Versioning REST APIs (Spring Boot REST APIs)

By Ramesh Fadatare (Java Guides)

Versioning REST APIs - Spring Boot REST API

- API versioning is the practice of transparently managing changes to your API.
- We will look at 4 ways of versioning a REST API.

1. Versioning through URI Path

2. Versioning through query parameters

3. Versioning through custom headers

4. Versioning through content negotiation

Why do we need to version our RESTful API?

- The best approach to versioning is NOT to do it. Yeah, that’s right. Do not version as long as versioning is not actually needed.
- **Build your services to backward compatible so that you can avoid versioning as much as possible!**
- One of the major challenges surrounding exposing services is handling updates to the API contract. **Clients may not want to update their applications when the API changes, so a versioning strategy becomes crucial.** A versioning strategy allows clients to continue using the existing REST API and migrate their applications to the newer API when they are ready.

When to version REST API

APIs only need to be up-versioned when a breaking change is made. Breaking changes include:

- Changing the request/response format (e.g. from XML to JSON)
- Changing a property name (e.g. from name to productName) or data type on a property (e.g. from an integer to a float)
- Adding a required field on the request (e.g. a new required header or property in a request body)
- Removing a property on the response (e.g. removing description from a product)

1. Versioning through URI Path

One way to version a REST API is to include the version number in the URI path.

http://www.example.com/api/1/products

http://www.example.com/api/v1/products

http://www.example.com/api/v2/products

http://www.example.com/api/v1/posts

http://www.example.com/api/v1/employees

In this approach we create a completely different URI for the new service.

http://www.example.com/api/v1/products

http://www.example.com/api/v2/products

Real-world examples:

Twitter, Pay Pal, Google etc

2. Versioning through query parameters

Another option for versioning a REST API is to include the version number as a query parameter.

Examples:

http://www.example.com/api/products?version=1

http://www.example.com/api/products?version=2

http://www.example.com/api/posts?version=1

http://www.example.com/api/employees?version=1

Real-world examples:

Google translation APIs - https://cloud.google.com/translate/docs/reference/rest

3. Versioning through custom headers

REST APIs can also be versioned by providing custom headers with the version number included as an attribute. The main difference between this approach and the two previous ones is that it doesn’t clutter the URI with versioning information.

Examples:

http://localhost:8080/api/products

headers=[X-API-VERSION=1]

http://localhost:8080/api/products

headers=[X-API-VERSION=2]

Pros: It doesn’t clutter the URI with versioning information

Cons: It requires custom headers

4. Versioning through content negotiation

The last strategy we are addressing is versioning through content negotiation.

In this approach, we use the Accept Header in the request.

Examples:

http://localhost:8080/api/products

headers[Accept=application/vnd.javaguides-v1+json]

http://localhost:8080/api/products

headers[Accept=application/vnd.javaguides-v2+json]

Real-world examples:

GitHub - https://docs.github.com/en/rest/overview/resources-in-the-rest-api

application/vnd.github.v3+json