JSON API Do's & Don'ts

Jan Varljen, Productive CTO

01

JSON API

json:api is a specification for building APIs in JSON

HTTP://JSONAPI.ORG/

CONCEPTS

1. Document structure
2. Fetching
3. CRUD
4. Errors
5. Do’s
6. Don’ts

02

DOCUMENT STRUCTURE

JSON STRUCTURE

- data
- included
- links
- meta

EXAMPLE

http://api.productive.io.dev/api/v2/1/projects?token=abc

RESOURCE

- id
- type
- attributes
- relationships

EXAMPLE

http://api.productive.io.dev/api/v2/1/projects/363?token=abc

RELATIONSHIPS

- data
- link

EXAMPLE

```
// ...
{
  "type": "articles",
  "id": "1",
  "attributes": {
    "title": "Rails is Omakase"
  },
  "relationships": {
    "author": {
      "links": {
        "self": "http://example.com/articles/1/relationships/author",
        "related": "http://example.com/articles/1/author"
      },
      "data": { "type": "people", "id": "9" }
    }
  },
  "links": {
    "self": "http://example.com/articles/1"
  }
}
// ...
```

INCLUDED

- related resources included inside the same response

EXAMPLE

http://api.productive.io.dev/api/v2/1/projects/363?token=abc

LINKS & META

- pagination links
- pagination metadata
- any other provisional metadata

EXAMPLE

http://api.productive.io.dev/api/v2/1/projects?token=abc

03

FETCHING

INCLUDING RELATED RESOURCES

- API may return related resources by default
- API may support the `include` request parameter

INCLUDE PARAMETER

- GET /projects/1?include=company
- GET /projects/1?include=project_manager
- GET /projects/1?include=company,project_manager
- GET /projects/1?include=project_manager.company

FILTERING RESOURCES

- use the `filter` query parameter for filtering resources

- JSON API is agnostic about the strategies supported by a server

FILTER PARAMETER

- GET /projects?filter[company_id]=41

- http://api.productive.io.dev/api/v2/1/projects?filter%5Bcompany_id%5D=41&token=abc

FRACTIONAL ATTRIBUTES

- API may return only specified attributes in the response
- use the `fields` query parameter for filtering attributes

FIELDS PARAMETER

- GET /projects/1?fields[projects]=name
- GET /projects/1?fields[projects]=name,currency
- GET /projects/1?fields[projects]=name,currency&fields[company]=name

SORTING RESOURCES

- API may support sorting resources
- use the `sort` query parameter for sorting

SORT PARAMETER

- GET /projects?sort=name
- GET /projects?sort=profit,name
- GET /projects?sort=-profit,name

PAGINATION

- API endpoint may be paginated
- pagination links are in the `links` part of the response
- http://api.productive.io.dev/api/v2/1/projects?token=abc

C(R)UD

CREATE

- POST /projects
- JSON:API request body
- 201 Created/202 Accepted/204 No content

EXAMPLE

```
{
    "data": {
        "type": "projects",
        "attributes": {
            "name": "project name"
        },
        "relationships": {
            "company": {
                "data": {
                    "type": "companies",
                    "id": "1"
                }
            },
            "project_manager": {
                "data": {
                    "type": "people",
                    "id": "1"
                }
            }
        }
    }
}
```

UPDATE

- PATCH /projects/1
- JSON:API request body
- partial updates
- 200 OK/202 Accepted/204 No content

EXAMPLE


```
{
    "data": {
        "type": "projects",
        "id": "1",
        "attributes": {
            "name": "new name"
        },
        "relationships": {
            "company": {
                "data": {
                    "type": "companies",
                    "id": "2"
                }
            }
        }
    }
}
```

DELETE

- DELETE /projects/1
- no request body
- 200 OK/202 Accepted/204 No content

05

ERRORS

ERRORS STRUCTURE

- errors (array)

ERROR STRUCTURE

- status
- title
- details
- source
- meta

EXAMPLE

http://api.productive.io.dev/api/v2/1/projects/363

http://api.productive.io.dev/api/v2/1/projects/99999?token=abc

EXAMPLE


```
{
  "errors": [
    {
      "status": "422",
      "title": "Invalid Attribute",
      "detail": "can't be blank",
      "source": {
        "pointer": "data/attributes/name"
      }
    },
    {
      "status": "422",
      "title": "Invalid Attribute",
      "detail": "can't be blank",
      "source": {
        "pointer": "data/attributes/project_manager"
      }
    },
    {
      "status": "422",
      "title": "Invalid Attribute",
      "detail": "can't be blank",
      "source": {
        "pointer": "data/attributes/company"
      }
    }
  ]
}
```

06

DO’S

FLAT STRUCTURE

- /projects/363/comments vs /comments?filter[project_id]=363

- every resource on own endpoint:
  - people/1/comments
  - companies/1/comments
  - invoices/1/comments
  - ....
  - becomes /comments?filter[x]

FLAT STRUCTURE

- less code duplications

- you need /comments route either way:
  - PATCH /comments/1
  - DELETE /comments/1

- more flexible:
  - /comments?filter[project_id][]=363&filter[project_id][]=364

PAGINATE EVERY ENDPOINT

- /projects
  - returns 10 projects

- /projects
  - returns 1000 projects
  - slow, blocking

- always paginate every endpoint

MAX PER PAGE

- /projects?page[size]=1_000_000
  - boom!

- always have a max_per_page defined
  - max_per_page = 100

- add info to meta

- have a system to override it by resource
  - /tags → max_per_page=300
  - /projects → max_per_page=50

POST CREATE RESPONSE

- return 201 instead of 204 on POST create

- 201 Created
  - resource included in the response

- 204 No content
  - response empty
  - consider the resource sent in the request is accepted by the server
  - almost never identical
  - think about timestamp columns like (created_at)

- always return 201 if possible

CUSTOM ACTIONS

- archive project
  - project.archived_at = 2017-04-18 10:25:49

- this should be set by the server not client

- PATCH with archived_at: {time_on_client} is wrong

- PATCH /projects/:id/archive
  - server sets timestamps and returns 200 OK with updated resource

EXTRACT RESOURCES

- update user password
  - password_reset_token
  - password_token_sent_at
  - password_token_valid_for
  - ...

- usually columns in the users table

- PATCH /users/1
  - bunch of “magic” attributes pollute the user resource
  - used only for password reset (rarely)

- API resources != database structure

EXTRACT RESOURCES

- password becomes first-class citizen → independent resource

- CREATE /passwords
  - create new reset password token

- GET /passwords/:token
  - get a password reset token

- PATCH /passwords/:token
  - make the password change (if token still valid)

CACHING

- cache every resource individually
- cache key example:
  - /{resource_name}/{object_key}/{serializer_file_digest}
- /projects/363-20170413144717638000000/7c4eba8cd5a6cd89d6e07ce1a51cc23f
- http://api.productive.io.dev/api/v2/1/projects/363?token=abc

CACHE READ MULTI

- http://api.productive.io.dev/api/v2/1/projects?token=abc
- 30 projects + 17 included resources → 47 cache store hits
- JSON API is agnostic about caching implementation
- write you're own caching layer
  - use read multi from cache (ideally only 1 cache request)
  - solve application specific problems (Productive has roles)

NANOSECONDS FOR CACHE

- many frameworks use timestamp in seconds for cache
  - “2017-04-18 10:25:26”
- multiple updates in single second – happens all the time
  - invalid responses
- use milliseconds or nanoseconds for cache

SEPARATION OF CONCERNS

- ProjectsController
  - handles authentication and authorization
- ProjectFilter
  - handles `filter` and `sort` param and filtering
- ProjectSerializer
  - handles `fields` param and fractional attributes
- ProjectPreloader
  - handles `includes` param and preloading relationships
- ProjectForm
  - handles validations and saving resources
- ...

07

DON’TS

NULLS ON PARTIAL UPDATES

- PATCH request can update a single attribute
- missing attributes and relations should not be treated as NULL

```
{
    "data": {
        "type": "projects",
        "id": "1",
        "attributes": {
            "name": "new name"
        }
    }
}
```

ABUSE DELETE VERB

- archive project
  - DELETE /projects/363

- archive touches a timestamp – does not actually REMOVE the resource

- DELETE is intended to REMOVE the resource
  - response is 204 No content

- archiving using DELETE will work but problems happen afterwards
  - compliant clients will remove the resource

BREAK CACHE

- resources are cached individually
- misplaced attributes can break cache
  - company_name


```
{
  "data": {
    "id": "363",
    "type": "projects",
    "attributes": {
      "name": "AAA",
      "company_name": "Mariplast"
    },
    "relationships": {
      "company": {
        "data": {
          "type": "companies",
          "id": "1"
        }
      },
      "project_manager": {
        "data": {
          "type": "people",
          "id": "1"
        }
      },
      "last_actor": {
        "data": {
          "type": "people",
          "id": "1"
        }
      }
    }
  },
  "included": [ … ] // 2 items
}
```

BREAK CACHE

- sometimes we need “counters”
- can also break cache if handled improperly

- two important things
  - cache on the database layer
  - manual cache refresh when needed


```
{
  "data": {
    "id": "363",
    "type": "projects",
    "attributes": {
      "name": "AAA",
      "open_tasks_count": 28
    },
    "relationships": { … } // 3 items
  },
  "included": [ … ] // 2 items
}
```

NEVER INCLUDE HAS-MANY RELATIONS

- big responses
- pagination problems
- filtering problems
- sorting problems

- prefer separate calls
  - /tasks?filter[project_id]=11

Thank you!

JAN@PRODUCTIVE.IO

Visit productive.io.