IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

Lazy Loading Based With Load On Demand In Web Browser

[1] Yogesh Patil, [2] Leena Deshpande

[1]Post Graduate Student, [2]Associate Professor

[1] Department of Computer Engineering, [2] Department of Computer Engineering,

[1]Vishwakarma Institute of Information Technology, Pune, India

Abstract: During this, we designed the editing function size and position adjustment, hiding and displaying, style editing, edit by device type etc. for every device type in order that contents will be dynamically converted in step with resolution or screen size in step with various devices and to be able to answer various devices by storing device information in an exceedingly component created through editing function. To boost the viewer of the online application and to boost the processing speed of the server, a component using the lazy loading technique was designed. In this, responsive web functionality and improve server throughput results show that the response function of the online function and therefore the improvement of the processing speed of the server are improved by the monitoring tool, and therefore the processing speed is improved in most browsers. All features can operate in real-time manners with our software architecture and loading mechanism, called Lazy Loading. The experimental results show that, with new software architecture and Lazy Loading, the online page can response 50% faster than loading the whole online page on the average and therefore the Lazy Loading can reduce the latent period of most pages by 50%.

***Index Terms* – Web application, lazy loading, component, load on demand, responsive functionality.**

I. Introduction

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. Unfortunately, such a typical and powerful pattern isn't trivial to implement in websites. There's no built-in library to accomplish lazy loading, and there are many factors that has to be considered, including threading, HTTP requests, memory management, and caching [1]. Unfortunately, these advantages often come at the price of decreased performance because certain optimizations become more difficult to perform when an optimizing compiler cannot assume that it's seen the full program. I make a study and a practical development of various functionalities with a frontend framework called angular. Specifically, the functionalities consist in using dynamic rendering of components, creating components in an exceedingly lazy loaded module and creating an online component with angular elements. As any front-end application grows in size the organization and architecture of the code become that rather more important. For a straightforward application that has maybe one or two pages, with not an excessive amount of business logic, we will go away with bad practices without an excessive amount of an adverse effect. At a bigger scale, these effects become magnified and may significantly impact your application. Lazy loading in Angular is one tool in your tool belt that you just can use to make an application that may scale easily because the application grows.

II. Problem Definition

Lazy loading could be a routing technique within which the JavaScript components load within the browser only their corresponding route is activated. The most aim of lazy loading is to boost the performance of the Angular app by restricting the unnecessary loading of components [2]. Allow us to understand this with an example. We've got created an Angular app without lazy loading. Allow us to assume that the app contains a login and a dashboard screen. Here, the login screen is that the starting screen of the app. whenever we load our app within the browser, both the screen scripts load within the memory of the browser, which eventually, reduces the performance of the app. Because we've got a really small app with only two screens, it'll not affect our app. But, consider a project that consists of many components. Therein case, it'll reduce the performance of the app. If we manage the identical example with lazy loading, then the JavaScript of login screen are going to be loaded at the beginning of the applying and therefore the JavaScript of the dashboard will only be loaded when the routes match for the dashboard [3].

III. Implementation of Lazy Loading

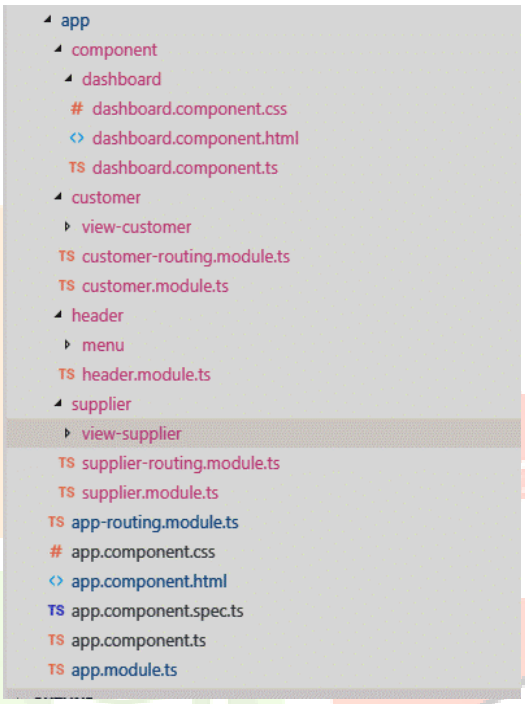
Create an app with the name LazyLoading Example. This app contains a header module for navigation, a dashboard component for the initial route, and two lazy modules named customer and supplier respectively. Both lazy modules contain a View component in them.

Step 1 - Create a new app with Routing

1) ng new LazyLoadingExample --routing

Step 2 - Creating modules and components

1) 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



```
app
  component
    dashboard
      dashboard.component.css
      dashboard.component.html
      dashboard.component.ts
  customer
    view-customer
    customer-routing.module.ts
    customer.module.ts
  header
    menu
    header.module.ts
  supplier
    view-supplier
    supplier-routing.module.ts
    supplier.module.ts
  app-routing.module.ts
  app.component.css
  app.component.html
  app.component.spec.ts
  app.component.ts
  app.module.ts
```

Fig. 1 Application directory structure in angular

After generating the modules and components, our directories look like this above image.

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 { }
```

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#CustomerModule'},
                         {path:'supplier', loadChildren:'./supplier/supplier.module#SupplierModule'}];
@NgModule({  imports: [RouterModule.forRoot(routes)],  exports: [RouterModule]  })
export class AppRoutingModule { }
```

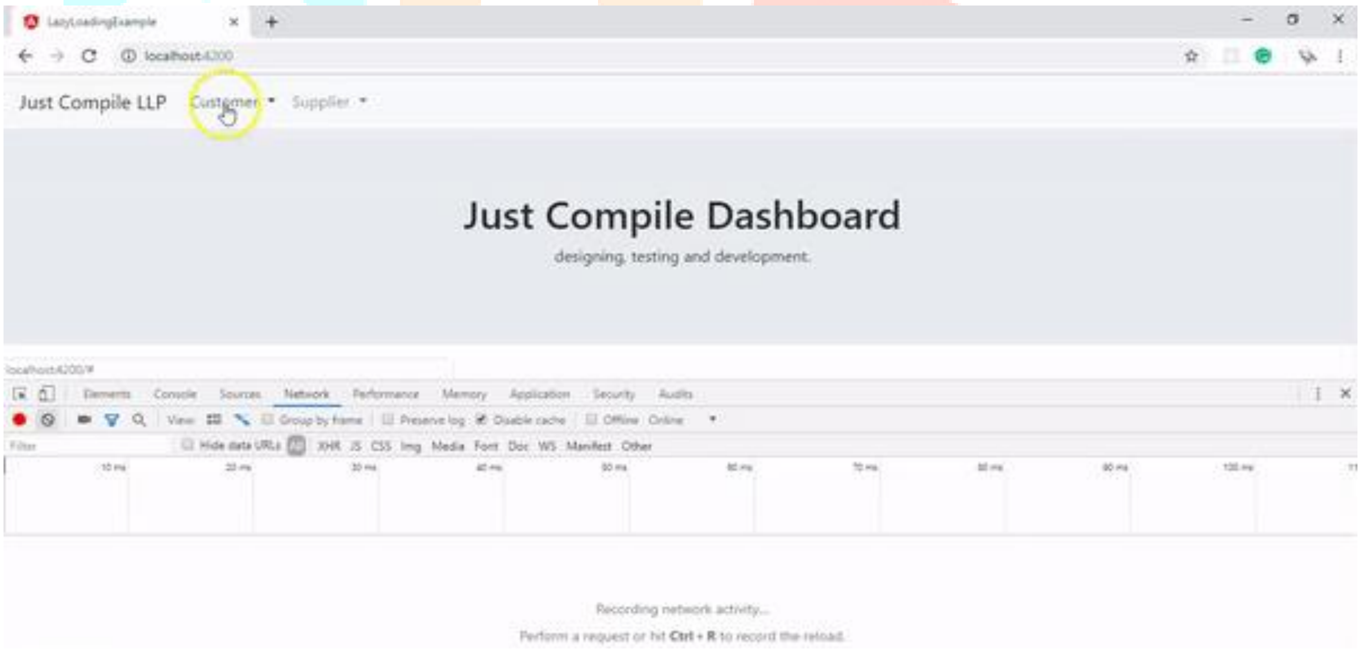
Step 5 – Add the navigation directive to the menu component.

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's confirm that 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'.


LazyLoadingExample
localhost:4200
Just Compile LLP
Customer
Supplier
Elements Console Sources Network Performance Memory Application Security Audits
Group by frame Preserve log Disable cache Offline Online
Filter Hide data URLs XHR JS CSS Img Media Font Doc WS Manifest Other

Just Compile Dashboard

designing, testing and development.

Recording network activity...

Perform a request or hit Ctrl + R to record the reload.

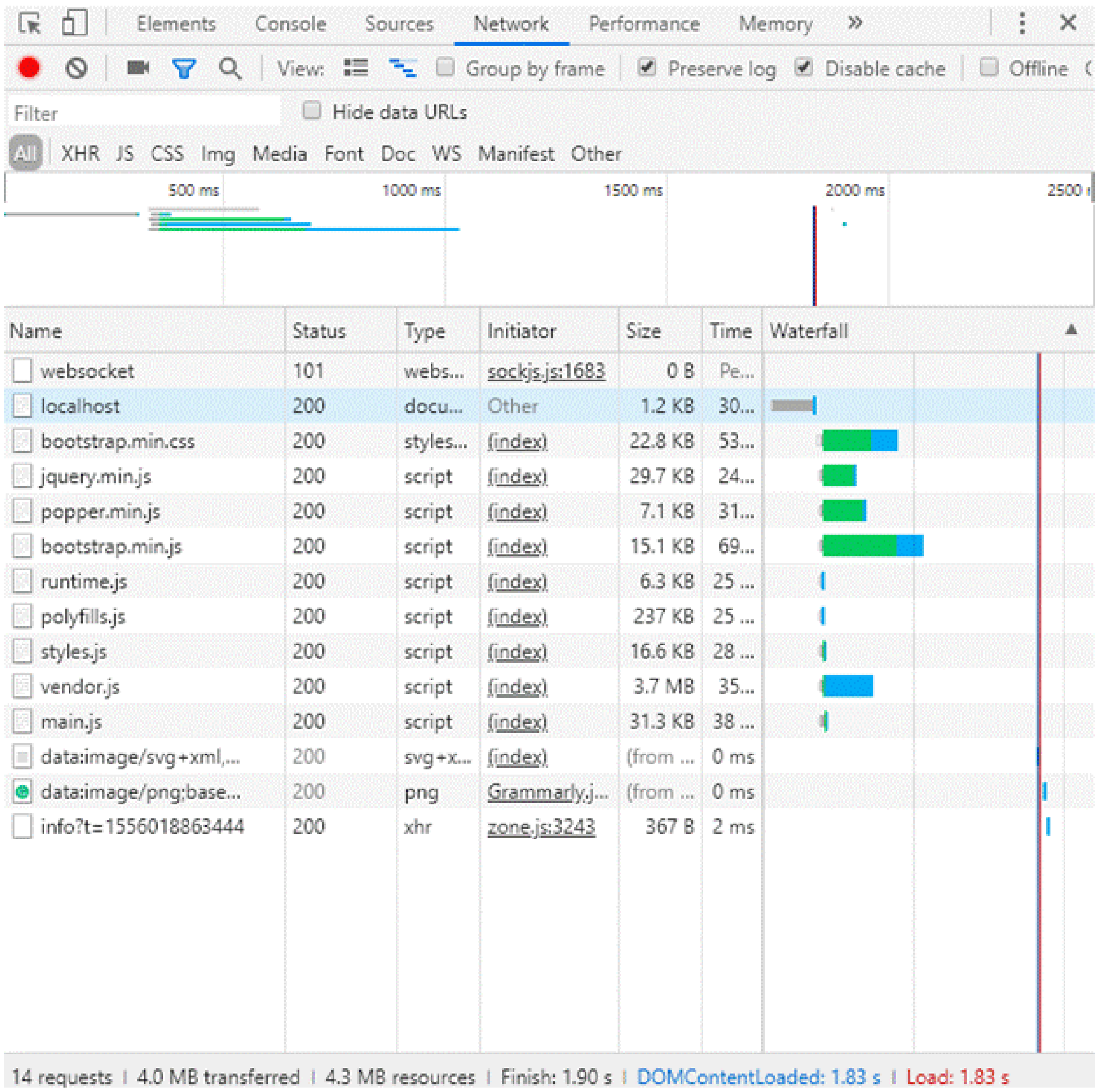
Fig. 2 Application result used to localhost in web browser

IV. Result

I have created the above same application without lazy loading, so we are able to easily compare the load time of the applying with or without lazy loading of results.

If you see the above of both the figures 3 or 4, you may find the finish load time. Within the case of without lazy loading, the finish time is 1.9 seconds and within the case of lazy loading, the finish time is 721 microseconds. It clearly shows that lazy loading improves the performance of the applying, you'll be able to load feature areas only if requested by the user, you'll be able to speed up load time for users that only visit certain areas of the applying, you'll be able to continue expanding lazy loaded feature areas without increasing the scale of the initial load bundle [5]

4.1 Without Lazy Loading:

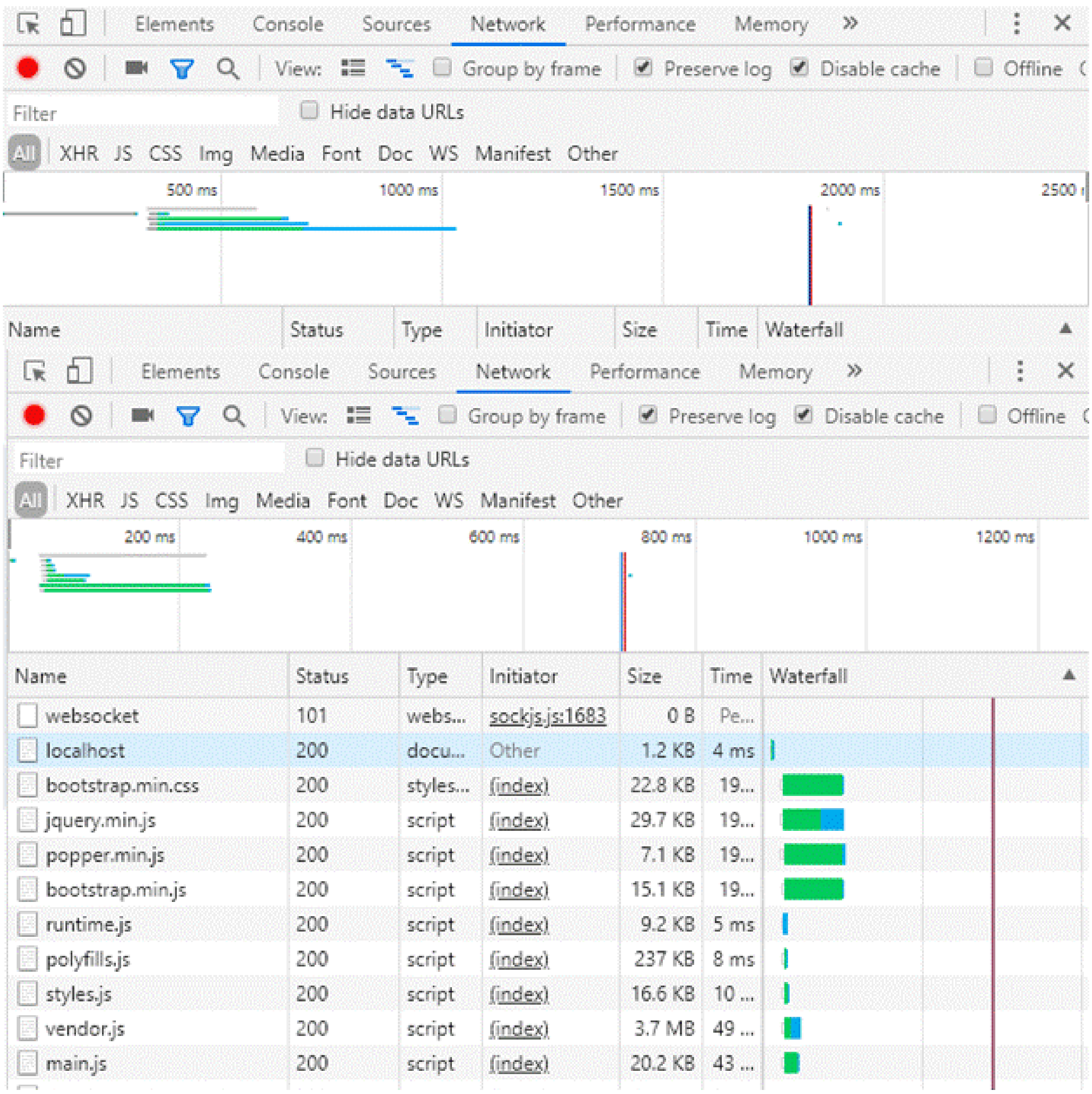


Name	Status	Type	Initiator	Size	Time	Waterfall
websocket	101	webs...	sockjs.js:1683	0 B	Pe...	
localhost	200	docu...	Other	1.2 KB	30...	
bootstrap.min.css	200	styles...	(index)	22.8 KB	53...	
jquery.min.js	200	script	(index)	29.7 KB	24...	
popper.min.js	200	script	(index)	7.1 KB	31...	
bootstrap.min.js	200	script	(index)	15.1 KB	69...	
runtime.js	200	script	(index)	6.3 KB	25 ...	
polyfills.js	200	script	(index)	237 KB	25 ...	
styles.js	200	script	(index)	16.6 KB	28 ...	
vendor.js	200	script	(index)	3.7 MB	35...	
main.js	200	script	(index)	31.3 KB	38 ...	
data:image/svg+xml,...	200	svg+x...	(index)	(from ...	0 ms	
data:image/png;base...	200	png	Grammarly.j...	(from ...	0 ms	
info?t=1556018863444	200	xhr	zone.js:3243	367 B	2 ms	

14 requests | 4.0 MB transferred | 4.3 MB resources | Finish: 1.90 s | DOMContentLoaded: 1.83 s | Load: 1.83 s

Fig. 3 without lazy loading result

4.2 With Lazy Loading



☐ Group by frame ☑ Preserve log ☑ Disable cache ☐ Offline ☐ Hide data URLs

Name	Status	Type	Initiator	Size	Time	Waterfall
websocket	101	webs...	sockjs.js:1683	0 B	Pe...	
localhost	200	docu...	Other	1.2 KB	4 ms	
bootstrap.min.css	200	styles...	(index)	22.8 KB	19...	
jquery.min.js	200	script	(index)	29.7 KB	19...	
popper.min.js	200	script	(index)	7.1 KB	19...	
bootstrap.min.js	200	script	(index)	15.1 KB	19...	
runtime.js	200	script	(index)	9.2 KB	5 ms	
polyfills.js	200	script	(index)	237 KB	8 ms	
styles.js	200	script	(index)	16.6 KB	10 ...	
vendor.js	200	script	(index)	3.7 MB	49 ...	
main.js	200	script	(index)	20.2 KB	43 ...	

Fig. 4 With lazy loading result

V. Result

Conclude that, most of the people won't wait around longer than five seconds for your page to load and a few will wait even less. If your application is slow to load then your customers are less likely to remain on your site which translates into fewer conversions. Lazy Loading to the system to boost the performance. It's recommended to use lazy loading if you've got plenty of routes and components. The new architecture can improve the performance by 50% on the average, comparing with regular webpage with-out AJAX properties. Our Lazy Loading can improve most pages by 38%. However, if the webpage includes modal windows, Lazy Loading can worsen the performance, definitely splinting an application into modules and using Lazy Load could be a great alternative to boost the loading time and also the initial file size of the applying.

VI. Acknowledgment

I would prefer to take this chance to increase my gratitude to Vishwakarma Institute of knowledge Technology, Pune, under Savitribai Phule Pune University for providing the platform for my work.

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

[3] Sandy Veliz. "Angular + Material Design | Instalación Angular Material". Medium, 2019 [Online]. Available: https://medium.com/@sandy.e.veliz/angular-material-designinstalaci%C3%B3n-angular-material-790caca5677b

[4] 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

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