NearForm

Top 10 REST API Mistakes and How to Avoid Them

Nigel Hanlon

Node.js Dublin

Who am I?

- Nigel Hanlon

- Working with NearForm as a Backend Developer for 2 ½ years so far.

- @nigelhanlon on Twitter
- github.com/nigelhanlon

This Talk

This talk will focus on ten of the most common and impactful mistakes made when developing modern REST APIs. From dreadful documentation to version-breaking mayhem, we will discuss what it truly takes to build an API you can be proud of and the pitfalls to avoid.

“Experience is simply the name we give our mistakes.”

Oscar Wilde

What is REST?

REST is an acronym for REpresentational State Transfer.

REST is..

- A design pattern you apply to your project.
- Separate to the platform and language you are using.
- Universal if followed correctly.

REST is not..

- An API but an API can be RESTful.
- A protocol, a web service or just following the HTTP standard.
- Difficult to implement, but requires forethought.

The Six Guiding Principles of REST

Client Server

Clear separation between server and client roles to simplify interactions.

Stateless

Every request must contain all of the information necessary to understand the request. Client stores state.

Cacheable

If a response is cacheable, then a client cache is given the right to reuse that response data for later, equivalent requests.

Uniform interface

Each endpoint should follow the same design principle and behaviour throughout the API.

Layered system

A client should not know or care if it is connected directly to an end server, or many serving the same endpoint.

Code on demand

You are free to return executable code to support additional features of your application. (Optional, think applets)

NearForm

Resources

Inconsistent resource naming

01

What's wrong with this?

```
/articles/listArticles/sortByNew
```

- You should never define actions when naming, that's what HTTP verbs are for.
- You should always use lowercase naming with dashes if needed.
- Filters and sorting should always be done via query parameters.

What's wrong with this?

```
/books/123/authors/8/categories/1
```

```
/categories/1/authors/8/books/123
```

- REST endpoints should be organised by largest to smallest resource.

What's wrong with this?

`/group/3/user/4`

`/groups/3/users/4`

- You should always try and use the plural form
- We are selecting from a collection so we should model them as such.

Resources

REST says exactly nothing about what URIs should look like. It's up to you to design a great API experience.

No actions in names

You are representing a resource. HTTP Verbs define actions

Resources are plural

Resources are collections, name them as such

Biggest to smallest

Each url segment should represent a smaller and smaller resource

Models, not database tables

Endpoints should describe your data model, not your database. (More on this later)

Consistency is key

If you define a naming convention, use it consistently.

Filters should be queries

Filtering and sorting should be done as a query parameter and not part of resource naming.

NearForm

Versioning

Not implementing versioning from the start

02

Version your API!

Software is never finished and neither is your REST API.

Follow best practice and..

- Version prefix endpoints, eg /v1/...
- Tolerate minor revisions, bump on major changes.
- Keep a changelog documenting revisions.

NearForm

Schemas

Ignoring the power of Schemas

03

An Example

```
{

  "title": "The Time Machine",

  "author": "H.G. Wells",

  "price": 9.99,

  "published":"1895-05-07T00:00:00.000Z",

  "available": true

}
```

```
const { title, author, price, published,
available } = req.body

if(!title || !author) { ...

}

if(!isNaN(price) || price < 0.99) { ...

}

if(!isNaN(new Date(published).getTime())){
...

}
```

With a Schema

```
{
  "title": "The Time Machine",
  "author": "H.G. Wells",
  "price": 9.99,
  "published":"1895-05-07T00:00:00.000Z",
  "available": true
}
```

```
{
  "required": ["title", "author", "price"],
  "type": "object",
  "properties": {
    "title":{
      "type": "string",
      "maxLength": 256
    },
    "price":{
      "type": "number",
      "minimum": 0.99
    },
    "published": {
      "type": "string",
      "format": "date-time"
    },
    "available: {
      "type": "boolean"
      "default": true
    }
  }
}
```

Benefits

Schemas are a great way to both enforce standards and document REST APIs.

Documentation for free

Tools like Swagger can automatically generate documentation from schemas

Schemas are portable

The same schemas can be used on both the front and backend.

Collaborate API Design

You can design and revise your API before final implementation.

Schemas can be enforced

Modern API frameworks can automatically return 400 errors for schema violations

Automated Testing

Tools like Postman and various test runners support OpenAPI Schemas out of the box

NearForm

Docs

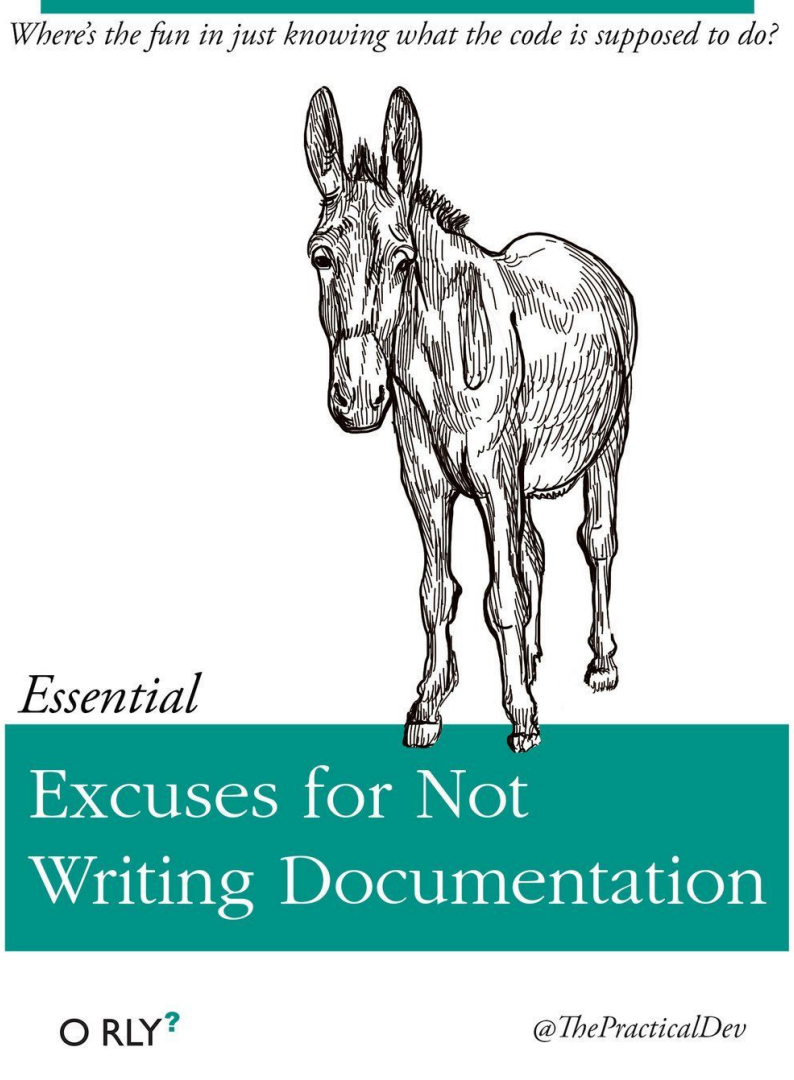
Undeadable, outdated or missing documentation

04

Documentation

We all love good documentation but hate writing it.

- The code is the documentation..
- Just look at the example JSON
- Here's a pdf with screenshots you can't search, copy or paste from.
- What documentation?



Wish List

- List of resources (Data model)
- Authentication guide
- Endpoint definitions
- Error definitions
- Example responses
- Code snippets
- Changelog

Some great examples

If you would like some inspiration, try these:

Stripe

https://stripe.com/docs/api

GitHub

https://developer.github.com/

Twitter

https://developer.twitter.com/en/docs

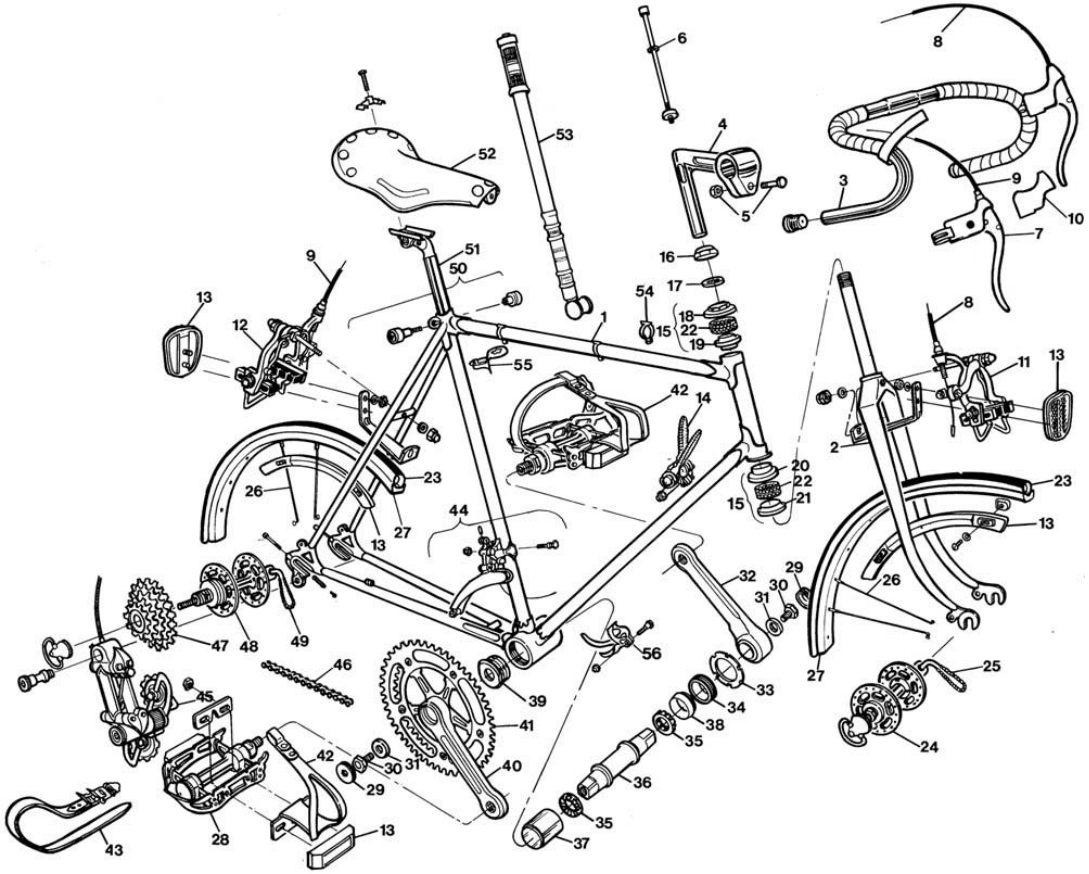
NearForm

Model

Poor data models cost bandwidth

05

Poor API design has terrible consequences.



An Example

```
GET /books/123

{

  "title": "JavaScript: The Good Parts",

  "author": 456,

  "category": 789

}
```

```
GET /authors/456

{

  "name": "Douglas Crockford"

}

GET /categories/789

{

  "category": "Software Development"

}
```

An Example

```
GET /books/123
```

```
{
  "title": "JavaScript: The Good Parts",

  "author": {
     "id": 456,
     "name": "Douglas Crockford"
  },

  "category": {
     "id": 789,
     "category": "Software Development"
  }
}
```

```
GET /authors/456
```

```
{

  "name": "Douglas Crockford"

}
```

```
GET /categories/789
```

```
{

  "category": "Software Development"

}
```

Best Practice

When building an API you must consider the resources you are trying to represent.

Minimise requests

Your API should aim to deliver relevant information in as few requests as possible.

Hydrate resources

If a resource is relevant, hydrate and deliver it rather than just an identifier.

Consolidate

Consolidate smaller resources if and when it makes your model cleaner.

NearForm

HTTP

Not using the right HTTP Verbs and Status

06

A Guide to HTTP Status Codes

1xx

Hold on!

2xx

Here you go.

3xx

Go away.

4xx

You screwed up.

5xx

I screwed up.


500 Error
200 OK
JAKE-CLARK.TUMBLR
imgflip.com

Try these..

Code	Status	When to use
202	Accepted	Useful for accepting requests that may take time to process.
204	No Content	Successful delete requests or changes that need no reply.
400	Bad Request	Generic cover all response but confirms the client is at fault.
401	Unauthorised	No authentication credentials presented.
403	Forbidden	User correctly authenticated, but they don't have the required permissions.
422	Unprocessable Entity	The request payload is correct but cannot be processed for some reason.
429	Too Many Requests	Return this when implementing rate limiting or quota based limits.
501	Not Implemented	A handy way to to say this endpoint will exist in the future
503	Service Unavailable	API is down for maintenance

A Guide to HTTP Verbs

GET

I want this

POST

Create this

PUT

Change this

DELETE

Delete this

PATCH

Change just this part

Putting it all together..

1. GET 200 OK
2. POST: 201 Created
3. PUT: 200 OK
4. PATCH: 200 OK
5. DELETE: 204 No Content

NearForm

Errors

Poor Error Responses

07

Errors

Internal Server Error

Errors

HTTP 422 Unprocessable Entity

Errors

```
{
    "statusCode": 422,
    "message": "Failed to create new event",
    "error": "Property eventDate cannot be in the past",
    "help": "http://awesome-events-api.com/docs/addEvent"
}
```

Error Handling Best Practice

A good error response is worth its weight in gold, but don't give too much away.

Return specific status codes

Stop returning 500 error codes for everything... please?

JSON Errors for a JSON API

Don't return plain text errors on a JSON API. They are next to useless.

Verbose 400s, strict 500s

400 errors are usually client side, but verbose 500s could expose sensitive internal data or attack vectors.

Include details

Where appropriate, include messages, error codes or details to help debug.

Use error arrays if needed

Sometimes you need to return more than one error, especially when using forms.

Consistency

Consistent error messages are much easier to handle by clients.

NearForm

Not Stress Testing

08

Reasons to stress test

1. Basic Benchmarks.
2. Memory Leaks.
3. Resource Starvation.
4. Capacity Planning
5. Performance Regression

Tools

AutoCannon

AutoCannon - A HTTP/1.1 benchmarking tool written in node.

Ox

Ox - Discover the bottlenecks and hot paths in your code, with flamegraphs.

Clinic.js - Tools to help diagnose and pinpoint Node.js performance issues.

NearForm

Paging

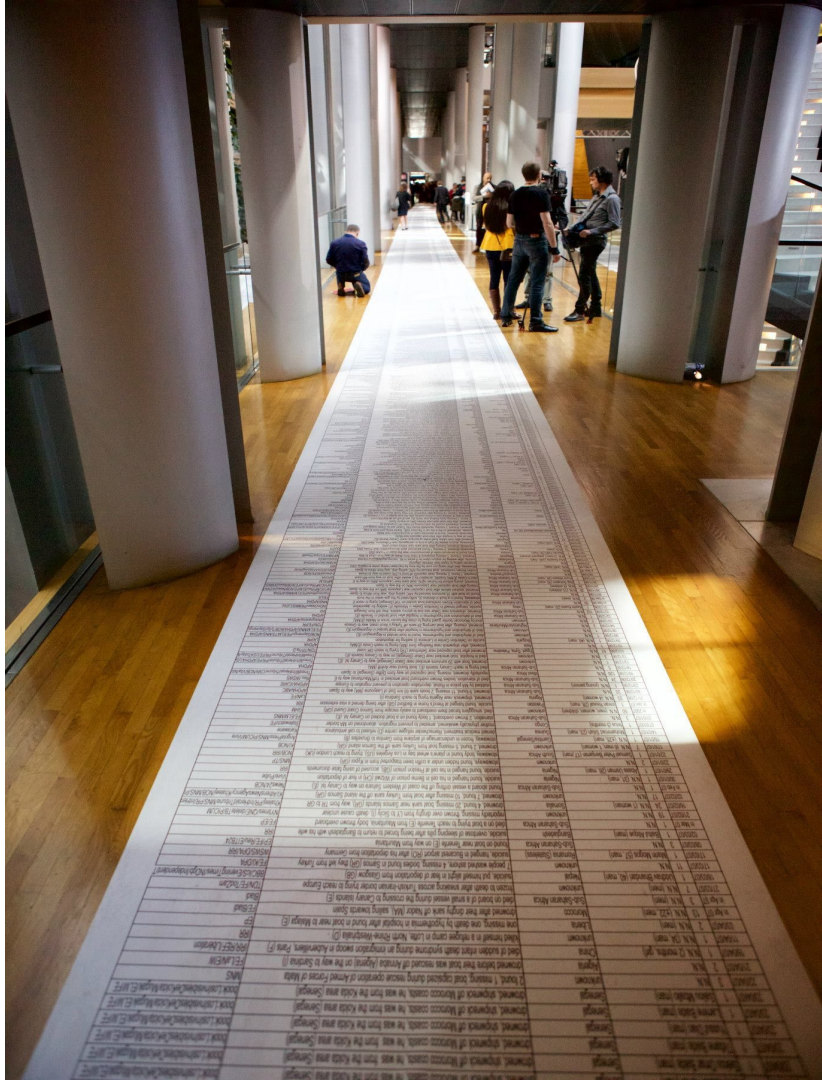
Prev | Mistake 9 of 10 | Next

09

Pagination

Most endpoints that return a list of entities should provide pagination.

- Improves frontend experience
- Allows for prefetching
- Less bandwidth intensive
- Lower DB loads (provided you index correctly).
- Filtering and sorting can return more relevant results faster.



Pagination

```
{
    "meta": {
        "page": { "size": 10, "offset": 20, "total": 100 }
    },
    "links": {
        "prev": "/example-data?offset=0&limit=10",
        "self": "/example-data?offset=20&limit=10",
        "next": "/example-data?offset=30&limit=10"
    },
    "data": [
        { "type": "examples", "id": "10" },
        { "type": "examples", "id": "11" },
        ...
        ..
        .
        ..
        ...
        { "type": "examples", "id": "20" }
    ]
}
```

Permission

Creating your own roles & permissions system

10

Don’t do it.

Permissions

```
{
    "username": "Nigel",
    "admin": true
}

{
    "username": "Bob",
    "admin": false
}
```

Permissions

```
{
    "username": "Nigel",
    "admin": true,
    "permissions": ["create", "delete", "view"]
}

{
    "username": "Bob",
    "admin": false,
    "permissions": ["view"]
}
```

Permissions

```
{
  "username": "Nigel",
  "roles": ["admin"]
}

{
  "username": "Bob",
  "roles": ["user"]
}

{
  "username": "Alice",
  "roles": ["editor"]
}
```

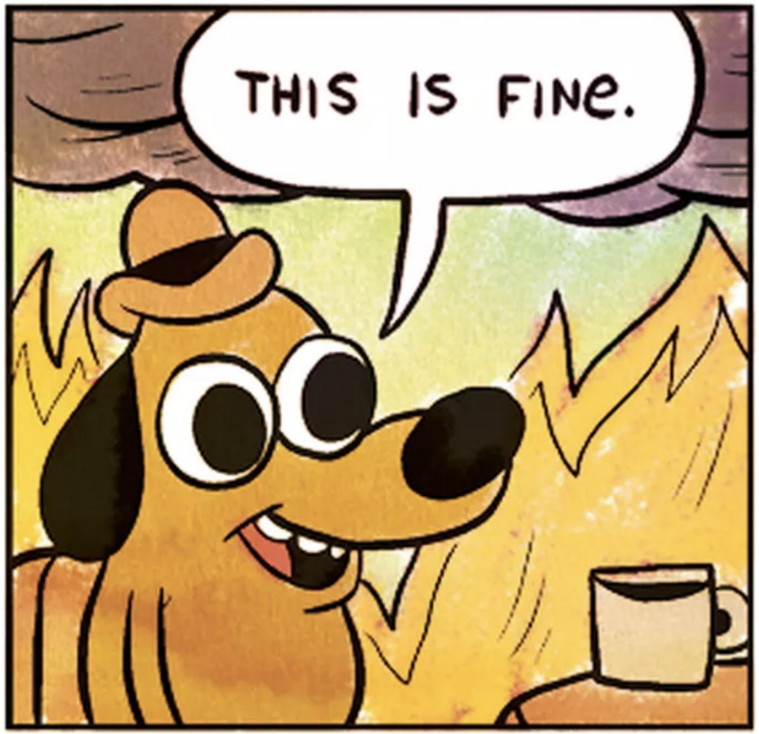
```
{
  "admin": [
    "create",
    "delete::all",
    "update::all",
    "view",
    "comment"
  ],

  "editor": [
    "create",
    "update:own",
    "delete::own",
    "view",
    "comment"
   ],

  "user": [
    "view",
    "comment"
  ],

  "guest": ["view"]
}
```

Permissions




```
{
    "editor": [
      "delete::comment::own::blogpost",
      "update::comment::own::blogpost",
      "delete::comment::own::gallery",
      "update::comment::own::gallery",
      "list::comment",
      "moderate::comment::own",
      "create::blogpost",
      "create::gallery",
      "delete::blogpost::own",
      "delete::gallery::own",
      "update::blogpost::own",
      "update::gallery::own",
      "update::tags::own",
      "delete::tags::own,
      "create::tags",
      "publish::blogpost::own",
      "unpublish::blogpost::own,
      "publish::gallery::own",
      "unpublish::gallery::own,
      "create::contributor"
      ...
    ]
}
```

Try these..

There are plenty of production ready solutions you can use.

Choose one that fits your needs.

Aim for the best fit with minimal overhead.

UDARU

https://nearform.github.io/udaru

ACCESS CONTROL

https://github.com/onury/accesscontrol

https://casbin.org/

GraphQL

Not at least giving it a try!

11?

Questions?

Thank you for your time.

United States: + 1 916 235 6459 nearform.com
International: + 353 1 514 3545 sales@nearform.com