Chapitre : Système d'autorisation par rôles (FastAPI)

C'est quoi ?

Un **système d'autorisation par rôles** permet de **limiter l'accès à certaines routes ou actions** d'une API en fonction du **niveau d'autorisations** de l'utilisateur (ex: admin, lead, opérateur).

Pourquoi on l'utilise ?

- Pour **protéger certaines routes** sensibles (ex: suppression, modifications critiques).
- Pour **organiser les droits** en fonction de profils métiers (ex: un "lead" peut ajouter une machine, un "operator" non).

Comment ça fonctionne ?

1. L'utilisateur s'identifie avec un **token** JWT ou autre.
2. On décode ce token et on vérifie son identité.
3. On regarde son **niveau d'autorisation** (ex: "operator", "lead", etc.)
4. On **bloque ou autorise l'accès** à la route en fonction de son rôle.

Vocabulaire essentiel

- **Authorization** ≠ Authentication : ici on parle des **droits**, pas juste de l'identité.
- **Depends** : permet d'écrire des "fonctions de vérification" utilisables dans les routes.
- **Raise HTTPException(403)** : signifie "interdiction d'accès".

Syntaxe et construction

On crée une fonction qui vérifie les autorisations :

```
from fastapi import Depends, HTTPException

def checkAuthorisation(user: dict = Depends(getCurrentUser)):
    if user['authorisation'] not in ('lead', 'supervisor'):
        raise HTTPException(status_code=403, detail="Not enough rights")
```

Puis on la branche sur une route :

```
@app.post("/secure")
def secureEndpoint(user: dict = Depends(checkAuthorisation)):
    return {"message": "OK"}
```

Représentation visuelle

```
[Request + token] → [getCurrentUser] → [checkAuthorisation] → route OK/403
```

Résultat

- Les routes sont **filtrées par niveau d'accès**.
- Tu peux protéger certaines opérations selon le profil utilisateur.
- Tu as un système d'autorisations **propre, réutilisable, maintenable**.