JSON
(and the argonauts)

INTERACTIVE 3D VERSION WITH SOURCE CODE

http://r1.my/klmug/11/

Warning: Some Assembly Required

PRESENTED BY - **@m_smalley**

NOT THIS JASON

NOR THESE ARGONAUTS

RATHER THESE

IMPORTANT DATES

1974

RELATIONAL DATABASES

1989

WORLD WIDE WEB

2002 = JSON.org / 2009 = MongoDB.org

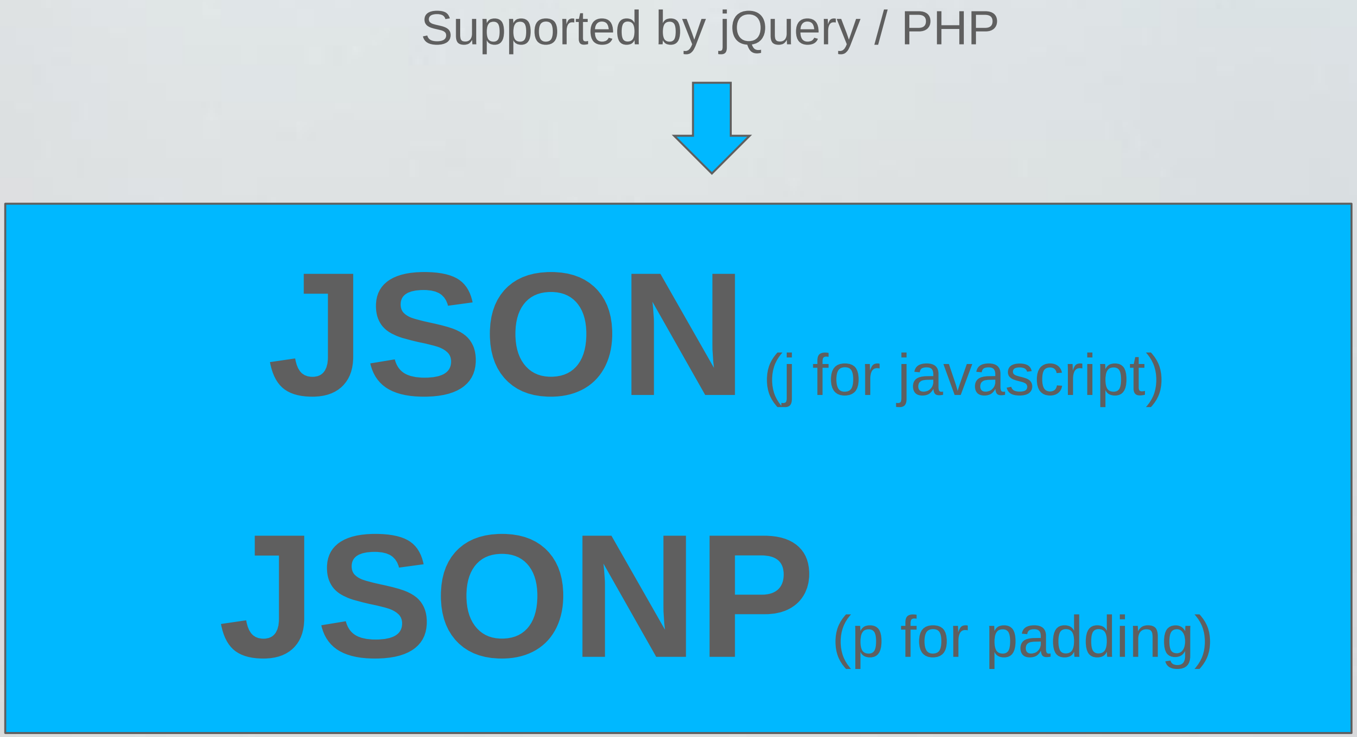
THE BIG THREE ONs

JSON (j for javascript)

JSONP (p for padding)

BSON (b for binary)

THE BIG THREE ONs



Supported by jQuery / PHP

JSON (j for javascript)

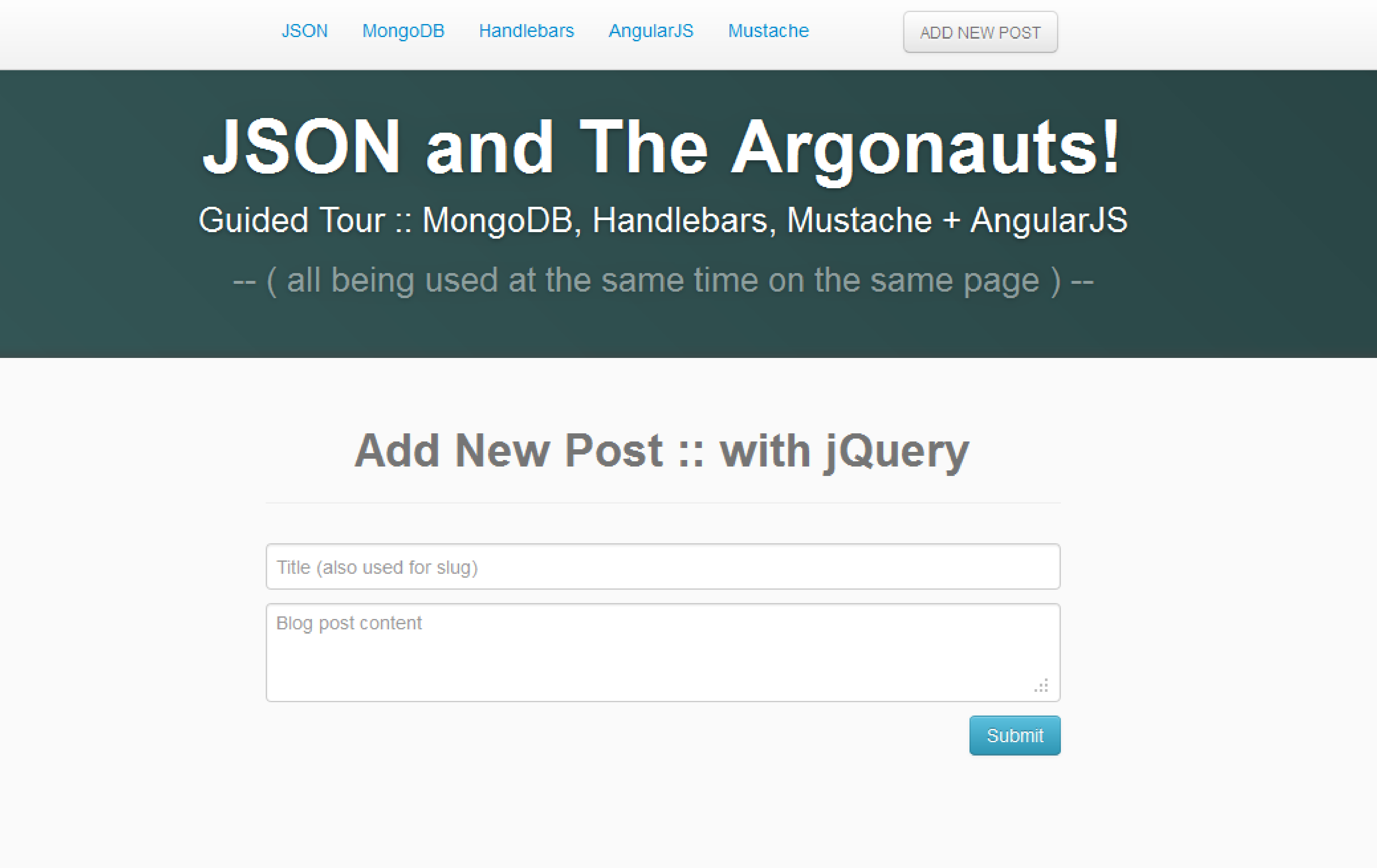
JSONP (p for padding)

BSON (b for binary)

```
[
  title: “Hello JSON World”,
  published: 1352229385,
  slug: “hello-world”,
  content: “<p>JSON has 6 data-types</p>”,
  tags: [“categories”,”test”,”etc”],
  comments: [
    { u: ”Jason”, c: “Thanks for sharing” }
  ]
]
```

```
callback_function([
  title: “Hello JSONP World”,
  published: 1352229385,
  slug: “hello-world”,
  content: “<p>JSONP is padded with a function</p>”,
  tags: [“categories”,”test”,”etc”],
  comments: [
    { u: ”Jason”, c: “Thanks for sharing” }
  ]
])
```

```
[
  “_id”: ObjectId(“50994e2e9bd64215053f7b22”),
  “title”: “Hello BSON World”,
  “published”: ISODate(“2008-02-03T00:00-00Z”),
  “slug”: “hello-world”,
  “content”: “<p>BSON has 11 data-types</p>”,
  “tags”: [“categories”,”test”,”etc”],
  “comments”: [
    { “u”: ”Jason”, “c”: “Including JS Code! :-)” }
  ]
]
```



JSON and The Argonauts!

Guided Tour :: MongoDB, Handlebars, Mustache + AngularJS

-- (all being used at the same time on the same page) --

Add New Post :: with jQuery

SOURCE CODE AVAILABLE – http://r1.my/klmug/11/

Before we can dive-in and take a closer look at the other Argonauts, we first need data.

In order to start adding data to MongoDB via AJAX and JSON whilst using PHP and jQuery, we will need a `<form>`. The source code for the " **add new post** " feature looks like this:

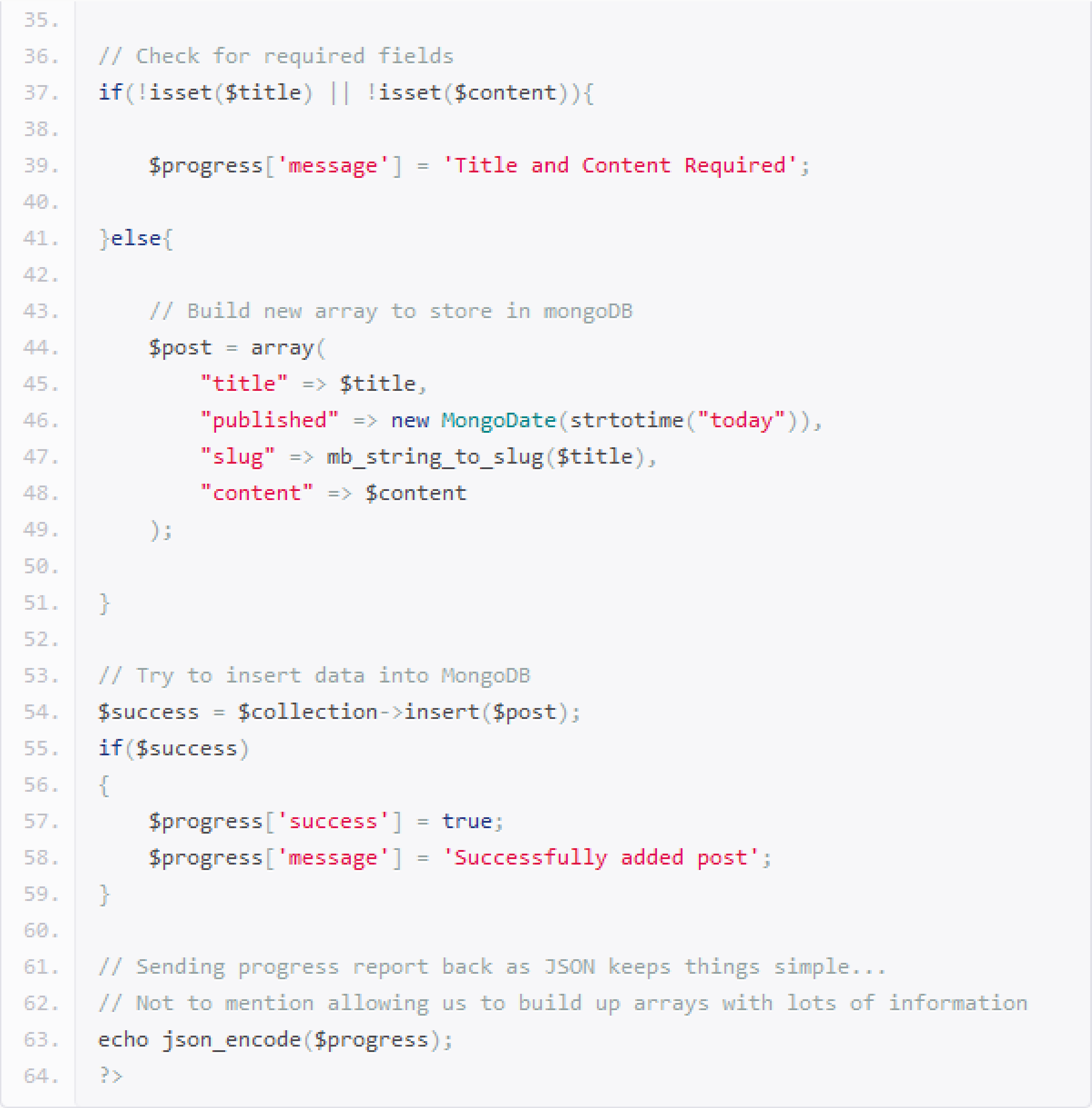
```
<form class="form-fluid">
    <input type="text" id="title" name="title" placeholder="Title" />
    <textarea id="content" name="content" placeholder="Content"></textarea>
    <button type="submit" class="btn btn-info pull-right">Submit</button>
</form>
```

And then we can apply some simple jQuery to it:

```
<script>
$(document).ready(function(){
    $('form.form-fluid').live('submit', function(e){
        var form = $(this);
        var data = $(form).serializeArray();
        var json = JSON.stringify(data);
        e.preventDefault();
        $.ajax({
            url: 'ajax/insert.php',
            dataType: 'JSON',
            type: 'POST',
            data: json,
            success: function(results){
                if(results.message) alert(results.message);
                if(results.success) $(form).find('input, textarea').val('');
            }
        });
    });
});
</script>
```

It's also worth noting that jQuery supports JSONp too - so assuming you have the mongoDB PHP-Drivers installed, `ajax/insert.php` file accessed above could then run the following:

```php
<?php
// Function for converting titles to slugs
function mb_string_to_slug($src)
{
    $src = preg_replace('/[^a-z0-9-]/', '-', strtolower(trim($src)));
    return preg_replace('/-+/', "-", $src);
}

// Array for sending back progress
$progress['success'] = false; $progress['message'] = 'Unable to add post';

// Connect to MongoDB
$m = new Mongo();
$db = $m->selectDB('argonauts');
$collection = new MongoCollection($db, 'posts');

// Quick and dirty way to ensure slugs are unique right out of the gate
$collection->ensureIndex(array('slug'=>1), array('unique'=>true));

// Decode JSON data into PHP array
if(isset($_POST['data'])) $data = json_decode($_POST['data']);
else $data = false;

// Create an array from the JSON data
$fields = false;
if(is_array($data)){
    foreach($data as $field){
        $fields[$field->name] = $field->value;
    }
}

// Could dump into database as is - but want to add a little structure
if(isset($data['title'])) $title = $data['title'];
if(isset($data['content'])) $content = $data['content'];
```



```php

// Check for required fields
if(!isset($title) || !isset($content)){

    $progress['message'] = 'Title and Content Required';

}else{

    // Build new array to store in mongoDB
    $post = array(
        "title" => $title,
        "published" => new MongoDate(strtotime("today")),
        "slug" => mb_string_to_slug($title),
        "content" => $content
    );

}

// Try to insert data into MongoDB
$success = $collection->insert($post);
if($success)
{
    $progress['success'] = true;
    $progress['message'] = 'Successfully added post';
}

// Sending progress report back as JSON keeps things simple...
// Not to mention allowing us to build up arrays with lots of information
echo json_encode($progress);
?>
```

Handlebars :: Minimal Templating

To be honest, Handlebars was one the first of this new generation of templating systems that I used - for it sounded perfect. A fork of Mustache.js that allowed for a bunch of extra cool things that others could not do...

Source Code:

Separated into 3 blocks here to prevent parent braces from spoiling PrettyPrint styling :p

```
{{#posts}} // The hash essentially starts a foreach
```

```
<section>
    <h1>
        <a href="{{slug}}">{{title}}</a>
        <label class="label label-info pull-right">{{date}}</label>
    </h1>
    {{{content}}}
</section>
```

```
{{/posts}} // The end of the foreach
```

In the live example, Handlebars.js gets it's data via the following PHP (in the header):

```
// Connect to MongoDB
$m = new Mongo();
$db = $m->selectDB('argonauts');
$collection = new MongoCollection($db, 'posts');

$results = array();
$all = $collection->find();
foreach($all as $result){
    $result['date'] = date('D / M / Y', $result['published']->sec);
    $results[] = $result;
}
```

Which is used with the following spaghetti-based mishmash:

```
<script>
var handlebars_data = new Object();
handlebars_data.posts = new Array();
<?php foreach($results as $result){ ?>
    handlebars_data.posts.push({
        title: "<?php echo $result['title']; ?>",
        content: "<?php echo $result['content']; ?>",
        date: "<?php echo $result['date']; ?>",
        slug: "<?php echo $result['slug']; ?>"
    });
<?php } ?>
var source = $("#handlebars-controller").html();
var template = Handlebars.compile(source);
var html = template(handlebars_data);
$("#handlebars-controller").html(html);
</script>
```

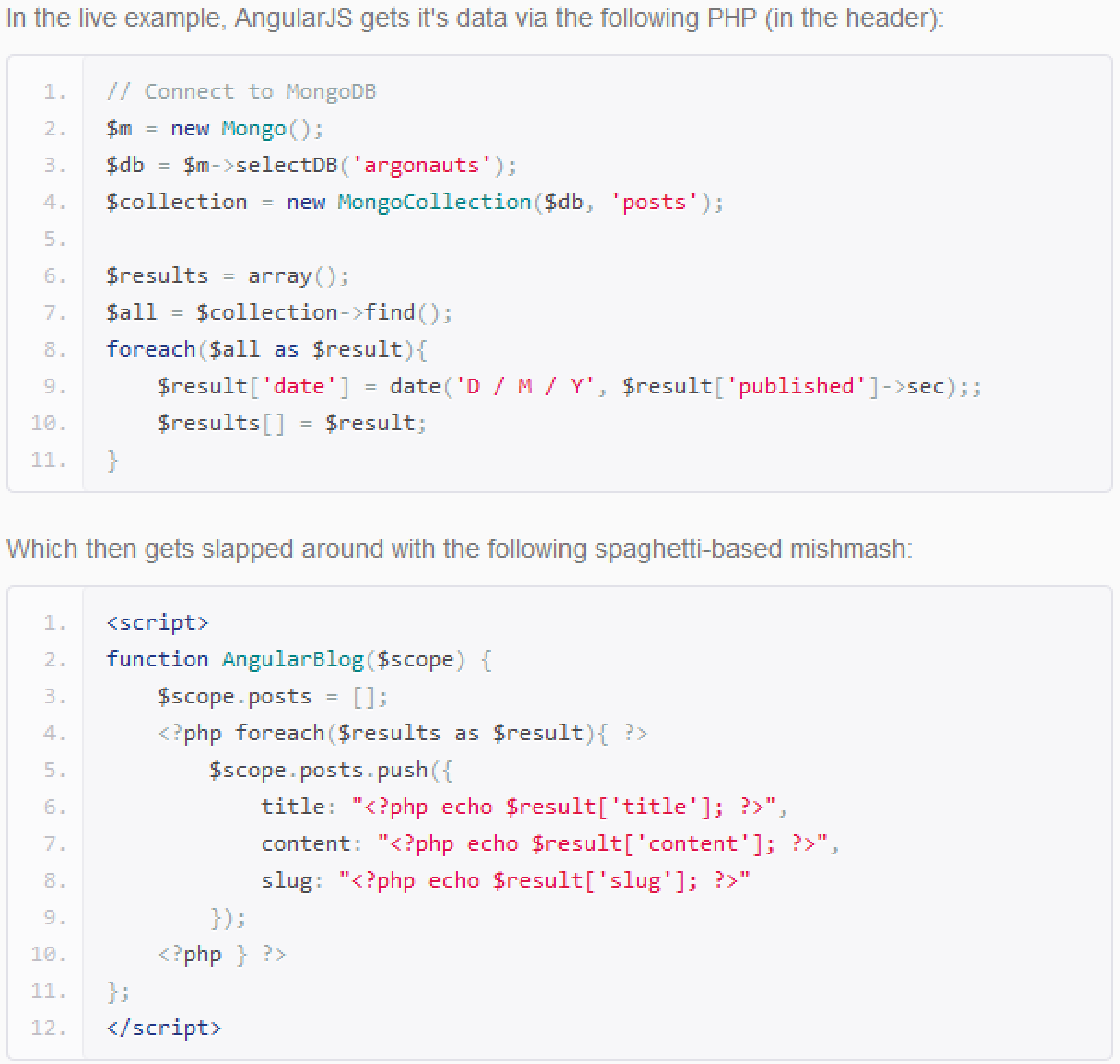
AngularJS :: Events on Steroids

We do not have the time to cover AngularJS events here, but at it's core, it is also a template-engine that utilizes inline data-attributes similar to Bootstrap.

Source Code:

Once again remember to replace the square braces with curly ones!

```
<div class="container-narrow" ng-controller="AngularBlog">
    <section ng-repeat="post in posts">
        <h1>
            <a href="[[post.slug]]">[[post.title]]</a>
            <label class="label label-info pull-right">[[post.date]]</label>
        </h1>
        <div ng-bind-html-unsafe="post.content"></div>
    </section>
</div>
```



In the live example, AngularJS gets it's data via the following PHP (in the header):

```
// Connect to MongoDB
$m = new Mongo();
$db = $m->selectDB('argonauts');
$collection = new MongoCollection($db, 'posts');

$results = array();
$all = $collection->find();
foreach($all as $result){
    $result['date'] = date('D / M / Y', $result['published']->sec);;
    $results[] = $result;
}
```

Which then gets slapped around with the following spaghetti-based mishmash:

```
<script>
function AngularBlog($scope) {
    $scope.posts = [];
    <?php foreach($results as $result){ ?>
        $scope.posts.push({
            title: "<?php echo $result['title']; ?>",
            content: "<?php echo $result['content']; ?>",
            slug: "<?php echo $result['slug']; ?>"
        });
    <?php } ?>
};
</script>
```

Mustache :: Swings Both Ways

The one thing that AngularJS or Handlebars cannot offer is server-side templating - extremely important from the context of creating a blog as it means the search engines and other bot-like mechanisms that spawn the internet will have a hard time figuring out what's going on.

This is why I eventually ended-up with Mustache, as it allowed me to use the same template tags within my HTML to be processed both by the server before the page is loaded, and via AJAX using JavaScript to re-process the DOM.

So the markup (in this case) is exactly the same as used with Handlebars:

Again separated into 3 blocks to prevent parent braces from spoiling PrettyPrint styling :p

```
{{#posts}} // The hash essentially starts a foreach
```

```
<section>
    <h1>
        <a href="{{slug}}">{{title}}</a>
        <label class="label label-info pull-right">{{date}}</label>
    </h1>
    {{{content}}}
</section>
```

```
{{/posts}} // The end of the foreach
```

In order to use Mustache on the server-side with PHP - we can do something like this:

```php
<?php
include('inc/class.mustache.php');
$m = new MustachePHP();
$template = file_get_contents('inc/template.mustache.php');
$mustache = $m->render($template, $results);
echo $mustache;
?>
```

The end results would get rendered on the server with plenty of SEO juice - as follows:

Hello Kuala Lumpur

Sun / Nov / 2012

Glad to celebrate our one year anniversary. Hope you enjoy the cake!

Hello San Fran

Sun / Nov / 2012

Thank you Francesca for the opportunity to speak and helping to organize this meet-up! :-)

LEARN MORE ABOUT

MongoDB

Kuala Lumpur MongoDB User-Group:
http://facebook.com/groups/klmug/

-- or stalk me on twitter --

@m_smalley