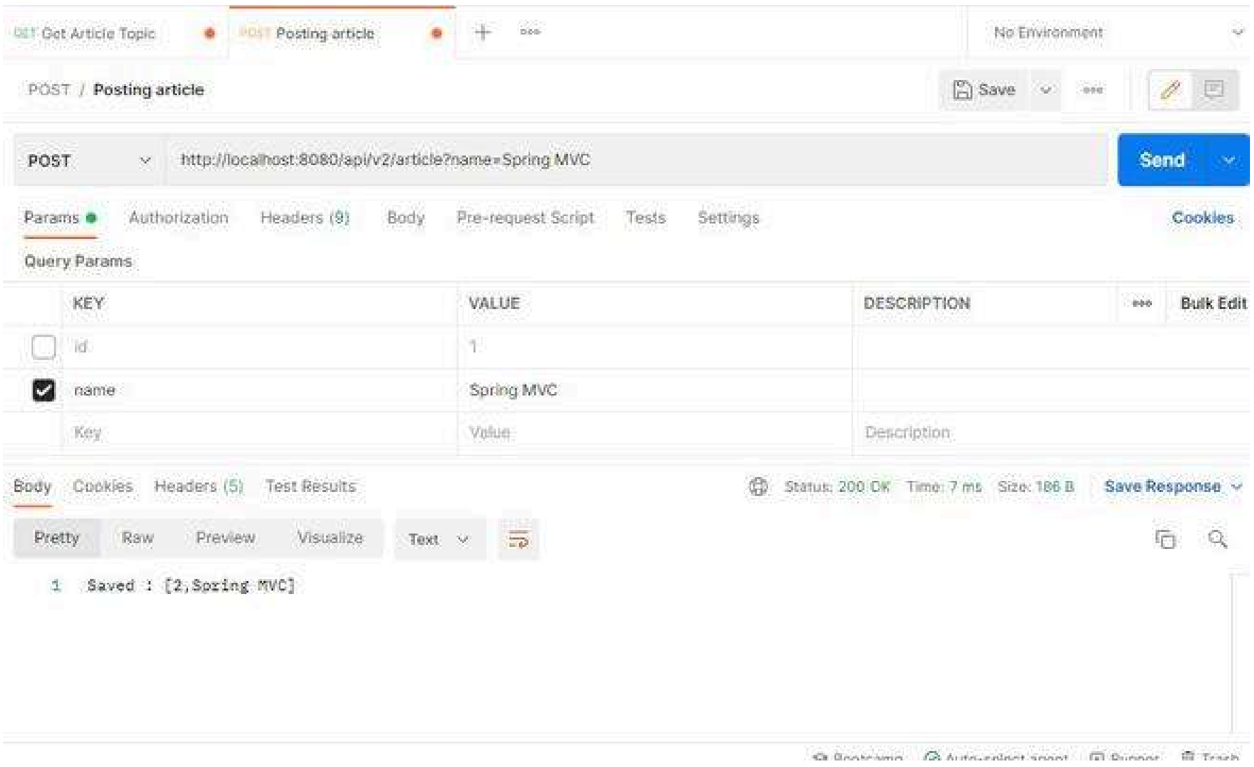
Specifying the Request Parameter Name

Let's say we have an endpoint /api/v2/article for posting article topics which takes a query parameter articleName as name.

Example:

```
PostMapping("/api/v2/article")
public ResponseEntity<String> postArticleTopic(@RequestParam("name") String
articleName) {
    if (articleTopics.containsValue(articleName)) {
        return ResponseEntity.badRequest().body("Article already exists");
    }
    int currentArticleID = ID++;
    articleTopics.put(currentArticleID, articleName);
    return ResponseEntity.ok("Saved: [" + currentArticleID + ", " +
articleTopics.get(currentArticleID) + "]");
}
```

Testing the APIs with postman: When we tried to get the article doesn't exist in our database, we save the request and return a response with 200 OK as status code.



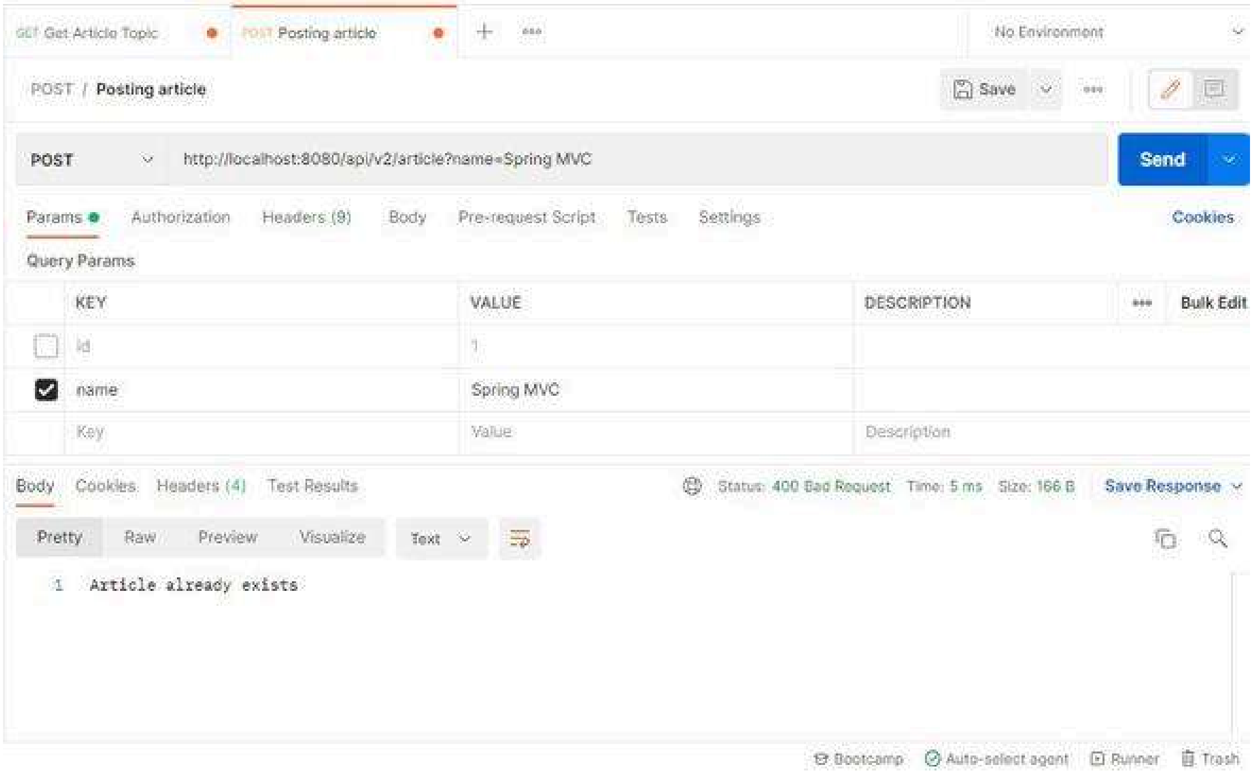
POST http://localhost:8080/api/v2/article?name=Spring MVC

	KEY	VALUE	DESCRIPTION
☐	id	1	
☑	name	Spring MVC	
	Key	Value	Description

Status: 200 OK Time: 7 ms Size: 186 B

```
Saved : [2,Spring MVC]
```

If the article already exists in the database, the API returns 'Article already exists' with a 400 BAD REQUEST status code.



```
POST http://localhost:8080/api/v2/article?name=Spring MVC
```

Query Params

	KEY	VALUE	DESCRIPTION
☐	id	1	
✓	name	Spring MVC	
	Key	Value	Description

Status: 400 Bad Request Time: 5 ms Size: 166 B

```
Article already exists
```

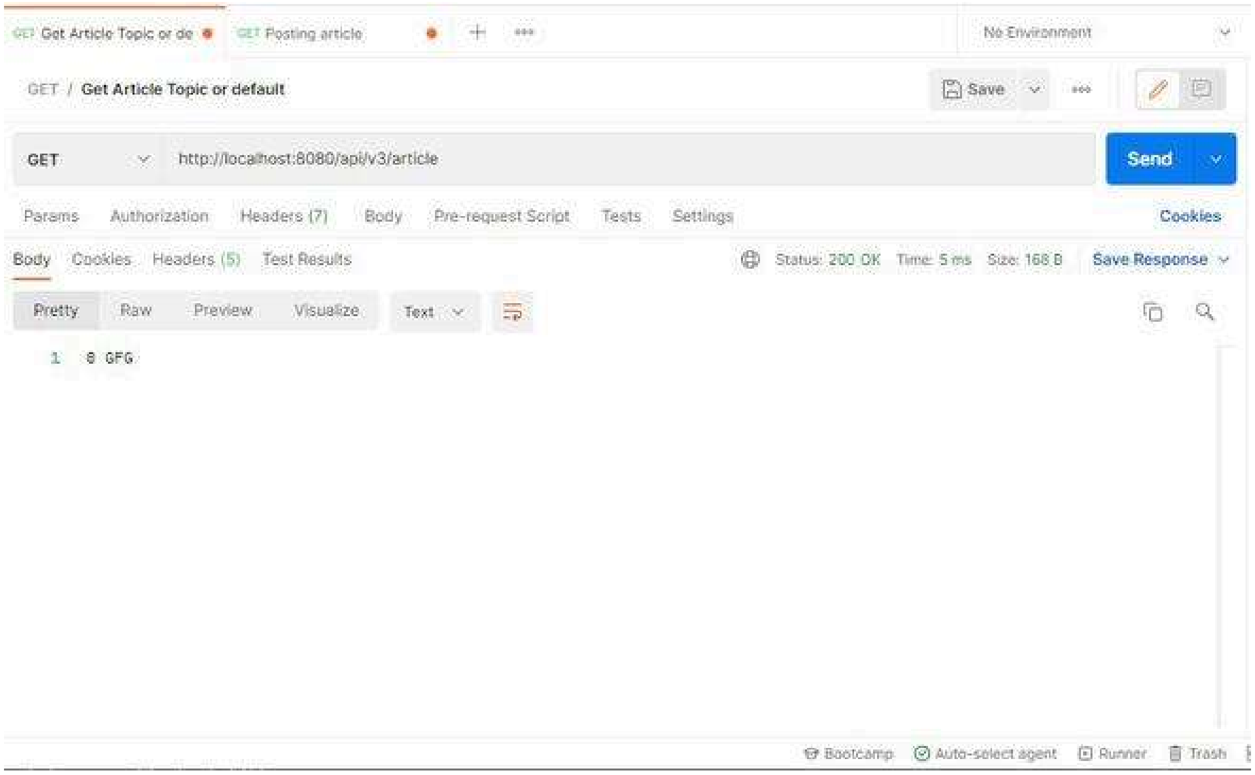
Using @RequestParam with Default Value

Let's say we have an endpoint /api/v3/article for getting article topics which takes a query parameter articleId. Here we have used defaultValue attribute which takes the default value if no value is provided, it defaults to 0.

Example:

```
@GetMapping("/api/v3/article")
public ResponseEntity<String>
getArticleTopicOrDefault(@RequestParam(defaultValue = "0") Integer
articleId) {
    if (!articleTopics.containsKey(articleId)) {
        articleId = 0;
    }
    return ResponseEntity.ok(articleId + " " +
articleTopics.get(articleId));
}
```

Testing the APIs with Postman: If no query parameter is provided, the default value is returned.

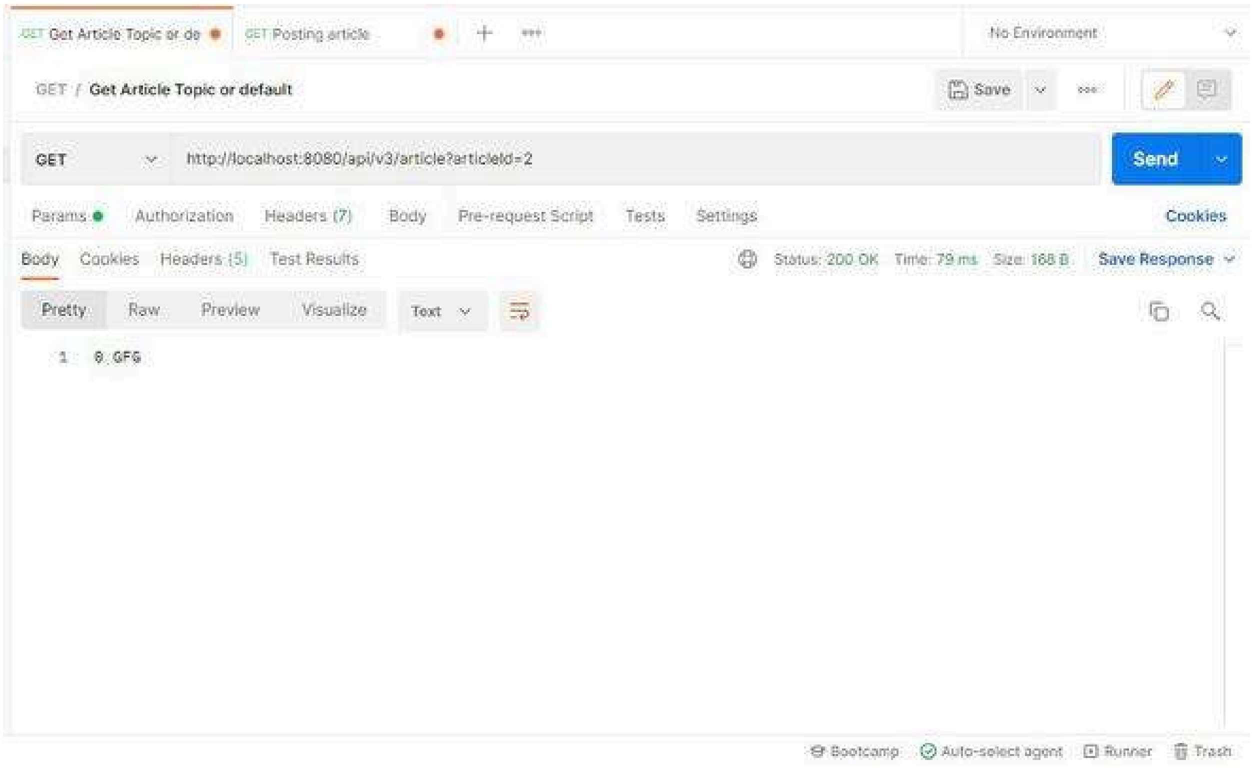

GET / Get Article Topic or default

GET http://localhost:8080/api/v3/article

Status: 200 OK Time: 5 ms Size: 168 B

```
0 GFG
```

If the provided value does not exist in the database, the default value is returned.


GET / Get Article Topic or default

GET http://localhost:8080/api/v3/article?articleId=2

Status: 200 OK Time: 79 ms Size: 168 B

```
0 GFG
```

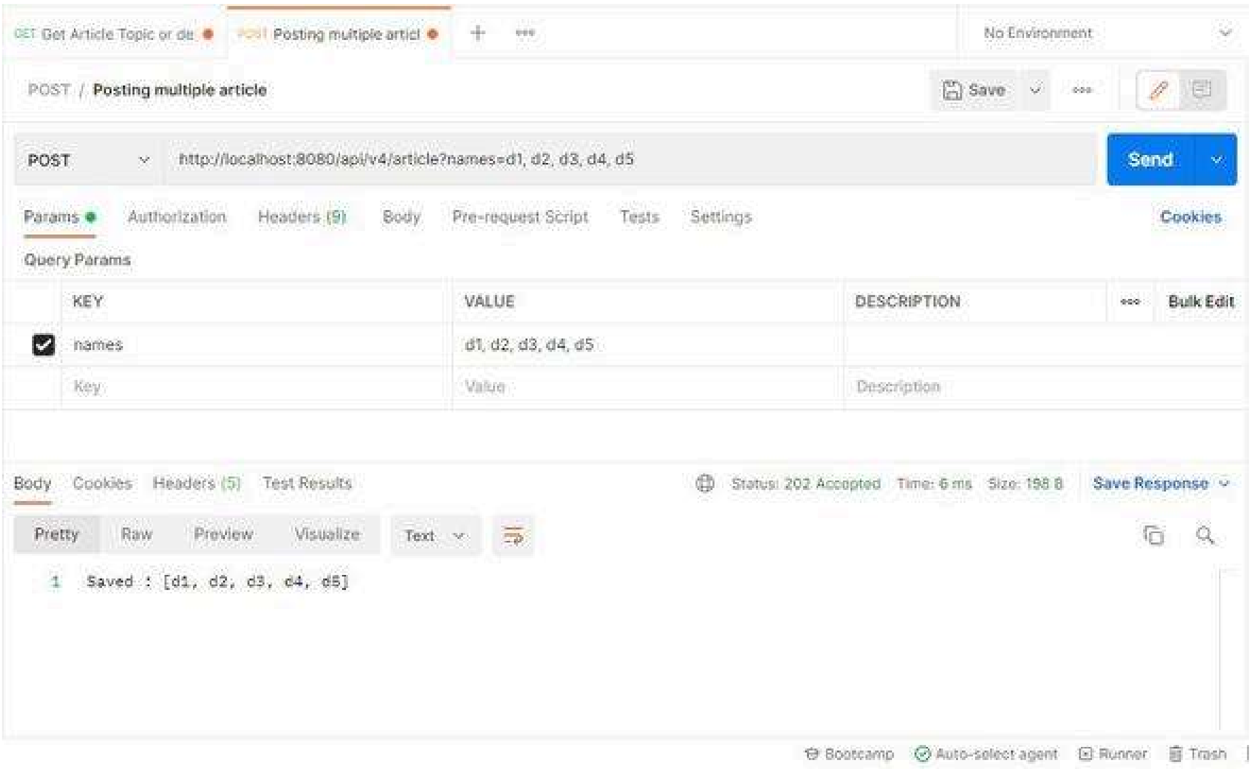
Using @RequestParam For Mapping Multiple Value Query Parameter

Let's say we have an endpoint /api/v4/article for posting article topics which takes a list as a query parameter.

Example:

```
@PostMapping("/api/v4/article")
public ResponseEntity<String> postMultipleArticleTopics(@RequestParam
List<String> names) {
    for (String topic : names) {
        articleTopics.put(ID++, topic);
    }
    return ResponseEntity.accepted().body("Saved: " + names);
}
```

Testing the APIs with Postman: The @RequestParam List<String> names annotation allows you to accept multiple values for the names parameter. The method saves each topic to the database and returns a success message.



```
http://localhost:8080/api/v4/article?names=d1, d2, d3, d4, d5
```

KEY	VALUE	DESCRIPTION
names	d1, d2, d3, d4, d5	
Key	Value	Description

Status: 202 Accepted Time: 6 ms Size: 198 B

```
Saved : [d1, d2, d3, d4, d5]
```

Complete Implementation

Here's a complete implementation of a Spring MVC controller that demonstrates the use of @RequestParam in various scenarios:

```
package com.example.springmvc.RequestParamAnnotation;
```

```java
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

@RestController
public class ArticleController {

    static int ID = 1;

    public static Map<Integer, String> articleTopics = new HashMap<>();

    static {
        articleTopics.put(0, "GFG"); // Default entry
    }

    @GetMapping("/api/v1/article")
    public ResponseEntity<String> getArticleTopic(@RequestParam Integer
articleId) {
        if (articleTopics.containsKey(articleId)) {
            return ResponseEntity.ok(articleId + " " +
articleTopics.get(articleId));
        }
        return ResponseEntity.badRequest().body("Article does not exist");
    }

    @PostMapping("/api/v2/article")
    public ResponseEntity<String> postArticleTopic(@RequestParam("name")
String articleName) {
        if (articleTopics.containsValue(articleName)) {
            return ResponseEntity.badRequest().body("Article already
exists");
        }
        int currentArticleID = ID++;
        articleTopics.put(currentArticleID, articleName);
        return ResponseEntity.ok("Saved: [" + currentArticleID + ", " +
articleTopics.get(currentArticleID) + "]");
    }

    @GetMapping("/api/v3/article")
    public ResponseEntity<String>
getArticleTopicOrDefault(@RequestParam(defaultValue = "0") Integer
articleId) {
        if (!articleTopics.containsKey(articleId)) {
            articleId = 0; // Fallback to default if the provided ID is
invalid
        }
        return ResponseEntity.ok(articleId + " " +
articleTopics.get(articleId));
    }

    @PostMapping("/api/v4/article")
    public ResponseEntity<String> postMultipleArticleTopics(@RequestParam
List<String> names) {
        for (String topic : names) {
            articleTopics.put(ID++, topic);
        }
        return ResponseEntity.accepted().body("Saved: " + names);
    }
}
```

> ***Note :*** *@RestController is a convenience annotation used for creating Restful controllers.*

If we are using Gradle as a build tool for a Spring Boot project, we need to add these dependencies to our build.gradle file.

> *dependencies {*
>
> *implementation 'org.springframework.boot:spring-boot-starter-web'*
>
> *implementation 'jakarta.validation:jakarta.validation-api:3.0.2' // Updated validation API*
>
> *compileOnly 'org.projectlombok:lombok'*
>
> *annotationProcessor 'org.projectlombok:lombok'*
>
> *testImplementation 'org.springframework.boot:spring-boot-starter-test'*
>
> *}*