# Deep Dive Into Mura.js and the JSON API

# Who am I?

# Who am I?

- Matt Levine
- CTO of Blue River
- Lead Architect of Mura

# What am I talking about?

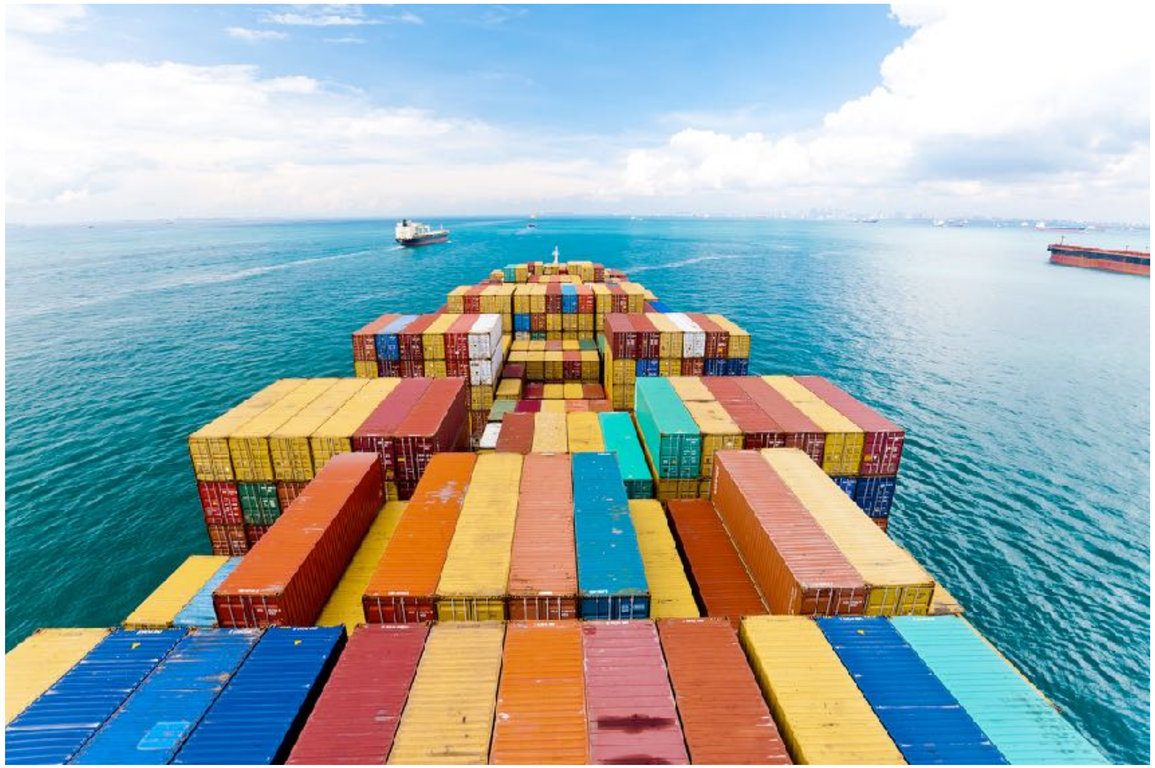
# What am I talking about?

- JSON/REST API
- Mura.js

# But first, a little background…

# The Monolithic Dev Stack is Dying!

# Because of Application Contianerization



# Application Containerization

**Application containerization** is an OS-level virtualization method used to deploy and run distributed **applications** without launching an entire virtual machine (VM) for each app. Multiple isolated **applications** or services run on a single host and access the same OS kernel.

http://searchitoperations.techtarget.com/definition/application-containerization-app-containerization

# Application Containerization

Leads to breaking applications into independent dedicated / specialized services.

# Application Containerization

In this new containerized world, modern applications need well refined APIs in order to allow these specialized services to interact with each other.

# Application Containerization

The technology that any particular service is built with is becoming less and less important.

# Application Containerization

But the ability for these discrete containerized services to interact with each other is critical.

# How Mura has Adapted

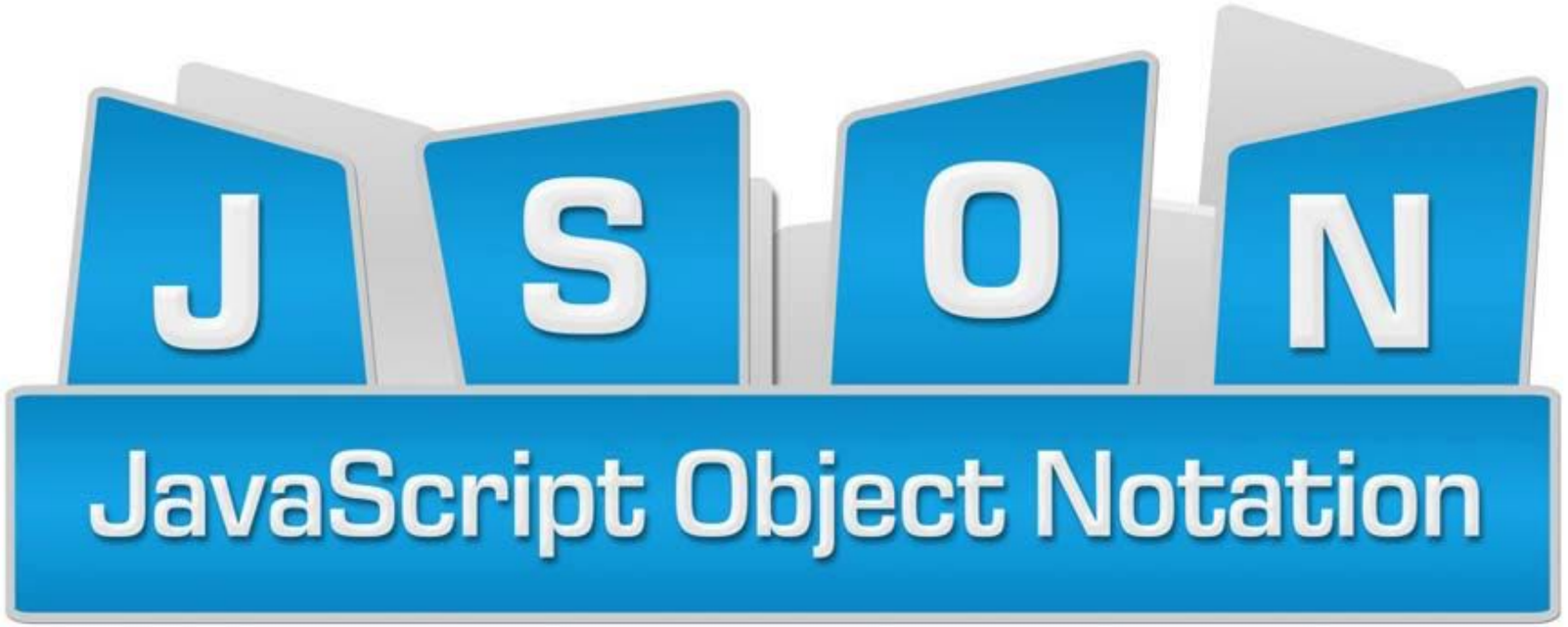
# How Mura has Adapted

- The rapid maturity of Mura's JSON/ REST API and Mura.js

# How Mura has Adapted

- The rapid maturity of Mura's JSON/ REST API and Mura.js
- The release of an official Mura image on DockerHub.

# JSON/REST API



# JSON/REST API

They are the same thing, but the JSON api uses cookies for state.

# JSON/REST API

The REST API can be great for service to service communications. This makes Mura's content easily accessible to other server based processes.

# JSON/REST API

Mura 7.0 introduced site level “Web Services” for REST authentication.

# JSON/REST API

## Supported REST authentication modes

- Basic
- OAuth2 (client_credentials)
- OAuth2 (authorization_code)
- OAuth2 (implicit)
- OAuth2 (password)

# JSON/REST API

Great introductory explanation of REST workflows:

https://alexbilbie.com/guide-to-oauth-2-grants/

# JSON/REST API

When doing traditional browser based JS we typically just use the JSON API. I'll mostly be using the JSON API in my examples in this presentation.

# How It Works

# How It Works

It's uses JSON:
"**J**ava**S**cript **O**bject **N**otation"

# How It Works

In it’s default state, there are only two kinds of objects.

# How It Works

In it’s default state, there are only kinds of objects.

- Entity Collection

# How It Works

In it’s default state, there are only kinds of objects.

- Entity Collection
- Entity Instance

# Entity Collection


```
{
	"apiversion": "v1",
	"data": {
		"pageindex": 1,
		"itemsperpage": 1000,
		"totalpages": 1,
		"totalitems": 1,
		"links": {
			"properties": "http://localhost:8080/index.cfm/_api/json/v1/default/Widget/properties",
			"self": "http://localhost:8080/index.cfm/_api/json/v1/default/?&method=findall&siteid=default&entityname=Widget&pageIndex=1",
			"entities": "http://localhost:8080/index.cfm/_api/json/v1/default"
		},
		"items": [...],
		"endindex": 1,
		"entityname": "Widget",
		"startindex": 1
	},
	"method": "findAll",
	"params": {
		"siteid": "default",
		"entityname": "Widget"
	}
}
```

# Entity Instance


```
{
	"apiversion": "v1",
	"data": {
		"widgetid": "F388399F-9DC6-FBF7-EFE46E460220AB89",
		"links": {
			"all": "http://localhost:8080/index.cfm/_api/json/v1/default/Widget",
			"properties": "http://localhost:8080/index.cfm/_api/json/v1/default/Widget/properties",
			"self": "http://localhost:8080/index.cfm/_api/json/v1/default/Widget/F388399F-9DC6-FBF7-EFE46E460220AB89",
			"entities": "http://localhost:8080/index.cfm/_api/json/v1/default"
		},
		"id": "F388399F-9DC6-FBF7-EFE46E460220AB89",
		"siteid": "default",
		"isnew": 0,
		"entityname": "Widget",
		"name": "My Widget"
	},
	"method": "findOne",
	"params": {
		"id": "F388399F-9DC6-FBF7-EFE46E460220AB89",
		"siteid": "default",
		"entityname": "Widget"
	}
}
```

# Self Describing

```
{
 “apiversion”:”v1”
}
```

The version of the API that handled the request

# Self Describing

```
{
 “data”:{…}
}
```

Only exists if the request did not throw an error. It contains the API request’s response.

# Self Describing

```
{
  “data”:{
     “links”:{
       “self”:”https://…”,
       “relateditems”:”https://…”
       }
    }
}
```

data.links provides related endpoints

# Self Describing

```
{
  “data”:{
    “id”:”CE15257E-AF7E-4B25-9FB4F58F0FE4700F”,
    “widgetid”:”CE15257E-AF7E-4B25-9FB4F58F0FE4700F”
    }
}
```

For entity instances the primary key value is normalized into data.id to remove the need to guess.

# Self Describing

```
{
 “error”:{
   “code”:”invalid_request”,
   “message”=“Insufficient Account Permissions"
 }
```

The error key only exists if an error has occurred

# Self Describing

When debuggingEnabled=true in your ./config/settings.ini.cfm the error key will contain the stacktrace serialized into JSON.

# Self Describing

```
{
 “method”:”findOne”
 }
```

Specifies the method that the API responded to.

# Self Describing

```
{
 “params”:{
   “args1”:”Hello”,
   “arg2”:”World”
  }
 }
```

The params keys species the arguments that the API responded to.

# Self Describing

The end results are a response that:

# Self Describing

The end results are a response that:

- Contains all the information that you need to resend the exact same request.

# Self Describing

The end results are a response that:

- Contains all the information that you need to resend the exact same request.
- By convention tells the consumer exactly how to transition to the current endpoint's related states (ie. links)

# Self Describing

## Built in Swagger support:

"Swagger is the world's largest framework of API developer tools for the OpenAPI Specification(OAS), enabling development across the entire API lifecycle, from design and documentation, to test and deployment."

# How to interact with the JSON API

# How to interact with the JSON API

Works like a normal REST-ful API

# How to interact with the JSON API

- FindOne: GET /_api/json/v1/:siteid/:entityName/:id
- FindNew: GET /_api/json/v1/:siteid/:entityName/new
- FindQuery: GET /_api/json/v1/:siteid/:entityName/$
- FindMany: GET /_api/json/v1/:siteid/:entityName/:ids/$
- Save: POST /_api/json/v1/:siteid/:entityName/
- Delete: DELETE /_api/json/v1/:siteid/:entityName/:id

# How to interact with the JSON API

Simple queries agains the API are easy

_api/json/v1/{siteid}/content?title=about

# How to interact with the JSON API

But complicated queries against the API can quickly become more complex.

/_api/json/v1/{siteid}/content?title[1]=eq^about&or[2]&title[3]=begins^contact

# How to interact with the JSON API

What's happening?

?title[1]=eq^about&or[2]&title[3]=begins^contact

# How to interact with the JSON API

You can use array syntax with the parameter names to indicate the order in which it should be applied:

?title**[1]**=……

# How to interact with the JSON API

This also allows you to query against the same property multiple times

?**title**[1]=about&or[2]&**title**[3]=begins^contact

# How to interact with the JSON API

And create parameter groupings:

?**openGrouping[1]**&title[2]=about&**or[3]**&title[4]=begins^contact**&closeGrouping[5]**

# How to interact with the JSON API

But what's going on here?

?title[1]=about&or[2]&title[3]**=begins^contact**

# How to interact with the JSON API

Just like you can use array syntax to impose order on your parameters.  You can prefix your parameter values to determine how the value is compared.

?title[1]=about&or[2]&title[3]**=begins^contact**

# How to interact with the JSON API

Supported parameter value prefixes:

- begins^
- ends^
- containsValue^
- in^
- notin^
- neq^
- gt^
- gte^
- lt^
- lte^

# How to interact with the JSON API

Sorting:

- Ascending: &sort=**title**
- Descending: &sort=**-title**
- Multiple: &sort=**-created,title**

# How to interact with the JSON API

You can limit the number of entities returned and control the pagination

- maxItems
- itemsPerPage
- pageIndex

# How to interact with the JSON API

You can limit the fields returned

/_api/json/v1/{siteid}/content?fields=title,summary

# How to interact with the JSON API

You can also tell the API to expand links. So instead of just a URL in the data.links object it can actually prefetch the expanded value.

/_api/json/v1/{siteid}/content?**expand=crumbs**
/_api/json/v1/{siteid}/content?**expand=all**&title=about

# How to interact with the JSON API

You can even do aggregate queries

?**count[1]=*&max[2]=created&min[3]=created&groupby[4]=type**

# How to interact with the JSON API

You can even do aggregate queries

?**count[1]=*&max[2]=created&min[3]=created&groupby[4]=title**

- Support options: sum, avg, count , max , min , groupby

# How to interact with the JSON API

You can even do aggregate queries

?**count[1]=*&max[2]=created&min[3]=created&groupby[4]=title**

- Support options: sum, avg, count , max , min , groupby
- Aggregate queries return generic “bean” entities

# How to interact with the JSON API

You can even do aggregate queries

?**count[1]=*&max[2]=created&min[3]=created&groupby[4]=title**

- Support options: sum, avg, count , max , min , groupby
- Aggregate queries return generic “bean” entities
- Aggregate queries against content nodes is restricted to administrative users when a site’s extranet is turned on.

# How to interact with the JSON API

You can allow aggregate queries against content nodes for non-administrative users when a site's extranet is turned.  You just add the setting AggregateContentQueries=true in your config/settings.ini.cfm and reload.

# How to interact with the JSON API

Special parameters for filtering content nodes

- liveOnly
- showNavOnly
- showExcludeSearch
- includehomepage

# How to interact with the JSON API

When not using REST via a Mura web service you must generate CSRF tokens for each save and delete action.

# How to interact with the JSON API

These tokens are single use only and must be generated with the value of the pimary key as it's context argument.

/_api/json/v1/{siteid}/generateCSRFTokens?context={primarykey}

# How to interact with the JSON API


```
{
    "apiversion": "v1",
    "data": {
        "csrf_token": "82BB2901D2A0E39C5DEA69416884D1D3",
        "csrf_token_expires": "18040300210722 8"
    },
    "method": "generateCSRFTokens",
    "params": {
        "siteid": "default",
        "context": "CE15257E-AF7E-4B25-9FB4F58F0FE4700F"
    }
}
```

# How to interact with the JSON API

One more very special API method

# How to interact with the JSON API

One more very special API method

/_api/json/v1/{siteid}/content/_path/{content_filename}

# How to interact with the JSON API

One more very special API method

/_api/json/v1/{siteid}/content/_**path/{content_filename}**

- It returns the rendered content of the corresponding content node.

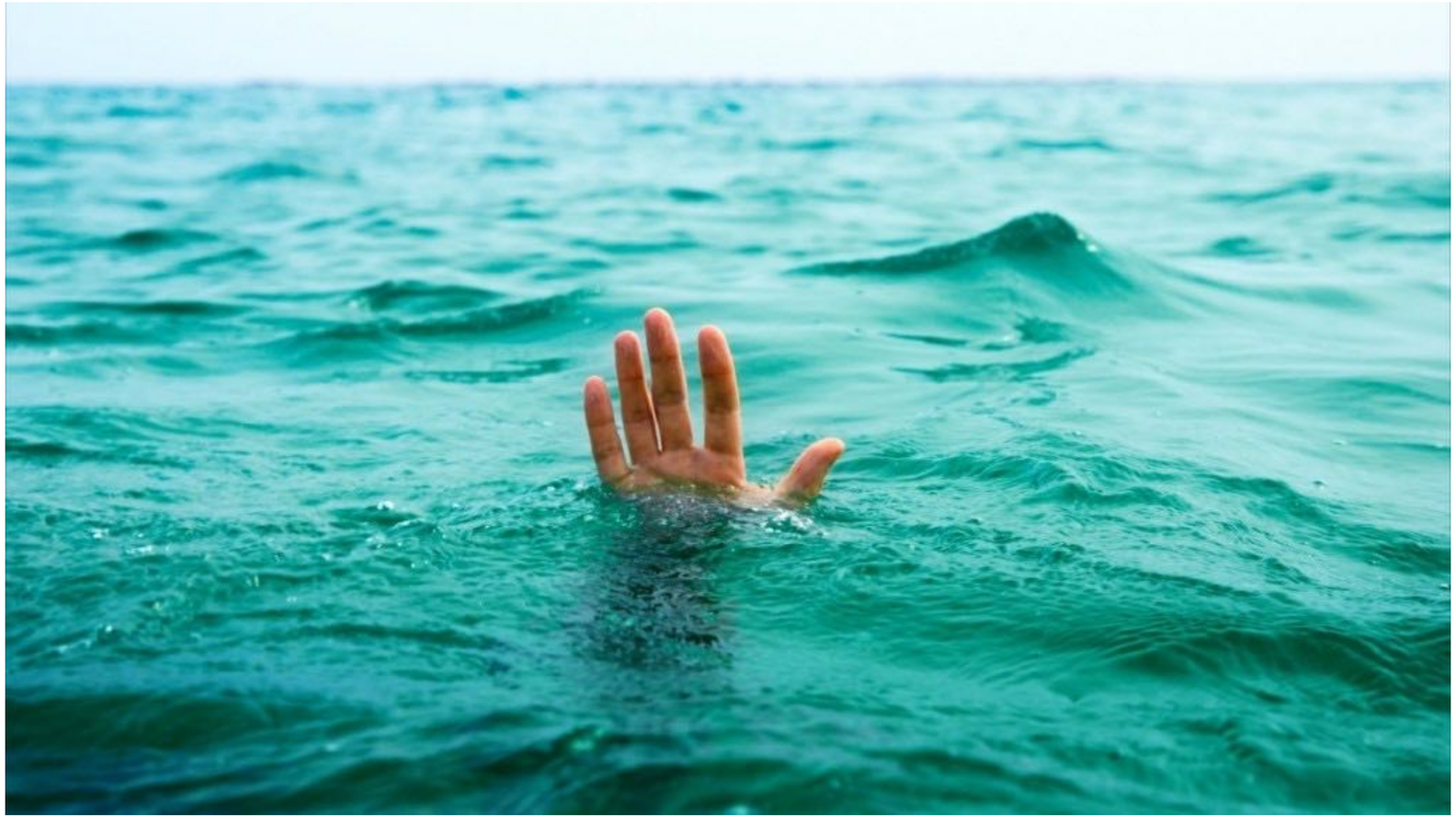
# How to interact with the JSON API

One more very special API method

/_api/json/v1/{siteid}/content/_**path/{content_filename}**

- It returns the rendered content of the corresponding content node in JSON format
- The data.body value is the result of content type rendering processes

# Are you still with me?



# Mura.js

# Mura.js

- A lightwieght jQuery replacement

# Mura.js

- A lightwieght jQuery replacement
- But it's other unique purpose is to provide seamless access to Mura's JSON/REST API

# Mura.js

- A lightwieght jQuery replacement
- But it's other unique purpose is to provide seamless access to Mura's JSON/REST API
- You can now forget everything you just learned ;)

# Mura.js

In Mura 7.1 Mura.js has been separated into a separate project

- https://github.com/blueriver/MuraJS

# Mura.js

And has been released as NPM

- https://www.npmjs.com/package/mura.js

# Mura.js

It’s best practice to always write any Mura.js snippets like this:

```
<script>
Mura(function(m){
 m('.my-class').on('click',
 function(e){
     m(this).html(‘clicked!’);
 });});
</script>
```

# Mura.js

This is because there is a new contentRenderer.cfc variable named **this.deferMuraJS**.

# Mura.js

This is because there is a new contentRenderer.cfc variable named **this.deferMuraJS**.

In Mura 7.1's default theme this is set to **true**.

# Mura.js

So any code in a Mura(fn) block will not execute until after the page's onReady event fires.

# Mura.js

So any code in a Mura(fn) block will not execute until after the page's onReady event fires.

```
<script>
Mura(function(){
 Console.log("I don't fire until everything is loaded!");
});
</script>
```

# Mura.js

This prevent's render blocking JS and allows you to do things like move a jQuery embed to your page footer.

```
<script>
Mura(function(){
  // use jQuery
});
</script>
```

# Mura.js

Remember this?

/_api/json/v1/{siteid}/content?title=about

# Mura.js

With Mura.js you can do this:

```
<script>
Mura(function(){
	Mura.getFeed('content')
	.where().prop('title').isEQ('about')
	.then(collection=>{
		collection.forEach(item=>{
			console.log(item.getAll());
		});
	});
});
</script>
```

# Mura.js

And this…

/_api/json/v1/{siteid}/content?title[1]=eq^about&or[2]&title[3]=begins^contact

# Mura.js

Now this.

```
<script>
Mura(function(){
	Mura.getFeed('content')
	.where().prop('title').isEQ('about').orProp('title').beginsWith('contact')
	.then(collection=>{
		collection.forEach(item=>{
			console.log(item.getAll());
		});
	});
});
</script>
```

# Mura.js

Supported parameter value prefixes:

- begins^
- ends^
- containsValue^
- in^
- notin^
- neq^
- gt^
- gte^
- lt^
- lte^

# Mura.js

Are now:

- beginsWith(criteria)
- endsWith(criteria)
- containsValue(criteria)
- isIn(criteria)
- isNotIn(criteria)
- isEQ(criteria)
- isNEQ(criteria)
- isGT(criteria)
- isGTE(criteria)
- isLT(criteria)
- isLTE(criteria)

# Mura.js

And this…

?**openGrouping[1]**&title[2]=about&**or[3]**&title[4]=begins^contact**&closeGrouping[5]**

# Mura.js

## Is now this.

```
<script>
Mura(function(){
	Mura.getFeed('content')
	.where().prop('title')
		.openGrouping()
			.isEQ('about').orProp('title').beginsWith('contact')
		.closeGrouping()
			.then(collection=>{
				collection.forEach(item=>{
					console.log(item.getAll());
				});
			});
});
</script>
```

# Mura.js

And this…

?**count[1]=*&max[2]=created&min[3]=created&groupby[4]=type**

# Mura.js

Is now this.

```
<script>
Mura(function(){
	Mura.getFeed('content')
	 .aggregate('max','created').aggregate('min','created').aggregate('groupby','type')
	.then(collection=>{
		collection.forEach(item=>{
			console.log(item.getAll());
		});
	});
});
</script>
```

# Mura.js

## Sorting:

```
<script>
Mura(function(){
	Mura.getFeed('content')
	 .sort('title','asc').sort('lastupdate','desc')
	.then(collection=>{
		collection.forEach(item=>{
			console.log(item.getAll());
		});
	});
});
</script>
```

# Mura.js

With all of this there are three main objects:

# Mura.js

There are three main objects for accessing data:

- Entity Feed (query)
- Entity Collection
- Entity Instance

# Mura.js

Most of the previous data access examples have been base around the feed object.

# Mura.js

But you may have noticed this:

```
<script>
 collection.forEach(item=>{
    console.log(item.getAll());
 });
</script>
```

# Mura.js

Entity Collection:

- length()
- item(index)
- index(item)
- getAll()
- forEach(fn)
- sort(fn)
- filter(fn)
- map(fn)
- reduce(fn,initValue)
- each(fn)

# Mura.js

Then there's the Entity object:

```
<script>
 collection.forEach(item=>{
    console.log(item.getAll());
 });
</script>
```

# Mura.js

The Entity object can be used to directly load data:

```
<script>
 Mura.getEntity('content')
  .loadBy('title','about')
  .then(content=>{
   console.log(content.getAll());
  });
</script>
```

# Mura.js

The Entity object provides seamless access to linked data:

```
<script>
  Mura.getEntity('content')
    .loadBy('title','about')
    .then(content=>{
      content.get('kids').then(kids=>{
          console.log(kids);
      });
    });
</script>
```

# Mura.js

Entity:

- exists()
- get(name,default)
- set(name,value)
- has(name)
- getAll()
- new()
- loadBy(prop,value,params)
- validate()
- hasErrors()
- getErrors()
- save()
- delete()

# Mura.js

The Entity object abstracts CSRF request token management:

```
<script>
 Mura.getEntity('content').loadby('title','about'),then(
     function(content){
          content.set('credits','Matt Levine').save()
          .then(content=>{
             //Do something
          });
    });
 </script>
```

# Mura.js

Remember that very special API method that returns rendered content?

/_api/json/v1/{siteid}/content/_path/{content_filename}

# Mura.js

```
<script>
Mura(function(){
 Mura.renderFilename('about')
  .then(content=>{
    console.log(content.getAll());
  });
});
</script>
```

# Mura.js

Mura.renderFilename() is the key to using Mura as a headless CMS.

# Mura as a Headless CMS

Mura.renderFilename() is the key to using Mura as a headless CMS.

# Mura as a Headless CMS

## Key Site Settings Properties

# Mura as a Headless CMS

## Key Site Settings Properties

- **Domain:** The domain of the remote site where the the content will be served on.

# Mura as a Headless CMS

## Key Site Settings Properties

- **Domain:** The domain of the remote site where the the content will be served on.
- **Is Remote:** true

# Mura as a Headless CMS

## Key Site Settings Properties

- **Domain:** The domain of the remote site where the the content will be served on.
- **Is Remote:** true
- **Remote Context:** The directory structure off of the remote site's web root where the site lives

# Mura as a Headless CMS

## Key Site Settings Properties

- **Domain:** The domain of the remote site where the the content will be served on.
- **Is Remote:** true
- **Remote Context:** The directory structure off of the remote site's web root where the site lives
- **Remote Port:** The port of the remote site

# Mura as a Headless CMS

## Key Site Settings Properties

- **Domain:** The domain of the remote site where the the content will be served on.
- **Is Remote:** true
- **Remote Context:** The directory structure off of the remote site's web root where the site lives
- **Remote Port:** The port of the remote site
- **Resource Domain:** The domain that Mura will use the access resource like css and js scripts that are dynamically loaded.

# Mura as a Headless CMS

If connecting from a different host, you need to make sure that you set the following response headers:

Access-Control-Allow-Origin
Access-Control-Allow-Credentials

# Mura as a Headless CMS

Access-Control-Allow-Origin: *

OR

Access-Control-Allow-Origin: https://youdomain.com:{port}

# Mura as a Headless CMS

Access-Control-Allow-Credentials: true

# Mura as a Headless CMS

A very simple demo

https://github.com/blueriver/MuraHeadlessAlpha

# Fin

# Questions?