International Research Journal of Engineering and Management Studies (IRJEMS)
Volume: 03 Issue: 04 | April -2019 ISSN: 1847-9790 || p-ISSN: 2395-0126

LOCAL DEAL FINDER

RONAK PATEL, BIRJESH DONGA, RITIK PATEL, SARITHA K

Final Year Btech.CSE, PIT Parul University, Vadodara, Gujarat, India

Abstract:

We developed a test framework and tools implemented as plugins for create and generate to book nearby hotels. We used our test environment to perform and we found important performance differences across the tested frameworks. These differences vary between the last print of the application when starting and its performance while already loaded. The architecture of Web applications has evolved in the last few years. The need to provide a native-like quality user experience has forced developers to move code to the client side (AngularJS). The dramatic increase in size of the AngularJS code was addressed first with the help of powerful libraries (jQuery) and more recently with the help of JavaScript frameworks. But although nowadays most Web applications use these powerful JavaScript frameworks, there is not much information about the impact on performance that the inclusion of the additional code will produce. One possible reason is the lack of simple, flexible tools to test the application when it is actually running in a real browser.

Introduction:

Local Deal Finder is a simply web application which is managed by Angular framework. It uses for hotel booking as well as to provide a platform to new start-ups. The main focus of this application is develop faster user-experience using this framework.

Angular JS is framework manage by Google, it help build responsive sites. Angular JS use to make a smooth web performance. Angular JS is a toolset for building the framework most suited to your application development. It is fully extensible and works well with other libraries. Every feature can be modified or replaced to suit your unique development workflow and feature needs.

Angular JS is a JavaScript framework. It can be added to an HTML page with a <script> tag. Angular JS extends HTML attributes with Directives, and binds data to HTML with Expressions. AngularJS extends HTML with new attributes.AngularJS is perfect for Single Page Applications (SPAs).AngularJS is easy to learn. The idea turned out very well, and the project is

now officially supported by Google AngularJS is a structural framework for dynamic web applications. It lets you use HTML as your template language and lets you extend HTML's syntax to express your application components clearly and succinctly. Its data binding and dependency injection eliminate much of the code you currently have to write. And it all happens within the browser, making it an ideal partner with any server technology. It was originally developed by MiskoHevery and Adam Abrons. HTML is great for declaring static documents, but it falters when we try to use it for declaring dynamic views in web-applications. AngularJS lets you extend HTML vocabulary for your application. The resulting environment is extraordinarily expressive, readable, and quick to develop

IMPLIMENTATION:

Explaining architecture

Let's start with a presentation of the system from an architectural point of view. I think that an image is worth more than a thousand word so I'll start from showing you this simple graph:

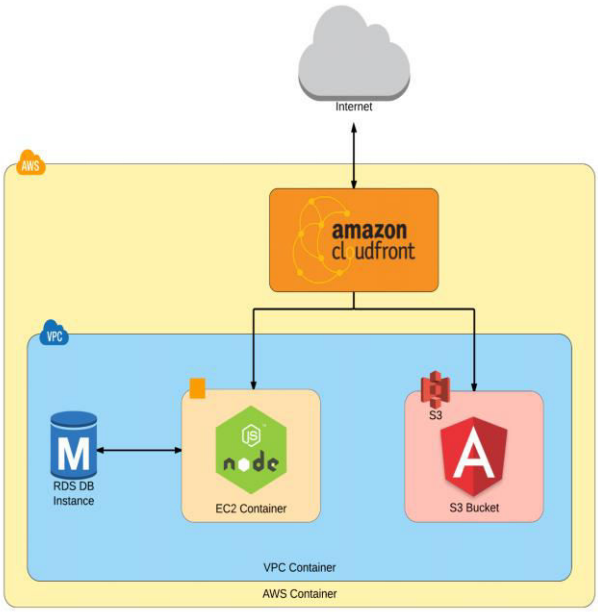


Diagram of AWS modules and relations between them

On the picture, there are all *AWS* presented and I'll try to characterize each of them shortly:

- **EC2 container** — a general purpose node which freely can be described as *VPS* (*Virtual Private Server*) thus it has own *OS* and you have direct access to it.
- **S3 bucket** — simply, a data container. You can keep here assets needed by your website: documents, images, videos and many more. It can be also used for serving static files (just like a bundled *Angular 2* application!).
- **RDS database** — *AWS* twisted name for a relational database instance.

- **VPC** — *Virtual Private Cloud* realize the concept of keeping all components inside one network. In this particular case, we place *EC2* container and *RDS* in the same *VPC* to make them communicate with each other.

- **CloudFront** — a gateway component which offers worldwide content delivery. There are many of *CloudFront* edge locations which ensure that user is redirected to the nearest one. If the content is available on that edge location, a user receives the answer immediately.

At the beginning of communication, when users tries to access the website, they are redirected to *CloudFront* server which is configured to serve a static content from *S3* bucket (our data container). If someone requested the same content from the same edge server before, *CloudFront's* cache object is returned instead of getting the object from *S3* bucket again. However, described caching behavior is not available when you serve static content directly from *S3* bucket container.

In addition to serving a static content, *CloudFront* instance redirects queries to the *node.js* backend server thus a user has a single endpoint both for backend and frontend server.

The last part of the puzzle is establishing a connection between *EC2* server and a database instance. Moreover, we should ensure that *DB* instance is not exposed to the Internet to avoid security problems.

To summarize, we must resolve following issues:

- Setup and expose *S3* bucket serving *Angular 2* application.
- Create *EC2* container which will be serving *node.js* server.
- Create a database (*RDS*).
- Establish connection between *EC2* container and *RDS* instance.
- Configure *CloudFront* to serve *Angular 2* and *node.js* applications.

In next paragraphs, I'll try to explain how to resolve mentioned cases. As a prerequisite, I assume that you have created *AWS* account and you have access to it using *aws-cli*.

S3 bucket setup

We will use the *S3* bucket for serving static content of a website. First, create an *S3* bucket **with public read access**. Then go to **Properties** page and make sure that **Static website hosting** is enabled. The last step is to

enable serving files from a bucket is adding proper access policy in **Permissions/Bucket Policy** page:Now your bucket can be read by everyone and you are ready to place your *Angular 2* application in it using

However, gulp HYPERLINK "https://github.com/pgherveou/gulp-awspublish"plugin to automate an upload process so after each build of the release application new version is automatically placed in *S3*.

Website URL patterns

It may be obvious for several reader, but I would like to briefly elaborate about creating routes for a website. After building some websites, I'm convinced that this is the most common pattern for routing:

- static content (frontend) — *www.example.com/** (excluding *api/*)
- dynamic content (backend) — *www.example.com/api/**

Such convention is pretty straight-forward and prevents from mixing backend and frontend paths. This will be very helpful when we will do configuration for *CloudFront*.

EC2 container for node.js server

Let's start from setting the heart of our system. As I mentioned before, *EC2* is a general purpose component and in our case, we will setup it as a *node.js*backend server.

Be aware when selecting **Network** in creation wizard so you can be sure that newly created container would be placed inside selected *VPC* instance. You can choose this option only once so step for a moment and consider what private cloud you will use.

The last thing, you must pay attention to during creation is S**ecurity Group**that can be described as firewall settings for *AWS* components. For *EC2* Linux instance you should have enabled at least *SHH* and *HTTP* port (this should be your *node.js* server port). These settings can be changed in the future thus it's not a big issue if you made mistake during configuration. Try to remember the name of that security group (or change it to remember it easily). It will be needed later for a database configuration.

Uff! We've finished! Make a coffee as a reward and wait a few minutes when *AWS* will be creating an *EC2* instance for you. Next, prepare yourself for a container's environment configuration. Make sure that your *node.js* server is exposed on*/api URL* because *CloudFront* would try redirect requests here.

RDS instance

Majority of the modern websites need a specific container for keeping persistent data. The most common approach for storing such data is creating a database. So now let's see how to launch a *DB* instance using *RDS* (*Relational Database Service*).

Log in into your *AWS* console, choose *RDS* and launch a new instance. The parameters you must be careful about during configuration:

- **VPC instance** — make sure that you create a database in same cloud instance you choose for *EC2* container
- **Public accessibility** — for security reasons you should disable access to a database from the Internet.
- **Backup** — (optional) if you want to save the state of your *DB* select this option.

After creating an instance it is essential to configure its Security Group. Go to **VPC console** and choose **Security Groups**. Select the database security group (it should be named *rds-xxxx*) and go to **Inbound rules,** then click **Edit**. You should be able to add a new rule. We need to create **All TCP, All UDP** and **All ICMP** rules. While you will be adding a new group, enter your *EC2* security group as a **Source**. As the result, you would have 3 new inbound rules and the source of each rule is your *EC2* instance security group.

We are done with configuration! Now let's check if our setup works. Do the following:

- Extract an address of your database instance. Try to run: awsrds describe-db-instances

*If you have the problem with access, go to your **IAM** console and set **AdministratorAccess** to your AWS CLI **User**.*

The output from that command will contain **ENDPOINT** section and this is basically the address of your database instance. I do not fully understand why there is no information about the endpoint of a DB instance in *AWS* console. Let's hope that guys from *AWS* will add such useful info in the near future.

2. Use the gathered address to perform connection to your database:

- Log in to your *EC2* instance.

(This is example command for MySQL instance. If you create another type of database, you need

to find similar command for connecting to the database.)

CloudFront configuration

explanation what CloudFront offers in AWS website:

Amazon CloudFront is a web service that speeds up distribution of your static and dynamic web content, such as .html, .css, .js, and image files, to your users. CloudFront delivers your content through a worldwide network of data centers called edge locations. When a user requests content that you're serving with CloudFront, the user is routed to the edge location that provides the lowest latency (time delay), so that content is delivered with the best possible performance. If the content is already in the edge location with the lowest latency, CloudFront delivers it immediately. If the content is not in that edge location, CloudFront retrieves it from an Amazon S3 bucket or an HTTP server (for example, a web server) that you have identified as the source for the definitive version of your content.

So if we sum up all this cool stuff, we can imagine *CF* as a caching gateway for our website. In our configuration, we want to cache requests to *S3* bucket because it contains static content (until we would like to deploy a new version of the *Angular* app to speed up user experience) but we don't want to do the same with requests for a backend as these are considered as a dynamic.

Before we take off, I feel obligated to warn you about making frequent changes for *CF* configuration — each save operation takes about 10 minutes to replicate so try to bulk all changes into one.

We have almost finished! This will be the last step we need in our setup: let's create **CloudFront Web** instance. In the wizard, we will setup *CF* for serving *S3* bucket and later we will add redirection for *EC2 node.js* container. During this part you must set:

- **Origin Domain Name:** choose S3 bucket.
- **Viewer Protocol Policy**: Redirect HTTP to HTTPS.
- **Allowed HTTP methods**: GET, HEAD, OPTIONS, PUT, POST, PATCH, DELETE.
- **Query String Forwarding and Caching**: choose No if your Angular application uses query string routes.
- **Compress Objects Automatically**: Yes.
- **Default Root Object**: write here your *index.html* filename.

If you want to know more about these settings read this article.

Congratulations, now you have your new *CloudFront* for any project you like. At this moment you should be able to enter your website (get the *Angular* part of it) when you enter the *CloudFront* address in the browser.

So let's go further and try to pass a non-root URL like *www.app.cloudfront.com/about*. Whoops! Seems like *CloudFront* does not know about that page. To avoid such situation just tell CF to redirect all 404 and 403 errors to *Angular 2* application in very simple way:

- Select *CloudFront* distribution.
- Enter **Error Pages** tab.
- Select **Create Custom Error Response.**
- Select **404** for **HTTP Error Code.**
- Set **TTL** to 0.
- Set **Customize Error Response** to Yes.
- In **Response Page Path** put the path to your index.html file.
- **HTTP response code**: set to 200.
- Do the same for **403** error code.

Now, all of 404 and 403 errors should be redirected to *Angular 2* application so you should deal with potentially incorrect URLs there.

"We are almost there" step we need to do is to enable the redirection of URL *www.cloudfront.com/api* to the node.js server. Moreover, it may be useful to remove caching behavior of *CloudFront* because of dynamic nature of responses. To achieve such effect follow steps listed below:

- In the **CloudFront console** edit the **Distribution Settings**.
- Go to **Origins** tab and **Create Origin**.
- Enter the address of your EC2 component as an **Origin Domain Name**.
- Set the **Origin Protocol Policy** to **HTTP Only**.
- Write your server **HTTP Port**.
- Click **Create**.

After that, you will create the custom *CloudFront* origin. In the moment I've written the article *CF* supports only *S3* buckets as origins. Every other component needs to be configured as a custom one.

OK, we've registered the origin and our final step is to create the route for it. So:

- Go to the **Behaviors** tab.
- Click **Create Behavior**.
- As a **Path Pattern** enter a wildcard of your backend base URL. In our case that will be */api/**.
- Choose previously created EC2 origin as **Origin**.
- In **Viewer Protocol Policy** choose: HTTP and HTTPS.
- **Allowed HTTP methods**: GET, HEAD, OPTIONS, PUT, POST, PATCH, DELETE.
- **Object caching:** Customize.
- Set all **TTL** to 0 as we don't want to cache responses.
- Probably you will need to disable **Query String Forwarding** too.
- **Compress Objects Automatically**: Yes.And that's it! Now we have fully functional *CloudFront* even for backend request. Take a note thatwe disabled caching via setting *TTL* of responses to 0. Itmeans that all messages come from backend are "hot" and a user browser will need to request a new data from backend when it's needed.

But what about HTTPS?

CloudFront instance by default is configured to use own certificate and it's ready to use *HTTPS*. That's it — you don't need to generate and sign a certificate. The connection between a user and your gateway can use both *HTTP* and *HTTPS*. But for inner communication, you can choose a more secure option.

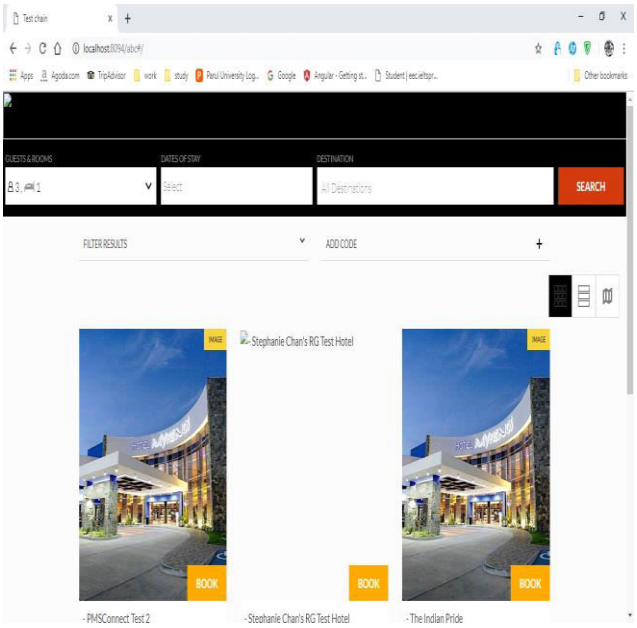
And we did it! After completing all above steps you have fully worked cloud server with a scalable inner organization. It can stand against user loads, it is safe and can be restored in case of failure. But the most important for fans of free solutions: all of presented here *AWS* components are available in free tier**.**

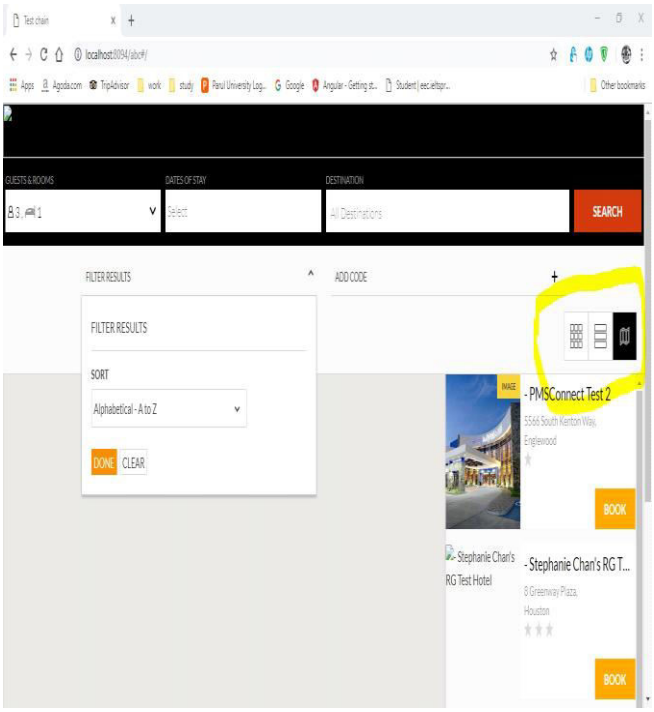
Proposed Work:

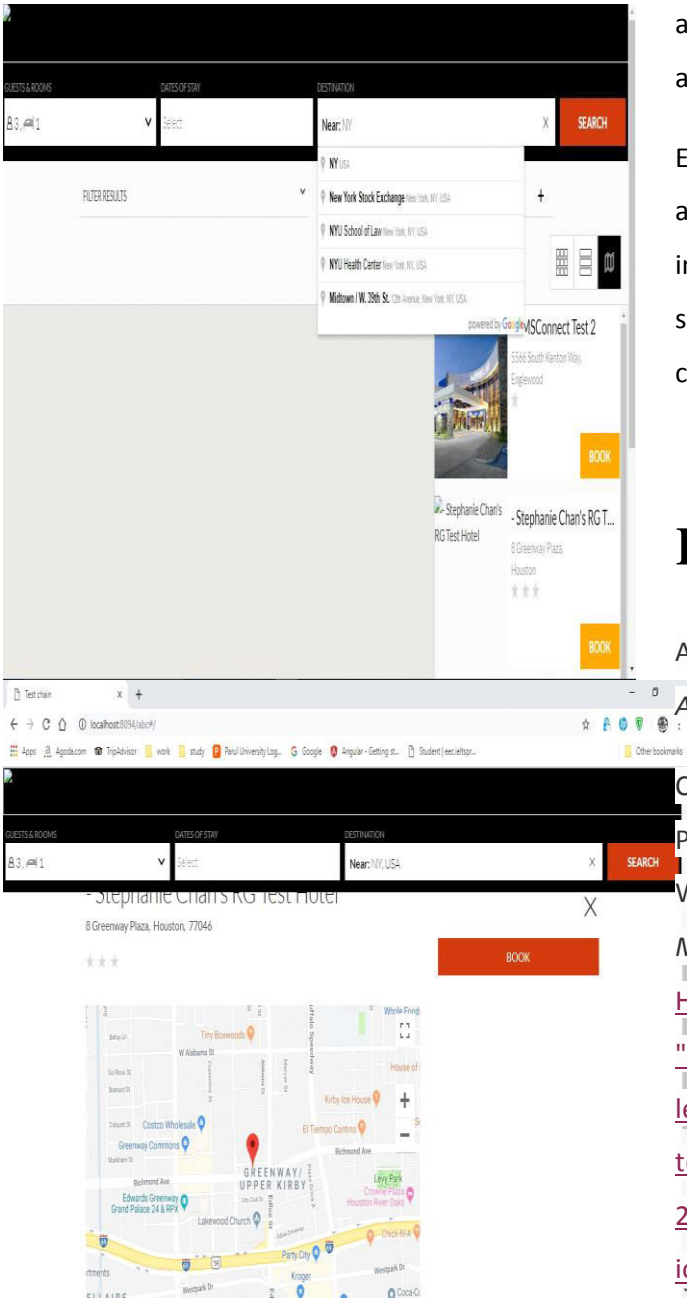
The recent introduction of Internet technology to general business has led to its wide-scale application in the hotel industry. Consumers have been increasingly using the Internet to search for accommodation-related information on hotel Websites. To facilitate a better understanding of e-commerce, hospitality and tourism researchers have shown the importance of establishing content-rich and user-friendly Websites. The existing hospitality

literature, however, has a very limited number of published articles that investigated the perceptions of hotel Website users on the importance of specific features on hotel Websites. The absence of prior studies on assessing the importance of hotel Website dimensions and attributes is particularly true in the comparison of online browsers who only search for information and online purchasers who have made online bookings. This article reports on a study that examined these two groups of international hotel Website users on their perceived importance level of specific dimensions and attributes on hotel Websites. Empirical evidence indicates that there was no significant difference in most of the included dimensions and attributes between these two groups of users. In addition, the respondents viewed that the included dimensions and attributes are important on Websites of 3-star or above hotels.

Screenshots:







agencies can also be benefited in different aspect in different departments.

Even every user can be benefited in different aspect like getting the better reviews from the integrated article section by introducing this section trusting online platform issues can be cut down.

Conclusion:

By using SAP suits, ERP one not only can develop their business growth but also can analyze financial risk as well as enterprise risk. Even though by using this method small

References:

Anonymous (2004). Online Frustrations. *Hotel Asia Pacific*, 5(2), 31.Google Scholar

Chung, T. & Law, R. (2003). Developing a Performance Indicator for Hotel Websites. *International Journal of Hospitality Management*, 22, 119–125.CrossRef HYPERLINK "http://scholar.google.com/scholar_lookup?title=Developing%20a%20Performance%20Indicator%20for%20Hotel%20Websites&author=T..%20Chung&author=R..%20Law&journal=International%20Journal%20of%20Hospitality%20Management&volume=22&pages=119-125&publication_year=2003"Google Scholar

Cox, B. (2002). *Online Travel — Still an E-commerce Star?* [Accessed on May 22, 2004]. www.internetnews.com/ec-news/article.php/1437521.Google Scholar

Doolin, B., Burgess, L. & Cooper, J. (2002). Evaluating the Use of the Web for Tourism Marketing: a Case Study from New Zealand. *Tourism Management*, 23, 557–561.CrossRef HYPERLINK "http://scholar.google.com/scholar_lookup?title=Evaluating%20the%20Use%20of%20the%20Web%20for%20Tourism%20Marketing%3A%20a%20Case%20Study%20from%20New%20Zealand&author=B..%20Doolin&author=L..%20Burgess&author=J..%20Cooper&journal=Tourism%20Management&volume=23&pages=557-561&publication_year=2002"Google Scholar

Greenspan, R. (2003). *Hotel Industry Makes Room for Online Bookings*. [Accessed on May 26, 2004] www.clickz.com/stats/markets/travel/article.php/1567141.Google Scholar

Morrison, A.M., Taylor, S., Morrison, A.J. & Morrison, A.D. (1999). Marketing Small Hotels on the World Wide Web. *Information Technology & Tourism*, 2(2), 97–113.Google Scholar

G. Bahmutov, Improving Angular web app performance example. Available at: http://bahmutov.calepin.co/improving-angular-web-app-performance-example.html (Accessed: 18 December 2014)