**Versioning APIs in Spring Boot**

Common versioning techniques: **URI versioning**,**request parameter versioning**, **custom header versioning**, and **content negotiation (accept header) versioning**.

**URI Versioning**

URI versioning involves adding the version number to the API’s URL. This is a straightforward and easily understandable method. However, it may not be the most elegant, as it can lead to lengthy URLs.

**Example:**

@RestController  
@RequestMapping("/api/v1/users")  
public class UserControllerV1 {  
  
 @GetMapping  
 public ResponseEntity<List<User>> getUsers() {  
 // Implementation here  
 }  
  
 // Other endpoints  
}

**Step-by-Step Breakdown**

1. **Annotation @RestController:** This annotation is used to define a controller in Spring Boot that handles HTTP requests. It combines @Controller and @ResponseBody, which means the return value is directly bound to the web response body.
2. **Annotation @RequestMapping("/api/v1/users"):** This specifies that all HTTP requests to /api/v1/users will be handled by this controller. Here, v1 in the URL explicitly indicates that this is version 1 of the API.
3. **Method getUsers():** This method is mapped to handle GET requests to the controller's base URL. It returns a ResponseEntity object containing a list of User objects, which represents the HTTP response including the status code, headers, and body.

**Request Parameter Versioning**

With this approach, the version number is passed as a request parameter. It’s less intrusive than URI versioning but might not be as intuitive for consumers.

**Example:**

@RestController  
public class UserController {  
  
 @GetMapping(value = "/api/users", params = "version=1")  
 public ResponseEntity<List<User>> getUsersV1() {  
 // Implementation here  
 }  
  
 @GetMapping(value = "/api/users", params = "version=2")  
 public ResponseEntity<List<User>> getUsersV2() {  
 // Implementation here  
 }  
}

**Step-by-Step Breakdown**

1. **No Version in URL:**Unlike URI versioning, the URL /api/users remains the same. The versioning is handled by a request parameter.
2. **Request Parameter:**The params attribute in the @GetMapping annotation is used to differentiate the API versions. For version=1, the getUsersV1 method will be called, and for version=2, the getUsersV2 method will be triggered.
3. **Flexible URL:**This method allows the URL to remain unchanged, offering flexibility and simplicity in API routing, though it requires clients to be aware of the version parameter.

**Custom Header Versioning**

Custom header versioning involves adding a custom header to the HTTP request, containing the version number. This method keeps the URI clean, but consumers must remember to include the custom header.

**Example:**

@RestController  
public class UserController {  
  
 @GetMapping(value = "/api/users", headers = "X-API-Version=1")  
 public ResponseEntity<List<User>> getUsersV1() {  
 // Implementation here  
 }  
  
 @GetMapping(value = "/api/users", headers = "X-API-Version=2")  
 public ResponseEntity<List<User>> getUsersV2() {  
 // Implementation here  
 }  
}

**Step-by-Step Breakdown**

1. **Headers for Versioning:** This technique utilizes the HTTP headers for version control. The headers attribute in @GetMapping specifies that this endpoint will only handle requests containing a header X-API-Version=1 or 2.
2. **Separation of Concerns:** Keeping the URI clean by not embedding version information in it, this method instead relies on clients sending the correct header.
3. **Client Responsibility:** Clients must remember to set the correct header (X-API-Version) to use the desired version of the API.

**Content Negotiation (Accept Header) Versioning**

Accept header versioning leverages the HTTP Accept header to specify the desired API version. This method adheres to the HATEOAS principle, as the version is part of the content negotiation.

**Example:**

@RestController  
public class UserController {  
  
 @GetMapping(value = "/api/users", produces = "application/vnd.example.api.v1+json")  
 public ResponseEntity<List<User>> getUsersV1() {  
 // Implementation here  
 }  
  
 @GetMapping(value = "/api/users", produces = "application/vnd.example.api.v2+json")  
 public ResponseEntity<List<User>> getUsersV2() {  
 // Implementation here  
 }  
}

**Step-by-Step Breakdown**

1. **Use of produces:** This attribute specifies the media type that the methods of the controller can produce. Here, it is used to differentiate the API versions based on the content type requested by the Accept header.
2. **Media Type Versioning:** Each method supports a different custom media type indicating the version (v1 or v2).
3. **Adherence to HATEOAS:** This approach follows the HATEOAS (Hypermedia as the Engine of Application State) principle by making the version an integral part of the media type, improving the self-descriptiveness of the response.