International Journal of Scientific Research in Science, Engineering and Technology
Print ISSN: 2395-1990 | Online ISSN : 2394-4099 (www.ijsrset.com)
doi : https://doi.org/10.32628/IJSRSET2183183

Lazy Loading Based with Load On Demand and Currency Support in Web Browser

[1]Shamali V. Bire, [2]Virendra Pawar

[1]Post Graduate Student, Master of Technology, Department of Computer Engineering, Vishwakarma Institute of Technology, Pune, Maharashtra, India

[2]Assistant Professor, Department of Computer Engineering, Vishwakarma Institute of Technology, Pune, Maharashtra, India

Article Info
Volume 8, Issue 3
Page Number: 473-478

Publication Issue :
May-June-2021

Article History
Accepted : 12 June 2021
Published: 22 June 2021

ABSTRACT

In this work our goal is to make a Web client application for the Real Estate Business. We already have a Stand-Alone Application for the same, so we are migrating from Online Application to the Web Client. Here for this we are using XPA tool which is a tool to move a project and make changes as per necessity. During this, we designed the editing function size and position adjustment, hiding and displaying, style editing, editing by device type etc. for every device type such that the contents will be dynamically converted according to resolution or screen-size in step with various devices. To be able to answer various devices, storing device information in an exceedingly component is created through editing function. With this, we have worked on Currency Format support in Web Client application and implemented a concept of Lazy Loading. Responsive web functionality improve server throughput that shows the response function of Web Client and therefore the processing speed is improved. All features can operate in real-time manners with our software architecture and loading mechanism, called Lazy Loading.

Keywords: Web application, lazy loading, component, load on demand

I. INTRODUCTION

According to the most recent reports, Around Millions of Euros, Dollars, Rupees were invested in the real estate market. Current web platforms for real estate advertise the selling and renting of real estate assets and provide the means for searching and listing different categories. By providing all features in a single centralized aggregating platform, real estate association management software will be a valuable tool to help association's efficiency. In this project we are migrating software to website. In this we have used the MagicXPA, DevExtreme tools and Web technology, which is a form of business process automation technology.

Along with this, we have worked on supporting currency format in Web Client Application and

implemented lazy loading. Lazy Loading describes the method of dynamically downloading and displaying content when a user scrolls down or across a sequence of information on the device screen. When images are included within the screen, it delivers a really powerful user experience. There are many factors that have to be considered, including threading, HTTP requests, memory management, and caching while implementing Lazy Loading.

II. PROBLEM DEFINITION

Supporting Numeric Currency specific formats in web Client Application with the Stand alone system we can easily define the currency because of the internal pre-defined Packages of magic. This tool provides all aspects of the application development and deployment process within a single end-to-end platform. It features a ready-made business application engine that simplifies the code-writing process and enables you to deploy to market faster, using fewer resources. Applications developed using this typically have fewer coding mistakes, undergo more thorough prototyping, benefit from greater business side input and optimization, and can be more easily adapted to changing business needs. The tool we are using enables us to focus more on the business logic of the application and less on what is happening behind the scenes. We have a stand-alone system and migrate it to the Web Client so some of those internal pre-defined packages would not support while migrating the application in to Web Client which we have solved, similarly we have encountered this currency format issue in web client and implemented Lazy Loading.

III. IMPLEMENTATION

Implementation of Lazy Loading and Supporting Currency format in Web Client is discussed below with steps of procedure.

A. Implementation of Lazy Loading

Create an app with the name 'LazyLoadingExample'. This app contains a header module for navigation, a dashboard component for the initial route, and two lazy modules named 'customer' and 'supplier' respectively. Both lazy modules contain a 'View' component in them.

Step 1: Create a new app with Routing

1) ng new LazyLoadingExample --routing

Step 2: Creating modules and components

2) ng g c component/dashboard --spec false --module=app

2) ng g m header/header --flat --module=app

3) ng g c header/menu --spec false --module=header

4) ng g m customer/customer --flat --routing

5) ng g m supplier/supplier --flat --routing

6) ng g c customer/view-customer --spec false --module=customer

7) ng g c supplier/view-supplier --spec false --module=supplier

After generating the modules and components, our directories look like the image shown in Fig.1.

Code for app.component.html

```
<app-menu></app-menu>
<router-outlet></router-outlet>
```

Step 3: Configure routes for lazy modules

Code for customer-routing.module.ts

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import {Routes, RouterModule} from
'@angular/router';
import { ViewCustomerComponent } from'./view-
customer/view-customer.component';
const routes: Routes = [ { path:'view',
component:ViewCustomerComponent } ];
@NgModule({ declarations: [ ], imports:
[CommonModule, RouterModule.forChild(routes)],
exports:[RouterModule] })
```

```
export class CustomerRoutingModule { }
```

Code for supplier-routing.module.ts.

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { Routes, RouterModule } from
'@angular/router';
import { ViewSupplierComponent } from './view-
supplier/view-supplier.component';
const routes: Routes = [ { path: 'view', component:
ViewSupplierComponent } ];
@NgModule({ declarations: [ ], imports:
[ CommonModule, RouterModule.forChild(routes)],
exports:[RouterModule]})
export class SupplierRoutingModule { }
```

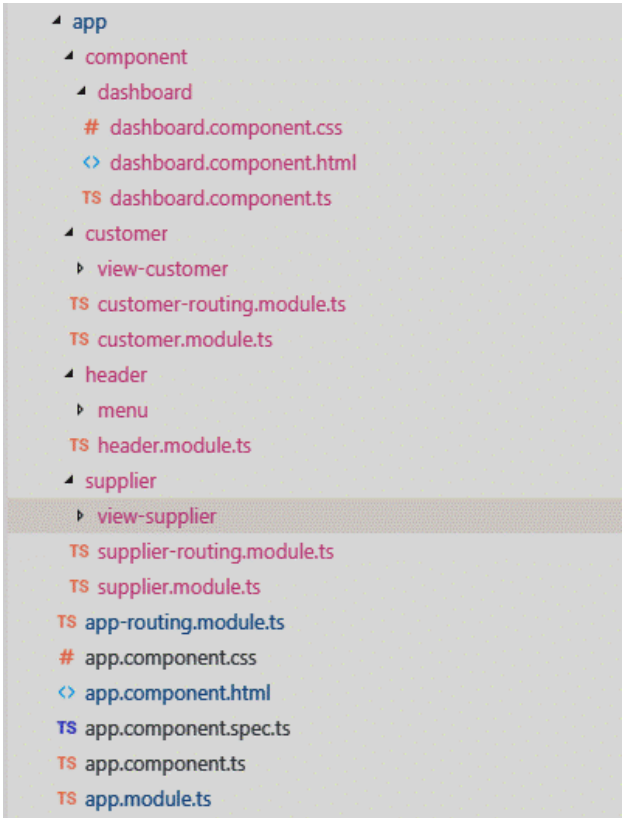

Figure 1: Application directory structure in angular

Step 4: Configure lazy routes for app.

In this step, point the lazy route to the lazy module from the app router. We are able to do that with the loadChildren property with the trail to the module file. Then, reference the module with a hash (#). This tells Angular to only load LazyModule when the lazy URL is activated.

Code for app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from
'@angular/router';
import { DashboardComponent } from
'./component/dashboard/dashboard.component';
const routes: Routes = [ { path:'', component:
DashboardComponent },{path:'customer',
loadChildren:'./customer/customer.module#Customer
Module'},
{path:'supplier',
loadChildren:'./supplier/supplier.module#SupplierMo
dule'}];
@NgModule({  imports:
[RouterModule.forRoot(routes)],  exports:
[RouterModule]  })
export class AppRoutingModule { }
```

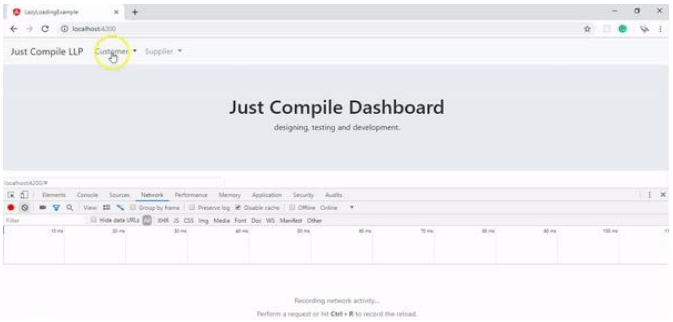

Figure 2 : Application result used to localhost in web browser

Step 5 – Add the navigation directive to the menu component.

Code for menu.component.html

```
<div class = "dropdown-menu" aria-labelledby =
"customerDropdown">
<a class="dropdown-item" routerLink =
"customer/view">View Customer</a>
```

```
</div>
<div class = "dropdown-menu" aria-labelledby =
"supplierDropdown">
<a class="dropdown-item" routerLink =
"supplier/view">View Supplier</a>
 </div>
```

Step 6 – Verify Lazy Loading is working.

Let us confirm that whether Lazy Loading is functioning. In Chrome, open developer tools and click on the "Network" tab. once you navigate to the lazy URL 'customer/view', you ought to see a customer-customer-module.js file rendered. During this demo, you'll be able to see it took 347ms and supplier-supplier-module.js loaded after you navigated to the lazy URL 'supplier/view'.

```
   align: "right",
   allowNegative: true,
   allowZero: true,
   decimal: ".",
   precision: 2,
   prefix: "",
   suffix: "",
   thousands: ",",
   nullable: true,
   min: null,
   max: null,
   inputMode: CurrencyMaskInputMode.FINANCIAL
};

@NgModule({
   imports: [
  // Material Modules
  MatTableModule,
       ...
  MagicAngularMaterialModule,

NgxCurrencyModule.forRoot(customCurrencyMaskC
onfig)

],
providers: [ExitMagicService],
```

Implementation of Supporting Currency format in Web Client

We can use ngx-currency package which supports all the attributes related to currency that xpa tool uses for numeric formats:

1. To install ngx-currency use following command : npm i ngx-currency

2. Add ngx-currency module in the library module magic-gen.lib.module.ts

```
import { CurrencyMaskInputMode,
NgxCurrencyModule } from "ngx-currency";

export const customCurrencyMaskConfig = {

})
```

3. In html, currently numeric field is generated like this way:

```
&lt;input
…
type=&#39;number&#39;
…
[formControlName]=&quot;mgc.V1&quot;
…
&gt;
&lt;mgError [magic]=mgc.V1&gt;
&lt;/mgError&gt;
```

4. Change the numeric field such that it is generated like this:

```
&lt;input
…
~~type=&#39;number&#39;~~
...
[formControlName]=&quot;mgc.V1&quot;
currencyMask
[options]=&quot;mg.getNumericPicture(mg.getValue(mgc.V2))&quot;
…
&gt;
```

Note: Here, type= ‘number’ is deleted.
‘currencyMask’ is directive and [options] is taking picture string from variable V2’s value. mg.getNumericPicture() takes picture in string and returns object which has all the keys needed for picture i.e. prefix, suffix, allow Negative etc..

IV.RESULT

We have created the above same application without Lazy Loading, so we are able to easily compare the finish load time when applying with Lazy Loading and without Lazy Loading of results.

If you refer both the Fig. 3 and Fig. 4, you may observe the ‘Finish load time’. Within the case of ‘Without Lazy Loading’, the finish time is 1.9 seconds and within the case of ‘With Lazy Loading’, the finish time is 721 microseconds. It clearly shows that Lazy Loading improves the performance of the application, you will be able to load feature areas only if requested by the user, you will be able to speed up load time for users that only visit certain areas of the application, you will be able to continue expanding lazy loaded feature areas without increasing the scale of the initial load bundle [5].

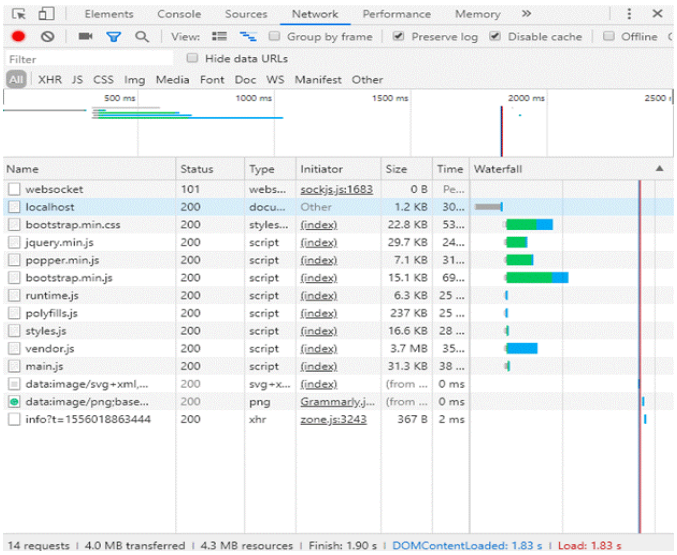

Figure 3: Application without Lazy Loading

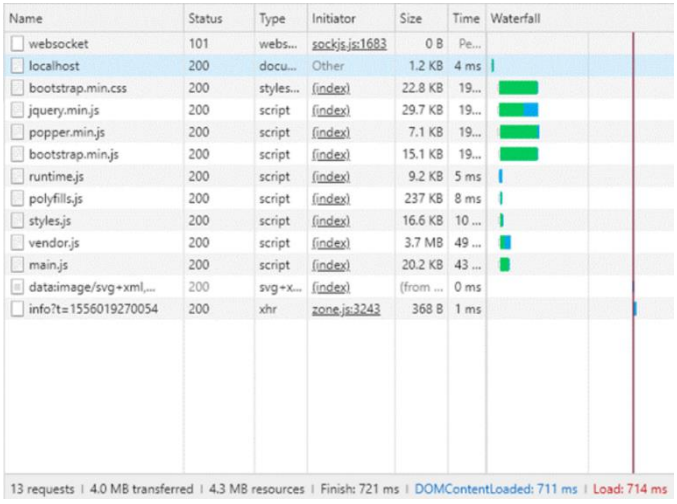

Figure 4: Application with Lazy Loading

V. CONCLUSION

User/Client will not wait around longer than five seconds for his/her page to load and a few will wait even less. If the application is too slow to load then their customers are less likely to remain on their site which translates into fewer conversions. Lazy Loading to the system is to boost the performance. It is recommended to use Lazy Loading if you have got plenty of routes and components. The new architecture can improve the performance by 50% on the average, comparing with stand-alone system with-out AJAX properties. Our new implementation can improve the speed and efficiency 38%. In this

project, we can add and manage Real-estate details, manage activities of Clients, their cells details. It is software which helps the user to work with the team easily. This portal reduces the efforts of standalone system and gives greater efficiency.

VI. ACKNOWLEDGEMENT

I would like to take this chance to express my gratitude to Magic Software Enterprise, Pune for taking me through this project and I am sincerely grateful to VIT Pune, Savitribai Phule Pune University for giving me this opportunity which made me transcend limitations and improve my knowledge.

VII. REFERENCES

[1]. B. Marco and P. Fraternali, “Large-scale model-driven engineering of web user interaction: The webml and webratio experience,” Science of Computer Programming, vol. 89, pp. 7187, 2014.

[2]. Stephen Fluin. “Why Developers and Companies Choose Angular”. Medium, 2017 [Online].

[3]. Available: https://medium.com/angular-japan-user-group/why-developers-andcompanies-choose-angular-4c9ba6098e1c

[4]. Sandy Veliz. “Angular + Material Design | Instalación Angular Material”. Medium, 2019 [Online].

[5]. Available:https://medium.com/@sandy.e.veliz/angular-material-designinstalaci%C3%B3n-angular-material-790caca5677b

[6]. Kevin Kreuzer. “The ultimate guide to set up your Angular library project”. Medium, 2019 [Online]. Available: https://medium.com/angular-in-depth/the-ultimate-guide-to-setup-your-angular-library-project-399d95b63500 Alligator [Online].

[7]. Available:https://www.digitalocean.com/community/tutorials/angular-lazy-loading

Cite this article as :

Shamali Bire, Virendra Pawar, "Lazy Loading Based with Load On Demand and Currency Support in Web Browser", International Journal of Scientific Research in Science, Engineering and Technology (IJSRSET), Online ISSN : 2394-4099, Print ISSN : 2395-1990, Volume 8 Issue 3, pp. 473-478, May-June 2021. Available at
doi : https://doi.org/10.32628/IJSRSET2183183
Journal URL : https://ijsrset.com/IJSRSET2183183