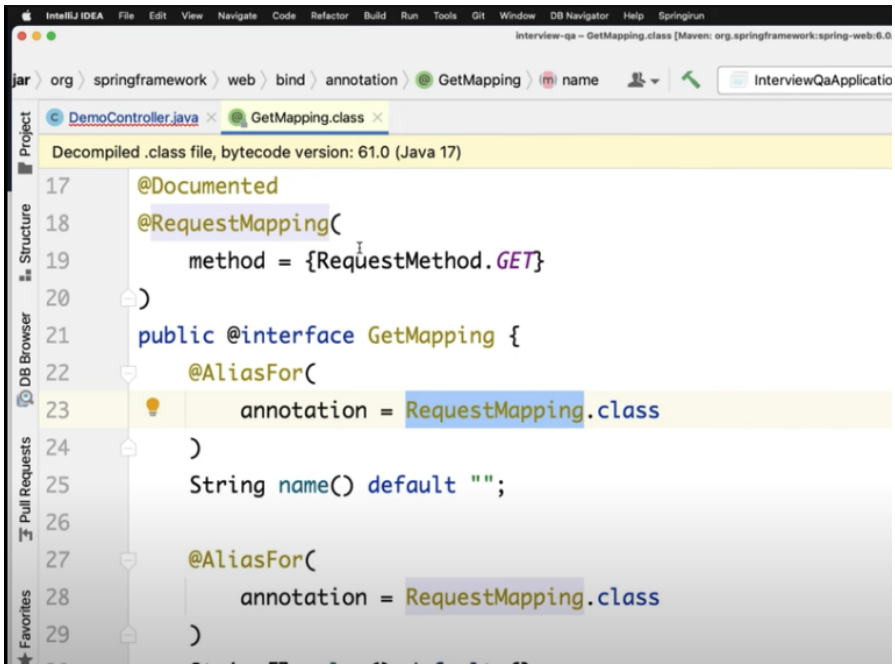
31. Have you worked on Restful webservices ? If yes What all HTTP methods have you used in your project ?

- POST — CREATE
- PUT — UPDATE
- GET — RETRIVE
- PATCH — PARTIAL UPDATE
- DELETE — REMOVE

32. How can you specify the HTTP method type for your REST endpoint?

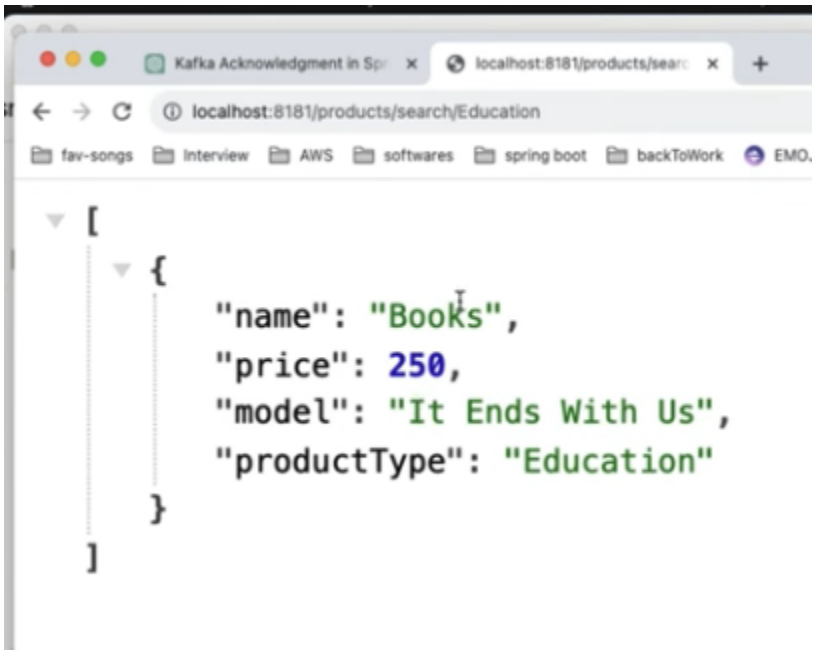
```
@GetMapping("/users")
@PostMapping
@PutMapping
@PatchMapping
@DeleteMapping
@RequestMapping(value = "/users",method = HttpMethod.POST)
public ResponseEntity<?> doSomeOperation(@RequestBody Object input){

}
```

RequestMapping is the parent annotation of all HTTP method annotations. Alternatively we can use Request mapping as shown above.



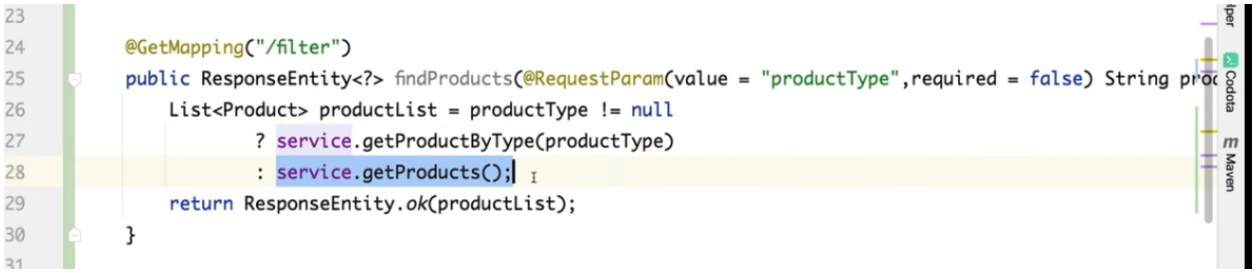
33. Scenario :

Can you design a rest endpoint , Assume that you have a Product database, and your task is to create an API to filter a list of products by productType.

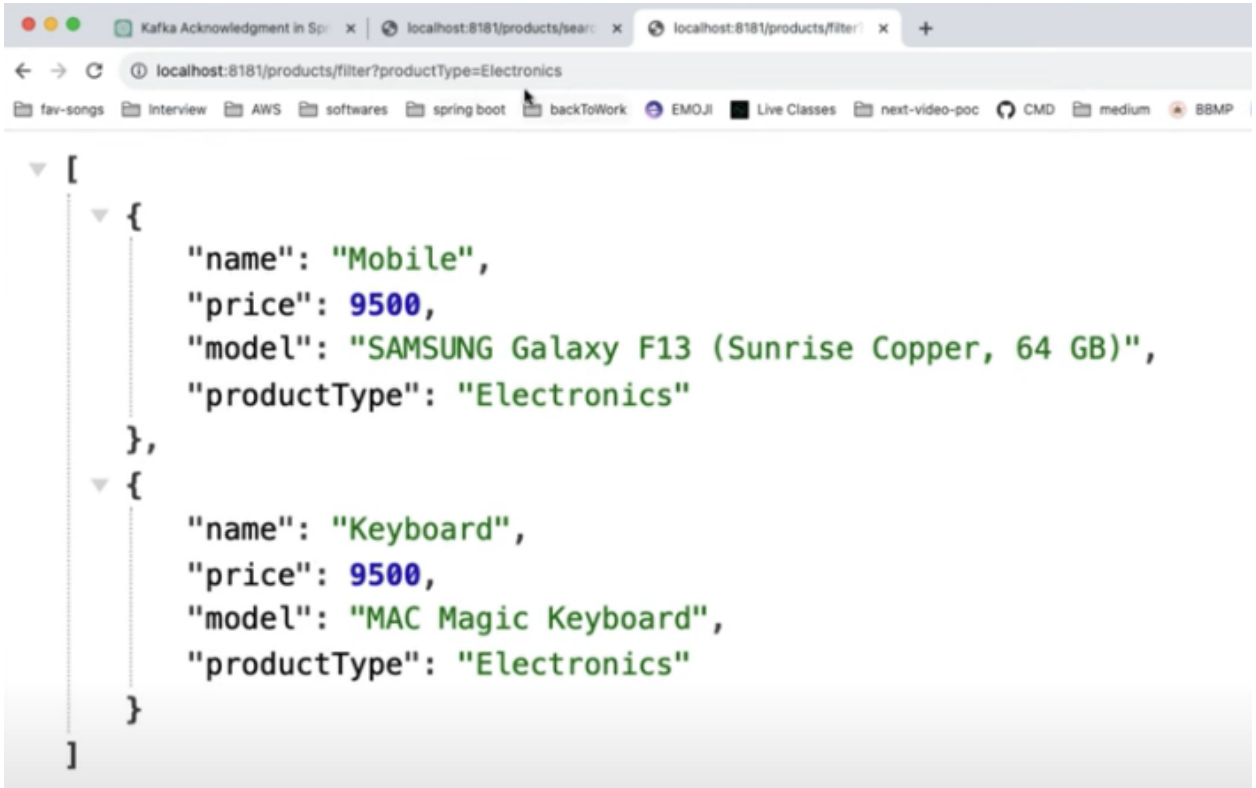


34. Scenario

Design endpoints in a way that takes "productType" as input. If the user provides this input, the endpoint should filter products based on the specified condition. If "productType" is not provided, the endpoint should return all the products.

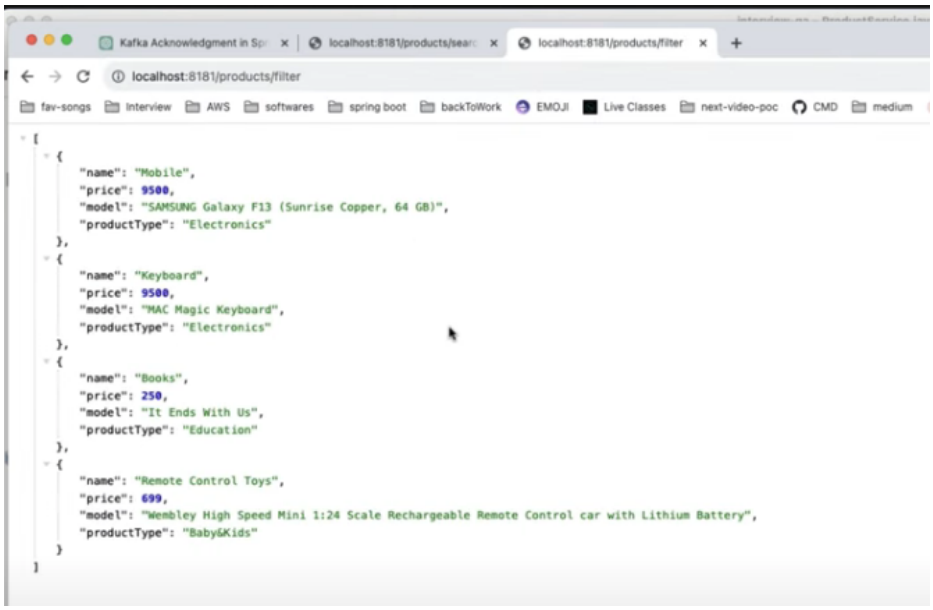


```
23
24      @GetMapping("/filter")
25      public ResponseEntity<?> findProducts(@RequestParam(value = "productType",required = false) String pro
26          List<Product> productList = productType != null
27                  ? service.getProductByType(productType)
28                  : service.getProducts();
29          return ResponseEntity.ok(productList);
30      }
31
```



localhost:8181/products/filter?productType=Electronics

```
[
  {
    "name": "Mobile",
    "price": 9500,
    "model": "SAMSUNG Galaxy F13 (Sunrise Copper, 64 GB)",
    "productType": "Electronics"
  },
  {
    "name": "Keyboard",
    "price": 9500,
    "model": "MAC Magic Keyboard",
    "productType": "Electronics"
  }
]
```



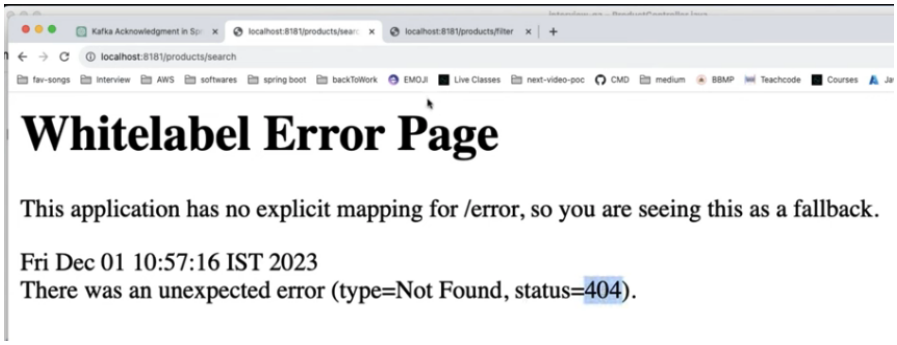
There is some option like defaultValue if not provide anything it will fetch this product type only.

```
findProducts(@RequestParam(value = "productType",required = false,defaultValue = "Electronics")
tList = productType != null
```

35. What is the difference between @PathVariable & @RequestParam

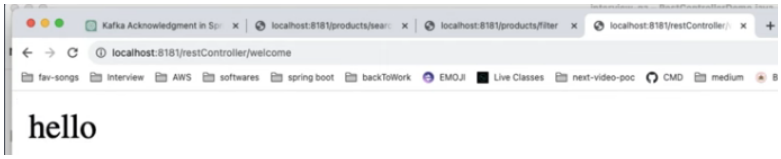
If the path variable is missing we get - 404. Incase of request param it is optional.

```
18  @GetMapping("/search/{productType}")//404
19  public ResponseEntity<?> getProducts(@PathVariable String productType){
20      List<Product> products = service.getProductByType(productType);
21      return ResponseEntity.ok(products);
22  }
23
24  @GetMapping("/filter")
25  public ResponseEntity<?> findProducts(@RequestParam(value = "productType",required = false) String product
26      List<Product> productList = productType != null
```

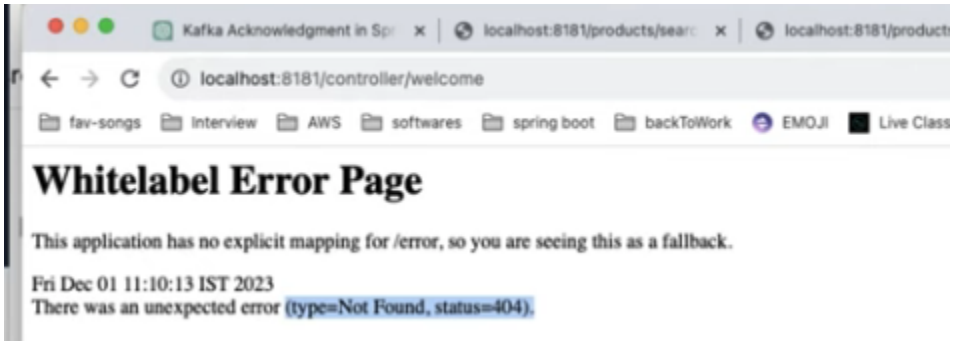


36. Why did you use @RestController why not @Controller ?

```
7   @RestController
8   public class RestControllerDemo {
9
10      @GetMapping("/restController/welcome")
11      public String welcome(){
12          return "hello";
13      }
14  }
```

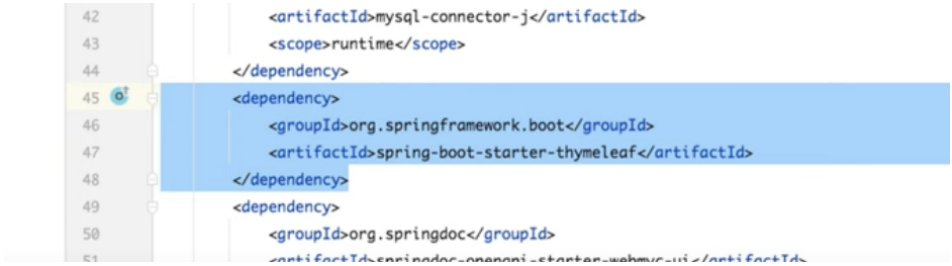



```
6   @Controller
7   public class ControllerDemo {
8
9       @GetMapping("/controller/welcome")
10      public String welcome(){
11          return "hello";
12      }
13  }
```

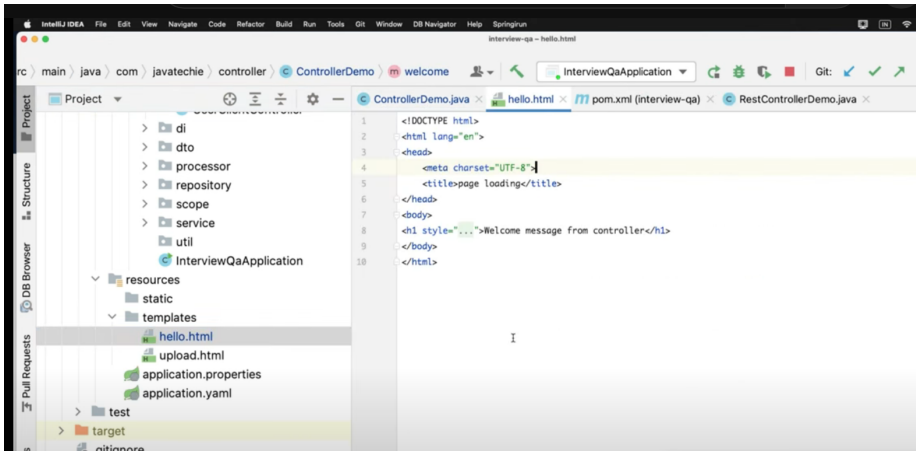


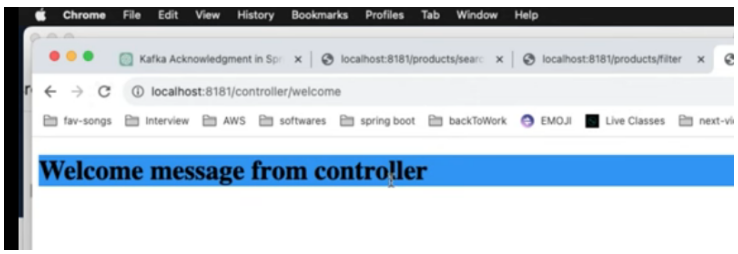
In case of controller it spring expect a page jsp/html page with name hello.jsp
Controller works like a model and view. How this works is explained below.

Add below dependency



Now add hello.html

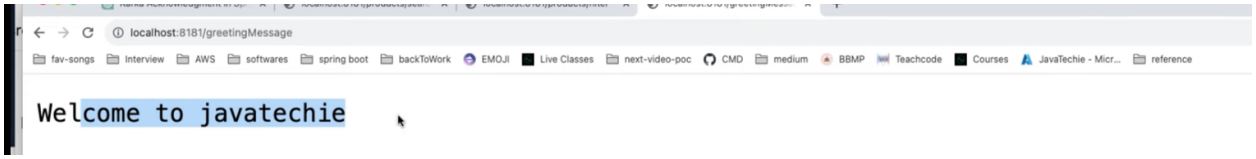




How can we make the controller work like a rest controller? We have to explicitly add mediatype and response body as shown below.

```
@GetMapping(value = "/message",produces = MediaType.TEXT_PLAIN_VALUE)
@ResponseBody
public String message(){
    return "Welcome to javatechie";
}
```

He changed to greetingMessage



RestController = Controller + ResponseBody

37. How can we deserialize a JSON request payload into an object within a Spring MVC controller ?

```java
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class Author {
    private long authorId;
    private String name;
    private int birthYear;
}
```

Keys in the request and variable names must match. This is how we pass it to endpoint. Shown below.

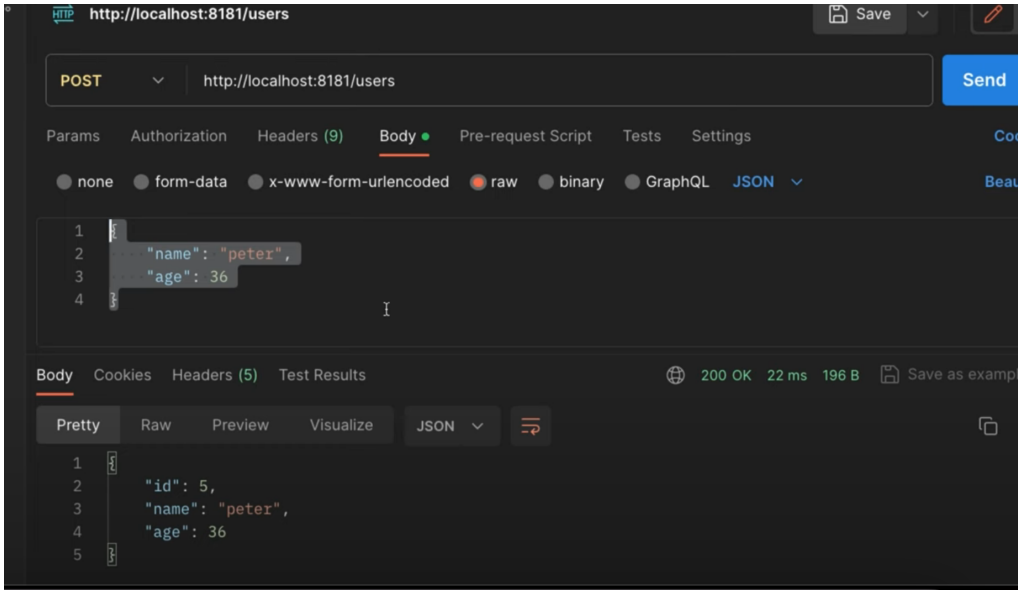
```java
    @PostMapping("/books")
    public String processBook(@RequestBody Book book){
        return book.getTitle()+" New book has been published on year "+book.getPubli
    }
}
```

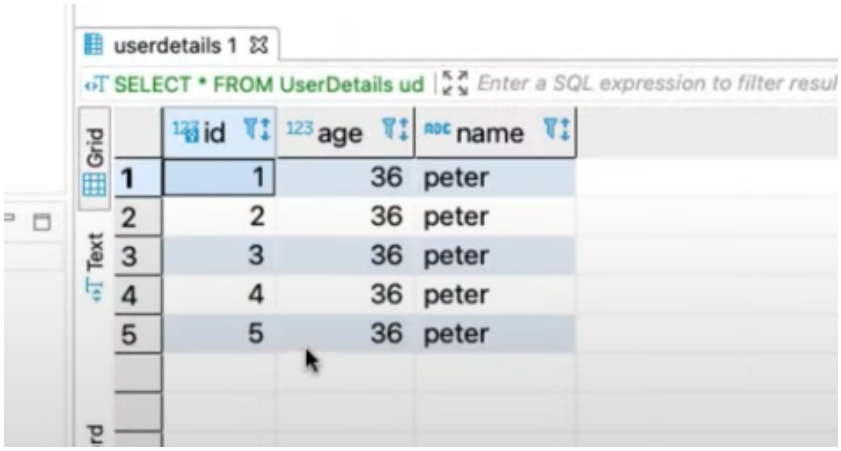
38. Can we perform update operation in POST http method if yes then why do we need Put Mapping or put http method?

```java
@PostMapping("/users")
public UserDetails addNewUser(@RequestBody UserDetails userDetails){
    return repository.save(userDetails);
}

@PutMapping("/users/{id}")
public UserDetails addAndUpdateUser(@PathVariable int id,@RequestBody UserDetails userDetails){
    UserDetails existingUserDetails = repository.findById(id).get();
    existingUserDetails.setName(userDetails.getName());
    existingUserDetails.setAge(userDetails.getAge());
    return repository.save(existingUserDetails);
}
```

Post is to create a record whereas put is to update an existing record.
Still we can do updates in the post mapping but it is a violation in restful design. In rest each http method has specific purpose and semantic meaning

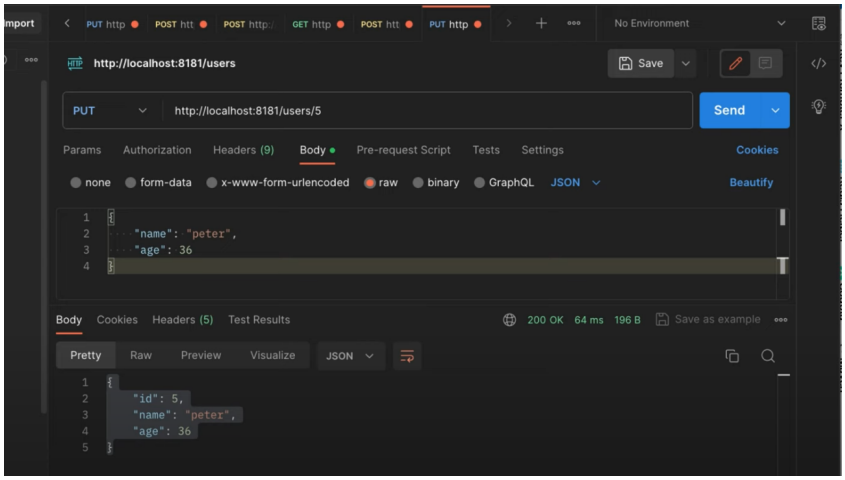
Post method should not be idempotent - create new resource for each hit.





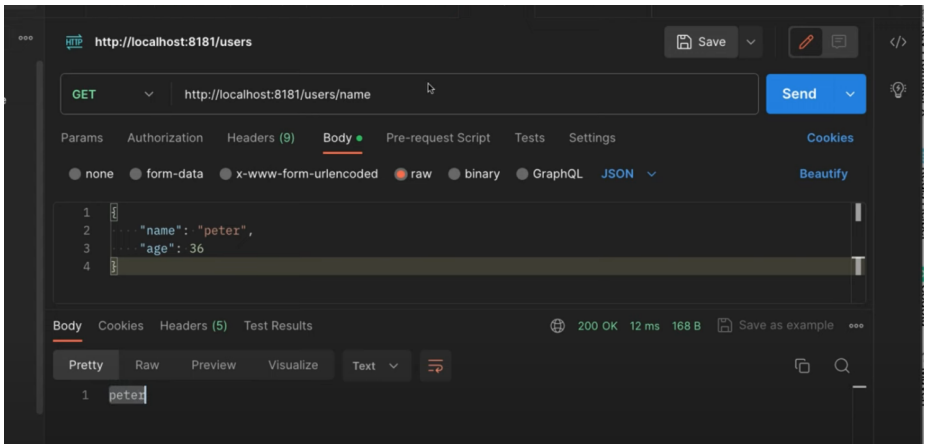
and put method should be idempotent -

Hit below endpoint n point of time but it should not increase record count in the db.



39. Can we pass Request Body in GET HTTP Method ?

We can pass technically but it is not recommended.



40. How can we perform content negotiation (XML/JSON) in Rest endpoint ?

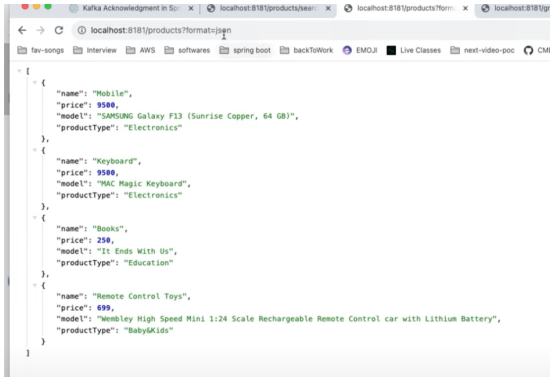
```
@GetMapping(produces = {"application/json","application/xml"})
public List<Product> products(@RequestParam(value = "productType", required = false) String produc
    return productType != null
            ? service.getProductByType(productType)
            : service.getProducts();
}
```

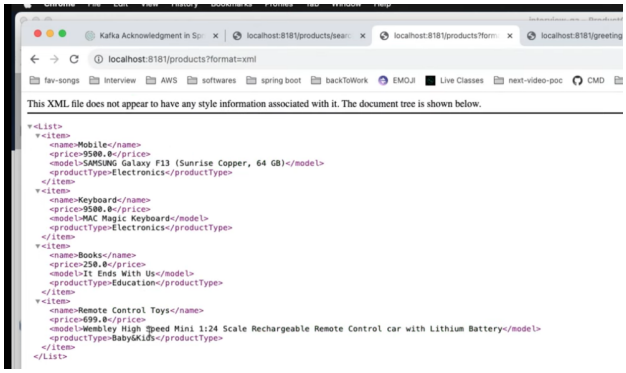
We need this one.

```
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-webflux</artifactId>
    </dependency>
    <dependency>
        <groupId>com.fasterxml.jackson.dataformat</groupId>
        <artifactId>jackson-dataformat-xml</artifactId>
        <version>2.13.0</version> <!-- Use the latest version available -->
    </dependency>

</dependencies>
```

```
# Enable content negotiation
spring.mvc.contentnegotiation.favor-path-extension=true
spring.mvc.contentnegotiation.favor-parameter=true
spring.mvc.contentnegotiation.defaultContentType=application/xml
```

By default it will give xml if we not pass content type



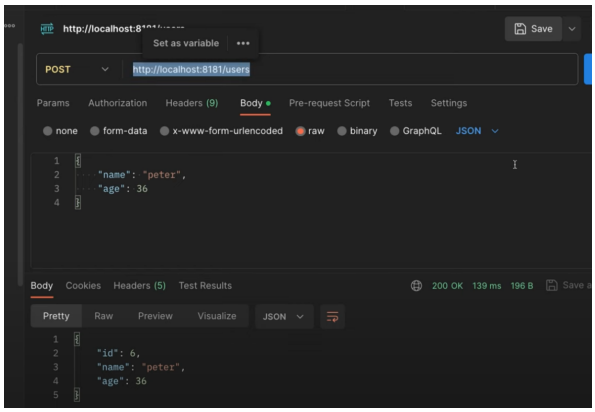


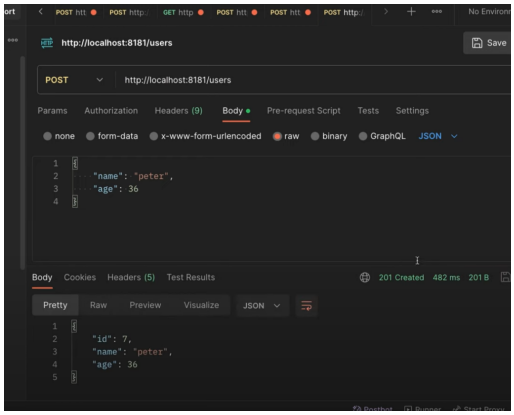
41. What all status code you have observed in your application ?

400 - bad request - some wrong input
404 - resource (url) not found
401 - authentication error
403 - authentication successful but not authorized that resource (url)
405 - method not allowed - get is accessed using post method like that
415 - unsupported media error (content type is not correct)
500 - Internal server error - server encountered unexpected condition that prevented fulfilling the request
502 - BadGateway. Accessing other endpoints which is unhealthy/down

42. How can you customize the status code for your endpoint ?

Previously the below post method returning 200 after change 201





Before and After.

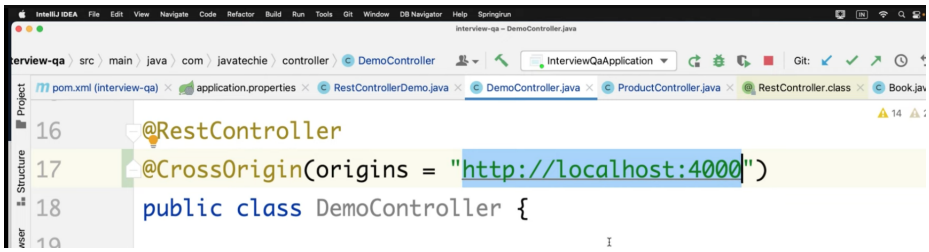
```
@PostMapping("/users") // not idempotent
@ResponseStatus(HttpStatus.CREATED)
public UserDetails addNewUser(@RequestBody UserI
    return repository.save(userDetails);
}
```

Similarly for put we can use no content

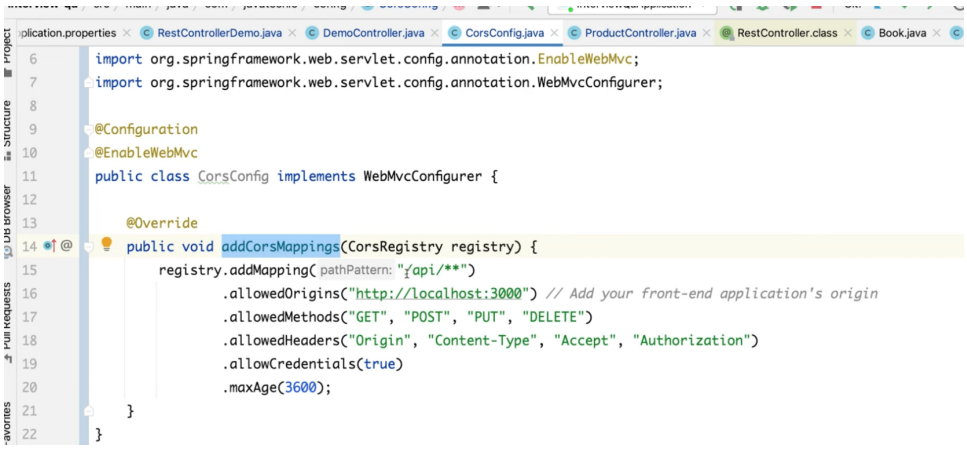
```
46      @PutMapping("/users/{id}") // idempotent
47      @ResponseStatus(HttpStatus.NO_CONTENT)
48  @   public UserDetails addAndUpdateUser(@PathVariable ir
49          UserDetails existingUserDetails = repository.finc
```

43. How can you enable cross origin ?

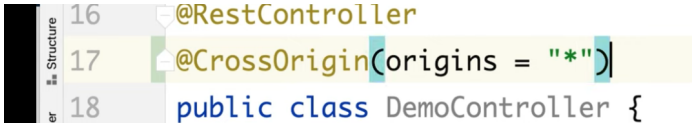
This will accept all the endpoints from the given origins only



The above one is annotation config the below one is java config

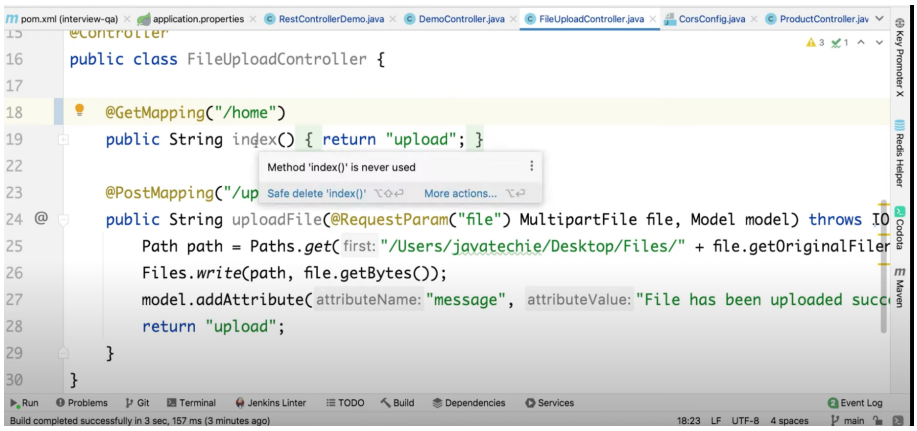


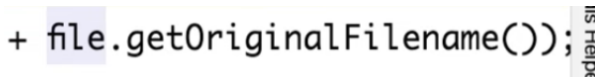
We can use wildcards as shown below

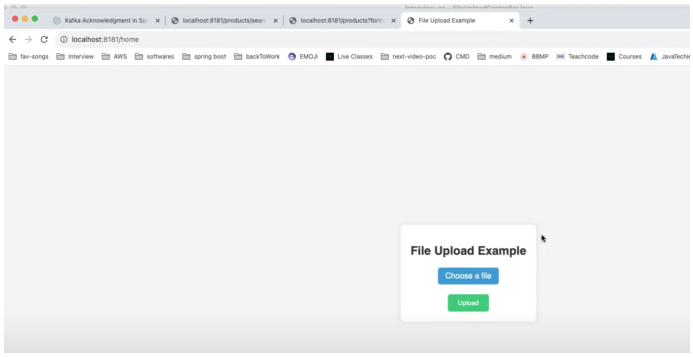


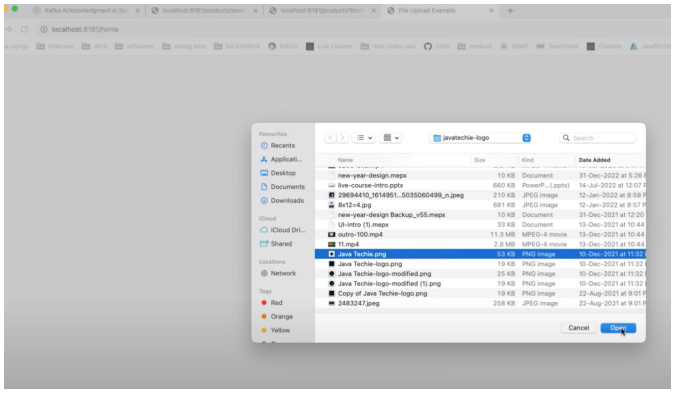

```
        .allowedOrigins("*") // Add your front-end application's origin
        .allowedMethods("GET", "POST", "PUT", "DELETE")
```

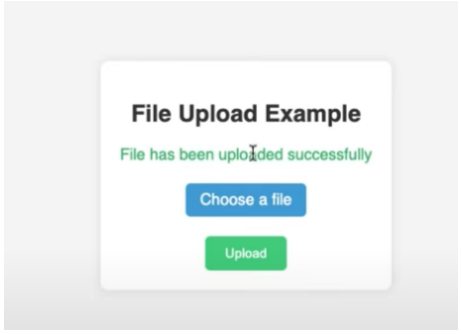
44. How can you upload a file in spring ?



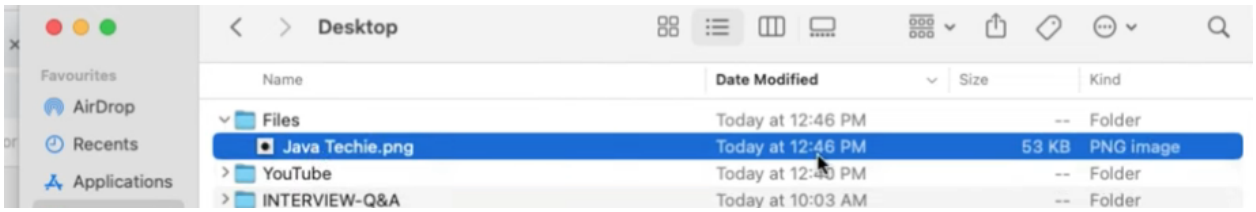








The location where the file is stored.



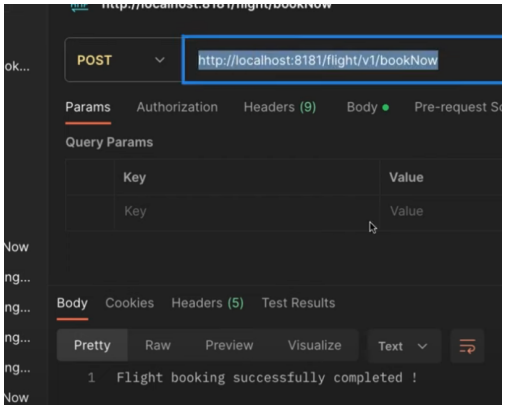
45. How do you maintain versioning for your REST API?

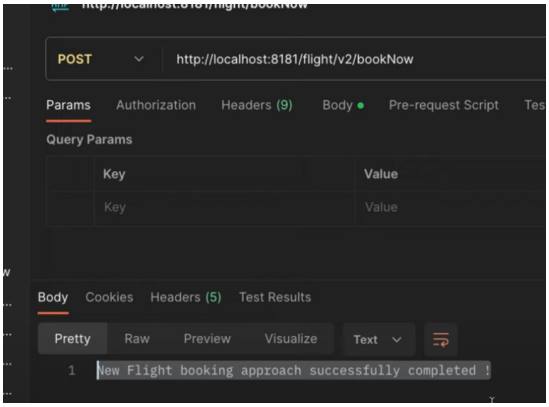
For backward compatibility and to add new features or changes we have to go for versioning our rest api.

1. Using versioning - like v1 and v2

FlightBookingController.java

```java
@RequestMapping("/flight")
public class FlightBookingController {


    @PostMapping("/v1/bookNow")
    public ResponseEntity<String> bookingTicket(@RequestBody Object request) {
        //execute actual logic
        return ResponseEntity.ok("Flight booking successfully completed !");
    }

    @PostMapping("/v2/bookNow")
    public ResponseEntity<String> bookingTicket2(@RequestBody Object request) {
        //latestLogic
        return ResponseEntity.ok("Flight booking successfully completed !");
    }

```

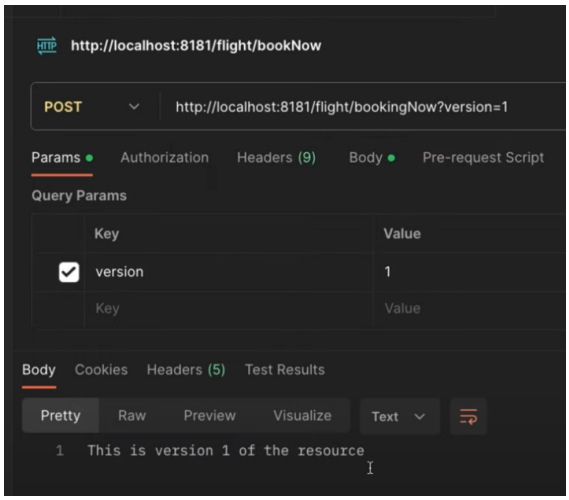


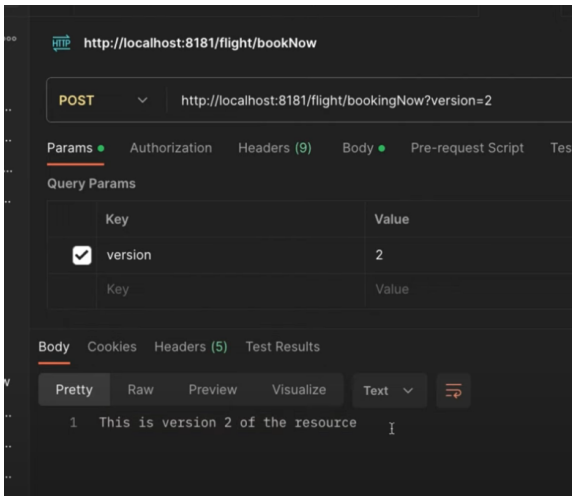


2. Using request params

```java
equestParam(@RequestBody Object request, @RequestParam(name = "version") int version) {
```

```
24
25      @PostMapping("/bookingNow")
26      public ResponseEntity<String> bookTicketVersionWithRequestParam(@RequestBody Object request,
27          //execute actual logic
28          if (version == 1) {
29              return ResponseEntity.ok("This is version 1 of the resource");
30          } else {
31              return ResponseEntity.ok("This is version 2 of the resource");
32          }
33      }
```

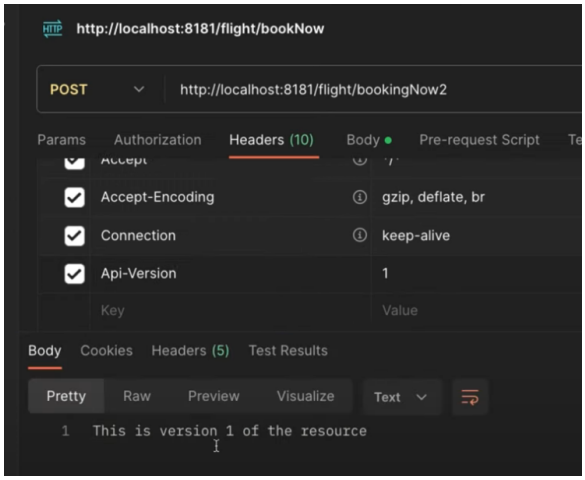


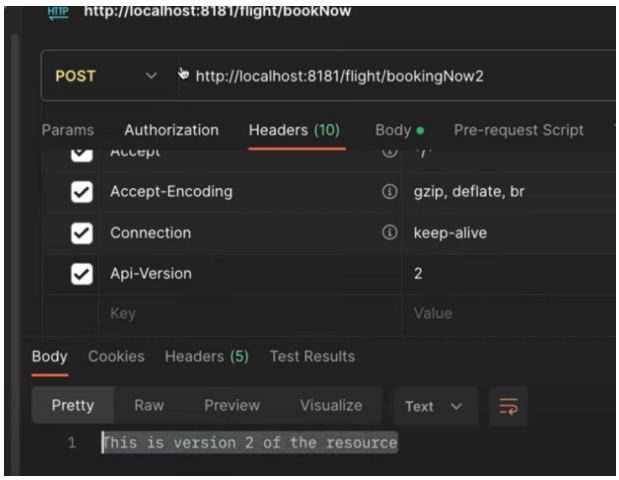


Using request header

```
35
36  /ersionWithHeaderParam(@RequestBody Object request, @RequestHeader(name = "Api-Version") int version) {
```

```
34
35      @PostMapping("/bookingNow2")
36      public ResponseEntity<String> bookTicketVersionWithHeaderParam(@RequestBody Ob
37          //execute actual logic
38          if (version == 1) {
39              return ResponseEntity.ok("This is version 1 of the resource");
40          } else {
41              return ResponseEntity.ok("This is version 2 of the resource");
42          }
43      }
44  }
```



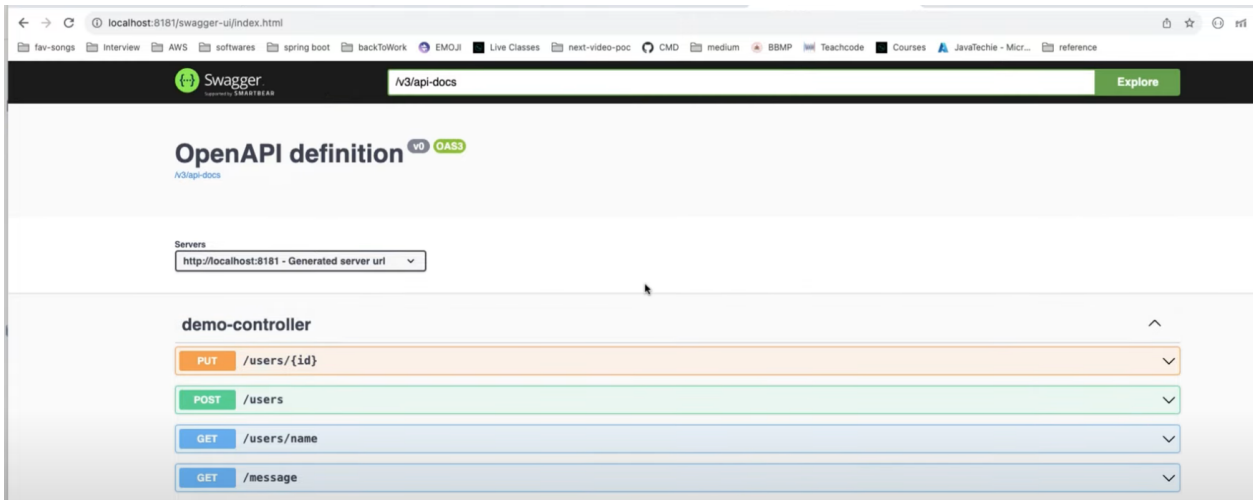


46. How will you document your rest API ?

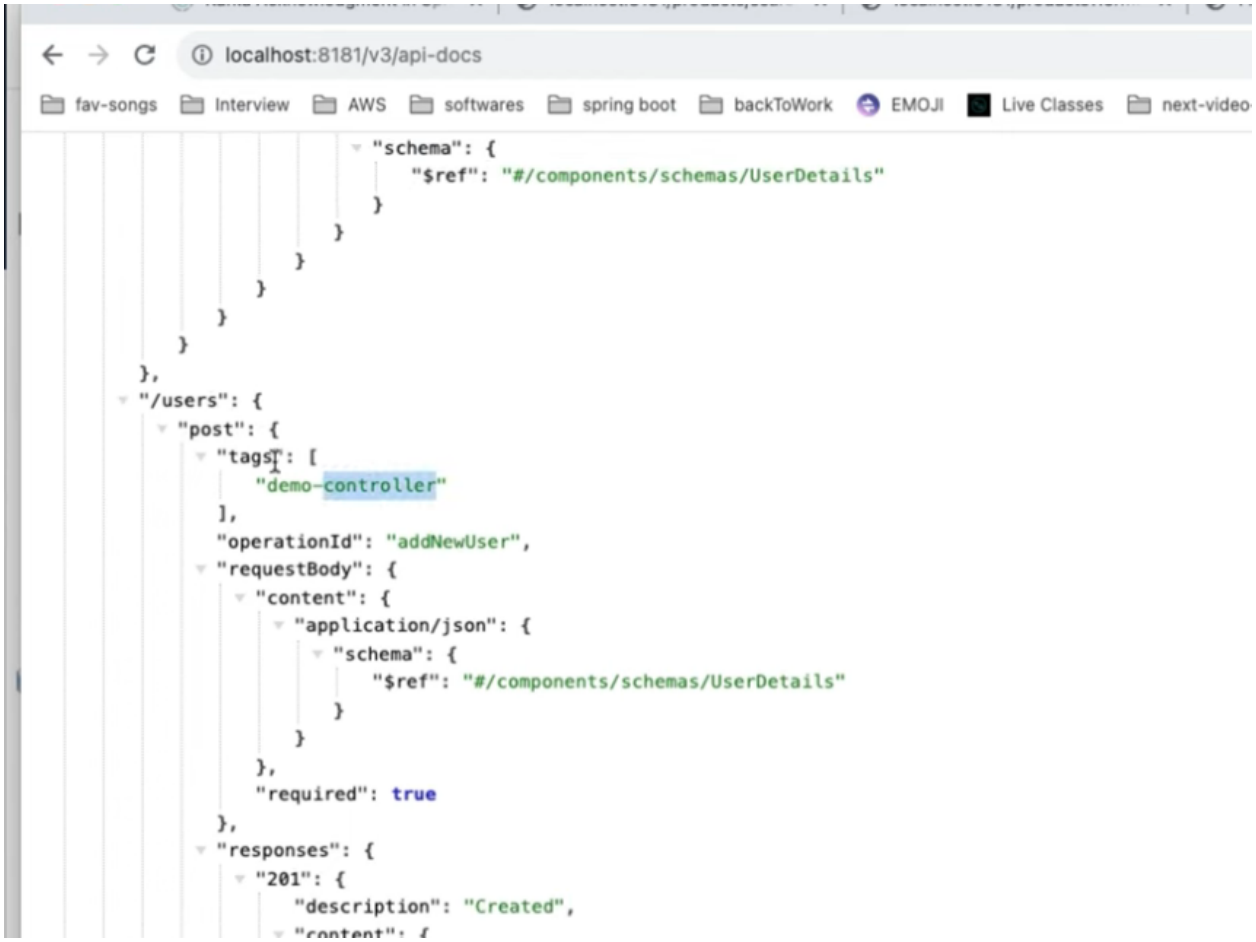
Previously it was swagger

```
        </dependency>
        <dependency>
            <groupId>org.springdoc</groupId>
            <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
            <version>2.0.4</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
```

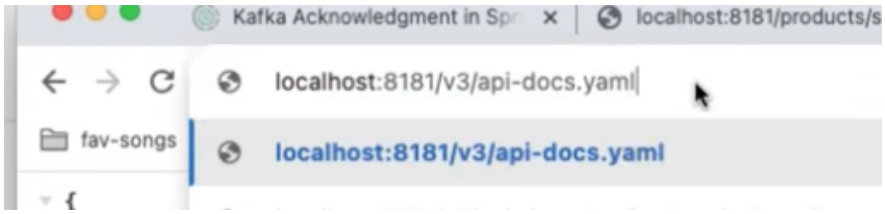
API Documents



API Description:



We can download this api description using the below url.



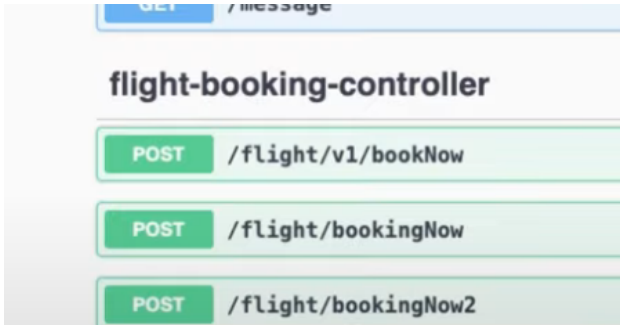
47. How can you hide certain REST endpoints to prevent them from being exposed externally?

Use @Hidden annotation as shown below. This will not be displayed in swagger.

```java
@PostMapping("/v1/bookNow")
public ResponseEntity<String> bookingTicket(@RequestBody Object request) {
    //execute actual logic
    return ResponseEntity.ok("Flight booking successfully completed !");
}

@PostMapping("/v2/bookNow")
@Hidden
public ResponseEntity<String> bookingTicket2(@RequestBody Object request) {
    //latestLogic
```

Run: InterviewQaApplication

We are seeing V1 but not V2.



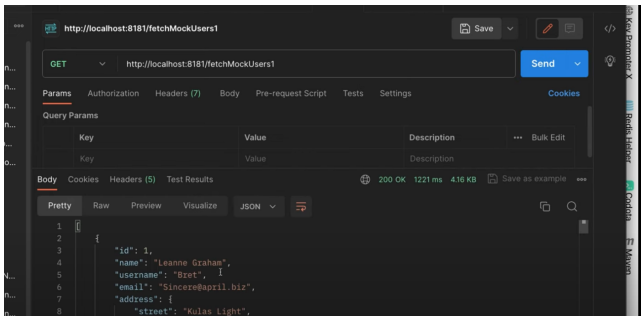
48. How will you consume restful API ?

1. RestTemplate
2. FeignClient
3. WebClient
4. Advance Rest Client

Create a rest template bean and autowire it.

```java
@Autowired
private RestTemplate template;
```

```
26
27      @GetMapping("/fetchMockUsers1")
28      public List<UserResponse> fetchMockUsersWithRestTemplate() {
29          return template.getForObject( url: "https://jsonplaceholder.typicode.com/users", List.class);
30      }
31
```

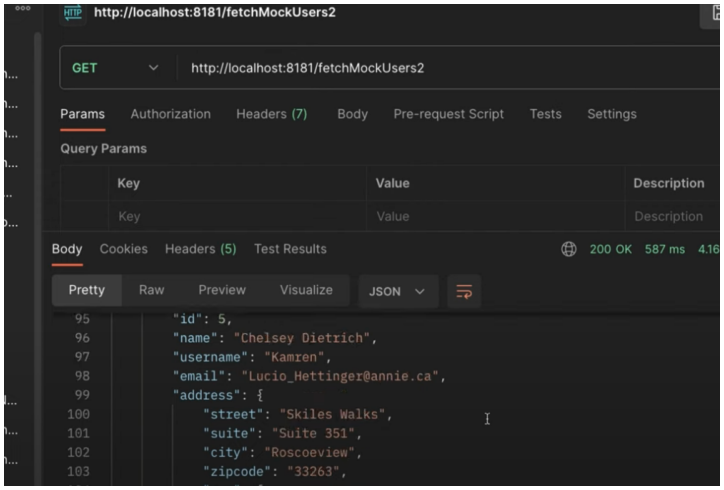


Feign Client -> proxy as interface

```
8
9   @FeignClient(url="https://jsonplaceholder.typicode.com",name = "USER-CLIENT")
10  public interface UserClient {
11
12      @GetMapping("/users")
13      public List<UserResponse> getUsers();
14  }
15
```

In controller
Autowire the above interface in the controller and use it like below.

```
32
33      @GetMapping("/fetchMockUsers2")
34      public List<UserResponse> fetchMockUsersWithFeignClient() {
35          return userClient.getUsers();
36      }
37
```



Webclient - unblocking and Asynchronous code.