The background is a dark navy blue. It features several overlapping, semi-transparent geometric shapes in various colors: bright green, cyan, magenta, orange, red, and blue. These shapes are arranged in a way that creates a sense of depth and movement, with some shapes appearing to be layered on top of others. The overall aesthetic is modern and tech-oriented.

**What is
JSON:API
and how can
I use it well?**

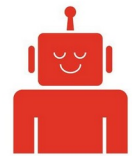


Hi!

I am Mateu

I am here because I am a decoupling
nerd

You can find me at @e0ipso



Lullabot

You will learn about...

JSON API

What is it?

Why use it?

Drupal module

What's the
status?

What are the
limitations?

Extend it

Solutions to hard
problems

**Stop deciding
unimportant things.
Start working**




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

Defines: Transport Interaction

```
GET /articles/1/relationships/comments HTTP/1.1  
Accept: application/vnd.api+json
```

Creative Commons specification



**Strongly driven by FE & UX
experts**

Steve Klabnik, Yehuda Katz, Dan Gebhardt, Tyler Kellen, Ethan



Why this one?

Since there are others, and there is GraphQL as well.



141 repos

That's a lot of traction



18 languages

And a lot of range

Client & Server

Total success!



With a highlight on its **flexibility**

Stays neutral on
implementation details and
gives you space. Also
provides extension system.



RTFM

Response to the typical problems

- > Multiple round trip requests
- > Bloated responses
- > Content discovery

They all have known solid solutions!



1. **TRANSPORT FORMAT**

The shape of the JSON object

**Resource
data
Info
(ID)**

**Supporting
Structure
(GLUE)**

**Attributes
&
Relationships
(DATA)**

**HATEOAS
&
Metadata
(HYPERMEDIA)**

FORMAT


```
{  
  "data": {  
    "type": "articles",  
    "id": "1",  
    "attributes": {...},  
    "relationships": {...},  
  },  
  "links": {...},  
  "meta": {...}  
}
```

FORMAT



```
{
```

```
...
```

```
  "attributes": {  
    "title": "Drupal 8!",  
    "body": "Lorem ipsum"
```

```
  ...
```

```
},
```

```
...
```

```
}
```

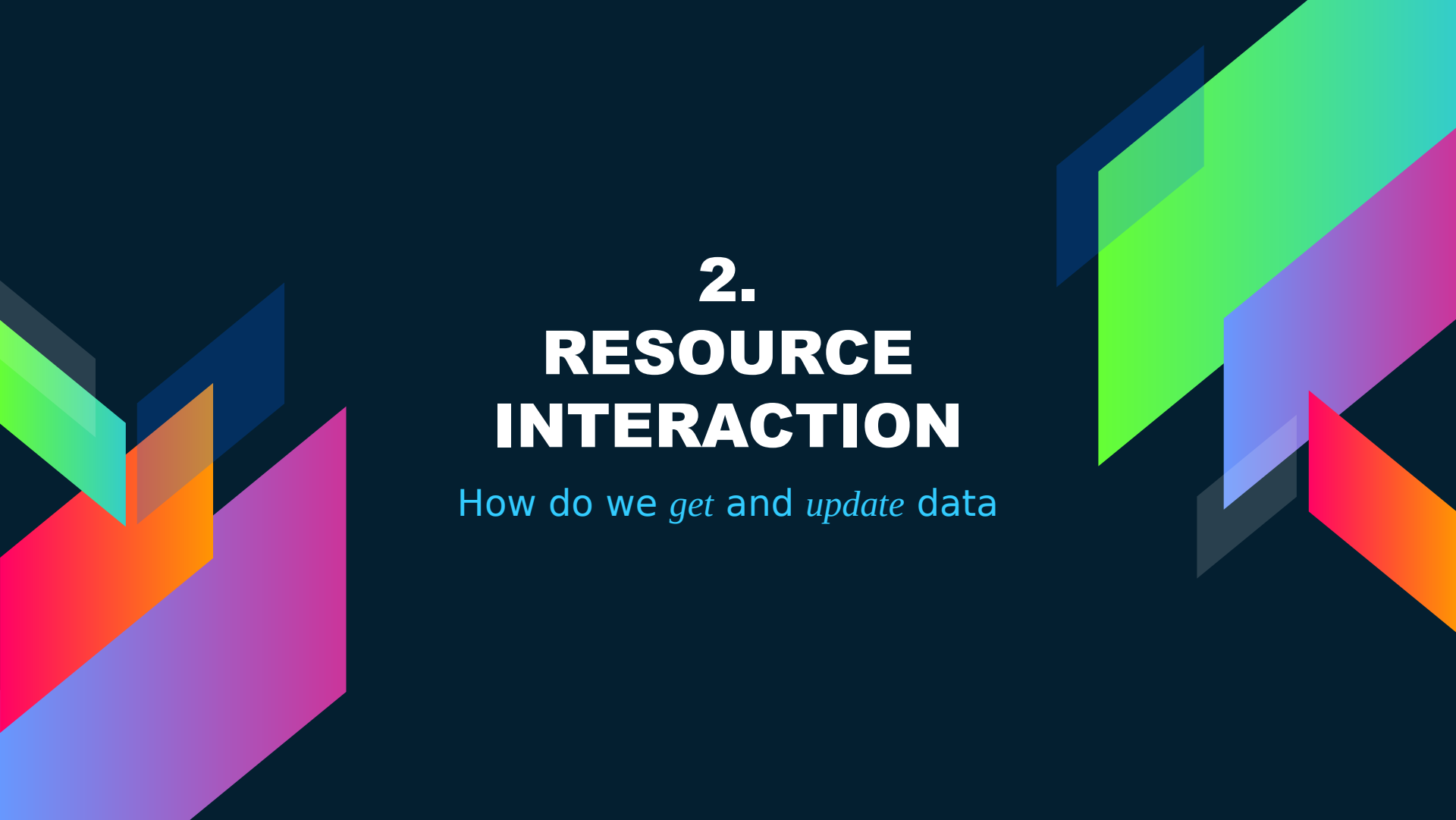
FORMAT

...

```
"relationships": {  
  "links": {  
    "self": "articles/1/relationships"  
  },  
  "tags": {  
    "data": [{  
      "type": "tags",  
      "id": "2"  
    }]  
  }  
}
```

...

FORMAT



2. RESOURCE INTERACTION

How do we *get* and *update* data

Uses REST

GET, POST, PUT, PATCH, DELETE, ...

Typical request

GET /articles HTTP/1.1
Accept: application/vnd.api+json



RESPONSE

/jsonapi/node/article



Response to the typical problems

- > Multiple round trip requests
- > Bloated responses
- > Content discovery

They all have known solid solutions!

The typical solutions

- > ⚠ Multiple round trip requests
- > ✅ Resource embedding
- > ⚠ Bloated responses
- > ✅ Sparse fieldsets
- > ⚠ Content discovery
- > ✅ Collections and filters

EXTREMELY SIMPLE

Your project will
have way more
stuff than this!



- > **1:** GET `articles/12`
- > **2:** GET `articles/12 => tags/34`
- > **3:** GET `articles/12 => tags/88`
- > **4:** GET `articles/12 => users/88`
- > **5:** GET `articles/12 => users/88 => images/5`
- > **6:** GET `articles/12/comments`
- > **7:** GET `articles/12 => comment/2`
- > **8:** GET `articles/12 => comment/2 => user/8`
- > **9:** GET `articles/12 => comment/2 => user/8 => image/9`
- > **10:** GET `articles/12 => comment/7 [...]`
- > **11:** GET `articles/12 => comment/7 [...]`
- > **12:** GET `articles/12 => comment/7 [...]`
- > **MORE!**



```
GET /articles/12?
```

```
include=
```

```
author,author.pic,
```

```
tags,
```

```
comment,comment.author,
```

```
comment.author.pic
```

**Resource
embedding**




```
GET /articles/12?  
fields[articles]=  
  title,  
  created
```

**Sparse
fieldsets**

...

```
"attributes": {  
  "title": "My article",  
  "uuid": "12345-1234-34",  
  "created": "10-05-2012",  
  "status": "1",  
  "body": {...},  
  "langcode": "en"  
}
```

...



“Give me the cover image and the publication year of all the albums of all the bands having one of the members under 35 currently living in Murcia.


Oh! And while you're at it, output the name of the band and that member as well.”

GET /bands?

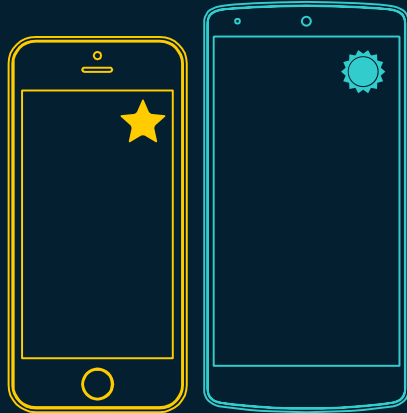
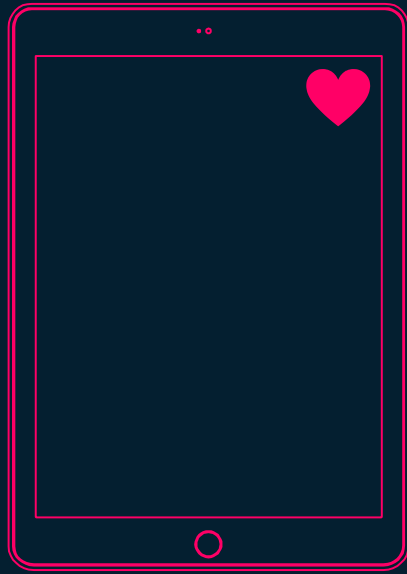
```
filter[members.city][value]=Murcia&  
filter[members.age][value]=35&  
filter[members.age][operator]="<="&  
include=albums,albums.cover,members&  
fields[bands]=name,albums,members&  
fields[members]=name&  
fields[albums]=publication&  
fields[images]=uri
```

**Collections
and filters**

WRITE URL QUERIES

A space-themed illustration featuring a blue planet with a ring and three small black dots, a pink rocket with a flame trail, and three yellow stars. The illustration is positioned to the right of the main title.

Every **API consumer** requests the resource data it needs. It can be different every time.



Every consumer has different data needs. The server (Drupal) cannot choose what those are.

Every resource 4 “endpoints”

1. `/bands/1234`
 - > **GET, PUT, PATCH, DELETE**
2. `/bands`
 - > **GET, POST**
3. `/bands/1234/albums`
 - > **GET**
4. `/bands/1234/relationships/albums`
 - > **GET, PATCH**

3. **PERFORMANCE**

How fast is the Drupal module?



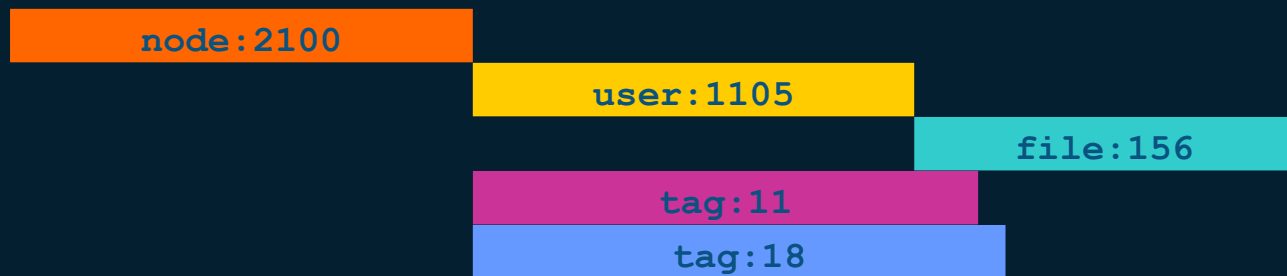
Benchmarking JSON API

- > `ab -v4 -k -c8 -n10 -A u:p`
- > `node:2100`
- > `include`
 - > **Author**
 - > **Author image**
 - > **Tags (2 tags)**

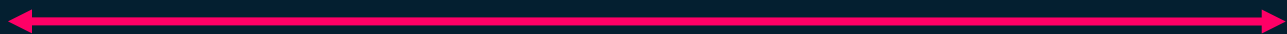
Benchmarking core HAL JSON

- > `ab -v4 -k -c8 -n10 -A u:p`
- > `node:2100`
 - > `user:1105`
 - > `file:156 (slow path)`
 - > ~~`tag:11`~~
 - > ~~`tag:18`~~

Results (core): anonymous

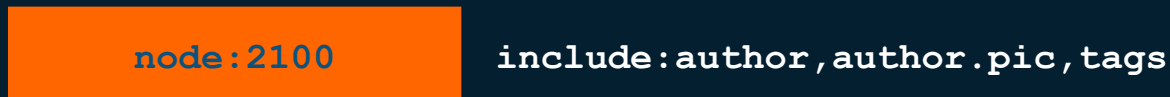
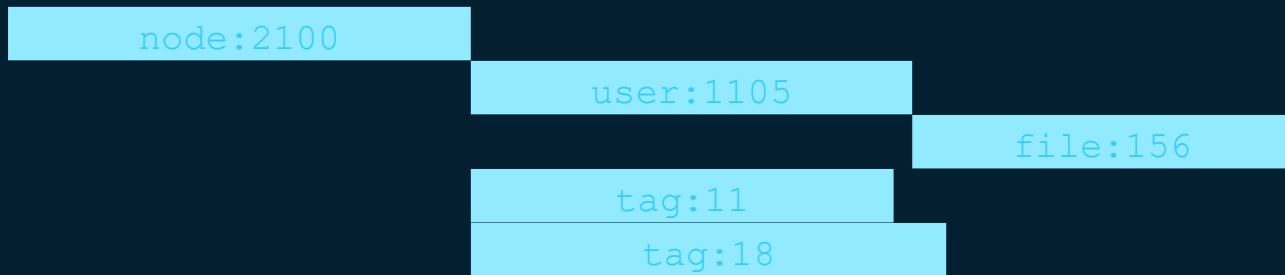


~ 21 ms



Using Keep Alive

Results (jsonapi): anonymous



~ 7 ms

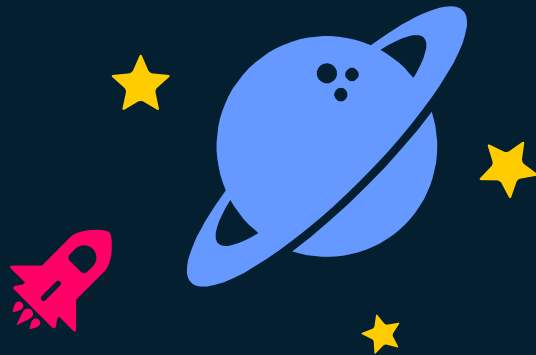


	Core (ms)	{json:api} (ms)
Anonymous	21	7
Auth	320	115
Uncached	392	182

[https://gist.github.com/e0ipso/
4b1b346b296fbf0c918450fef5b0b3d7](https://gist.github.com/e0ipso/4b1b346b296fbf0c918450fef5b0b3d7)

AVOID BOOTSTRAPS

And unnecessary HTTP round trips.



4.

DRUPAL MODULE

Our implementation of the
standard.



It is in Drupal core!



Oriented to entity bundles

- > Each resource is a bundle (content type)
- > `/jsonapi/node/page`
- > Automatically enabled (can be disabled)
- > You can do **any** entity query as filter

Customize your API with JSON:API Extras

- > `/node/page` → `/pages`
- > `field_text` → `text`
- > Disable fields ☹
- > Disable resources ☹
- > Fields enhancers (preprocessors) 🛠

JSON:API cross bundles

- Separate contrib
- Allows you to have bundles per entity type:
 - <https://.../jsonapi/node>
 - <https://.../jsonapi/media>


JSON:API hypermedia

- Enhances support for *links*
- Enables HATEOAS
 - Let's you define custom behaviors about the returned data.

```
{  
  "data": {  
    "type": "product",  
    "id": "1234-...-abcd",  
  },  
  "links": {  
    "add-to-cart": {  
      "href": "/purchases",  
      "rel": "add"  
      ...  
    }  
  }  
}
```

HYPERMEDIA

```
{  
  "href": "/purchases",  
  "rel": "add",  
  "params": {  
    "title": "Buy now!",  
    "confirm": "Yes, I am sure.",  
    "data": {  
      "type": "product",  
      "id": 1  
    }  
  }  
}
```





Buy now!

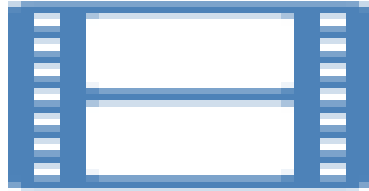
```
{  
  "href": "/wishlist/items",  
  "rel": "add",  
  "params": {  
    "title": "Save for later",  
    "confirm": false,  
    "data": {  
      "type": "product",  
      "id": 1  
    }  
  }  
}
```

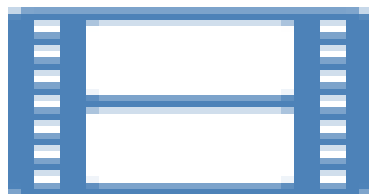


Save for later



Do not have the client app
check the product stock API







JSON:API Resources

- > **It doesn't do anything by itself.**
- > **It handles non-entity resources**  **.**
- > **For custom routes and data.**

- `/jsonapi/me`
- `/jsonapi/user/{id}/reminders`
- `/jsonapi/node/posts/{id}/retweet`

OpenAPI

- > The leading standard to describe APIs
- > There is a whole talk about it!

Credits

Special thanks to all the people who made and released these awesome resources for free:

- > Presentation template by [SlidesCarnival](#)
- > Photographs by [Startupstockphotos](#)
- > [Creative Commons images](#)