
Web API 設計のベストプラクティス

Web API 設計のベストプラクティス

[Enchant](https://www.enchant.com/) の開発者、

Vinay Sahni さんが書いた記事。

2015 年の記事なのでちょっと古めですが、

明確なベストプラクティスがない API 設計において一度は読んでおきたい素晴らしい記事。

RESTful な URL にする

- CRUD 操作

```
GET /tickets # チケットの一覧を取得
GET /tickets/12 # 指定したチケットを取得
POST /tickets # チケットを作成
PUT /tickets/12 # 指定したチケットを更新する
DELETE /tickets/12 # 指定したチケットを削除する
```

- 関連モデルを操作

```
GET /tickets/12/messages
```

RESTful な URL にする

- CRUD 操作にマッチしない操作
  - 例1: リソースに対してアクティベーションさせる操作

```
PUT /gists/:id/star
DELETE /gists/:id/star
```

  - 例2: 複数のリソースを横断するようなアクション

```
GET /serach
```

SSL 通信を用いる

- NG

```
http://hoge
```

- OK

```
https://hoge
```

API 仕様書を作る

著名な API の仕様書

- Github API
- Stripe

良い API 仕様書のポイント

- 誰でもアクセスできるところにある。
- リクエストとレスポンスの例が記されている。
- リクエストはコピーできるようになっている。
- API の破壊的な変更のスケジュールなどをお知らせできる。

バージョンは URL に含める

- NG
  - バージョン指定をヘッダに含める

```
# endpoint
/users

# Request Header
Accept: application/json; version=v1
```

- OK
  - バージョン指定を URL に含める

```
# endpoint
/v1/users
```

リクエストパラメータを活用する

フィルタ

```
GET /tickets?state=open
```

ソート

```
GET /tickets?sort=-priority,created_at
```

検索

```
GET /tickets?q=return
```

よく使うパラメータがあるならエイリアスを作る

```
GET /tickets/recently_closed
```

レスポンスのフィールドを絞れるようにする

```
GET /tickets

# Got response
{
  id: 1,
  subject: "Sample Title",
  coutomer_name: "John Doe",
  created_at: "2022-04-29",
  updated_at: "2022-04-30",
  ...
}

GET /tickets?fields=id,subject

# Got resoponse
{
  id: 1,
  subject: "Sample Title"
}
```

作成/更新後はリソースをフルで返す

```
POST /tickets

# Got response
{
  id: 2,
  subject: "Created ticket",
  created_at: "2022-04-30",
  updated_at: "2022-04-30
}
```

フィールドの命名規則を考える

API は **スネークケース** を使う。

JSON はデフォルトで整形する

- NG

```
{id:1,subject:"Sample Title",coutomer_name:"John Doe",created_at:"2022-04-29",updated_at:"2022-04-30"}
```

- OK

```
{
  id: 1,
  subject: "Sample Title",
  coutomer_name: "John Doe",
  created_at: "2022-04-29",
  updated_at: "2022-04-30"
}
```

要素はラップしない

- NG

```
{
  "data" : {
    "id" : 123,
    "name" : "John"
  }
}
```

- OK

```
{
  "id" : 123,
  "name" : "John"
}
```

追加/更新のリクエストボディも JSON を使う

- NG

```
POST /tickets
Content-Type: application/x-www-form-urlencoded
subject=sample&customer_name=john
```

- OK

```
POST /tickets
Content-Type: application/json
{
  subject: "sample",
  customer_name: "john"
}
```

ページング情報はレスポンスヘッダに入れる

- NG

```
{
  data: {
    tickets: [
      {
        id: 1,
        subject: "sample"
      }
    ]
  },
  page: 2,
  per_page: 100,
  total_count: 5000
}
```

ページング情報はレスポンスヘッダに入れよう

- OK

```
Link: <https://tickets?page=3&per_page=100>; rel="next", <https://tickets?page=50&per_page=100>; rel="last"
{
  tickets: [
    {
      id: 1,
      subject: "sample"
    }
  ]
}
```

関連データを埋め込む手段を作る

```
GET /tickets/12?enbed=customer.name,assign_user

# Got response
{
  id: 12,
  subject: "sample",
  customer: {
    name: "John"
  },
  assign_user: {
    id: 1,
    name: "Jane",
    ...
  }
}
```

HTTP メソッドを上書き可能にする

- 更新

```
POST /tickets/12
X-HTTP-Method-Override: PUT
```

- 削除

```
POST /tickets/12
X-HTTP-Method-Override: DELETE
```

リクエスト制限情報をレスポンスヘッダに入れる

- Twitter の API で採用されている制限情報

```
X-Rate-Limit-Limit: 100 # 一定期間内にリクエストできる回数
X-Rate-Limit-Remaining: 99 # 次の期間までにリクエストできる残り回数
X-Rate-Limit-Remaining: 600 # 次の期間が来るまでの秒数
```

トークン認証を使う

- Basic 認証

```
Authorization: Basic dXNlcjpwYXNzd29yZA==
```

キャッシュの情報をレスポンスヘッダに入れる

レスポンスヘッダにリソース情報を変換したものを含めることで変更を検知する。

- ETag を用いる方法
  - リソースのハッシュか、チェックサムを含める

```
ETag: "qa3311fa"
```

- Last-Modified を用いる方法
  - リソースの最終更新日時を含める

```
Last-Modified: Sat, 30 Apr 2022 03:21:05 GMT
```

エラーメッセージはちゃんと返す

- NG

```
HTTP/1.1 422 Unprocessable entry
{}
```

エラーメッセージはちゃんと返す

- OK

```
HTTP/1.1 422 Unprocessable entry
{
  "code" : 1024,
  "message" : "Validation Failed",
  "errors" : [
    {
      "code" : 5432,
      "field" : "first_name",
      "message" : "First name cannot have fancy characters"
    }
  ]
}
```

- 良いエラーメッセージのポイント
  - ユニークなエラーコード
  - 原因の説明
  - (バリデーションエラーであれば)問題のあるフィールドとその理由

HTTP ステータスコードを活用する

- HTTP レスポンスステータスコード
  - https://developer.mozilla.org/ja/docs/Web/HTTP/Status

実際にやってみた

- Github
  - https://github.com/Horse-race-common-info-api/api