Website Development Using JavaScript Templates and CDN

An alternative solution with sample code, using Full-Stack JavaScript

# Abstract

The aim of this document is to present an alternative solution-sample to web-development using full-stack Javascript. The solution includes JSON data served by a backend process (server), UI templates using Mustache, and the use of a content delivery network (CDN) for caching front end scripts. Amazon Cloudformation is used in the sample but it could be any CDN.

# Introduction

Javascript full-stack development can include client frameworks like AngularJS, ReactJS for the client side and Node on the server side serving up data to be consumed for display by the client. This is a common architectural pattern used and is illustrated below.

Data Store

AngularJS

Browser

NodeJS

NodeJS returns JSON in the above scenario usually, and could also be referred to as the API server. REST is usually the intended ‘architectural-style’.

As an alternative solution:

* We could still have the JSON REST API server running on NodeJS.
* For the data-store, while Mongo DB is preferred; though for this sample, we just use text files containing JSON formatted data.
* Again REST is the intended architectural style.
* For the front-end, plain JS with JQuery has been used – though this is just for demo. The main thing here is that it is hosted on a CDN. The advantage here is the CDN servers tend to be closer to the client location, hence improving n/w download performance/speed.
* Also templates are used to process/filter JSON data. This saves a bit of ‘manual’ coding. The trick here is to keep updated on the features templates offer which are constantly improving. For example, if you only want to display the name of products, rather than looping through the list in code to get product names, templates readily handle it for you – all the template needs is a reference to the file containing JSON data.

CDN for static files

Browser

Node server serving JSON data

2

2

1

|  |
| --- |
| 1. Browser gets 'page stub’ |
| 2. Browser gets static data |
| 3. Browser gets json data and templates (todo – can template be moved to CDN?) |
| 4. Browser assembles final code |

Unfortunately I’m not too clear on how Mustache/Node organises stuff under the hood.

# Description of sample code

## Main

The ‘main’ file is productApplication.js. This file contains routes to the different URL’s used in the application. The main routes are described below:

External routes (end-user) are:

/pages/home

* This list products available in the store.

/pages/products/<productName>

* This is accessible by clicking on links on the home page, which lists products by name. Ideally of course, each product should map to a resource in true rest spirit. I have just returned json from the product selected by parsing the json database (products.json file). I haven’t use templates in this case as I could not pass the name of the product to the template. It would seem that the proper way would be to have a file for each product (e.g. Product named aa would be described in a file called aa.json – Also the templating engine js files on the CDN seem to only resolve .json files on the nodeJS server – Hence almost forcing one to use REST as intended.)

/products.json (this is more of a data verification for testers)

Internal routes used by CDN and the templating engine have the following prefixes:

/templates/ and /content/

## Page stub

All files under /pages/ use the templating engine – mustache in this case. I have only one page that lists all products at <http://localhost:8080/pages/home>. Here ‘home’ maps to ‘home.js’ file. More URLs can be added using ‘home’ as an example. All of them use the below ‘page-stub’. Pls note that the {{PAGE\_NAME}} mustache tags highlighted below are replaced with URL resource name (which would be ‘home’ in the case of http://localhost:8080/pages/home)

*Page stub: (basic.html)*

<!DOCTYPE html>

<html>

<head>

<title>Photo Album</title>

<!-- Meta -->

<meta http-equiv="content-type" content="text/html; charset=utf-8" />

<!-- Stylesheets -->

<link rel="stylesheet" href="http://ec2-52-42-161-61.us-west-2.compute.amazonaws.com/site1/content/style.css"

type="text/css" />

<!-- Javascript -->

<script src="http://ec2-52-42-161-61.us-west-2.compute.amazonaws.com/site1/content/jquery-3.1.1.js"

type="text/javascript"></script>

<script src="http://ec2-52-42-161-61.us-west-2.compute.amazonaws.com/site1/content/**{{PAGE\_NAME}}.**js"

type="text/javascript"></script>

<script src="http://ec2-52-42-161-61.us-west-2.compute.amazonaws.com/site1/content/mustache.js"></script>

</head>

<body>

Loading... If this fails to disappaer in a second, then look in your

Javascript console for errors.

</body>

</html>

As, can be seen form the above html, the script tags point to content on the content server.

Also notice the {{PAGE\_NAME}} mustache tags – these are replaced by the page name from the URL.

Our home URL is <http://localhost:8080/pages/home>. The home.js file is on the content server. It contains a reference to the map JSON data and mustache templates. The final result is displayed on the browser.

*home.js*

var tmpl, // Main template HTML

tdata = {}; // JSON data object that feeds the template

// Initialise page

var initPage = function() {

// Load the HTML template

$.get("/templates/home.html", function(d){ **-🡪 1**

tmpl = d;

});

// Retrieve the server data and then initialise the page

$.getJSON("/products.json", function (d) { **-🡪 2**

$.extend(tdata, d.data);

});

// When AJAX calls are complete parse the template

// replacing mustache tags with vars

$(document).ajaxStop(function () {

var renderedPage = Mustache.to\_html( tmpl, tdata ); **-🡪 3**

$("body").html( renderedPage );

})

}();

1. home.js gets the template used by the UI to filter/format data (home.html)

2 home.js gets the json data from file (products.json)

3. Mustache templating engine then combines both and information is displayed on browser.

# Advantages of Templating/CDN solution

* Faster n/w download times, hence improving end user performance.
* Templating helps to easily filter and display data

# Disadvantages of Templating/CDN solution

* Using CDN, I wasn’t able to pass individual product name values to templates. I reckon this is intentional – as the user is forced to have individual product details in their own files. This would be as REST intended, wherein the resource is the URL. (The resource in its own json file would be picked up by the template though)
* Templates (atleast mustache) don’t seem to be mature enough to filter rows – but this is related to the point above where probably that was the intent all along.
* CDN costs/complexity.

# Running code sample

(Assumptions includes that node is installed. Download code from: https://github.com/kkrishnan212/node\_PIP/tree/master/productsStore)

* npm install
* node productApplication.js

Code can be downloaded from: <https://github.com/kkrishnan212/node_PIP/tree/productApplication/productApplication>

# Testing

* This assumes you have the CDN set-up with static content. (In this case, all file under the content folder.
* Use the following URL on your browser

<http://localhost:8080/pages/home>

This should list products in store by name.

* Click on a product – This should give details of the product in json format. It would seem that the only way to use templates and CDN together in this case to format the data would be to have individual product data in its own file.

# CDN

I have used Amazon CloudFormation as the CDN. I used the LAMP template in CloudFormation.

I then copied over the files in the contet folder onto the LAMP server and the path where the files are in LAMP are referenced in the basic.html file.