Designing API Versions

Introduction

Why You Should Version APIs?

Types of APIs

Public

Partner

Private

Types of APIs

Public
(Versioning MUST)

Partner
(Versioning MUST)

Private
(Versioning OPTIONAL)

API versioning should be planned & designed from the beginning

Enhancing existing APIs to support new features

Ways to Version APIs

Ways to Version APIs

No versioning

URL versioning

Query String Versioning

Header versioning

Media type versioning

Know the pros, cons, and ideal use case of each versioning type

Approach 1: No Versioning

No Versioning

Change the existing API directly

No Versioning - Example

Existing API:

GET /api/colleges/10

```
{
        "id": xxx,
        "name": xxx
}
```

Changed API:

GET /api/colleges/10

```
{
        "id": xxx,
        "name": xxx,
        "address": xxx
}
```

No Versioning

Change the existing API directly

There may be breaking changes

No Versioning – Breaking Change Example

Existing API:

GET /api/colleges/10

```
{
        "id": xxx,
        "students": [ xxx, xxx, xxx ]
}
```

Changed API:

GET /api/colleges/10

```
{
        "id": xxx
}
```

GET /api/colleges/10/students

```
{
        [xxx, xxx, xxx]
}
```

Pros & Cons of No Versioning

No additional development effort required.

Good for small teams.

No way of tracking changes.

Can break existing applications.

No Versioning

Change the existing API directly

There may be breaking changes

Ideal for Private APIs

Approach 2: URL Versioning

URL Versioning

Add the version as part of URL

URL Versioning - Example

No Versioning:

/api/colleges/10

URL Versioning:

/api/v1/colleges/10

/api/v2/colleges/10

URL Versioning - Example

Existing API:

GET /api/v1/colleges/10

```
{
        "id": xxx,
        "students": [ xxx, xxx, xxx ]
}
```

Changed API:

GET /api/v2/colleges/10

```
{
        "id": xxx
}
```

GET /api/v2/colleges/10/students

```
{
        [xxx, xxx, xxx]
}
```

URL Versioning

Add the version as part of URL

Need to support existing versions

URL Versioning - Example

Existing API:

GET /api/v1/colleges/10

```
{
        "id": xxx,
        "students": [ xxx, xxx, xxx ]
}
```

Changed API:

GET /api/v2/colleges/10

```
{
        "id": xxx
}
```

GET /api/v2/colleges/10/students

```
{
        [xxx, xxx, xxx]
}
```

URL Versioning

Add the version as part of URL

Need to support existing versions

There may be breaking changes

Pros & Cons of URL Versioning

Simple to use.

Very clear in usage.

Need to change URL every time.

Difficult to handle URL linking in responses.

URL Versioning

Add the version as part of URL

Need to support existing versions

There may be breaking changes

Ideal for simple APIs

URL Versioning - One of the popular versioning used in production

Approach 3: Query String Versioning

Query String Versioning

Add the version as query string to request

Query String Versioning - Example

No Versioning:

/api/colleges/10

Query String Versioning:

/api/colleges/10?version=1

/api/colleges/10?version=2

Query String Versioning - Example

Existing API:

GET /api/colleges/10?version=1

```
{
        "id": xxx,
        "students": [ xxx, xxx, xxx ]
}
```

Changed API:

GET /api/colleges/10?version=2

```
{
        "id": xxx
}
```

GET /api/colleges/10/students?version=2

```
{
        [xxx, xxx, xxx]
}
```

Query String Versioning

Add the version as query string to request

Query string to have default value

Query String Versioning - Example

With Version:

GET /api/colleges/10?version=1

```
{
        "id": xxx,
        "students": [ xxx, xxx, xxx ]
}
```

Without Version:

GET /api/colleges/10

```
{
        "id": xxx
        "students": [ xxx, xxx, xxx ]
}
```

Pros & Cons of Query String Versioning

Existing integrations aren’t affected.

URL structure remains same.

Client shouldn’t forget to pass the query string.

Difficult to handle URL linking in responses.

Query String Versioning

Add the version as query string to request

Query string to have default value

Used for any use cases

Approach 4: Header Versioning

Header Versioning

Custom header added to indicate version

Header Versioning - Example

No Versioning:

/api/colleges/10

Header Versioning:

Header:

X-API-Version=2

Request:

/api/colleges/10

Header Versioning - Example

Existing API:

Headers: X-API-Version=1

GET /api/colleges/10

```
{
        "id": xxx,
        "students": [ xxx, xxx, xxx ]
}
```

Changed API:

Headers: X-API-Version=2

GET /api/colleges/10 { … }

Headers: X-API-Version=2

GET /api/colleges/10/students { … }

Header Versioning

Custom header added to indicate version

Consider default version if not passed

Header Versioning: Default Value

Explicit Header Version:

Headers: X-API-Version=2

GET /api/colleges/10/students

{ … }

Default Header Version Value:

GET /api/colleges/10

{ … }

Version=1

Pros & Cons of Header Versioning

Existing integrations aren’t affected.

URL structure remains same.

Versioning logic is separated from API.

Client shouldn’t forget to pass the header.

Difficult to handle URL linking in responses.

Header Versioning

Custom header added to indicate version

Consider default version if not passed

Ideal if clients are experienced

Approach 5: Media Type Versioning

Media Type Versioning

Include the version along with the Media Type header

Media Type Versioning - Example

GET /api/colleges/10

Request Header:

Content-Type: application/vnd.cms.v1+json

Response Body:

```
{
        "id": xxx,
        "students": [ xxx, xxx, xxx ]
}
```

Media Type Versioning - Example

Existing API:

Content-Type: application/vnd.cms.v1+json

GET /api/colleges/10

```
{
        "id": xxx,
        "students": [ xxx, xxx, xxx ]
}
```

Changed API:

Content-Type: application/vnd.cms.v2+json

GET /api/colleges/10 { … }

Content-Type: application/vnd.cms.v2+json

GET /api/colleges/10/students { … }

Media Type Versioning

Include the version along with the Media Type header

Ideal if response consists of URL linking

Pros & Cons of Media Type Versioning

Ideal form of API versioning.

Easier to handle URL linking in responses.

Complex compared to other versioning approaches.

Not cache-friendly.

STEP 21: Identify the API Versioning Scheme to Use

Which Versioning Approach to Use?

Ways to Version APIs

No versioning

URL versioning

Query String Versioning

Header versioning

Media type versioning

Selecting the Versioning Approach

Choose as per your requirement

Consider the client experience

Consider the performance

RECOMMENDED APPROACH:
URL or Query String Versioning

STEP 22: Set the API Version for the Current Version

Designing API Versions

Summary