import { HttpClientModule } from '@angular/common/http';

app.module.ts

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

@Injectable() export class HomeComponent implements OnInit,  OnDestroy {

configUrl = './assets/content/home.json';

HomeModel = [];

HTAListData: string [];

wCCListData: string [];

  constructor(

              private httpService: HttpClient

              ) { }

  ngOnInit() {

    this.httpService.get(this.configUrl).subscribe(

      data => {

        this.HomeModel = data as string [];

        this.HTAListData = this.HomeModel['HTAList'];

        this.wCCListData = this.HomeModel['wCCList'];

      },

      (err: HttpErrorResponse) => {

        console.log (err.message);

      }

    );

  }

1. Sending POST request

 onCreatePost(postData: Post) { //model

    // Send Http request

    this.http

      .post<{ name: string }>(

        'https://search-itinerary.firebaseio.com/posts.json',

        postData

      )

      .subscribe(responseData => {

        console.log(responseData);

      });

  }

page.component.ts

1. GETting Data

 private fetchPost(){

    this.isFetching = true;

    this.http

    .get<{[key: string]: Post}>('https://search-itinerary.firebaseio.com/posts.json') .subscribe(responseData => {

        console.log(responseData);

      });

    .pipe(map(responseData =>{

        const postArray: Post[] = [];

        for (const key in responseData){

          if(responseData.hasOwnProperty(key)){

            postArray.push({...responseData[key], id: key});

          }

        }

        return postArray;

      })

    )

    .subscribe(data =>{

      this.isFetching = false;

      this.loadedPosts = data;

    })

  }

page.component.ts

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

import { HttpClient } from '@angular/common/http';

import { Observable } from 'rxjs';

import { map, catchError } from 'rxjs/operators';

@Injectable({

  providedIn: 'root'

})

export class RestApiService {

  items = [];

  constructor(private http: HttpClient) { }

  getAllScheduleService(): Observable<any> {

    return this.http.get('http://api.tvmaze.com/schedule/full').pipe(

      map(this.extractData),

      catchError(this.handleErrorObservable)

    );

  }

  getAllScheduleDetails(id): Observable<any> {

    return this.http.get(`http://api.tvmaze.com/shows/${id}/episodes`).pipe(

      map(this.extractData),

      catchError(this.handleErrorObservable)

    );

  }

  private extractData(res: Response) {

    let body = res;

    return body;

}

private handleErrorObservable (error: Response | any) {

  console.error(error.message || error);

  return Observable.throw(error.message || error);

}

}

1. Using RxJS operator to transfer Response data

 private fetchPost(){

    this.isFetching = true;

    this.http

    .get<{[key: string]: Post}>('https://search-itinerary.firebaseio.com/posts.json')

    .pipe(map(responseData =>{

        const postArray: Post[] = [];

        for (const key in responseData){

          if(responseData.hasOwnProperty(key)){

            postArray.push({...responseData[key], id: key});

          }

        }

        return postArray;

      })

    )

    .subscribe(data =>{

      this.isFetching = false;

      this.loadedPosts = data;

    })

  }

page.component.ts

1. Using types with HttpClient

 private fetchPost(){

    this.isFetching = true;

    this.http

    .get<{[key: string]: Post}>('https://search-itinerary.firebaseio.com/posts.json')

    .pipe(map(responseData =>{

        const postArray: Post[] = [];

        for (const key in responseData){

          if(responseData.hasOwnProperty(key)){

            postArray.push({...responseData[key], id: key});

          }

        }

        return postArray;

      })

    )

    .subscribe(data =>{

      this.isFetching = false;

      this.loadedPosts = data;

    })

  }

page.component.ts

1. Delete Posts

onClearPosts() {

    this.http.delete('https://search-itinerary.firebaseio.com/posts.json')

    .subscribe(() =>{

      this.loadedPosts = [];

    });

  }

page.component.ts

1. Handling Errors

private fetchPost(){

    this.isFetching = true;

    this.http

    .get<{[key: string]: Post}>('https://search-itinerary.firebaseio.com/posts.json')

    .pipe(map(responseData =>{

        const postArray: Post[] = [];

        for (const key in responseData){

          if(responseData.hasOwnProperty(key)){

            postArray.push({...responseData[key], id: key});

          }

        }

        return postArray;

      })

    )

    .subscribe(data =>{

      this.isFetching = false;

      this.loadedPosts = data;

    }, error =>{

        this.error = error.message;

        console.log(error);

    });

  }

page.component.ts

1. Using subject to Errors Handling

import { Subject } from 'rxjs';

export class AppComponent implements OnInit {

  error = new Subject<string>();

  onCreatePost(postData: Post) { //model

    // Send Http request

    this.http

      .post<{ name: string }>(

        'https://search-itinerary.firebaseio.com/posts.json',

        postData

      )

      .subscribe(responseData => {

        console.log(responseData);

      }, error =>{

        this.error.next(error.message);

      }

      );

  }

}

post.service.ts

import { Subject, Subscription } from 'rxjs';

export class AppComponent implements OnInit {

  private errorSub: Subscription;

  constructor(private http: HttpClient) {}

  ngOnInit() {

    this.errorSub = this.postsService.error.subscribe(errorMessage =>{

      this.error = errorMessage;

    })

  }

page.component.ts

1. Catch Error Operator

import { map, catchError } from 'rxjs/operators';

import { Subject,throwError, Subscription } from 'rxjs';

export class AppComponent implements OnInit {

  private fetchPost(){

    this.isFetching = true;

    this.http

    .get<{[key: string]: Post}>('https://search-itinerary.firebaseio.com/posts.json')

    .pipe(map(responseData =>{

        const postArray: Post[] = [];

        for (const key in responseData){

          if(responseData.hasOwnProperty(key)){

            postArray.push({...responseData[key], id: key});

          }

        }

        return postArray;

      }),

      catchError( errorRes =>{

       return throwError(errorRes);

      })

    )

    .subscribe(data =>{

      this.isFetching = false;

      this.loadedPosts = data;

    }, error =>{

        this.error = error.message;

        console.log(error);

    });

  }

page.component.ts

1. Setting Headers

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

export class AppComponent implements OnInit {

  private fetchPost(){

    this.isFetching = true;

    this.http

    .get<{[key: string]: Post}>('https://search-itinerary.firebaseio.com/posts.json'),

    {

      headers: new HttpHeaders({'Custom-Header': 'Hellow'})

    }

    .pipe(map(responseData =>{

        const postArray: Post[] = [];

        for (const key in responseData){

          if(responseData.hasOwnProperty(key)){

            postArray.push({...responseData[key], id: key});

          }

        }

        return postArray;

      })

    )

    .subscribe(data =>{

      this.isFetching = false;

      this.loadedPosts = data;

    });

  }

page.component.ts

1. Adding Query Parameter

import { HttpClient, HttpHeaders, HttpParams } from '@angular/common/http';

export class AppComponent implements OnInit {

  onFetchPosts() {

    this.fetchPost();

  }

  private fetchPost(){

    this.isFetching = true;

    this.http

    .get<{[key: string]: Post}>('https://search-itinerary.firebaseio.com/posts.json'),

    {

      headers: new HttpHeaders({'Custom-Header': 'Hellow'}),

      params: new HttpParams().set('print', 'pretty')

    }

    .pipe(map(responseData =>{

page.component.ts

Adding Mulitple Query Params

private fetchPost(){

    this.isFetching = true;

    let searchParams = new HttpParams();

    searchParams = searchParams.append('print', 'pretty');

    searchParams = searchParams.append('custom', 'key');

    this.http

    .get<{[key: string]: Post}>('https://search-itinerary.firebaseio.com/posts.json'),

    {

      headers: new HttpHeaders({'Custom-Header': 'Hellow'}),

     // params: new HttpParams().set('print', 'pretty')

      params: searchParams

    }

1. Observing different types of response

import { HttpClient, HttpHeaders, HttpParams } from '@angular/common/http';

export class AppComponent implements OnInit {

  onFetchPosts() {

    this.fetchPost();

  }

  private fetchPost(){

    this.isFetching = true;

    this.http

    .get<{[key: string]: Post}>('https://search-itinerary.firebaseio.com/posts.json'),

    {

      headers: new HttpHeaders({'Custom-Header': 'Hellow'}),

      params: new HttpParams().set('print', 'pretty')

    }

    .pipe(map(responseData =>{

page.component.ts

1. Interceptor

Interceptors in Angular, is a simple way provided by the framework to intercept and modify the application's http requests globally before they are sent to the server

import { HttpInterceptor, HttpRequest, HttpHandler } from '@angular/common/http';

export class AuthInterceptorService implements HttpInterceptor {

  intercept(req: HttpRequest<any>, next: HttpHandler) {

    console.log('Request is on its way');

    return next.handle(req);

  }

}

Auth-interceptor.service.ts

import { BrowserModule } from '@angular/platform-browser';

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

import { FormsModule } from '@angular/forms';

import { HttpClientModule, HTTP\_INTERCEPTORS } from '@angular/common/http';

import { AppComponent } from './app.component';

import { AuthInterceptorService } from './auth-interceptor.service';

@NgModule({

  declarations: [AppComponent],

  imports: [BrowserModule, FormsModule, HttpClientModule],

  providers: [

    {

      provide: HTTP\_INTERCEPTORS,

      useClass: AuthInterceptorService,

      multi: true

    }

  ],

  bootstrap: [AppComponent]

})

export class AppModule {}

app-module.ts

1. Manipulating Request object

import { HttpInterceptor, HttpRequest, HttpHandler } from '@angular/common/http';

export class AuthInterceptorService implements HttpInterceptor {

  intercept(req: HttpRequest<any>, next: HttpHandler) {

    console.log('Request is on its way');

    console.log(req.url);

    const modifiedRequest = req.clone({

      headers: req.headers.append('Auth', 'xyz');

    })

    return next.handle(modifiedRequest);

  }

}

Auth-interceptor.service.ts

1. Response Interceptors

import { HttpInterceptor, HttpRequest, HttpHandler, HttpEventType } from '@angular/common/http';

import { tap }  from 'rxjs/operators'

export class AuthInterceptorService implements HttpInterceptor {

  intercept(req: HttpRequest<any>, next: HttpHandler) {

    console.log('Request is on its way');

    console.log(req.url);

    const modifiedRequest = req.clone({

      headers: req.headers.append('Auth', 'xyz');

    })

    return next.handle(modifiedRequest).pipe(

      tap(event =>{

        console.log(event);

        if(event.type === HttpEventType.Response){

          console.log('Response arrived, Body data: ');

          console.log(event.body);

        }

      })

    );

  }

}

Auth-interceptor.service.ts

1. Mulitple Interceptors

import { HttpInterceptor, HttpRequest, HttpHandler, HttpEventType } from '@angular/common/http';

import { tap } from 'rxjs/operators';

export class LoggingInterceptorSerice implements HttpInterceptor{

    intercept(req: HttpRequest<any>, next: HttpHandler){

        console.log('Outgoing Request');

        console.log(req.url);

        console.log(req.headers);

        return next.handle(req).pipe(

            tap(event =>{

            if(event.type === HttpEventType.Response){

                console.log('Incoming Response');

                console.log(event.body);

            }

        }));

    };

}

logging-interceptor.service.ts

import { HttpInterceptor, HttpRequest, HttpHandler } from '@angular/common/http';

export class AuthInterceptorService implements HttpInterceptor {

  intercept(req: HttpRequest<any>, next: HttpHandler) {

    const modifiedRequest = req.clone({

      headers: req.headers.append('Auth', 'xyz');

    })

    return next.handle(modifiedRequest);

  }

}

Auth-interceptor.service.ts

@NgModule({

  declarations: [AppComponent],

  imports: [BrowserModule, FormsModule, HttpClientModule],

  providers: [

    {

      provide: HTTP\_INTERCEPTORS,

      useClass: AuthInterceptorService,

      multi: true

    },

    {

      provide: HTTP\_INTERCEPTORS,

      useClass: LoggingInterceptorSerice,

      multi: true

    }

  ],

  bootstrap: [AppComponent]

})

App-module.ts