

Credential Provider

! → validarea și verificarea
codurilor de acces

Metode de ac

→ crea o auth și de ac
stergere o auth => Aceste metode
relează UserModel's credential
manager care este responsabilul
și să stie unde să atragă/
și să scrie credentiale, de ex
local storage / federated storage

Credential Intercept Validator

prin un parametru
generic care extinde
în Credential Model

permete Keycloak

să stie că acest furnizor
poate fi folosit și

pt a valida o auth pt un
Authenticator

isValid()

credential-data

secret-data

json

Vault Provider → token FAM

EventListener Provider

⇒ Atâtăzi a asculta cum este
trimis un token FAM și pt a-l stoca
în vault cu cum trebuie

Code Acces

severul Keycloak generă un cod
de acces pe care îl trimite mai departe
pt FAM.

când tără străot temporar enderec
pt a putea face verificarea

Cache Infinispan

→ este pentru distribuția uniformă a datelor din cache în
cluster

Cum ar trebui să fie stocat codul de acces?

⇒ **Infinispan cache**

→ este un sistem de caching și stocare a datelor in-memory = rapid și

scalabil

- securitate:
 - controlul accesului => permite configurația controlului accesului și lista de utilizatori, astfel încât doar clientii autorizați să poată accesa și manipula cache-ul.
 - securitatea transportului => atunci când facem distribuția uniformă între client și Infinispan poate fi configurația folosind SSL/TLS pt acripta comunicația între moderni, prevenind interceptarea datelor în transmisiune.
 - securitatea stocării: deși Infinispan este un sistem de stocare in-memory, există opțiuni pt a persista datele în moduri securizate folosind criptarea la disc pt stocarea offline.

• criptarea datelor din cache

• securitatea clientului => modernele de interfață și pot aloca clientului

• timp de expirare.

1. **Credential Provider** => este componenta care gestionează credențialele specifice, în acest caz codurile de acces. Aici se va implementa logica pt generația, stocarea și validarea codurilor de acces. Acest provider va interacționa cu **Infinispan** cache sau cu un Vault provider pt a retine codurile de acces în mod sigur.

2. **Authenticator** => Aceasta este responsabilă pt procesul de autentificare propriu-zis, adică verificarea dacă codul de acces introdus de utilizator este valid. Authenticatorul va verifica codul de acces în locația de stocare temporară și va compara codul introdus de utilizator cu cel stocat. Dacă codul corespunde și este înălțat (nu a expirat), atunci utilizatorul va fi autentificat.

3. Vault Provider \Rightarrow În ceea ce nu dorește să stocă codurile de acces direct în cache (poate din motive de securitate suplimentare), ai putea să folosești un Vault Provider. Aceasta introducează în sistem extors de gestionare a secretelor (de ex: HashiCorp Vault), unde codurile de acces pot fi stocate în mod securizat) ???? \Rightarrow Trba criptare ai. doar Keycloak să poată crita/decripta datele secrete.
4. Event Listener Provider \Rightarrow Această componentă asază și reacționează la diferite evenimente din cadrul Keycloak. De ex: logare când un cod este generat, când expira un cod \Rightarrow acțiunea necesară, când un token FCM este trimis de către un utilizator să îl stochăm în vault.
5. Infusepm Cache \Rightarrow este sistemul de caching folosit de Keycloak pt a stoca temporar codurile de acces. Infusepm oferă performanță înaltă și este bine integrat în Keycloak, făcându-l o opțiune bună pt stocarea datelor care trebuie să fie rapid accesibile și care nu sunt persistente pe termen lung.
6. Secret Data \Rightarrow Acest model de date conține informații sensibile, cum ar fi codul de acces în sine. Secret Data ar trebui să fie stocat într-un mod securizat și să fie accesabil doar sistemului de auth pt validare.
7. Credential Data \Rightarrow Acest model de date conține metadate legate de credențiale, și care codul de acces va expira. Aceasta agită la det. cum ar fi data și ora la care codul de acces este învăluit atunci când este introdus de utilizator doar un cod de acces este învăluit atunci când este introdus de utilizator.
8. Credential Model \Rightarrow este folosit pt a reprezenta credențialele unui utilizator care include SecretData (ex: cod acces) + Credential Data - legată de utilizator.

SecretData

- \rightarrow cod acces
- \rightarrow token FCM

Credential Data

\rightarrow data de expirare

- \rightarrow data la care a fost emis
- \rightarrow data când a fost ultima dată emisă.

Event Listener Provider

\rightarrow Ex de utilizare: trimiterea unei token de la un util.

Vault

(VS)

- securitate ridicată
- specializat în stocarea secretelor
- poate servi mai multe aplicații
- rotatia și gestionarea cheilor de mătă al secretelor
- accesul la secrete mai lent =>
=> aplicare în securitatea și operațiile de criptare/decriptare

Infinispan

- oferă acces rapid la date
- setare timp de expirare
- utilizare în keycloak mai simplă,
deoarece este de la integrat și folosit pt sesiuni și alte date temporare

Vault Provider

- tokenFCM

criptat / => decriptare

key session

Infinispan

- code access

Deci fluxul va fi următorul :

- să am Credential Data în core specific date normale aici (metadate) (1)
- să am SecretData în core stocat doar identificatorul unic al credentialelor. (1)
- Credential Model în core associaz un utilizator de o credencială și acesta tipă le credenciale primărește să fie reușită și de restul sistemului.

Authenticator

⇒ ar trebui să se ocupe de validarea, genera
rearea și salvarea
șifriror asociat.

- ⇒ se va genera codul la finalizarea succes a primului factor
și salvat în infiniSpan
- ⇒ validarea se va face după trimiterea push notif către util.

Credential Provider

⇒ este folosit mai ales pt a gestiona credentialele
utilizatorilor, cum ar fi parolele. Logica de validare + stocare a tokenelor
FCM atunci când sunt interceptate.

Credential Data

Secret Data

=> folosite în general pt a gestiona detalii legate
de credentiale în Keycloak.

Aici trebuie să implementăm reguli pt adăugarea, actualizarea, și
recuperarea tokenelor RRF din vault. \leftrightarrow creare unei noi
tip de credentiale în Keycloak care reprezintă tokenurile
FCM și sauva logica pt a manipula aceste credentiale.

Token Hardware

- ① Utilizatorul se conectă la aplicația web, implementând primul factor de autentificare. (username + parolă Keycloak)
- ② După Keycloak va selecta cel de-al 2-lea factor =>
 - va genera un string și îl va trimite mai departe utilizatorului
 - dispozitivul utilizatorului trimite prin NEXU acesta monograma tokenului hardware
 - tokenul hardware va solicita introducerea unui PIN
 - tokenul se menține cu ajutorul cheii private monograma și o trimită înapoi la Keycloak.
 - dacă semnatura este validă (Keycloak va verifica semnatura cu ajutorul cheii publice asociate utilizatorului) => semnatura este validă atunci utilizatorul este autentificat cu succes.

Tokenul hardware va contine:

Când se va furniza monograma către tokenul hardware se va trimite și informație suplimentară despre aplicația web.

Security:

- ① Este imposibil să atacurile de tip **phishing** ⇒ dacă utilizatorul ajunge pe un site legitim ⇒ nu va avea acces la cheia privată de token doar dacă îl fură.
- ② Cheia privată de pe tokenul hardware nu este în niciun moment accesibile sistemele de spionaj al altor utilizatori. ⇒ dacă computerul utilizatorului are programe malware, acesta nu poate extrage cheia privată din token. ⇒ totuși procesarea criptografică este realizată de token și îndărătușă de atac.

De legătură suntem valui de la PN de cel de la Token Hardware.