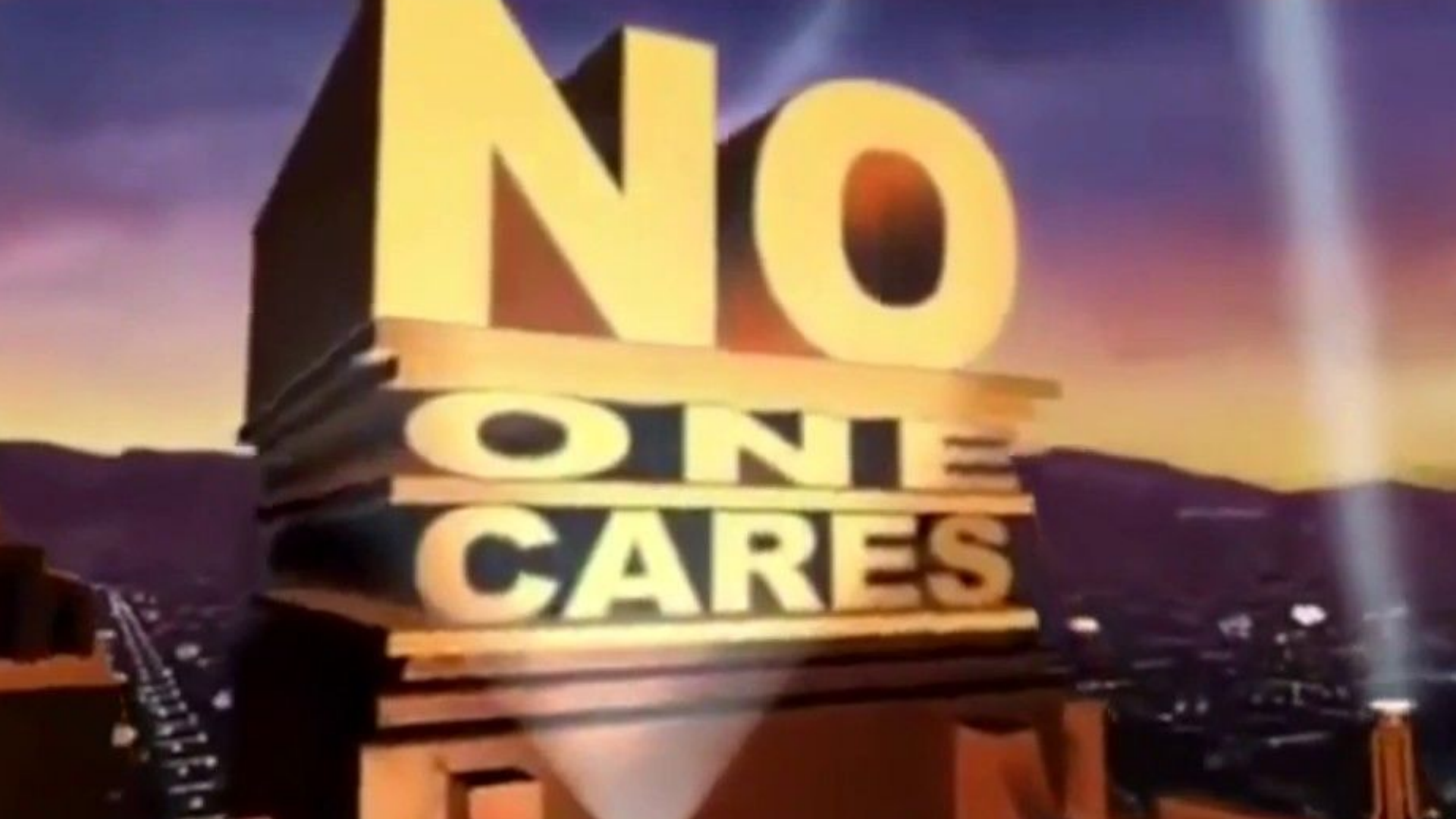
Building APIs with ColdFusion

Part 1: Start coding APIs today

Online CF Meetup
October 15th 2020

Gavin Pickin


Who am I??


NO
ONE
CARES

Who am I?

- Software Consultant for Ortus Solutions
- Work with ColdBox, CommandBox, ContentBox, APIs and VueJS every day
- Working with Coldfusion for 22 years
- Working with Javascript just as long
- Love learning and sharing the lessons learned
- From New Zealand, live in Bakersfield, Ca
- Loving wife, lots of kids, and countless critters

@gpickin on twitter

http://www.ortussolutions.com

Why should I give this talk?

- I build a lot of APIs for Customers
- I build, maintain and convert a lot of legacy ColdFusion sites
- I have seen many different ways to build an API

- I'm going to share some of the lessons I have learned.
- Hopefully you'll learn something
- Hopefully it will be useful.

Is this how I build APIs today?

No

I will be doing another talk in November, Part 2, where I show you how I build APIs today.

- Not everyone can start a greenfield project
- Not everyone can use frameworks, and libraries
- Not everyone NEEDs that, they just need a little API in their existing app

What is an API

- API is an application programming interface

More commonly when someone says API today they think of an JSON API, usually we hope for a REST API but not always.

APIs can be xml, soap, but its very common to talk about JSON apis, and that's what we're talking about today.

REST JSON API vs JSON API

What is REST

REST is acronym for REpresentational State Transfer. It is architectural style for distributed hypermedia systems and was first presented by Roy Fielding in 2000 in his famous dissertation.

All REST are APIs but not all APIs are REST.

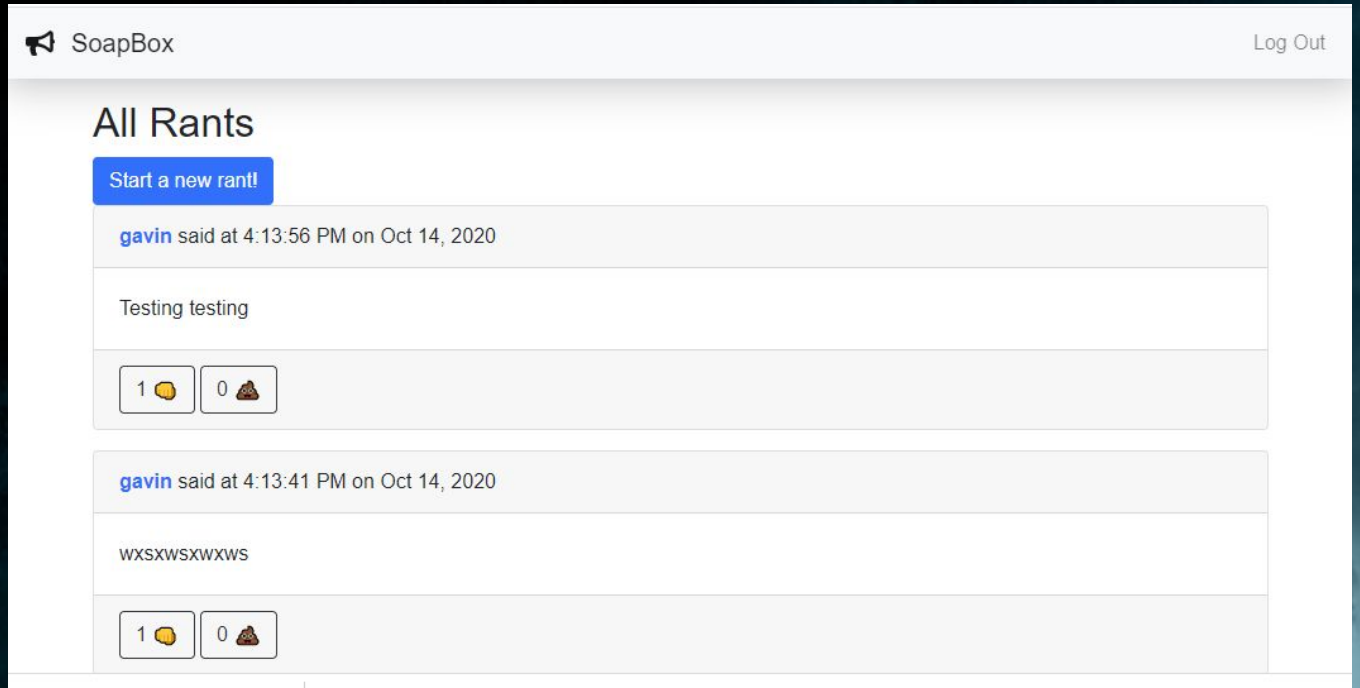
Guiding Principles of REST

- Client–server
- Stateless
- Cacheable
- Uniform interface
- Layered system
- Code on demand (optional)

The key abstraction of information in REST is a resource. Any information that can be named can be a resource: a document or image, a temporal service, a collection of other resources, a non-virtual object (e.g. a person), and so on. REST uses a resource identifier to identify the particular resource involved in an interaction between components.

What App are we working on today?

SOAPBOX - a twitter clone we built for our ColdBox Zero to Hero Training.
Modified for this presentation



How did I modify Soapbox for today?

- I have purposely removed a lot of things so it should resemble a very legacy app
  - I removed the framework - no ColdBox
  - I removed ColdBox modules like BCrypt
  - I am not using dependency injection like wirebox
  - I removed the OO (objects)
  - I removed registration
- I did leave in a few modern ish components
  - CommandBox to start the server
  - I am using .env files and cfconfig files for quick and easy datasource setup
  - I have a couple of CFCs for some business logic
- I have to say - it was PAINFUL to build this after being spoiled by ColdBox

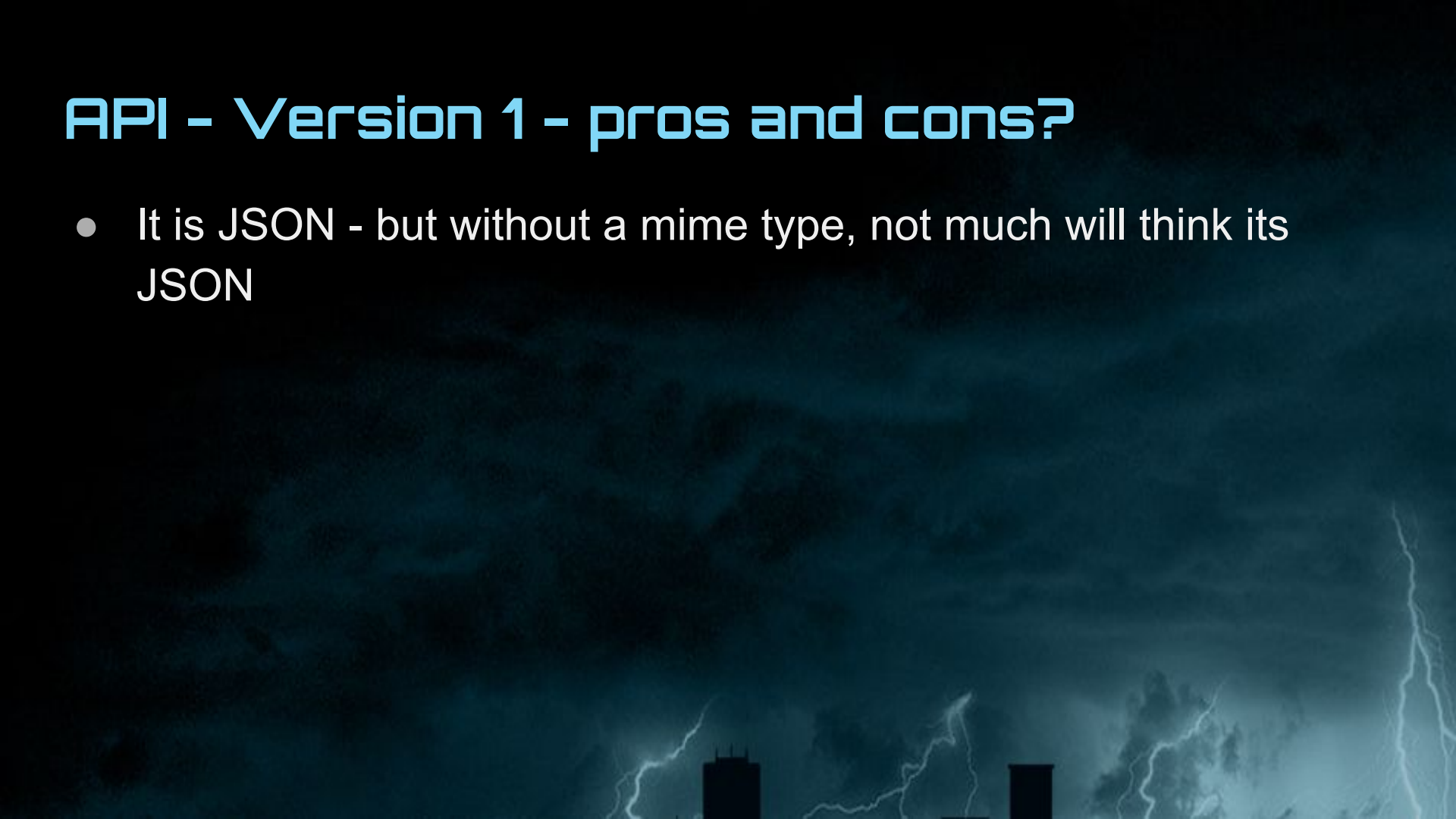
API - Version 1

- Version 1, we put a CFM file in a folder (api/v1/users/index.cfm) and then return JSON from our cfm page
- Look in the browser: http://127.0.0.1:59636/api/v1/users/?userid=5

- Look in postman

API - Version 1 - pros and cons?

- It is JSON - but without a mime type, not much will think its JSON



API - Version 2

- Version 2, we put a CFM file in a folder (api/v1/users/index.cfm) and then return JSON from our cfm page
- We add a mime type this time
- Look in the browser: http://127.0.0.1:59636/api/v2/users/?userid=5
- Look in postman

API - Version 2 - pros and cons?

- It is JSON and with the mime type, now our browser extensions and postman process this response as JSON
- Don’t CFCs support API like responses already?

API - Version 3

- Let's use a CFC for our API endpoint
- In the browser:
  http://127.0.0.1:59636/cfcs/users.cfc/?method=get&userid=5
  This errors because there is no remote function called get
- Let's make a new CFC for versioning reasons
  http://127.0.0.1:59636/api/v3/users.cfc/?method=get&userid=5
  Look in the browser and postman

API - Version 3 - pros and cons?

- It's easy to make a CFC into an API returning CFC with the access="remote"

- Isn't WDDX a pain to work with?
- Can we get json back instead?
  - We can add returnformat=json to the function
  - We can set a param in our application for url.returnformat = json so it defaults to json instead of wddx.

So far so good?

What API stuff haven’t we covered yet?

So far so good?

What API stuff haven't we covered yet?

- Errors?

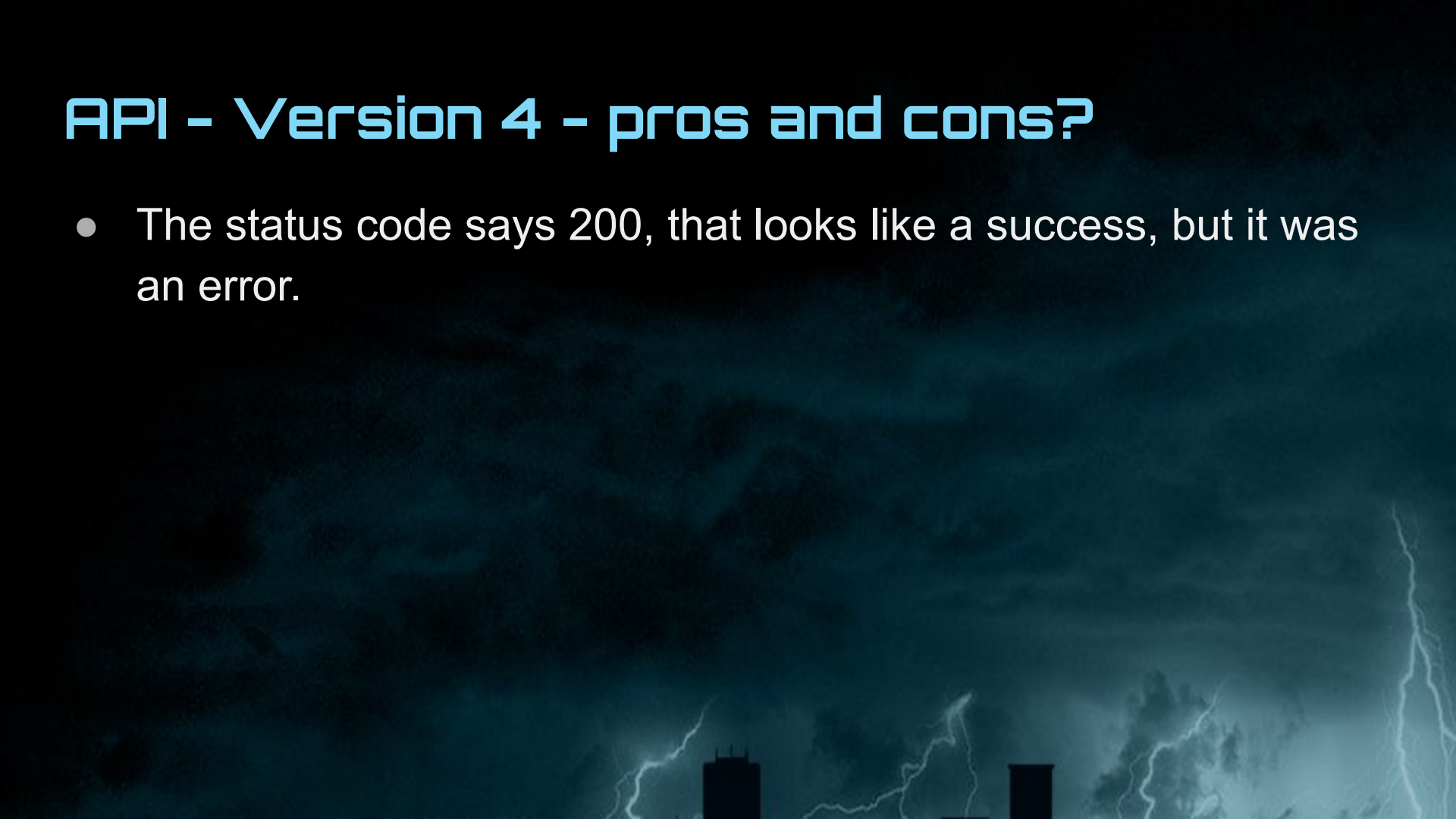
  http://127.0.0.1:59636/api/v3/users.cfc/?method=get&userid=12

API - Version 4

- Let's add some error handling into our user.cfc
  http://127.0.0.1:59636/api/v4/users.cfc/?method=get&userid=12

API - Version 4 - pros and cons?

- The status code says 200, that looks like a success, but it was an error.

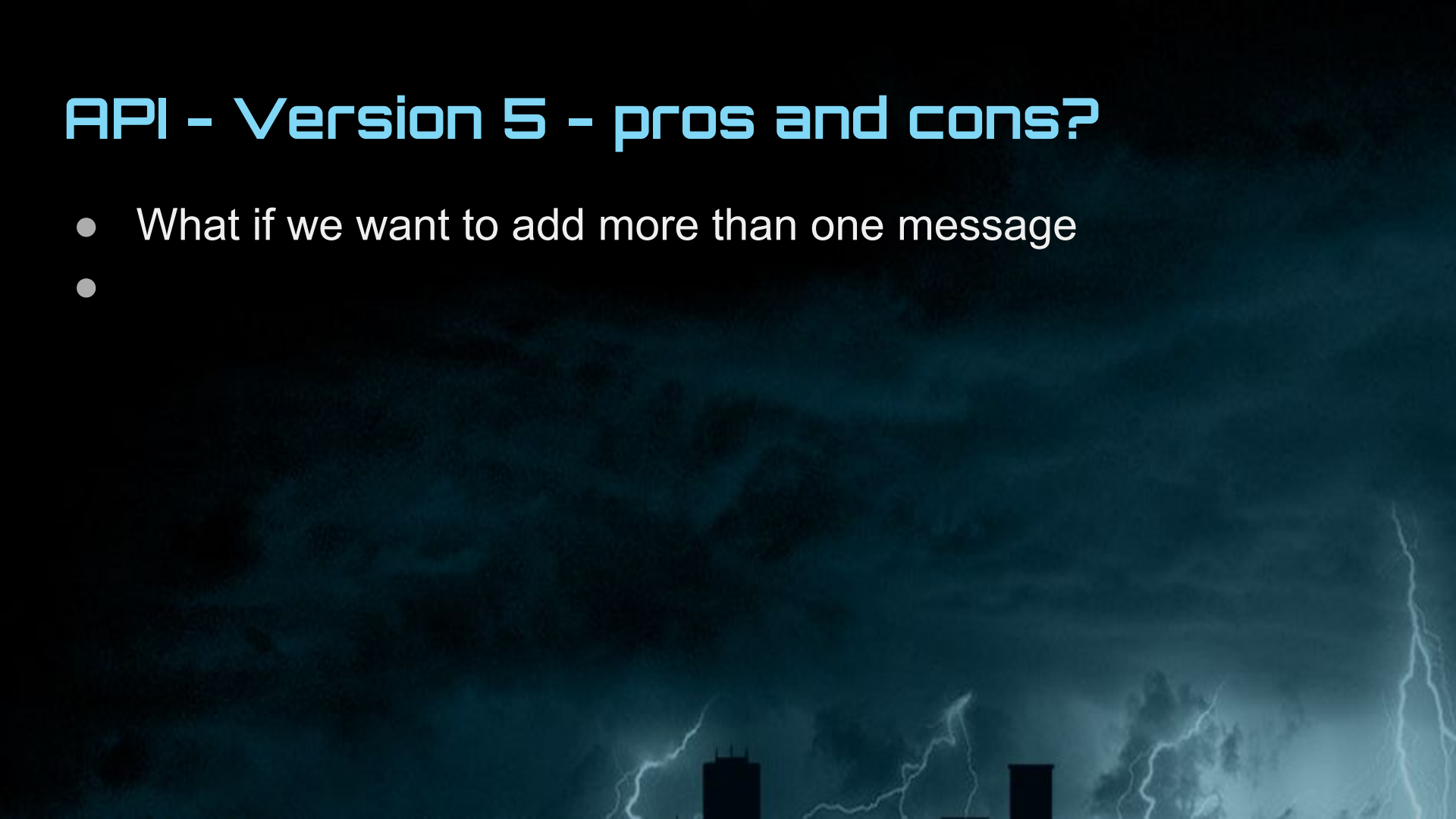


API - Version 5

- Let’s add some a here for status code and status text into our user.cfc
  http://127.0.0.1:59636/api/v5/users.cfc/?method=get&userid=12

API - Version 5 - pros and cons?

- What if we want to add more than one message
-



API - Version 6

- Let’s add an array of error messages into our user.cfc
  http://127.0.0.1:59636/api/v6/users.cfc/?method=get&userid=12

API - Version 6 - pros and cons?

- Consistency?
- Now we have to check the status code to know how to handle our response, sometimes it's a struct with values in it, sometimes it's an array, sometimes it has error messages
- What if we want messages with our data packet on a success?
- What if we want an easier way to look for an error?

API - Version 7

- Let’s add an a boolean for error for success and failures
- Lets add a data value to be consistent for success and failures
- http://127.0.0.1:59636/api/v7/users.cfc/?method=get&userid=12
  http://127.0.0.1:59636/api/v7/users.cfc/?method=get&userid=5

API - Version 7 - pros and cons?

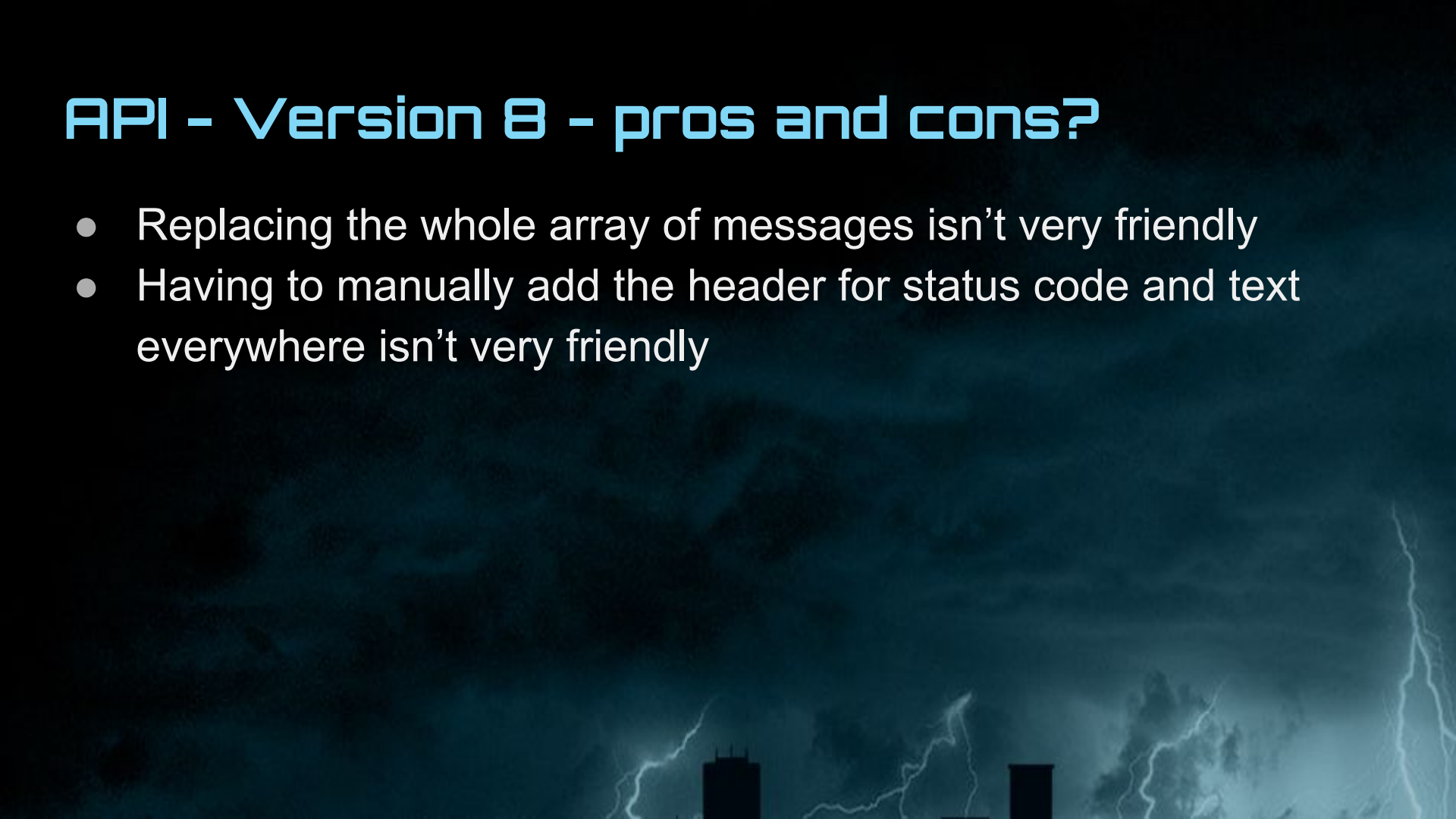
- Someone who doesn’t work on the API much, gets confused, are we calling our boolean success, or error?
- What if we want to add in something new, we’ll have to change our responses EVERYWHERE for that.

API - Version 8

- Let’s create a OBJECT to give our response consistency and structure, and make it more modern (chaining)
- http://127.0.0.1:59636/api/v8/users.cfc/?method=get&userid=12
  http://127.0.0.1:59636/api/v8/users.cfc/?method=get&userid=5

API - Version 8 - pros and cons?

- Replacing the whole array of messages isn't very friendly
- Having to manually add the header for status code and text everywhere isn't very friendly



API - Version 9

- Let’s add a helper method for addMessage which appends one or more messages
- Lets add support for the status code and text in the response
- http://127.0.0.1:59636/api/v9/users.cfc/?method=get&userid=12
  http://127.0.0.1:59636/api/v9/users.cfc/?method=get&userid=5

API - Version 9 - pros and cons?

- What about more Headers? CORS etc?
- What about more helpers for repetitive steps like:
  - setting status code and text at the same time.
  - Having to set a status text for regular status codes
  - Setting the error to true and adding an error message
- What about validation of your inputs?
- What about all of the boiler plate even with our response object and our helpers?

API - Version X

- Let’s add a helper for setErrorMessage
- Let’s add a helper for setStatusCode
- Lets add support for more headers
  http://127.0.0.1:59636/api/vx/users.cfc/?method=get&userid=12
  http://127.0.0.1:59636/api/vx/users.cfc/?method=get&userid=5

API - What else?

- Authentication (Logged In - Cookies / Sessions / JWT)
- Authorization (Permissions)
- HTTP Verbs GET vs POST vs DELETE etc
- Routing /users/8/rants for example

We can build all that... but...

- Why reinvent the wheel?
- There are frameworks that already solve these problems
- There are modules that already solve these problems
- There are templates that already solve these problems

Check some of these out and more in:

Building APIs - Part 2

In November with Charlie and the Online CF Meetup

Questions

Repo: https://github.com/gpickin/ocfm-2020-building-apis-part1

Database: In repo /workbench/20201014_soapbox.sql

Slides: In Repo in the readme.md file.