## 🔐Chapitre : Authentification et sécurité (Token JWT avec FastAPI)

---

### 🧰1. C'est quoi ?

C'est un système qui permet à un utilisateur de **s'authentifier** (preuve d'identité) via un **nom d'utilisateur et un mot de passe**. Une fois connecté, il reçoit un **jeton (token JWT)**, qu'il doit envoyer dans chaque requête suivante pour prouver qu'il est bien connecté.

---

### 🕐2. Pourquoi on l'utilise ?

- **Sécuriser les accès** : empêcher l'accès aux routes sensibles aux utilisateurs non connectés.
- **Identifier un utilisateur** : savoir *qui* fait une requête.
- **Standard d'API REST** : le token évite d'avoir une session en mémoire côté serveur.

---

### ⚙️3. Comment ça fonctionne ?

1. L'utilisateur envoie un POST `/token` avec son login et mot de passe.
2. Si les identifiants sont bons → le serveur crée un **token JWT** (crypté avec une clé secrète), qui contient l'identité de l'utilisateur.
3. L'utilisateur garde ce token côté client.
4. À chaque nouvelle requête → il met ce token dans l'en-tête HTTP `Authorization: Bearer <token>`.
5. Le serveur **vérifie la signature du token** (grâce à la clé secrète), puis extrait les infos (comme l'utilisateur).

---

### 📚4. Vocabulaire essentiel

- **JWT (JSON Web Token)** : petit fichier texte sécurisé contenant des infos (souvent l'utilisateur).
- **payload** : contenu du token (ex. : `{"sub": "toto"}`)
- **signature** : permet de vérifier que le contenu n'a pas été falsifié
- **OAuth2PasswordBearer** : outil de FastAPI pour lire le token envoyé par le client
- **Depends** : outil de FastAPI pour injecter automatiquement des fonctions dans les routes protégées

---

### 🧱5. Syntaxe & construction

#### 🕰️Installer les dépendances :

```
pip install python-jose[cryptography] passlib[bcrypt]
```

### 🕰️1. Créer un token JWT

```python
from jose import jwt

def createAccessToken(data, secret, expiresDelta):
    toEncode = data.copy()
    expire = datetime.utcnow() + expiresDelta
    toEncode.update({"exp": expire})
    return jwt.encode(toEncode, secret, algorithm="HS256")
```

### 🕰️2. Route POST /token (login)

```python
@app.post("/token")
def login(form_data: OAuth2PasswordRequestForm = Depends()):
    # vérifier login/mdp
    # créer token avec createAccessToken()
    return {"access_token": token, "token_type": "bearer"}
```

### 🕰️3. Lire le token dans d'autres routes

```python
oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")

@app.get("/secure")
def read_secure(token: str = Depends(oauth2_scheme)):
    payload = jwt.decode(token, SECRET_KEY, algorithms=["HS256"])
    user = payload.get("sub")
    return {"user": user}
```

---

## 📖6. Représentation visuelle

```
Client → POST /token {login, password}  → Serveur → vérifie → crée JWT →
retourne {access_token}

Client → GET /machines (avec token dans headers)
    ↳ Authorization: Bearer <token>
            Serveur → vérifie signature JWT → extraie user → continue
```

---

## 🔗7. Résultat attendu

- Un token sécurisé contenant les infos utilisateur, à insérer dans les headers `Authorization`.
- Un backend capable de :
- générer ce token (`POST /token`),
- et de le lire/valider sur des routes protégées (`Depends(oauth2_scheme)`).