**Angular Mock-up Documentation**

**Challenges Faced during the development:**

1. **Fixing sidebar component and navbar static and not allowing content to expand:**

Sol: Created another div to occupy the sidebar space and then included the sidebar in that div. The side bar is fixed and the outside div will make sure that the content div is not taking the entire space.

1. **Making the sample-pages accordion:**

Sol: Used a click event on the sample pages button, and used class binding to apply the dropdown class which toggles the display property of the sample pages children: i.e., Auth page and error page.

1. **Making the dashboard component layout:**

Sol: Divided the dashboard component to three parts.

1. Dashboard-top:

It contains statistics tiles and the chart tile.

1. Dashboard-middle:

It contains the Order history table, the server load and performance tile.

1. Dashboard-middle:

It contains the three tiles of equal size.

To make the design possible, we used rows and columns of the bootstrap. All the data inside these HTML is extracted from a sample JSON file, to reduce the code inside the HTML page.

1. **Inserting Charts with JSON data:**

**Sol:** Used chart.js website to display the charts inside the HTML page. Installed the package using npm.

In the TS file imported the Chart, registerables from ‘node-modules/chart.js’ and then used the predefined register method inside the chart.js file.

**Code to register:** Chart.register(…registerables)

The data for the chart is saved inside the data set value of the chart.js chart file.

The type of the chart is set to a line chart and the other one is a bar chart.

Changed the labels inside the data set array to match the requirement.

Used “ fill : true “ property to fill the colour below the lines inside the line chart.

Used scales property to start the scale of graph to always start at zero.

import { Component, OnInit } from '@angular/core';

import {Chart, registerables} from 'node\_modules/chart.js'

Chart.register(...registerables);

  RenderChart() {

    new Chart('piechart', {

      type: 'line',

      data: {

        labels: ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'June'],

        datasets: [{

          label: 'Total Followers',

          data: [10000, 13400, 16200, 12439, 18500, 21100],

          borderWidth: 3,

          fill : true

        }]

      },

      options: {

        scales: {

          y: {

            beginAtZero: true

          }

        }

      }

    });

  }

1. **Making API call to the server for the data inside Order History table:**

import { Injectable } from '@angular/core';

import { HttpClient, HttpErrorResponse } from '@angular/common/http'

import { Observable, throwError } from 'rxjs';

@Injectable({

  providedIn: 'root'

})

export class TestserviceService {

  url = "../assets/sample-jsons/webTraffic.json"

  customerDetailsApi = "https://reqres.in/api/users?per\_page=7";

  // ordersApi = "https://reqres.in/api/unknown?per\_page=6"

  constructor(private \_http : HttpClient) { }

  getCustomerDetails() {

    return this.\_http.get<JSON>(this.customerDetailsApi);

  }

  // getOrderDetails() {

  //   return this.\_http.get<JSON>(this.ordersApi);

  // }

  errorHandler(error : HttpErrorResponse) {

    return throwError(error.message || "Unknown Server Error");

  }

}

**Fig : Inside the testservice service file for API calling**

Sol: Created a service with name “ testservice ”. Imported HTTPClient module inside that service file.

Created a method called getCustomerDetails(). Inside that method by using the http.get method returned the data received from the server.

1. **Routing between the page:**

To route between the components, we used the angular routing service.

Inside the routing file and inside the Routes type array, specify the paths for our components, and the respective components that should be displayed.

After setting the path and the component. Used click events inside the HTML file and

assigned it to the elements.

const routes: Routes = [

  // {path : '', redirectTo : 'dashboard', pathMatch : 'full'},

  {path : '', redirectTo : 'dashboard', pathMatch : 'full'},

  {path : 'login', component : LoginComponent},

  {path : 'dashboard', component : DashboardComponent},

  {path : 'sample-pages/auth-page', component : AuthPageComponent},

  {path : 'sample-pages/error-page', component : ErrorPageComponent},

  {path : 'ui-elements', component : UiElementsComponent},

  {path : '\*\*', component : PageNotFoundComponent}

];

**Fig: Inside the app.routing.module.ts file**

  constructor(private \_router: Router, private \_activatedRoute : ActivatedRoute) { }

  ngOnInit(): void {

  }

  gotoDashboard() {

    this.\_router.navigate(['dashboard'], {relativeTo: this.\_activatedRoute});

  }

  // gotoSamplePages() {

  //   this.\_router.navigate(['sample-pages'], {relativeTo : this.\_activatedRoute});

  // }

  gotoUi() {

    this.\_router.navigate(['ui-elements'], {relativeTo: this.\_activatedRoute});

  }

  gotoAuthPage() {

    this.\_router.navigate(['sample-pages/auth-page'], {relativeTo : this.\_activatedRoute});

    // window.open('sample-pages/auth-page');

  }

**Fig: Inside the component’s .ts file**

When we click the element the click event will be fired and it will trigger a method. Inside that method by using the **Router module and Activated Router module** from **‘@angular/router’** to navigate to different components.

The **navigate** method inside the Router module is used to navigate to components based on the respective path.

1. **Creating validation for the Auth Page:**

The Auth page is a sample page of the design of the login page for the website. To create a form using angular we can use :

1. Template Driven Forms (TDF)
2. Reactive Forms

Here, we used the reactive forms which will result in less code inside our HTML file and it is easy for unit testing.

Used bootstrap classes for the styling of the login form.

Inside the TS file imported the FormBuilder, Validators from ‘@angular/forms’.

Created a form group using the Formbuilder and by using the Validators created some input validations.

Inside the HTML file every input is assigned with a class name of “ form-control “ and the formControlName is used to bind the input fields with the form Group values inside our TS file.

  authForm = this.\_fb.group({

    userName : ['', [Validators.required, Validators.minLength(5), Validators.maxLength(10), forbiddenNameValidator(/password/)]],

    password : ['', [Validators.required]],

    confirmPassword : ['', [Validators.required]]

  })

  get getUserName () {

    return this.authForm.get('userName');

  }

  get getPassword () {

    this.passValidator = this.authForm.get('password')?.value;

    return this.authForm.get('password');

  }

  get getConfirmPassword () {

    // return this.authForm.get('password')?.value === this.authForm.get('confirmPassword')?.value

    return this.passValidator === this.authForm.get('confirmPassword')?.value;

  }

  constructor(private \_fb : FormBuilder) { }

  ngOnInit(): void {

  }

Created custom password validation using the regular expression which makes sure that the entered password matches the password requirements.

A screenshot of a computer

Description automatically generated with medium confidence When all the validations are matched then all the input fields are surrounded with green border and the form is ready to submit.

1. **Toggling the state of the sidebar (Hidden and Visible):**

**Sol:** To toggle the state of the sidebar whenever we click on the hamburger menu. We have assigned a click event to the hamburger. So, when we click the hamburger, the event gets fired and that calls a function.

When the function is called it makes the side bar position to the left at a high value so that it will not be visible inside the web page.

And simultaneously another class is added to the content div which makes the content div to take the entire width of the screen.

1. **Creating popup menus for notifications, messages, and apps buttons:**

**Sol:** To create popups for the three buttons, we have taken three divs with the respective data inside the divs.

We took two classes to show and hide the content inside the popups. The popups will only be visible when we click on the buttons, and they will disappear once we click anywhere else while the popup is open.

To do this we have added a click event to each of the buttons. So, when we click on the button then the click event gets fired which will toggle the classes which shows and hides these popups.

To make the popups disappear when we click outside. We have added a click event to the window. We have given a condition inside the window click event so that the popup will disappear when we click anywhere other than the button.

messagesDrop () {

    const mess = document.querySelector('.messages');

    const messDrop = document.querySelector('.messages-popup')

    const messDismiss = document.querySelector('.view-all-messages');

    mess?.addEventListener('click', () => {

      messDrop?.classList.toggle('messages-pop-dis');

    });

    messDismiss?.addEventListener('click', () => {

      messDrop?.classList.remove('messages-pop-dis');

    });

    window.addEventListener('click', function(e) {

      if(e.target != mess && e.target != messDismiss) {

        messDrop?.classList.remove('messages-pop-dis')

      }

    });

**Fig: Function which toggles one of the popup**

**References:**

* <https://angular.io/docs>
* <http://preview.themeforest.net/item/rippleui-bootstrap-4-responsive-admin-dashboard-template/full_screen_preview/23362041?_ga=2.40294048.325693335.1669616051-1507218261.1668662180>
* https://www.youtube.com