SESSIONSTORAGE

hbtw\_user -> daros del usuario logeado

1.- ng new hacienda\_buzon\_tributario\_web

2.- cd hacienda\_buzon\_tributario\_web

3.- ng g c auth/layout --skip-tests

4.- ng g c auth/pages/login --skip-tests

5.- ng g c auth/pages/register --skip-tests

6.- Controlados que va a fungir con enlace entre las diferentes paginas

ng g c dashboard/layout/layoutDash --skip-tests –flat

import { RouterModule } from '@angular/router'; -> (manejo de rutas)

\*\*\*.html -> agregamos el router-outlet “Seguirá creciendo”

<router-outlet>

En la carpeta assets copiamos los fonts a usar y en el archivo style.css global creamos las reglas al inicio del archivo y los demás estilos se ponen en los css de cada componente

[ FONT ]\*/

@font-face {

  font-family: Poppins-Regular;

  src: url('./assets/fonts/poppins/Poppins-Regular.ttf');

}

@font-face {

  font-family: Poppins-Medium;

  src: url('./assets/fonts/poppins/Poppins-Medium.ttf');

}

@font-face {

  font-family: Poppins-Bold;

  src: url('./assets/fonts/poppins/Poppins-Bold.ttf');

}

@font-face {

  font-family: Poppins-SemiBold;

  src: url('./assets/fonts/poppins/Poppins-SemiBold.ttf');

}

Para el manejo de arreglos JSON, en "compilerOptions" dentro del archivo "tsconfig.json"

"resolveJsonModule": true,

7,. Manejo de estilos

ng add @angular/material

npm install primeflex

angular.json

styles: [

…

"./node\_modules/primeflex/primeflex.css"

]

8.- Manejo de errores

npm install sweetalert2

9.- Manejo de pagina no encontrada

ng g c dashboard/pages/notFoundPage --skip-tests --skip-selecto

10.- Configuracion para acortar los paths relativos en tsconfig.json

{

…

"compilerOptions": {

…

"paths": {

      "@dashboard/\*": ["./src/app/dashboard/\*"],

      "@auth/\*":   ["./src/app/auth/\*"],

      "@environments/\*": ["./src/environments/\*"],

      "@shared/\*": ["./src/app/shared/\*"],

    }

}

}

Ejemplo de aplicación import { SidemenuComponent } from '@shared/sidemenu/sidemenu.component';

11.- app.router.ts -> genweramos las rutas

export const routes*: Routes* = [

  {

    path: 'auth',

    //canActivate: [isNotAuthenticatedGuard],

    loadComponent: *()* => import('./auth/layout/layout.component').then(c => c.LayoutComponent),

    children: [

      {

        path: 'login',

        title: 'Login',

        loadComponent: *()* => import('./auth/pages/login/login.component').then(c=> c.LoginComponent)

      },

      {

        path: '',

        redirectTo: 'login',

        pathMatch: 'full'

      }

    ]

  },

  {

    path: 'dashboard',

    //canActivate: [isAuthenticatedGuard],

    loadComponent: *()* => import('./dashboard/layout/layout-dash.component').then(c => c.LayoutDashComponent),

    children: [

      /\*{

        path: 'refrendo/:idConcepto/:formulario',

        title: 'Refrendo Vehicular',

        loadComponent: () => import('./portal-servicios/smyt/pages/refrendo-page/refrendo-page.component'),

      },\*/

…

    ]

  },

  {

    path: '',

    redirectTo: 'auth',

    pathMatch: 'full'

  },

  {

    path: '\*\*',

    loadComponent: *()* => import('./dashboard/pages/not-found-page/not-found-page.component').then(c => c.NotFoundPageComponent)

  }

];

12.- App.component.htm -> borramos todo solo dejamos el <router-outlet>

login.component.ts Agregamos código html, librerías necesarios y creamos el formulario

layout.component.ts Agregamos código html, RouterModule y Css

13.- Crear la carpeta en src/environments, manejo de variables globales. Creamos dos archivos para manejo de variables en pruebas y en producción

environments.ts y environments.prod.ts

Editamos angular.json

“projects”[

“hacienda\_buzon…”{

“architect”: {

build: {

…

"configurations": {

            "production": {

              "fileReplacements": [{"replace": "src/environments/environments.ts", "with": "src/environments/environments.prod.ts"}],

…

}

}

}

}

]

14.- app.config.ts -> inyectar HttpClientModule e importarlo -> manejo global de peticiones

15.- ng g s auth/services/authService ->

>ng g s auth/services/authService --skip-tests

inyectar HttpClient

16.- Crear las interfaces auth/interfaces/login-respose-struct.interface.ts y auth/interfaces/user-struct.interface.ts

>ng g i auth/interfaces/loginResponseStruct

>ng g i auth/interfaces/userStruct

>ng g i auth/interfaces/authStatusStruct -> maneja la estructura de logeado, no logeado, etc. Es un ENUM

17.- auth-service.service.ts Agregamos el método de login y agregamos toda la funcionalidad del servico para manejar el registro

Login recibirá parámetros de tipo LoginRequestStruct

Generamos la interfaz auth/interfaces/login-request-struct

> ng g i auth/interfaces/loginRequestStruct

18.- login.component.ts inyectamos AuthService implementamos el método login y hacemos el llamado al servico de logeo

Inyectamos Routes para el manejo de rutas

Inyectamos Swal para las notificaciones a los usuarios

19.- Cremos un servicio de validaciones /auth/services/validationService -> Su función es validar dos campos y eliminar la validación del vacío

>ng g s auth/services/validation --skip-tests

En login.component.ts inyectomos validationService

20.- Se crea el apartado shares y dentro el componente datos del contribuyente que será compartido con la pagina de alta de contribuyente y para facturar cuando no se registren

>ng g c shared/components/taxpayerDaata --skip-tests

21.- Editamos auth/pages/register, agregamos formulario y en

22.- Se crea carpeta en shared con nombre data que contendrá archivos que consuman todos los modulos, en este caso creamos un archivo JSON con mensajes que se manejaran en los formularios “messages.json”

23.- crear una clase “message\_handling.class.js”

Importamos messages.json para manejar los errores de manera global en la clase

import MessagesList from '@shared/data/messages.json';

Creamos una interfaz con la estructura del mensaje a obtener

> ng g i shared/interfaces/messageStruct

24.- Creamos un servicio con consultas generales en shared “general”

> ng g s shared/services/general --skip-tests

Inyectamos HttpClient

En los environments agregamos la ruta a **servicios-hacienda** para obtener municipios, etc

BASE\_URL\_SERVHACIENDA: 'https://app.hacienda.morelos.gob.mx/',

Cremos método de consulta de municipios

Creamos la interfaz de respuesta de consulta de municipios

> ng g i shared/interfaces/municipiosResponseStruct

Creamos la interfaz de petición de consulta de entidades

> ng g i shared/interfaces/entidadesRequestStruct

25.- Se crear en sared/components “snackBar” que se usara en los componentes que lancen mensajes.

26.- en register se importa taxpayer-daata.components y se incluye para embeber en el HTML

27.- se crea la ruta para registro en app.routes

{

        path: 'register',

        title: 'Register',

        loadComponent: () => import('./auth/pages/register/register.component').then(c => c.RegisterComponent)

      },

28.- en login.component.html se agrega la ruta “registro”

29.- en general.service creamos el metodo para consultar entidades federativas

30.- se hieron adecuaciones a registro.component, taxpayer-data.component

31 .- Se crea dashboard/pages/servicesMenu

>ng g c dashboard/pages/servicesMenu --skip-tests --skip-selector

Crear en dashboard/pipes/servicesMenuImage

>ng g p dashboard/pipes/servicesMenuImages --skip-tests

Cramos un JSON con los datos a mostrar en el menu shared/data/portal-menu.json

Creamos Interfaz con la estructura de portal-menu.json

>ng g i dashboard/interfaces/portalMenu

Se agregan las imagenes a usar en la carpeta assets/images/servicemenu

32.- en app.routes.ts dentro de la ruta “dashboard” se crea la ruta hija ”sevices-menu” que apuntara a “service-menu.component.ts”

33.- En el “login.component.ts” en el apartado **quieres continuar sin registro** lo apuntamos a la ruta **sevices-menu**

34.- Creamos dashboard/pages/portalHaciendaMenu-> encargado de mostrar el menu de opciones de portal de hacienda

> ng g c dashboard/pages/portal-hacienda-services/portalHaciendaMenu --skip-tests --skip-selector

35.- creamos en shared/components/toolbar y shared/components/sidenav

> ng g c shared/components/sidenav --skip-tests

> ng g c shared/components/toolbar --skip-tests

36.- Crear Interfaz para el manejo de respuesta de la estructura de conceptos que se mostraran en el navbar “dashboar/interfaces/conceptsResponseStruct

> ng g i dashboard/interfaces/conceptsResponseStruct

37.- \* crear componente SPINNER para manejo de loading “shared/components/loadSpinner”

> ng g c shared/components/loadSpinner --skip-tests

38.- Usamos la interface de manejo de datos de usuario logeado ubicada en “auth/interfaces/user-struct.interface.ts” esto en el componente “sidenav.component”

39.- se crea pipe dashboard/pipes/haciendaMenuImages para el manejo de path de imágenes de menú de hacienda. se importa el componente en el portal-hacienda-menu

40.- Generar Servicio para manejar las peticiones APIREST de servicios de hacienda ”dashboard/services/serviciosHaciendaportal”

> ng g s dashboard/services/serviciosHaciendaPortal --skip-tests

41.- editamos el app.routes y agregamos la ruta para navegar al menú de portal de hacienda si es que se selecciono la opcion

42.- Cremos en dashboard/pages/portal-hacienda-services/smyt/refrendo sin selector

> ng g c dashboard/pages/portal-hacienda-services/smyt/refrendo --skip-tests --skip-selector

43.- Crear componente en dashboard/components/smyt/vehicleData

>ng g c dashboard/components/smyt/vehicleData --skip-tests

Creamos el script de formato para fecha en “shared/formats/custom-date-format.ts

Importamos el formato de fecha a vehicle-data.component.ts

Instalar momento y MomentAdapter para manejo de fechas

> npm i moment

> npm i @angular/material-moment-adapter@17

Crear archive de consulta de servicios SMYT dashboard/services/smyt/smyt

> ng g s dashboard/services/smyt/smyt --skip-tests –flat

Crear interface de manejo de la estructura de las oficinas

> ng g i dashboard/interfaces/smyt/officesResponseStruct

Crear archive de datos JSON correspondiente a las oficinas disponibles e importamos

Al servicio

“dashboard/data/smyt/oficinas\_tramite.json

Crear interface de manejo de la estructura de tipo de vehiculos

> ng g i dashboard/interfaces/smyt/vehicleTypeResponseStruct

Crear archive de datos JSON correspondiente a las tipo de vehiculo disponibles e importamos al servicio

“dashboard/data/smyt/vehicle\_type.json

Crear la interfaz par el manejo de los datos para validar un vehículo, son los datos que se envían al API

> ng g i dashboard/interfaces/smyt/vehicleDataRequestStruct

Crear la interfaz par el manejo de los datos de respuesta del API de validación de datos vehiculares

> ng g i dashboard/interfaces/smyt/vehicleDataResponseStruct

Creamos un servicio de validaciones generales

>ng g s dashboard/services/validator –skip-tests –flat

Crear interfaz con la estructura para el manejo de mensajes dentro de los componentes

“dashboard/interfaces/message-struct.interfaz.ts

Instalamos libreria para encriptar datos en loca storage

    > npm i --save-dev @types/crypto-js

Creamos la secret key en el environment

  SECRET\_KEY\_CRYPTO: '3\*U{m1q#t$>.I!/<0/^NXe"|6...gE',

En shared/classes creamos dos clases de nombre data-decrypt.js y data-encrypt.js las cuales serán instanciadas desde los componentes o servicios que las requieran

En shared/interfaces generamos una interfaz genérica que será usada para que el servicio comunique la respuesta a la pagina o componente con los atributos message y success

Creamos en shared/components/datataxpayer/ los componentes taxpayerDatacontact, taxpayerRecurrentContrib y taxpayerPass mismos que serán ocupados en el modulo de registro de usuarios y cuando se va a facturar sin un registro

\* ya se creo el taxpayer-daata.component que contiene los datos generales del uasuario

> ng g c shared/components/datataxpayer/taxpayerDatacontact --skip-tests

> ng g c shared/components/datataxpayer/taxpayerRecurrentcontrib --skip-tests

> ng g c shared/components/datataxpayer/taxpayerPass --skip-tests

Crear las interfaces para manejo de datos de tipo emailList, phonelist, taxList, vehicleList y conssesionList, todas estas en auth/interfaces y agregamos los atributos que coicidan con el Back-End

>ng g i auth/interfaces/register-user/emailUSerStruct.interfaz

>ng g i auth/interfaces/register-user/phoneUSerStruct.interfaz

>ng g i auth/interfaces/register-user/TaxUSerStruct.interfaz

>ng g i auth/interfaces/register-user/VehicleUSerStruct.interfaz

>ng g i auth/interfaces/register-user/concesionUSerStruct.interfaz

>ng g i auth/interfaces/register-user/registerUserStruct.interfaz

Para los componentes que usan tipo de contacto creamos arreglos en JSON para el consumo de esta info shared/data/ “tipo\_contacto.json”

Generamos las validaciones en taxpayer-datacontact .- Checar este archivo se manejaron validaciones mas especializadas

Generamos servicio encargado de las validaciones dashboard/services

>ng g i auth/interfaces/register-user/generalUserStruct.interfaz

Se genero en taxpayer-pass expresión regular para validar un password

En auth/interface generamos la clase que dará fomato a los datos de respuesta “RegisterDataResponce”

>ng g i auth/interfaces/registerDataResponse.interfaz

En este punto se termino la parte de **REGISTRO**

Construimos ina interfaz con la estructura lo que se va a guardar en localstorage shared/interfaces

>ng g i shared/interfaces/storageDataStruct.interfaz

REFRENDO

\*\* <http://blog.enriqueoriol.com/2017/11/httpclient-vs-http-angular.html>

\*\* https://medium.com/@nicowernli/angular-captura-todos-los-errores-de-httpclient-mediante-un-httpinterceptor-2cead03bb654