

**YOUR API IS
BAD**

**AND YOU SHOULD FEEL
BAD**

Who is this guy?



Ben Edmunds

Open Source

Author

PHP Town Hall Podcast

CTO at Mindfulware

Who is this guy?



Ben Edmunds

@benedmunds

<http://benedmunds.com>

What Will We Cover?

Slim PHP Framework

RESTful API Basics

API Design

Implementing your API

Versioning, Auth, & Rate Limiting

Slim PHP

Simple

Similar to Sinatra

FastRoute

Testable

Workshop

SETUP BASE PROJECT

<https://github.com/benedmunds/>

OpenWest2018_API_Workshop

RESTful API Basics



What is REST?

Representational **S**tate **T**ransfer

HTTP Methods

GET, POST, PUT, PATCH, DELETE

Expressive

Real-World Usage

What is REST?

Formats

JSON*

XML

Anything

What is REST?

Most widely used data format is JSON

Javascript **O**bject **N**otation

```
{  
  key: 'value',  
  anotherKey: 'value2'  
}
```

What is REST?

JSON usage in Javascript = familiar

```
var object = {  
  key: 'value',  
  anotherKey: 'value2'  
};
```

What is REST?

JSON encoding is supported natively in most languages

PHP

```
echo json_encode(array(  
    'key' => 'value'  
));
```

Outputs

```
{key:'value'}
```

What is REST?

JSON decoding is just as simple

PHP

```
$json_data = "{key:'value'}";  
print_r(json_decode($json_data));
```

Outputs

```
array(  
    'key' => 'value'  
)
```



Warning

This will be Opinionated

API Design

Simple

Expressive

Intuitive

API Design

Simple

Expressive

Intuitive

STABLE

API Design

Simple

Expressive

Intuitive

STABLE

CONSISTENT

API Design

Use the Facebook API

DO THE OPPOSITE

API Design

Document!

Document!

Document!

API Design

To ADD or not to ADD

API

Driven

Development

API Design

HTTP Status Codes

HTTP Status Cats
to the Rescue!

API Design

Status **2xx** = Success



200
OK

API Design

Status **3xx** =
Redirect



301

Moved Permanently

API Design

Status **4xx** = Client
Errors



406

Not Acceptable

API Design

Status **5xx** = Service
Errors



599

Network connect timeout error

API Design

VERBS

API Design

HTTP Methods

POST/PUT	=	Create
GET		Read
PUT/PATCH		Update
DELETE		Delete

API Design

POST/PUT = Create

PUT if all values are known

api.domain.com/user

API Design

GET = READ

api.domain.com/user/2

API Design

PUT = IDEMPOTENT

Needs all info

Basically, include the ID

API Design

Put vs Post

Different Opinions

PUT = create (idempotent)

POST = create (unknown resource)

PUT/PATCH = update (resource is known)

API Design

PUT = UPDATE

api.domain.com/user/2

{

id: 2,

first_name: 'bob',

last_name: 'cat'

}

API Design

PATCH = IDEMPOTENT

Needs all identifying info

Basically, include the ID

Does not need all info

Series of changes

API Design

PATCH = UPDATE

api.domain.com/user/2

```
{  
  id: 2,  
  first_name: 'bob'  
}
```

API Design

PATCH = UPDATE

api.domain.com/user/2

```
[  
  {id: 2, first_name: 'bob'}  
  {id: 2, first_name: 'bobby'}  
]
```

API Design

DELETE = DELETE

api.domain.com/user/2

API Design

CONSISTENCY

API Design

URL endpoints represent data

/user	=	user
/company	=	company

API Design

A single Object maps to a singular URL

api.domain.com/user/2

api.domain.com/company/3

API Design

Objects map to plural URLs

api.domain.com/users

api.domain.com/companies

API Design

Common actions across most objects

GET /user/2

GET /company/3

DELETE /user/2

DELETE /company/3

EVERYBODY JUST CHILL OUT



I GOT THIS

API Design

<https://site.com/api/statuses>

Maps to all of the statuses

“Statuses” could be:

- SQL table

- NoSQL collection

- Aggregate Data

API Design

<https://site.com/api/status/1234>

Maps to a single status with
the ID of 1234

API Design

Creating

If you know all of the data
(including the ID)

PUT <https://site.com/api/status/1234>

API Design

Creating

If you know all of the data
(including the ID)

PUT <https://site.com/api/status/1234>

Idempotent

No matter how many times you send
this data only one resource should
ever be created

API Design

Creating

If you know all of the data
(including the ID)

PUT <https://site.com/api/status/1234>

```
{  
  id: 1234,  
  retweeted: false,  
  active: true  
}
```

Implementation

Creating

If you don't know all of the data

POST <https://site.com/api/status>

Implementation

Creating

If you don't know all of the data

POST <https://site.com/api/status>

Unique

Each time you post this data a new resource should be created

Implementation

Creating

If you don't know all of the data

POST <https://site.com/api/status>

```
{user_id: 1,  
  text: 'Test Status'  
}
```

Implementation

Creating

If you don't know all of the data

POST <https://site.com/api/status>

```
{user_id: 1,  
  text: 'Test Status'  
}
```

Response

```
{succes: true,  
  id: 123  
}
```

API Design

Reading

Get all statuses

GET <https://site.com/api/statuses>

API Design

Reading

Complex requests that require additional data

GET <https://site.com/api/statuses>

API Design

Reading

Get all of the statuses that have been retweeted and are active

GET <https://site.com/api/statuses>?
retweeted=1&active=1

API Design

Updating

We know all of the identifying information so we PUT updates

Implementation

Updating

We know all the data

PUT <https://site.com/api/status/123>

```
{id: 123,  
  user_id: 1,  
  text: 'Test Status 2'  
}
```

Implementation

Updating

We know some of the data

PATCH <https://site.com/api/status/123>

```
{id: 123,  
text: 'Test Status 3'  
}
```

API Design

Delete

Just pass the identifying information

DELETE <https://site.com/api/status/123>

```
{id: 123}
```

API Design

Relationships

Get the user that posted a status

GET <https://site.com/api/status/1234/user>

Workshop

WALKTHROUGH I

Status Routes

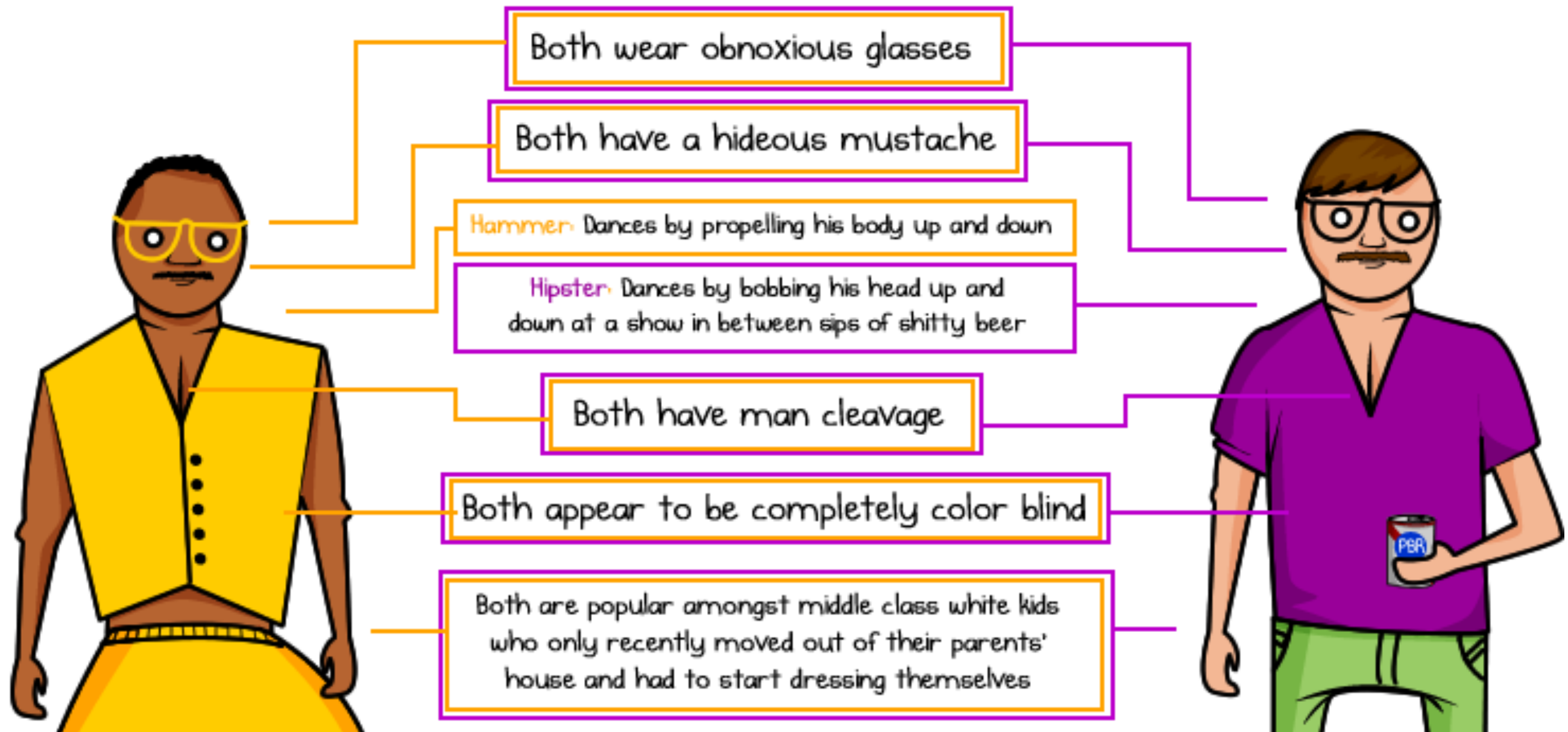
Workshop

WALKTHROUGH 2

TESTS

Hammer Pants VS Hipsters

A visual comparison



Versioning

How to handle versioning?

Each version has a separate route -

GET <https://site.com/api/v1/statuses>

GET <https://site.com/api/v2/statuses>

Versioning

How to handle versioning?

Route newest API through
<https://api.site.com>

GET <https://site.com/api/statuses>

=

GET <https://site.com/api/v2/statuses>

Versioning

How to handle versioning?

Accept Headers
(HATEOAS)

Accept: application/vnd.github.user.v4+json

Workshop

WALKTHROUGH 3

VERSIONING

HATEOAS

**Hypermedia
as the
Engine
of
Application State**

THE ONLY THING I LOVE



IS HATE

-OAS

HATEOAS

Content Negotiation

Hypermedia Controls

HATEOAS

Content Negotiation

Request “Accept” Header
(JSON, XML, YAML, etc)

Accept: application/x-yaml
Accept: application/json

HATEOAS

Content Negotiation

OPTIONS Http Request

Allow: GET, PUT, POST

HATEOAS

Hypermedia Controls

Links to Related Content

```
{ success: true,  
  id: 123,  
  links:  
    {  
      "rel": 'self'  
      "url": '/status/123'  
    },  
    {  
      "rel": 'user'  
      "url": '/status/123/user'  
    }  
}
```

HATEOAS

Hypermedia Controls

Links to Related Content

```
{ success: false,  
  error: {  
    "code": 'errorFail'  
    "message": 'You have Failed!'  
    "url": 'http://domain.com/docs/errorFail'  
  }  
}
```

Workshop

WALKTHROUGH 4

HATEOAS



PERMISSION

DENIED

Authentication

**Who's the
Consumer?**

User Service

Internal Service

Authentication

Consumer = User

OAuth

Client

Server

Request Token ->

<- Access Token

Authentication

Consumer = User

OAuth 2

Standard

Many Diff “Flows”

Complicated Spec

Authentication

Consumer = User

OAuth 2

Client

Server

Request ->

w/AccessToken

Authentication

**Consumer =
Internal**

Access Token = Service Key + Client Key

Client

Server

Request ->

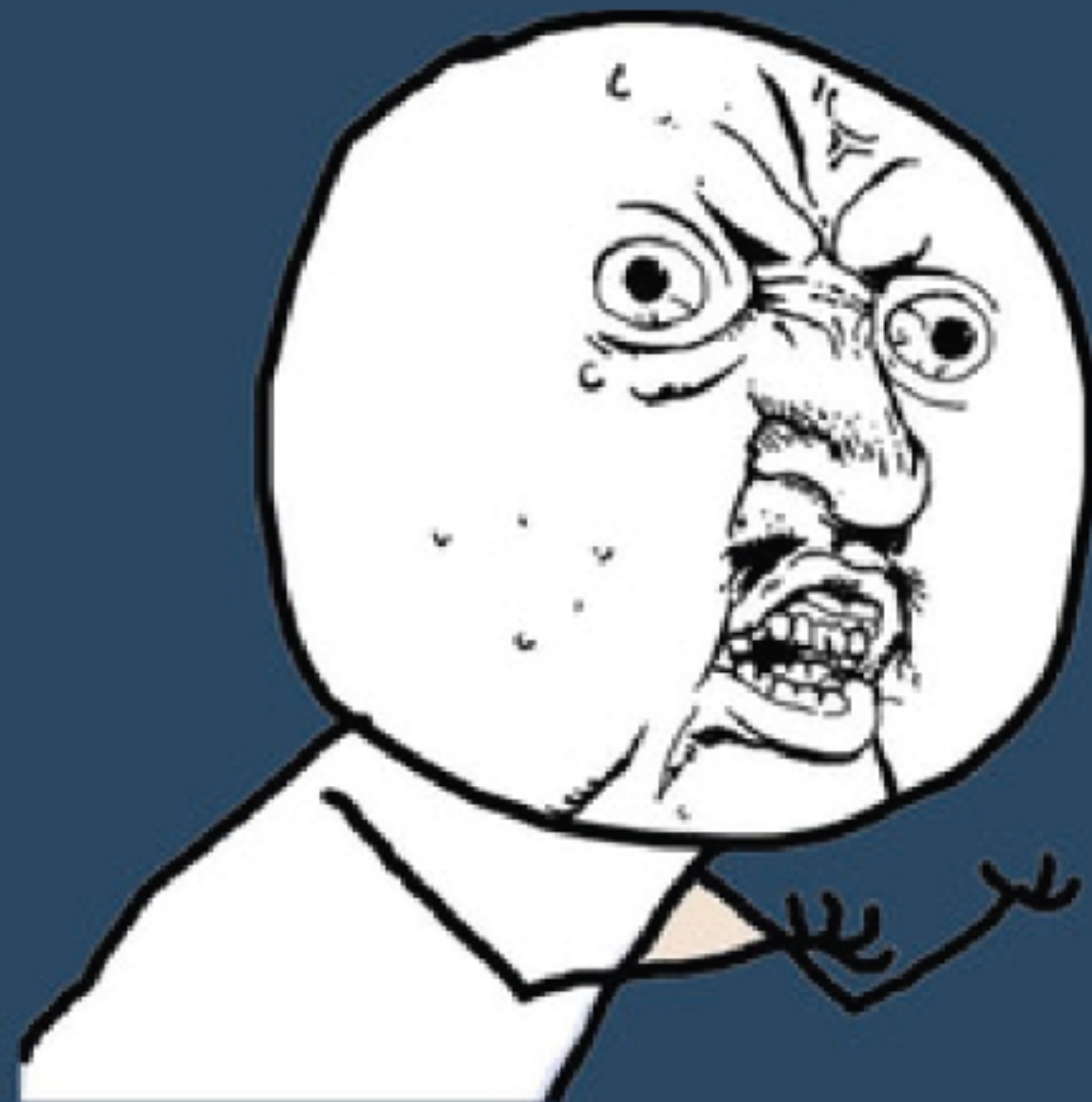
w/AccessToken

Authentication

Data Hash

SHA1 (\$DATA)

YOUR API'S DOWN



Y U NO RATE LIMIT?

Rate Limiting

Prevent Abuse

Maintain

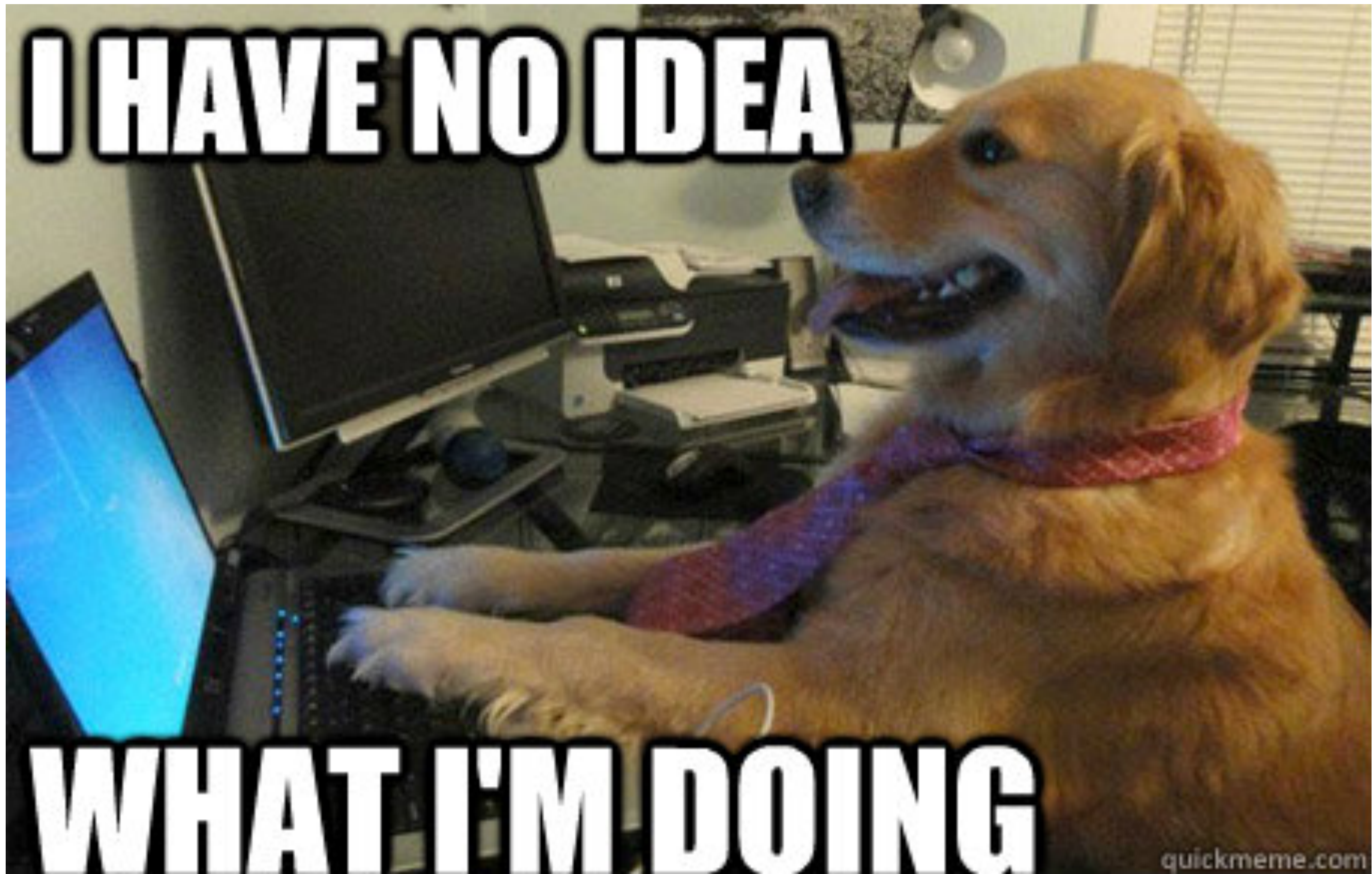
Availability

Client Key

IP Address

User ID

I HAVE NO IDEA



WHAT I'M DOING

Debugging

**How to debug as
you develop?**

CURL

Automated Tests

Rested on OSX

Workshop

WALKTHROUGH 6

BONUS - CLIENT

Go Make
Cool Things

Resources

Book -

Build APIs You Won't Hate

<https://leanpub.com/build-apis-you-wont-hate>

Tutorial -

Demystifying REST

<https://tutsplus.com/tutorial/demystifying-rest/>

Resources

Blog -

OAuth 2 Simplified

[http://aaronparecki.com/articles/
2012/07/29/1/oauth2-simplified](http://aaronparecki.com/articles/2012/07/29/1/oauth2-simplified)

GuiltTrip.io



**STOP PLAYING GAMES
AND FINISH YOUR
SIDE PROJECT!**

Each month we'll make you feel like
s%!* for not finishing your side project.

STOP NEGLECTING YOUR SIDE PROJECT! \$5/MONTH

Q/A TIME!

<https://joind.in/talk/0033b>

[https://github.com/benedmunds/
OpenWest2018_API_Workshop](https://github.com/benedmunds/OpenWest2018_API_Workshop)

Ben Edmunds

@benedmunds

<http://benedmunds.com>