EXPLICATION DE L'AUTHENTIFICATION

1. Mot de passe

- On ne stocke jamais de mot de passe en clair
- On stocke un hash crypt
- A la connexion on re hash le mot de passe saisi pour vérifier (bcrypt.compare)

2. JWT

- Le serveur fabrique un token signé (jwt.sign) qui contient un payload (ex: {id, username}) + une expiration
- Le serveur ne stocke pas le token
- Le front stocke le token (local storage) et le renvoi à chaque requête

3. Route protégée (middleware)

- Le middleware lit Authorization: Bearer <token>
- Vérifie la signature + expiration (jwt.verify)
- Si OK → il accroche req.user puis next()

SIGNUP

```
sequenceDiagram
    participant F as Front
    participant B as Back (Express)
    participant H as bcrypt
    participant DB as DB (Postgres)

    F->>B: POST /api/signup {username, password}
    B->>DB: SELECT id FROM users WHERE username=?
    DB-->>B: rows (0 ou 1)

    alt username dispo
        B->>H: bcrypt.hash(password)
        H-->>B: passwordHash
        B->>DB: INSERT users(username, passwordHash)
        DB-->>B: user {id, username}
        B-->>F: 201 {user}
    else username déjà pris
        B-->>F: 409 {error: "Username already taken"}
    end
```

LOGIN

```
sequenceDiagram
participant F as Front
participant B as Back (Express)
participant H as bcrypt
participant J as JWT
participant DB as DB (Postgres)

F->>B: POST /api/login {username, password}
B->>DB: SELECT * FROM users WHERE username=?
DB-->>B: user {id, username, passwordHash} (ou null)
```

```
alt user existe
B->>H: bcrypt.compare(password, passwordHash)
H-->>B: true/false

alt password OK
B->>J: jwt.sign({id, username}, secret, exp=1h)
J-->>B: token
B-->>F: 200 {token}
F->>F: localStorage.setItem("token", token)
else password KO
B-->>F: 401 {error: "Invalid credentials"}
end
else user n'existe pas
B-->>F: 401 {error: "Invalid credentials"}
end
```

1. Le front créer une requête HTTP

Le navigateur envoie littéralement :

```
POST /api/login
Content-Type: application/json
{
  "username": "alice",
  "password": "1234"
}
```

C’est **ça** qui transporte le username et le password.

2. Express transforme ça en req.body

Grâce à :

```
app.use(express.json());
```

Express lit le JSON et fabrique :

```
req.body = {
  username: "alice",
  password: "1234"
}
```

3. Le controller lit le req.body

```
const { username, password } = req.body;
```

4. return res.json({ token });

Le front ne sait jamais “si le mot de passe est bon”.

Il sait seulement : “est-ce que le serveur m’a donné un token ?”

L’utilisateur est connecté ici : localStorage.setItem("token", data.token);
“J’ai un token → je suis connecté.”

5. **Comment le front vérifie que la connexion est encore valide ?**

Quand l’app démarre : validateToken()

qui fait :

```
GET /api/me
Authorization: Bearer <token>
```

Si le token est :

- valide → le serveur renvoie `{ user }` → front connecté
- expiré / faux → 401 → front déconnecté

Le mot de passe sert à obtenir le token.

Le token sert à prouver qu’on est connecté.

```
flowchart TD
    A[App démarre] --> B{Y a-t-il un token<br/>dans localSt
orage ?}

    B -->|Oui| D[Appelle GET /api/me<br/>avec Bearer token]
    B -->|Non| C[User = null<br/>Non connecté]
```

```
    D --> E{Réponse du serveur ?}

    E -->|401 / erreur| F[Supprimer le token<br/>User = nul
l]
    E -->|200 + user| G[User = données reçues<br/>Connecté]

    G --> H[L'app affiche<br/>les pages protégées]
    C --> I[L'app affiche<br/>login/signup]
```

ROUTE PROTÉGÉES

```
sequenceDiagram
participant F as Front
participant B as Back (Express)
participant M as Middleware requireAuth
participant J as JWT
participant C as Controller me

F->>B: GET /api/me (Authorization: Bearer <token>)
B->>M: requireAuth(req,res,next)

alt header Authorization absent
M-->>F: 401 {error:"No token provided"}
else header présent
M->>J: jwt.verify(token, secret)
alt token valide (signature + exp OK)
J-->>M: decoded {id, username, iat, exp}
M->>M: req.user = decoded
M->>C: next()
C-->>F: 200 {user:{id, username}}
else token invalide/expiré
J-->>M: error
M-->>F: 401 {error:"Token invalid"}
end
```

```
end
```