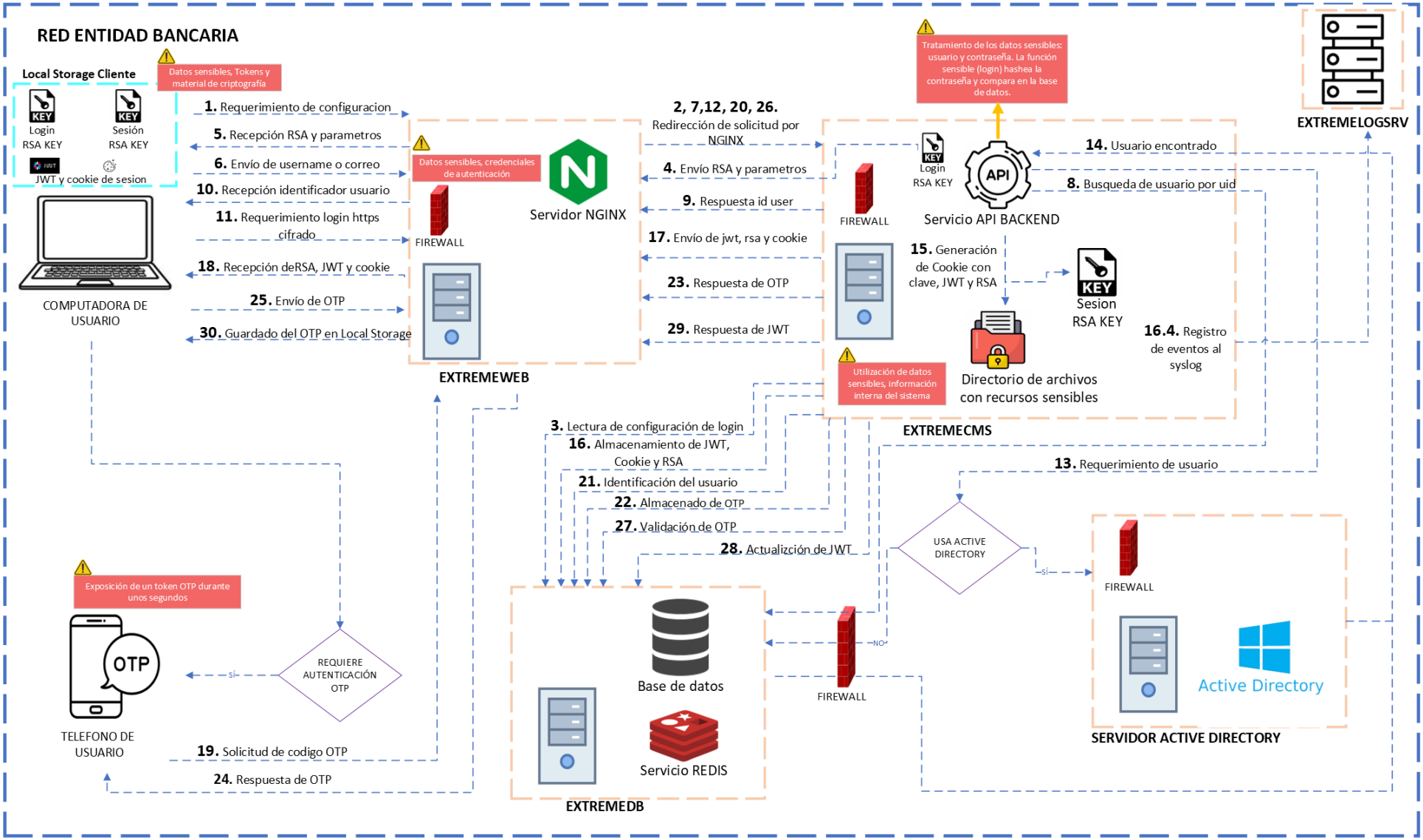
GRÁFICO DE FLUJO DE LOGIN



NARRATIVA DEL FLUJO DE LOGIN EN EL SISTEMA CMS

Entradas:

1. Usuario.
2. Contraseña.
3. OTP

Salidas:

1. Token JWT identificador de sesión
2. Cookie de sesión
3. RSA de sesión
4. Información del Usuario

Narrativa:

Para el flujo se observan hostnames que son colocados de forma práctica para un mejor entendimiento, no obstante, la distribución de servidores, bases de datos, y otros recursos pueden ajustarse de diferentes formas según la entidad requiera, pero el esquema de comunicación y el flujo en general se mantendrá. El flujo da inicio de la siguiente manera:

1. Lo primero que ocurre en una computadora del cliente al cargar la página del login es enviar un request https tipo GET al backend para obtener información de tipo de login configurado (autenticación con captcha y encriptado de datos).

2. Mediante proxi inverso configurado en el NGINX se toma la solicitud y se la envía al backend para que este devuelva la respuesta deseada.

3. Se procesa la búsqueda de datos y se envía a consultar la configuración del login a la base de datos.

4. El backend prepara la respuesta de la configuración con el RSA público para que este pueda ser almacenado por el front y hacer el envío del POST encriptando con el RSA público, las claves del RSA se encuentran ubicados en la ruta que indica la configuración del appSettings del backend del login, en el atributo RSACONFIG/PK (para la clave publica) y RSACONFIG/PBK (para la clave privada) comúnmente se recomienda ubicarlos en el directorio *"/Extreme/CMS/Keys".* La funció n que se encarga de obtener el RSA publico para la utenticación es *GetPubRSAPathHandler.Handle*

5. El cliente recibe la información de login y almacena en el local storage el RSA y la configuración del login.

6. El cliente comienza escribiendo su correo o user name (dependiendo de la configuración si es o no active directory), cuando lo termina de escribir se envía una solicitud al backend para obtener la confirmación que el usuario existe.

7. La solicitud de confirmación del cliente es redirigida por el proxi inverso del nginx.

8. La api recibe la llamada y envía a buscar el usuario por su correo, o username. De esta forma se obtiene un identificador.

9. Se envía la respuesta del id del usuario al front.

10. El cliente recibe el id y lo almacena para usarlo en el envío de la contraseña.

11. El cliente llena termina de escribir la contraseña, que es considerada como 'dato sensible' y se envían mediante una llamada *post* al NGINX con protocolo *https,* además en este punto se encripta toda la data haciendo uso del RSA para que el backend con la clave privada lo descencripte.

12. El requerimiento es recibido por el servidor EXTREMEWEB, el cual proporciona la aplicación del *frontend* y recibe solicitudes *https* para redirigirlas internamente al servidor *backend* por proxi inverso

13. El *backend* recibe el requerimiento con la función *SessionAndEncryptValidation* desencripta el cuerpo de la solicitud con la llave privada del RSA y utiliza la función sensible de *LogginCommandHandler.Handle* para manejar obtener el usuario por su id y enviar a comparar la contraseña (dato sensible) para así autenticar. Dentro de este nivel se encuentran dos opciones de autenticación:

    13.1. Servidor Base de Datos EXTREMEDB: Se consulta la tabla de usuarios (Users) por medio de la función *UserMiddlewareRepository.GetUserByMail* para obtener aquel que coincida con el usuario ingresado en el *login*.
    13.2. Servidor LDAP: Se consulta al servicio LDAP mediante la función *ActiveDirectoryRepository.AutenticateUser* para obtener y validar el que las credenciales ingresadas en el *login* están oficialmente registradas. En este caso, no se utiliza la función Bcrypt; en su lugar, se procede directamente a la generación del token. Esto implica saltarse el paso 9 y avanzar directamente al paso 10.

14. Al realizar la autenticación usando la base de datos se genera una contraseña con un *hash* obtenido por medio de Bcrypt, librería con la que se cifran las contraseñas por medio de SHA-256, luego para llevar a cabo una comparación entre las cadenas obtenidas y los datos ingresados por el usuario (nombre de usuario y contraseña) se siguen los siguientes pasos dentro de la función sensible:
    14.1. Se aplica un *hash* a la contraseña que se está enviando con BCRYPT, obteniendo así un hash válido
    14.2. BCRYP, con el método *verify,* compara los hashes (el que se genera con el que estaba ya en la base de datos en el atributo use_Password) y verifica si las contraseñas son iguales por medio de los hashes.
    14.3. Al coincidir las contraseñas, se procede con el paso 10; en caso contrario, se entra en una excepción y se emite un mensaje de credenciales incorrectas.

15. Se crea una cookie de sesión y un JWT, ambos para el manejo de sesiones. La cookie de sesión se crea en base a una configuración precargada en el arranque del proyecto por medio de *RegisterServices.* La generación del JWT consiste en lo siguiente:
    15.1. Se arman datos de usuario para enviarlos como parte de la respuesta al cliente. Estos datos indican si el usuario que se está autenticando debe enviar posteriormente un OTP para genera un Token básico con la función *JwtProvider.GenerateTknOTP*, o en su lugar, al no requerir OTP, se genera un Token completo con la función *GenerateTokenHandler.Handle* esta función a su vez llama a la función *JwtProvider.Generate*.
    15.2. Se registran los datos del usuario en el token de tal manera que logremos identificar al usuario logueado en cada requerimiento del sistema.
    15.3. Se registra la empresa y oficina del usuario, al igual que se define la fecha de expiración del JWT
    15.4. El Token se genera con una clave secreta que se encuentra en el AppSettings del *backend* en el atributo JWT/SecretKey. Esta clave secreta es un dato sensible.
    15.5. El AppSettings es un recurso sensible. Sin embargo, al estar cifrado, la data que contiene, como la clave de firmado de JWT, no permite su fácil lectura o edición.
    15.6. Se genera un RSA publico y privado de forma aleatoria con la librería *RSACryptoServiceProvider* con un parámetro de 2048 de *System.Security.Cryptography* esto es llamado en la función *CreateRSAHandler.Handle*.
16. Cuando se generan la cookie y el JWT estos son almacenados en las bases de datos. En este punto es opcional configurar si se prefiere almacenar en REDIS o en la base de datos mencionada anteriormente. La cookie, en caso de guardarse en la base de datos se guarda en la tabla de CookiesSessions. Luego de que la base de datos guarda el Token en la tabla SessionsStorage, ocurre lo siguiente:
    16.1. Se obtiene el Id del token guardado en la base de datos.
    16.2. Se genera un nuevo token el cual contiene solo la información del identificador, obtenido en el paso previo, esto se lo hace con la función *JwtProvider.GenerateBasic().*
    16.3. Antes de dar la respuesta al cliente, en el encabezado de esta se debe incluir una cookie de sesión para que el cliente pueda incluirla en sus siguientes solicitudes y de esta manera poder identificarlo. Esta cookie se genera con una clave la cual se almacena en un xml el mismo que se encuentra en un directorio especificado en el AppSettings
    16.4. Se envía un guardado al log de eventos como un registro de un evento ocurrido haciendo uso de *SyslogLoging.LogingModule*, si en algún caso la autenticación falla por un error se registra en el log por medio de *LogRecord.SaveLogError* al igual que también se guarda un registro en *LogRecord.SaveLogError*

17. El token y la cookie generados son enviados como respuesta al *frontend*.

18. El *frontend* recibe el token y la cookie de sesión, listo para guardarlo en el *local storage.* De manera que, ni el token ni la cookie contienen aún información sensible, solo la información necesaria para reconocer al usuario logueado.

19. Si el usuario necesita un OTP el *front* se lo solicitará, por lo que tendrá que ir a la aplicación de un teléfono para solicitar un nuevo código OTP. Si no tiene iniciada una sesión debe hacerlo, esta sesión de la aplicación se la hace por medio de las credenciales de autenticación y el

proceso es el mismo redactado en los pasos del 1 al 8, finalizando con el almacenamiento de un Token de aplicación que se genera en la función *LogginAppHandler.Handle.*

20. La aplicación móvil hace un requerimiento al *backend,* pasando primero por el NGINX, para obtener un OTP.

21. El *backend* procesa el requerimiento identificando primero al usuario que está haciendo la solicitud del OTP para generar el OTP numérico en base al identificador de usuario como semilla para la generación, para ello se consume la función *GenerateOTPHandler.Handle*.

22. Se va a almacenar el OTP a la tabla Users en el atributo LastOTP con la fecha en que caduca dicho OTP en el atributo OTPExpiration. Se aplica un registro en el log de eventos por medio de *SyslogLogging.LoggingModule* y de igual manera si hay un error lo registra en *LogRecord.SaveLogError*

23. El *backend* envía esta respuesta (OTP) al *frontend*.

24. El cliente obtiene esta respuesta en la aplicación móvil, el código OTP solo dura unos segundos antes de caducar y cambiar, por lo que, aunque es un dato sensible no es necesario eliminarlo. Durante el envío y recepción de los datos sensibles toda la comunicación se realizó por SSL resultando en mantener los datos seguros.

25. El usuario envía el OTP obtenido por medio del cliente web.

26. El NGINX redirecciona ese envío al *backend* para que este lo pueda validar.

27. El *backend* al recibir la solicitud valida que el envío se lo haga con una sesión válida obteniendo los datos del JWT, esto se lo hace con dos funciones *AuthorizationPermissionService.GetClaims* y *AuthorizationPermissionService.GetOriginalToken* valida el OTP recibido, este OTP se va a comparar con el que está en la base de datos validando que solo se lo use si está dentro de la última fecha de caducidad en el registro de la tabla Users en el atributo OTPExpiration, si la fecha es menor a la actual se procede a comparar si no entonces se rechaza, esto se lo hace en la función *ValidateOTPHandler.Handle*

28. Al coincidir el *backend* genera un nuevo JWT y lo actualiza en la base de sesiones para que el usuario pueda hacer uso finalmente de la API, primero obtiene los datos de la sesión actual con *TokenStorageRepository.GetSessionStorage,* para modificar su token y luego enviarlo a actualizar con *TokenStorageRepository.UpdateToken.* Se registra un log de evento en caso usando *SyslogLogging.LoggingModule*

29. Se envía el nuevo token al *frontend* pasando por NGINX

30. Por último, el cliente guarda el token actualizado para comenzar a consumir la API del *backend* haciendo uso del mismo token.