

Configuración librería encriptación

Esta librería funciona como un repositorio centralizado de componentes, permitiendo compartir y reutilizar elementos visuales de manera consistente en todas las aplicaciones. Al mantener un sistema de diseño unificado, garantiza la coherencia visual y mejora la eficiencia del desarrollo, reduciendo la duplicación de código y simplificando el mantenimiento de la interfaz de usuario.

Uso

Para comenzar, sigue estos pasos:

- Instale la librería de npm crypto-js [Enlace a la librería](#)

```
npm i crypto-js
```

- Crea un archivo **.npmrc** en la raíz del proyecto, el cual va a contener la configuración para instalación de la librería alojada en Nexus Repository.
- Agregue la dependencia al archivo **package.json** (**Recuerde verificar la versión actual de la librería**) e instale las dependencias.

```
"crypto-util-library":  
"http://10.1.40.130:8081/repository/npm-hosted/crypto-util-library/-/crypto-util-library-1.0.0.tgz"
```

- Crea un servicio para centralizar y gestionar la encriptación y desencriptación de datos, utilizando los métodos proporcionados por la librería **crypto-util-library**. Esto facilita la reutilización del código y garantiza un manejo seguro y consistente de la información en toda la aplicación.

```
import { CryptoUtil } from 'crypto-util-library';
```

```
decryptResponse(data: any) {  
    return this.cryptoUtil.decryptObject(data);  
}
```

```
encryptData(data: any) {  
    return this.cryptoUtil.encryptObject(data);  
}
```

- Configura un **interceptor** encargado de la encriptación y desencriptación de los datos en las solicitudes HTTP. Este interceptor recibe como parámetro un archivo de configuración que permite definir una lista de URLs excluidas, en las cuales no es necesario aplicar estos procesos. De esta manera, se optimiza el rendimiento y se garantiza que solo las comunicaciones sensibles sean protegidas mediante encriptación.

Interceptor

```
import { CryptoConfig } from './crypto.config';
import { CryptoUtilLibraryService } from '../services/crypto-util-library.service';

export const cryptoInterceptorInterceptor = (config: CryptoConfig):
HttpInterceptorFn =>
(request, next) => {
  const cryptoService = inject(CryptoUtilLibraryService);
  const shouldSkip = config.excludedUrls.some(url =>
    request.url.includes(url)
  );
  if (shouldSkip) {
    return next(request);
  }
  if (request.body) {
    return from(cryptoService.encryptData(request.body)).pipe(
      switchMap(encryptedBody => {
        const clonedRequest = request.clone({ body: encryptedBody });
        return next(clonedRequest).pipe(
          switchMap(event => {
            if (event instanceof HttpResponse && event.body) {
              const decryptedBodyPromise =
cryptoService.decryptResponse(event.body);
              return from(decryptedBodyPromise).pipe(
                map(decryptedBody => event.clone({ body: decryptedBody }))
              );
            }
            return [event];
          })
        ),
        catchError(error => {
          if (error instanceof HttpErrorResponse && error.error) {
            const decryptedErrorPromise =
cryptoService.decryptResponse(error.error);
            return from(decryptedErrorPromise).pipe(
              switchMap(decryptedError => {
                const clonedError = new HttpErrorResponse({
                  ...error,
                  error: decryptedError,
                  url: error.url ?? undefined
                });
                return throwError(() => clonedError);
              })
            );
          }
          return throwError(() => error);
        })
      )
    );
  }
  return next(request);
});
```

```
    );
  }
  return next(request).pipe(
    switchMap(event => {
      if (event instanceof HttpResponse && event.body) {
        const decryptedBodyPromise =
cryptoService.decryptResponse(event.body);
        return from(decryptedBodyPromise).pipe(
          map(decryptedBody => event.clone({ body: decryptedBody }))
        );
      }
      return [event];
    }),
    catchError(error => {
      if (error instanceof HttpResponseError && error.error) {
        const decryptedErrorPromise =
cryptoService.decryptResponse(error.error);
        return from(decryptedErrorPromise).pipe(
          switchMap(decryptedError => {
            const clonedError = new HttpResponseError({
              ...error,
              error: decryptedError,
              url: error.url ?? undefined
            });
            return throwError(() => clonedError);
          })
        );
      }
      return throwError(() => error);
    })
  );
};
```

[←Regresar](#)

From:
<http://wiki.adacsc.co/> - Wiki

Permanent link:
<http://wiki.adacsc.co/doku.php?id=ada:howto:sicoferp:factory:new-migracion-sicoferp:front:configuracion-libreria-encriptacion&rev=1738169950>

Last update: 2025/01/29 16:59

