rangle.io

The Web Inverted

www.rangle.io

@rangleio

150 John St., Suite 501
Toronto, ON Canada
M5V 3E3

1-844-GO-RANGL

BUILDING AN ANGULARJS HACK STACK

Nick Van Weerdenburg

CEO, rangle.io

Yuri Takhteyev

CTO, rangle.io

rangle.io
The Web Inverted

RANGLE.IO is a lean/agile JavaScript development and consulting firm focused on building next-generation web and mobile applications for and with our clients.

CORE CAPABILITES

Continuous Delivery	User Experience Design
Ionic & PhoneGap	Responsive Design
AngularJS	Node.js
HTML5	JavaScript

The Company History

Founded
2013

20+
Successfully completed client projects

32+
developers & designers on the team, 45 total staff

Leaders
in HTML5 & JavaScript

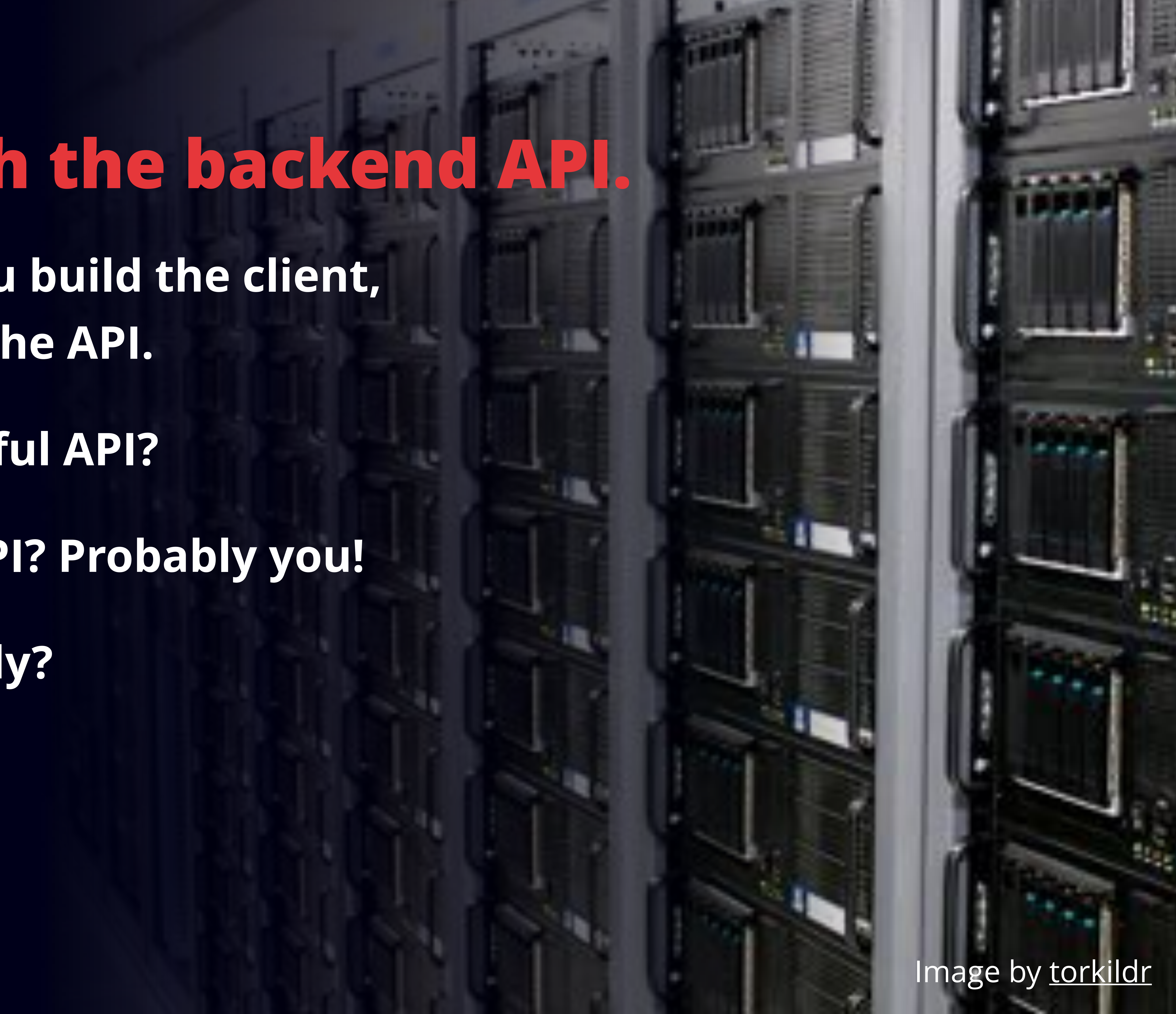
Specialists
in cross-platform application development

What's a Hack Stack?

A setup that allows you to work with a broken API delivered late.

OR

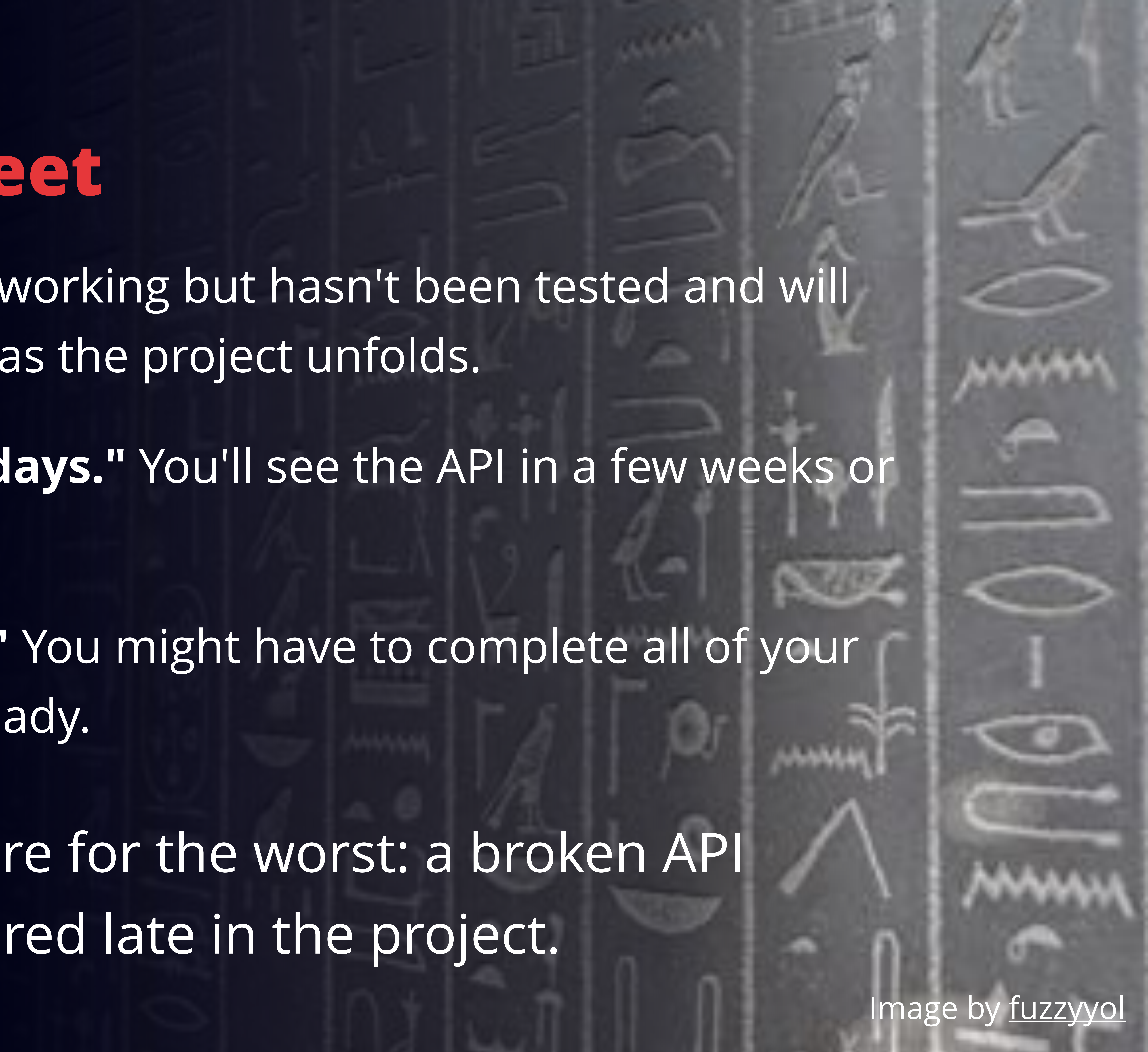
Allows you to build quickly with-out too much investment in a back-end



Working with the backend API.

- A common case: you build the client, someone provides the API.
- What's a good RESTful API?
- Who will test the API? Probably you!
- When will it be ready?

Image by torkildr



Translation Sheet

- **"It's working now."** It's working but hasn't been tested and will probably be redesigned as the project unfolds.
- **"We'll have it in a few days."** You'll see the API in a few weeks or months.
- **"We are working on it."** You might have to complete all of your work before the API is ready.

☛ Prepare for the worst: a broken API delivered late in the project.

Image by fuzzyyol

Why Such Poor Predictions?

- The existing API is low-level, and doesn’t fit needs for a REST API and client access
- The prior data-base schema doesn’t map well to the future REST API JSON document schema
- Legacy business rules are scattered through-out the prior view layer, resulting in a large effort to implement in new API

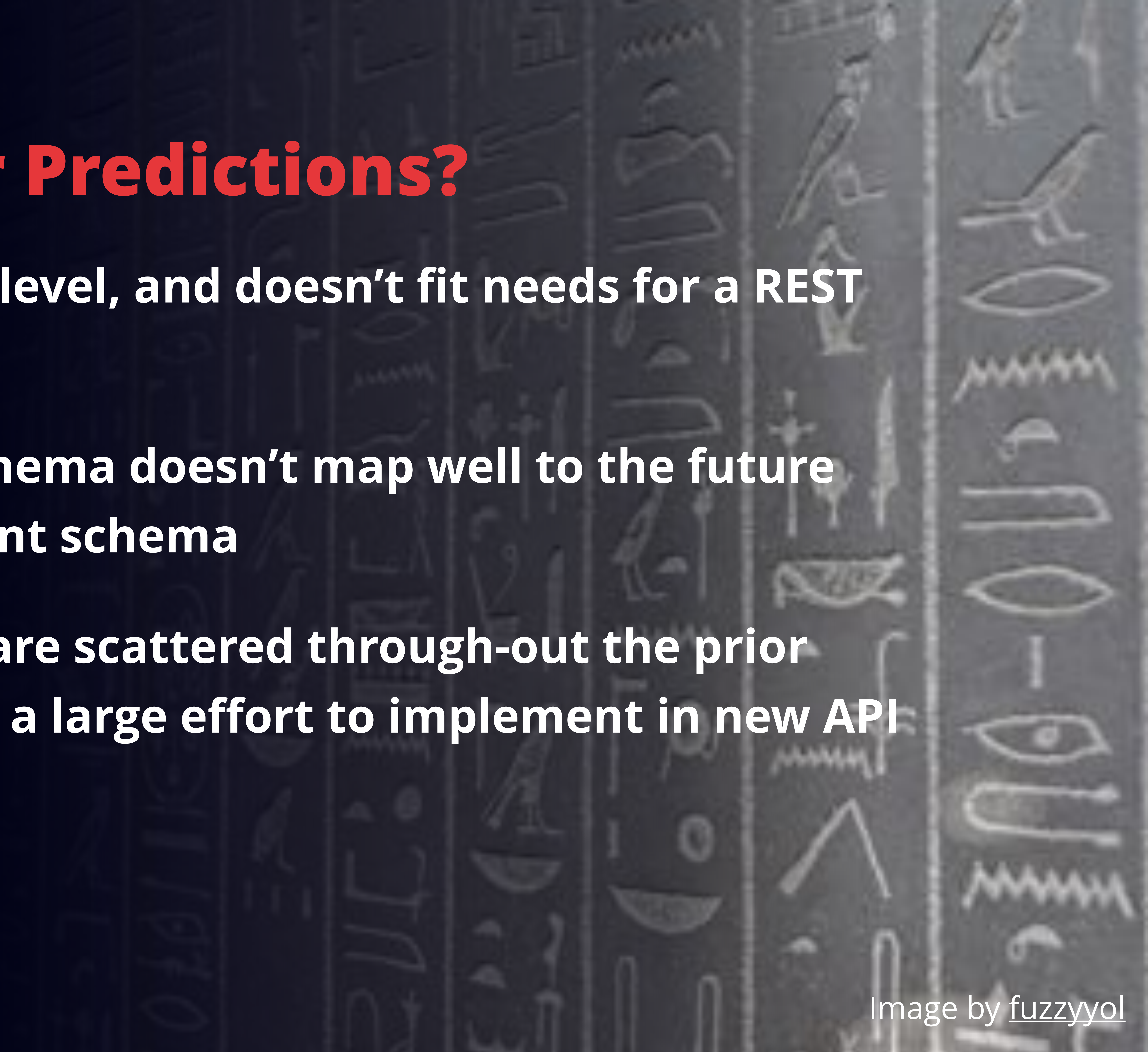


Image by fuzzyyol

“Hack Stack" to the Rescue

- A setup that allows you to work with a broken API delivered late.
- Or work on a quick prototype
- Not a library or a tool – rather, a set of best practices based on our experience.

Image by ejmc

Hack Stack 101

Getting Started on the Hack Stack

Document the API

- Allows you to start making assumptions.
- Can unearth problems that would later lead to delays.
- Apiary.io can be useful, but a Google Doc works fine.

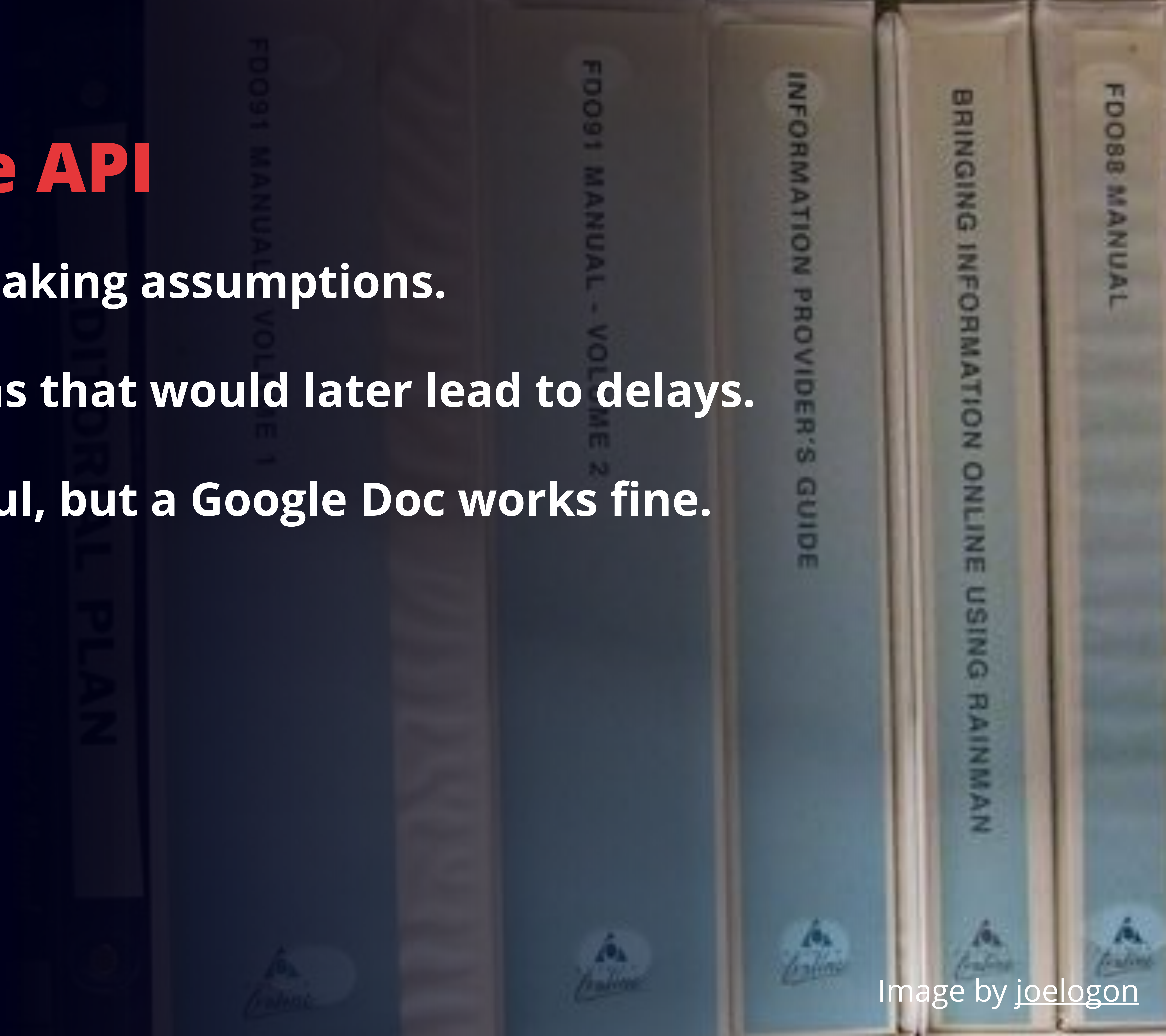

FDO91 MANUAL - VOLUME 1
FDO91 MANUAL - VOLUME 2
INFORMATION PROVIDER'S GUIDE
BRINGING INFORMATION ONLINE USING RAINMAN
FDO88 MANUAL

Image by joelogon

Mocking the API

- TDD: the first thing to try.
- Apiary.io etc: too limiting.
- A mock server: can work well, but expensive.
- Client-side mocking: our preferred solution.

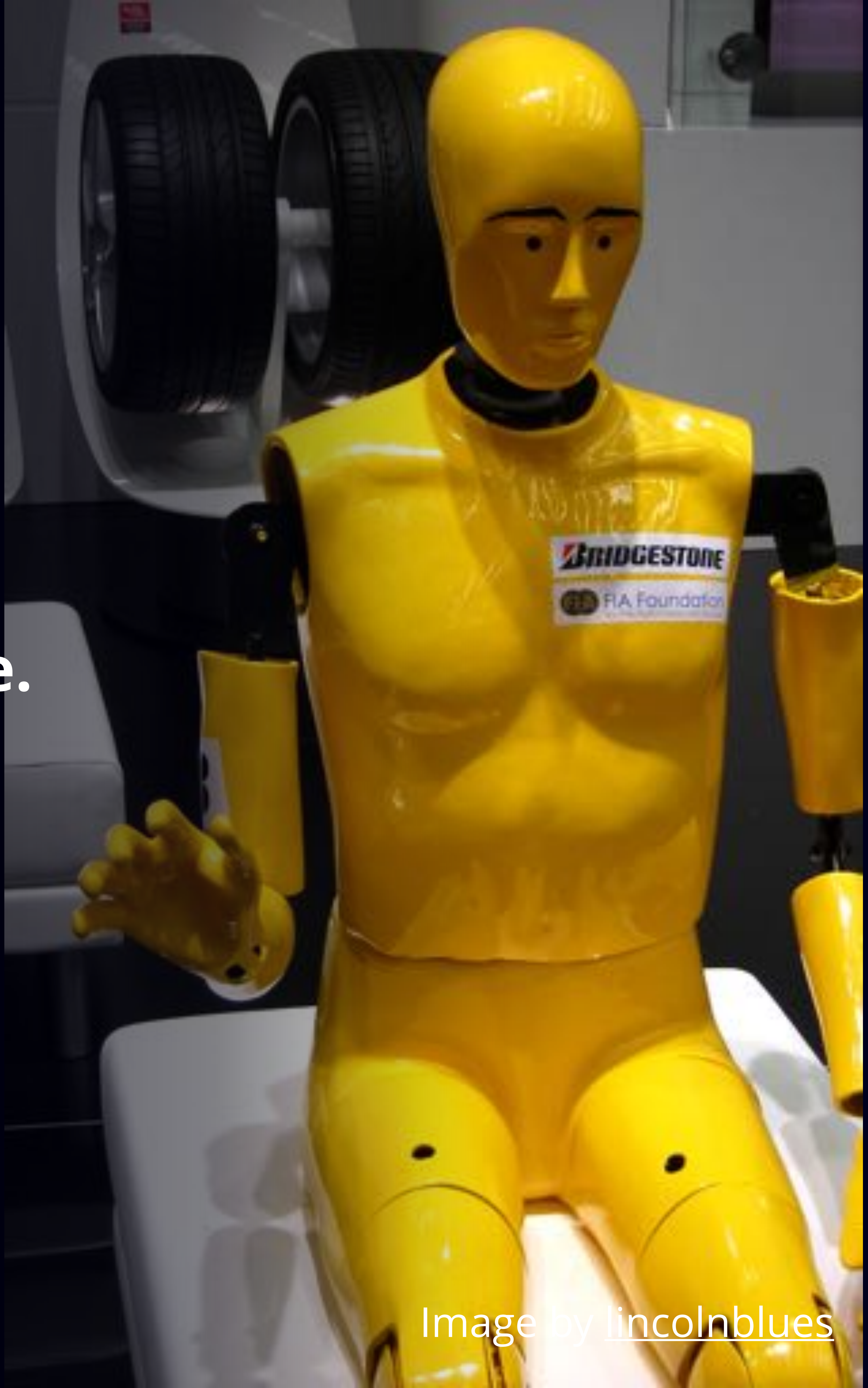


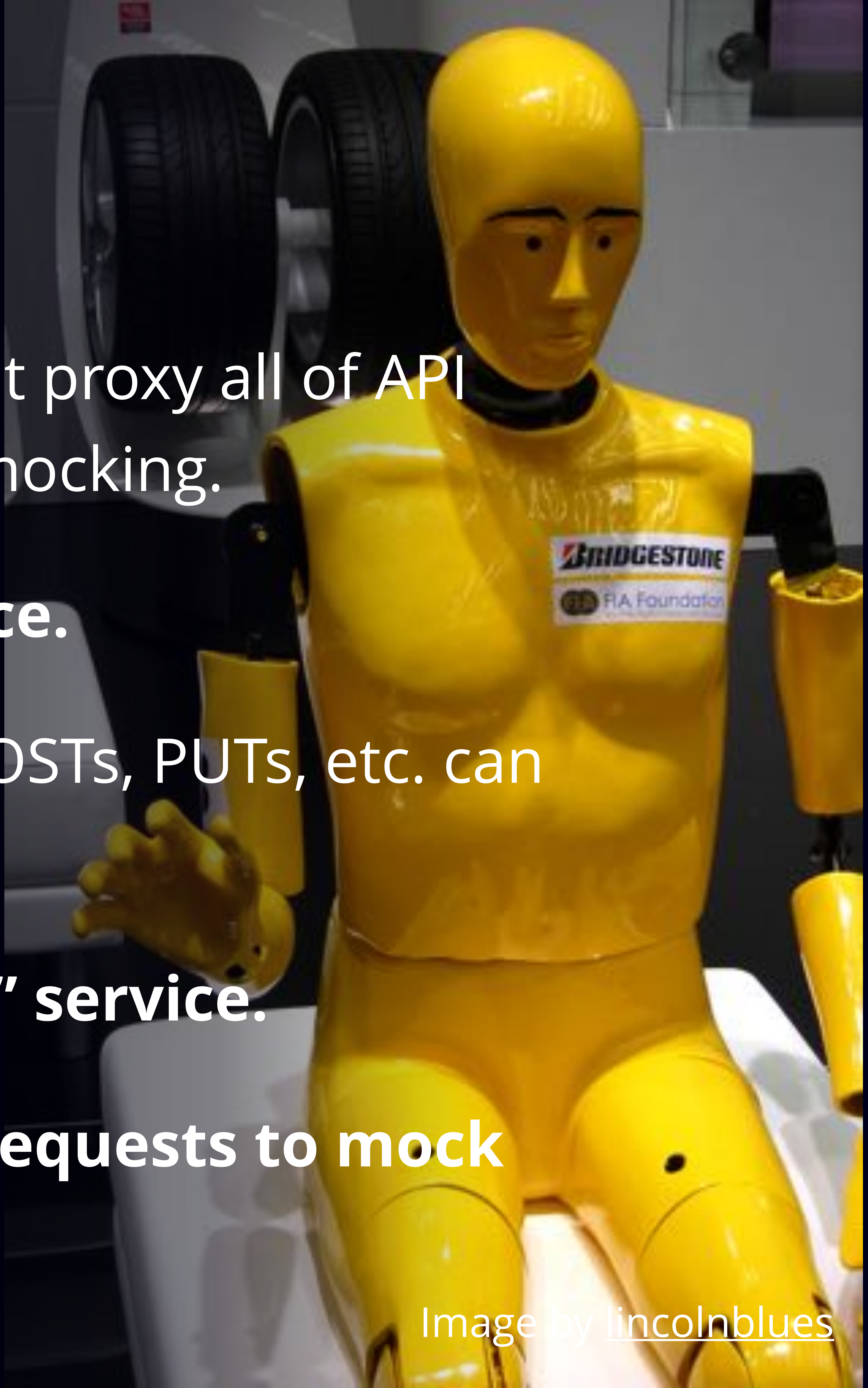
Image by lincolnblues

Scenario: The /tasks/ Endpoint

- We expect to eventually have a `/tasks/` endpoint that would give us REST access to tasks.
- We've agreed on what the returned JSON would look like.
- We've also agreed on how problems are going to be handled.
- Now we are waiting for the endpoint.

Image by lincolnblues

Client-Side Mocking



- **Leave our "tasks" service out of it.** But have it proxy all of API calls through "api" proxy that will be in on the mocking.
- **Put mock data into "tasksMocksData" service.**
- **Put mock logic into "tasksMocks" service.** POSTs, PUTs, etc. can modify data in memory.
- **Refactor common logic into "mockEndpoint" service.**
- **The "api" service (or similar) will direct API requests to mock services when appropriate.**

Image by lincolnblues

The “Real” Service

```
.service('tasks', function(api) {
  var service = this;
  service.getTasks = function() {
    return api.get('/tasks/');
  };
});
```

The Data – Keep It Separate

```
.value('tasksMockData', {
  BASIC_TASK_LIST: [{
    "taskId": "114a8455-3ea6-4d15-9e17-4f51c0728f9b",
    "ownerId": "ece21bd8-c99f-49fc-a1f0-5bc9bfb86ab9",
    "description": "Make green eggs and ham.",
    "date": "2013-03-04T21:42:36 +04:00"
  }, {
    "taskId": "1e387178-c22b-11e4-8dfc-aa07a5b093db",
    "ownerId": "28a74904-c22b-11e4-8dfc-aa07a5b093db",
    "description": "Fix the roof.",
    "date": "2014-07-17T20:42:36 +04:00"
  }]
});
```

Generating the Data

- http://www.json-generator.com/
- https://www.uuidgenerator.net/
- https://placekitten.com/

Image by lincolnblues

Trivial Mock Logic

```
.service('tasksMocks', function(tasksMockData, $q) {
  var service = this;
  var taskList = tasksMockData.BASIC_TASK_LIST;
  service.getTasks = function() {
    return $q.when(taskList);
  };
});
```

☛ Make sure to return promises.

Mock Likely Problems

- Slow connection: mock with a timeout.
- Dropped connection.
- Server-side errors.
- Loss of authentication.

Image by lincolnblues

Mocking Latency

```
.service('tasksMocks', function(tasksMockData, mockEndpoint, $q) {
  var service = this;
  var taskList = tasksMockData.BASIC_TASK_LIST;
  service.getTasks = function() {
    return mockEndpoint.waitRandomTime(80, 300)
      .then(function() {
        return taskList;
      });
  };
});
```

☛ Control latency with a constant.

Mocking Dropped Calls

```
.service('tasksMocks', function(tasksMockData, mockEndpoint, $q) {
  var service = this;
  var taskList = tasksMockData.BASIC_TASK_LIST;
  service.getTasks = function() {
    return mockEndpoint.waitRandomTime(80, 300)
      .then(function() {
        return mockEndpoint.maybeDropConnection(0.50);
      });
      .then(function() {
        return taskList;
      });
  };
});
```

When the API Arrives

One day, the real API does arrive.

Testing the API

- Test it with Postman.
- Maybe test it with `supertest`.

Image by torkildr

Dealing with Changes

- Run the API server locally. (Easier with Vagrant!)
- Control your schedule.
- Settup a toggle between client-side mocks and the real API.

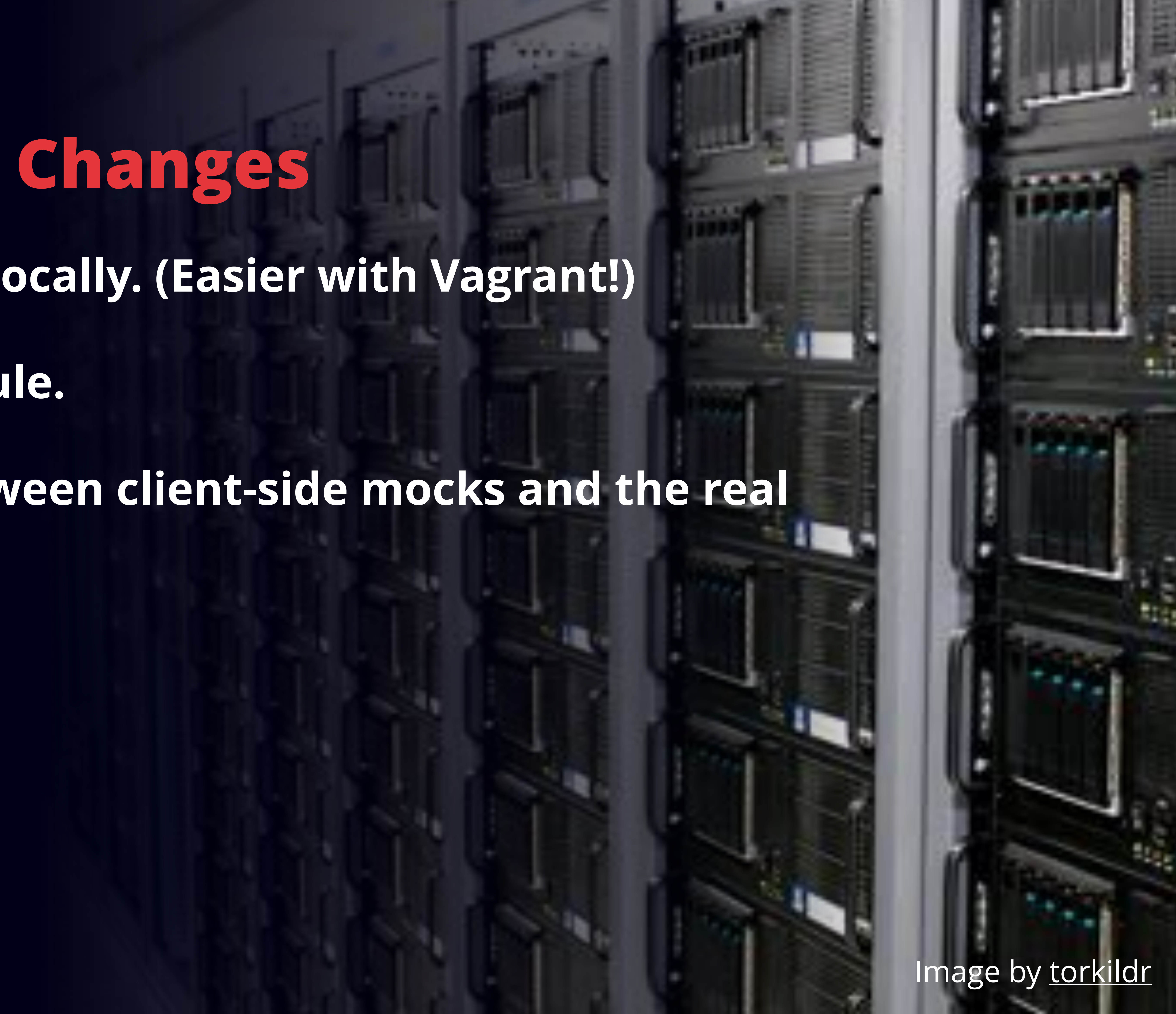
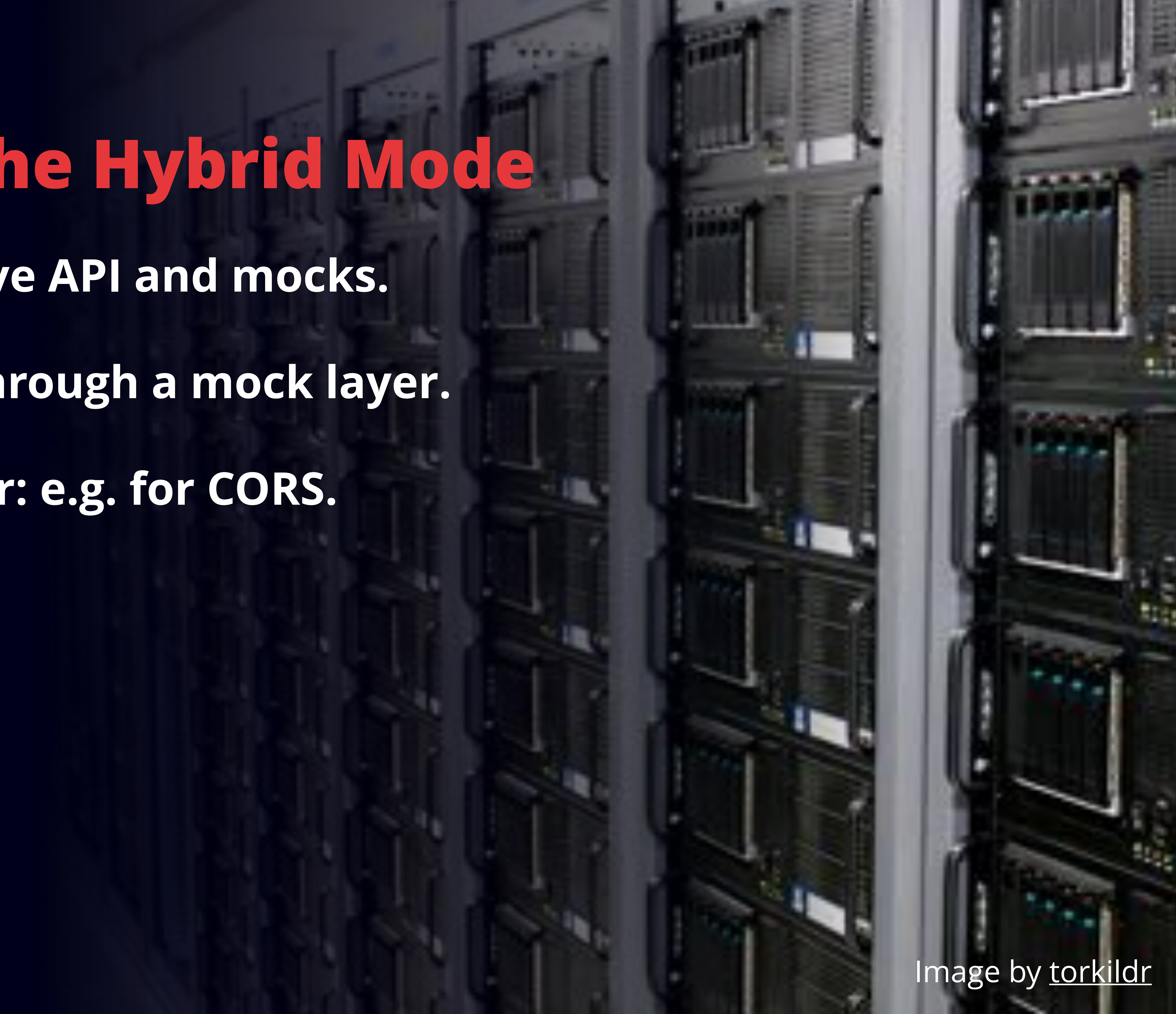


Image by torkildr



Working in the Hybrid Mode

- Mixing data from live API and mocks.
- Filtering API data through a mock layer.
- Using a proxy server: e.g. for CORS.

Image by torkildr

Mixing Real and Mock Data

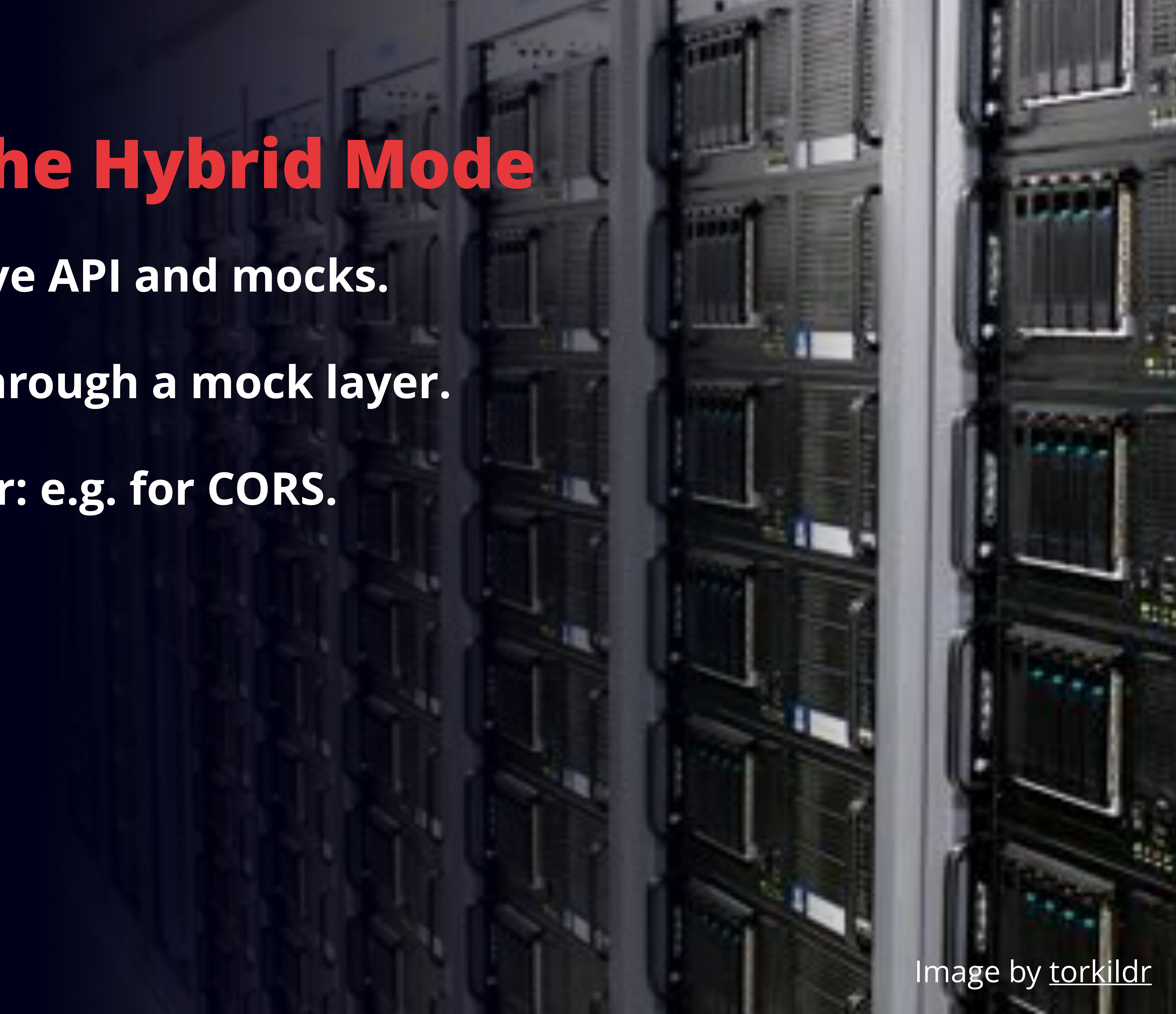
```
.service('tasksMocks', function(tasksMockData, mockEndpoint, $q,
  users) {

  var service = this;
  var taskList = tasksMockData.BASIC_TASK_LIST;
  service.getTasks = function() {
    angular.forEach(taskList, function(task, index) {
      task.ownerID = users[index % users.length].userId;
    });
    return $q.when(taskList);
  };
});
```

Proxying the Server

```
var express = require('express');
var request = require('request');
var app = express();
app.all('/api/(*)', function(req, res) {
  var url = 'https://api.example.com/v2/' + req.params[0];
  req.pipe(url).pipe(res);
});
app.listen(8080);
```

☛ Surely not for use in production.



Working in the Hybrid Mode

- Mixing data from live API and mocks.
- Filtering API data through a mock layer.
- Using a proxy server: e.g. for CORS.

Image by torkildr

THANK YOU!

Nick Van Weerdenburg
CEO, rangle.io

@n1cholasv

n1cholasv

Yuri Takhteyev
CTO, rangle.io

@qaramazov

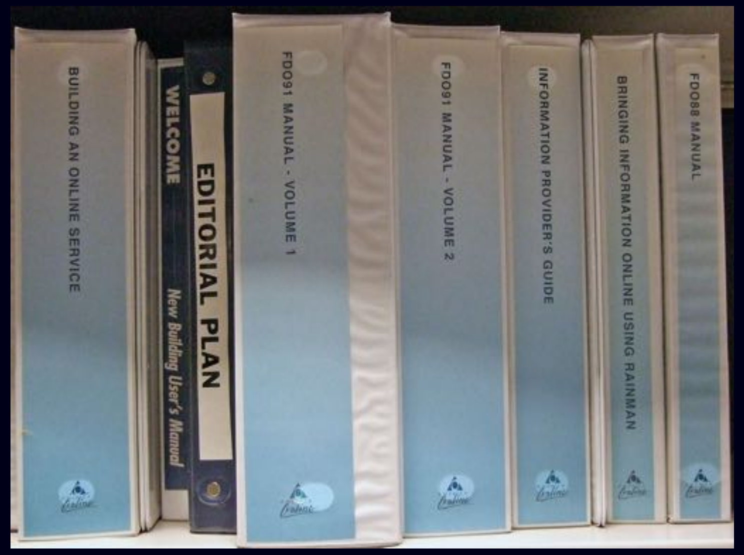
yuri

rangle.io
The Web Inverted

Image Credits

by torkildr

by fuzzyyol


BUILDING AN ONLINE SERVICE
WELCOME
EDITORIAL PLAN
FD001 MANUAL - VOLUME 1
FD001 MANUAL - VOLUME 2
INFORMATION PROVIDER'S GUIDE
BRINGING INFORMATION ONLINE USING RAINMAN
FD008 MANUAL
by joelogon

by ejmc

by lincolnblues