★ @coder_aishya ★

HttpInterceptor Examples

Setting the new headers

Often we need to **return an API key to an authenticated API endpoint via a request Header.** Using Interceptors, we can simplify our application code to handle this automatically. Let's make a simple use case of attaching an API header key to each request.

```
export class HeaderInterceptor implements HttpInterceptor {
  intercept(httpRequest: HttpRequest<any>,
            next: HttpHandler): Observable<HttpEvent<any>>
  {
    const API_KEY = '123456';
    return next.handle(httpRequest.clone(
            { setHeaders: {API_KEY } }));
  }
}
```

@coder_aishya

Convert XML response to JSON

```
export class XmlInterceptor implements HttpInterceptor {
 constructor(@Inject(XmlParser) private xml: XMLParser) {}

 intercept(req: HttpRequest<any>, next: HttpHandler) {
    return next.handle(req).pipe(
      // proceed when there is a response; ignore other
events
      filter(event => event instanceof HttpResponse),
      map(
        (event: HttpResponse<any>) => {
          if (this.xml.validate(event.body) !== true) {
            // only parse xml response, pass all other
responses to other interceptors
            return event;
          }

          // {responseType: text} expects a string response
          return event.clone({ body:
JSON.stringify(this.xml.parse(event.body)) });
        },
        // Operation failed; error is an HttpErrorResponse
        error => event
      )
    );
  }
}
```

@coder_aishya

Adding the Authorisation token

```
export class TokenInterceptor implements HttpInterceptor {
  intercept(httpRequest: HttpRequest<any>,
            next: HttpHandler): Observable<HttpEvent<any>>
  {
    //Get token from some service
    const token: string =authService.Token;
    if (token) {
       req = req.clone({headers:
       req.headers.set('Authorization', 'Bearer ' +
                       token) });
     }
  }
}
```

@coder_aishya

Error Handling

```
export class ErrorInterceptor implements HttpInterceptor {

    constructor(private authenticationService: AuthService,
private router: Router) { }

    intercept(request: HttpRequest<any>, next:
HttpHandler): Observable<HttpEvent<any>> {

        return next.handle(request).pipe(catchError(err =>
        {
            if (err.status === 401) {
                this.authenticationService.logout();
                this.router.navigate(['/login']);
            }

            const error = err.error.message ||
                err.statusText;
            return throwError(error);
        }))
    }
```

@coder_aishya

Modify Response

```
export class DemoInterceptor implements HttpInterceptor {
 intercept(req: HttpRequest<any>, next: HttpHandler):
             Observable<HttpEvent<any>> {

    return next.handle(req).map(resp => {
        const myBody = [{
          'id': '1',
            'name': 'coderaishya',
            'html_url': 'www.coderaishya.com',
            'description': 'description'
          }];

         // on Response
        if (resp instanceof HttpResponse) {
            console.log(resp);
            console.log(resp.body);
            resp = resp.clone<any>({ body: myBody});
            return resp;
        }
    });
  }
}
```

@coder_aishya

Cancel the current Request

```
import { EMPTY } from 'rxjs';

export class DemoInterceptor implements HttpInterceptor {
  intercept(req: HttpRequest<any>, next: HttpHandler):
            Observable<HttpEvent<any>> {
      if (NotLoggedIn) {
        return EMPTY;
      }

      return next.handle(request);
  }
}
```

@coder_aishya

Change the Requested URL

```
export class DemoInterceptor implements HttpInterceptor {
  const baseURL="https://www.github.com/";

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

    const newReq = req.clone({
      url: baseURL + req.url;
    });

    return next.handle(httpsReq);
  }
}
```

@coder_aishya

Caching

```
export class CachingInterceptor implements HttpInterceptor
{
  constructor(private cache: CacheService) {}

intercept(req: HttpRequest<any>, next: HttpHandler):
Observable<HttpEvent<any>> {
    // continue request if not cacheable
    if (!this.canCache(req)) {
      return next.handle(req);
    }
  const cachedResponse = this.cache.get(req.urlWithParams);
    // return HttpResponse so that HttpClient methods
       return a value
    return cachedResponse
      ? of(new HttpResponse({ body: cachedResponse.body }))
      : sendRequest(req, next, this.cache);
  }

  canCache(req: HttpRequest<any>): boolean {
      // only cache `todo` routes
      return req.url.includes('todos');
    }
}
```

Follow me for more

@aishwarya-dhuri

@coder_aishya