Quelques éléments à propos de l'authentification

- *Capability/apiKey* : token opaque
- Session
- Token transparent : jwt

Authentification par "capability"

- **Principe** : un token opaque dans l'URL ou dans un header permet l'accès à une ressource
- **Intérêts** :
  - simple, pas de transport d'identifiant/mot de passe
- **Risque** : partage accidentel de l'URL/token
- **Utilisation** :
  - Partage de ressources ,
  - Clé d'API (API Key) pour identifier 1 client
  - *refresh* token pour authentification expirée
- **Conseil** : usage unique, re-génération après chaque utilisation, ou usage côté serveur (API key)

Cliquez pour ajouter un titre

- Le token peut être transporté dans l'URL ou dans un header

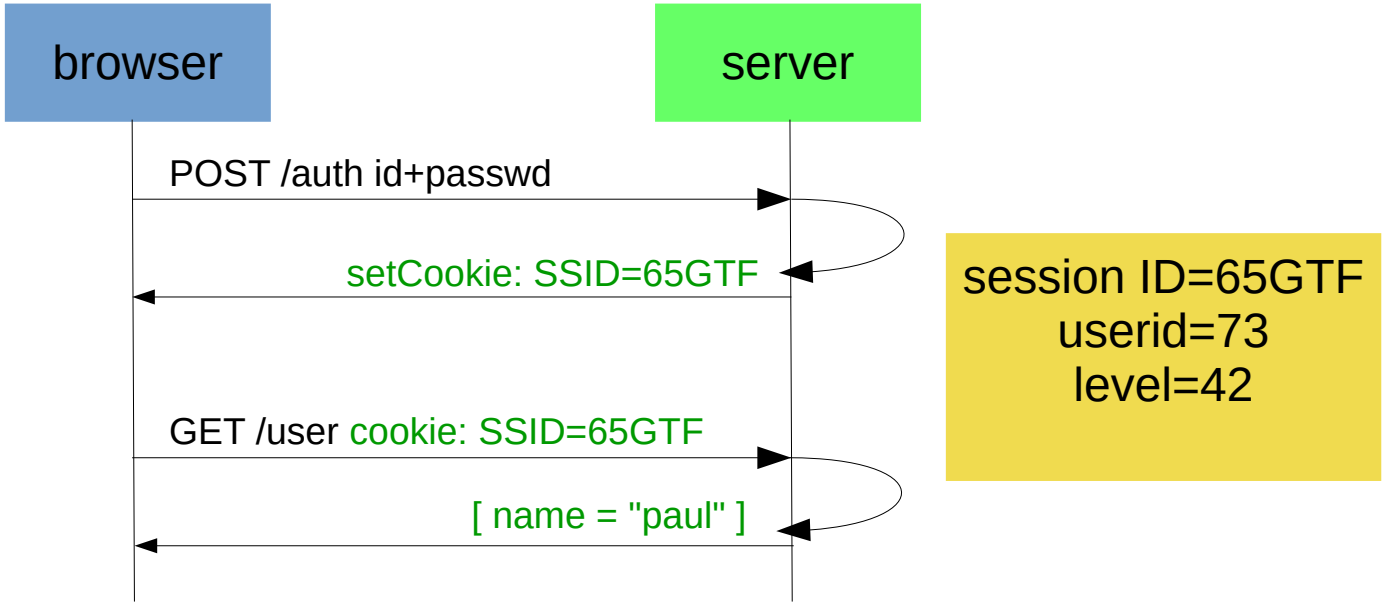
```
GET /orders/345443876?token=FTSRghxu78hskkx9Nqfr345h7G
GFDE21h
```

```
GET /orders/345443876
X-app-token: FTSRghxu78hskkx9Nqfr345h7GGFDE21h
```

```
GET /orders/345443876
Authorization: Bearer FTSRghxu78hskkx9Nqfr345h7GGFDE21h
```

Authentification basée sur la session

- **Principe** : un profil spécifiant un niveau d'accès est conservé en session sur le serveur après authentification
  - l'identifiant de session est échangé dans un cookie
- **Intérêt** : le cookie de session est positionné automatiquement par le navigateur



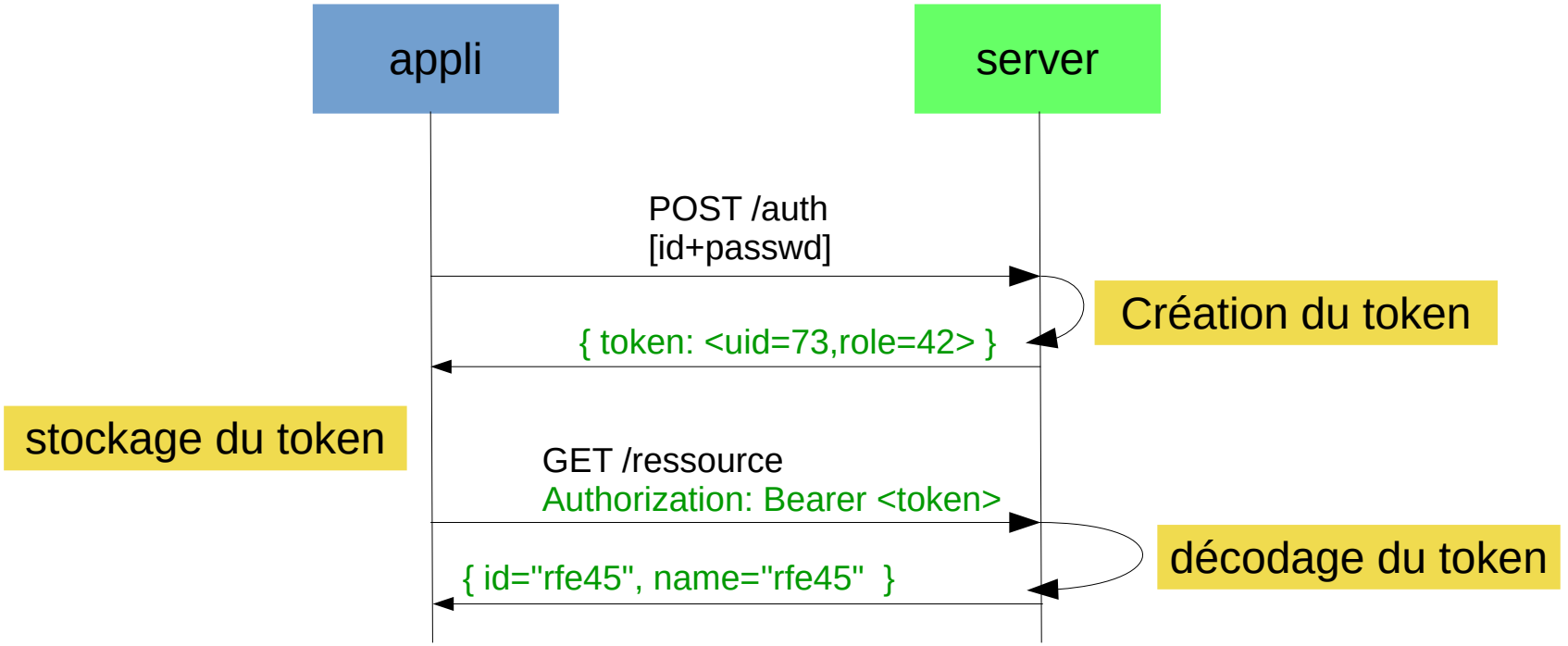
Session

- **Risques** :
  - vol de la session : interception du cookie de session
  - CSRF : l'exécution non choisie d'une requête malveillante se fait au sein de la session en raison du positionnement automatique du cookie
- **Utilisation :**
  - authentification et contrôle des droits d'accès dans une application web classique pouvant fonctionner sans javascript

```
GET /commandes/345443876
Cookie:PHPSSID=FTSRghxu78hskkx9Nqfr345h7GGFDE21h
```

Authentification basée sur un token type JWT (Json Web Token)

- **Principe** : un token JWT est un token **transparent** (= lisible) transportant des informations (identité, rôle, expiration...)
- Le token est conservé par le client et n'a **pas besoin d'être stocké côté serveur**



Token JWT

- Le token est construit et ***signé*** par le serveur, il est le seul à pouvoir vérifier la signature grâce à un **secret**
- **Intérêts pour l'authentification :**
  - stockage côté client : session et stockage côté serveur inutile
  - bien adapté aux api et micro-services (stateless)
  - pas de problème de CSRF
- **Risques :** vol du token

```
GET /commandes/345443876
Authorization: Bearer u78hskkx9Nqfr345h7GGFDE21h2598Jendt
```

Access Token / refresh Token

- Un token JWT contient des informations d'identité utiles au contrôle d'accès → ***access token***
- Il a une **durée de vie limitée**, parfois courte
- Pour éviter de demander au client de se ré-authentifier et de fournir et transporter à nouveau ses credentials, on peut fournir en plus un ***refresh token*** : token à durée de vie plus longue permettant de demander la re-génération d'un *access token*

Autres utilisations des token JWT

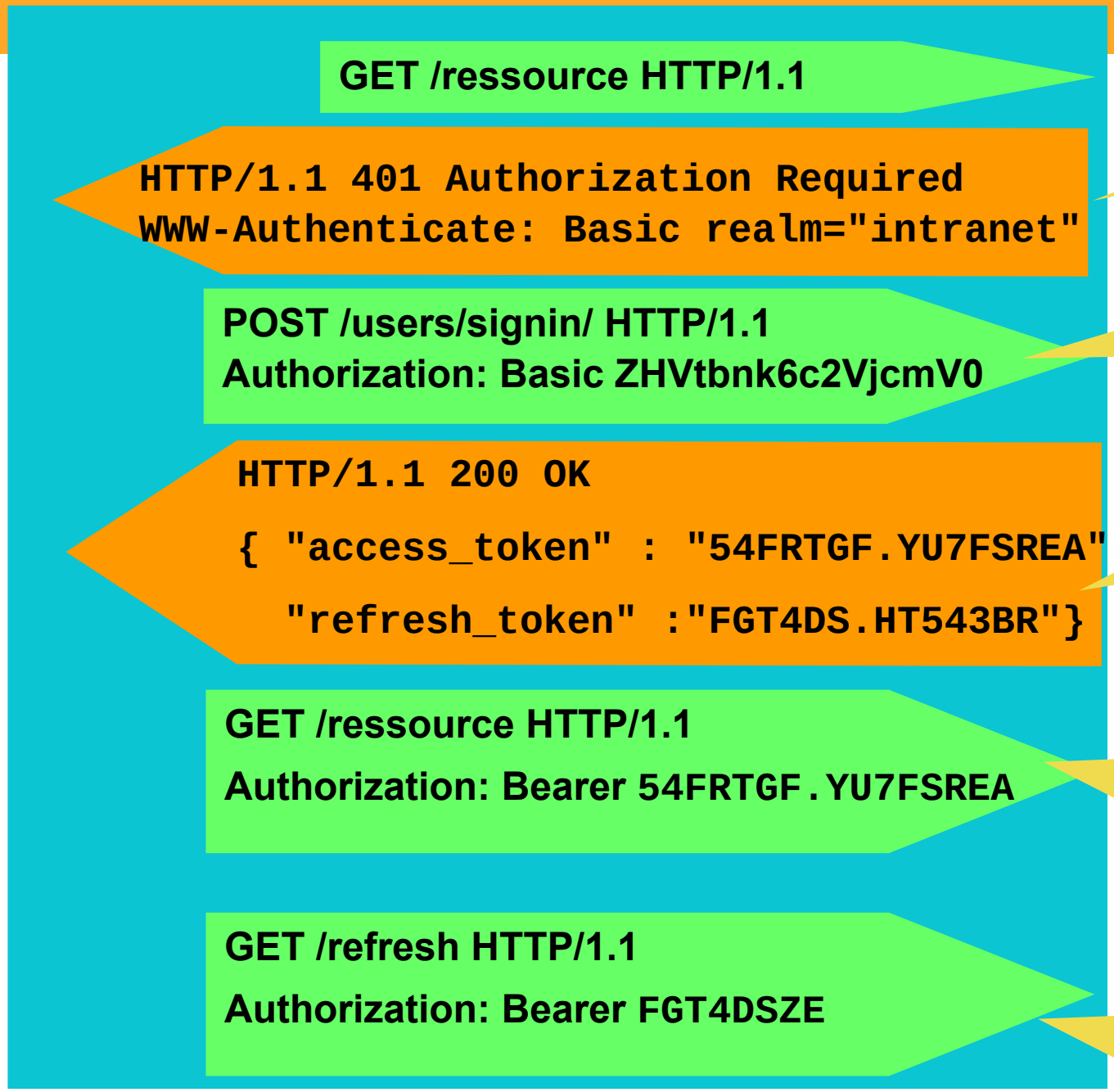
- Activation de compte / renouvellement de mot de passe : identifiant de compte et date d'expiration sont transportés dans le token
- Partage de ressource : identifiant de ressource et date d'expiration sont transportés dans le token
- Clé d'API : identifiant de client et date d'expiration dans le token

```
POST /activate?token=u78hskkx9Nqfr345h7GGFDE21h2598Jendt
POST /reset?token=a58hsNqfr345h7GGFDE21jq65etgs
GET /ressource/5gtsr54?token=8hsNqfr345h7GGFDE
```

Authentification à base de token JWT : scénario

1) Le client émet 1 requête d'accès vers une ressource
2) Le serveur répond avec un code 401 et une URL de redirection vers l'authentification
3) Le client fait 1 demande vers le serveur d'authentification en utilisant l'url et en transmettant les credentials
4) Le serveur vérifie les credentials, génère un *access token* et un *refresh token* et les retourne au client
5) Le client place l'*access token* dans toutes ses requêtes vers le serveur
6) Pour chaque requête, le serveur contrôle la validité de l'*access token* **et** réalise le contrôle d'accès
7) Lorsque l'*access token* n'est plus valide, le client utilise le *refresh token* pour obtenir un nouvel *access token*

Scénario dans le contexte HTTPS



realm : identifiant de la restriction

credentials : **iduser:passwd** encodé en Base64

retourne un access token + refresh token

les requêtes transportent l'access token en mode **Bearer** dans le header Authorization

le refresh token est également transporté en mode Bearer

Les tokens JWT (rfc 7519)

- Un token JWT comprend 3 parties :
  - un **entête** : spécifie le type de token et de hash
  - un **contenu** (payload) : des données placées par le serveur
  - une **signature** : permet au serveur de vérifier la validité du jeton, généré par un algo utilisant une clé secrète (HMAC ou RSA)
- Le contenu : un objet json
  - des propriétés **prédéfinies** standardisées
  - des propriétés liées à **l'application**

Principe de génération / validation

- A la **génération** d'un token, le serveur **signe** en hashant **le contenu du token** à l'aide d'un algo utilisant une clé secrète ; la signature est incluse dans le token

- Pour **contrôler la validité** d'un token JWT transmis par un client, le serveur extrait le contenu du token puis
  - vérifie sa validité du point de vue des dates (exp, nbf)
  - calcule la signature en utilisant la même clé, et s'assure qu'elle est identique à celle reçue, ce qui garantit que le contenu n'a pas été modifié

Jwt : le payload

- Propriétés prédéfinies (optionnelles) :
  - "**iss**" : "issuer", identifie l'émetteur du token
  - "**sub**" : "subject", le sujet du token : en général ID de l'utilisateur authentifié
  - "**aud**" : "audience", destinataires du token – le serveur recevant un token doit vérifier qu'il lui est bien destiné
  - "**iat**" : "issued at", date d'émission du token
  - "**exp**" : "expires", date d'expiration du token
  - "**nbf**" : "not before", date de validité du token
  - "**jti**" : "jwt id", identificateur unique du token

Exemple

base64Url(header)

```
{
  "alg" : "HS512",
  "typ" : "JWT"
}
```

```
{
  "iss" : "https://auth.myapp.net",
  "aud" : "https://api.myapp.net",
  "sub" : "HA56F-D4EB7",
  "iat" : 1513330929,
  "exp" : 1513334529,
  "data": {
     "user" : "jean.neymar@gmal.com",
     "role" : 100
}
```

base64Url(payload)

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.
eyJpc3MiOiJodHRwOi8vYXV0aC5teWFwcC5uZXQiLCJhdWQiOiJodHRwOi8vYXBpLm15YXBwLm5ldCIsImlhdCI6MTUxMzMzMDkyOSwiZXhwIjoxNTEzMzM0NTI5LCJkYXRhIjp7InVzZXIiOiJlYW4ubmV5bWFyQGdtYWwuY29tIiwicm9sZSI6MTAwfX0.
kg65cK2uP2fAStQPhm1klpvqnTVPM9uZWyBFLBTuv-4
```

header

payload

signature

HMACSHA512(**base64Url(header)** . **base64Url(payload)**, **SECRET_KEY**)

En PHP

- Utiliser une librairie : firebase/php-jwt , lcobucci/jwt
- Exemple avec firebase/php-jwt :

```php
use Firebase\JWT\JWT;

$payload = [ 'iss'=>'http://auth.myapp.net',
                        'aud'=>'http://api.myapp.net',
                        'iat'=>time(),
                        'exp'=>time()+3600,
                        'sub' =>  $user->id,
                        'data' => [
                          'role' =>  $user->role,
                          'user' =>  $user->email
                        ]
] ;

$token = JWT::encode( $payload, $secret, 'HS512' );
```

- Décoder le token et vérifier sa signature :

```php
use Firebase\JWT\JWT;
use Firebase\JWT\Key;
use Firebase\JWT\ExpiredException;
use Firebase\JWT\SignatureInvalidException ;
use Firebase\JWT\BeforeValidException;

try {
  $h = $rq->getHeader('Authorization')[0] ;
  $token = sscanf($h, "Bearer %s")[0] ;
  $payload = JWT::decode($token, new Key($secret,'HS512' ) ) ;

} catch (ExpiredException $e) {

} catch (SignatureInvalidException $e) {

} catch (BeforeValidException $e) {

} catch (\UnexpectedValueException $e) { }
```

- Décoder le token sans vérifier la signature : le contenu est toujours lisible – dans les cas où on peut faire confiance

```php
$token_line = $request->hasHeader('Authorization') ?
            $request->getHeaderLine('Authorization') : null;

list($token) = sscanf($token_line, "Bearer %s");


// decode without validation
list($header, $payload, $signature) = explode('.', $token);
$payload = json_decode(base64_decode($payload), true) ;
```

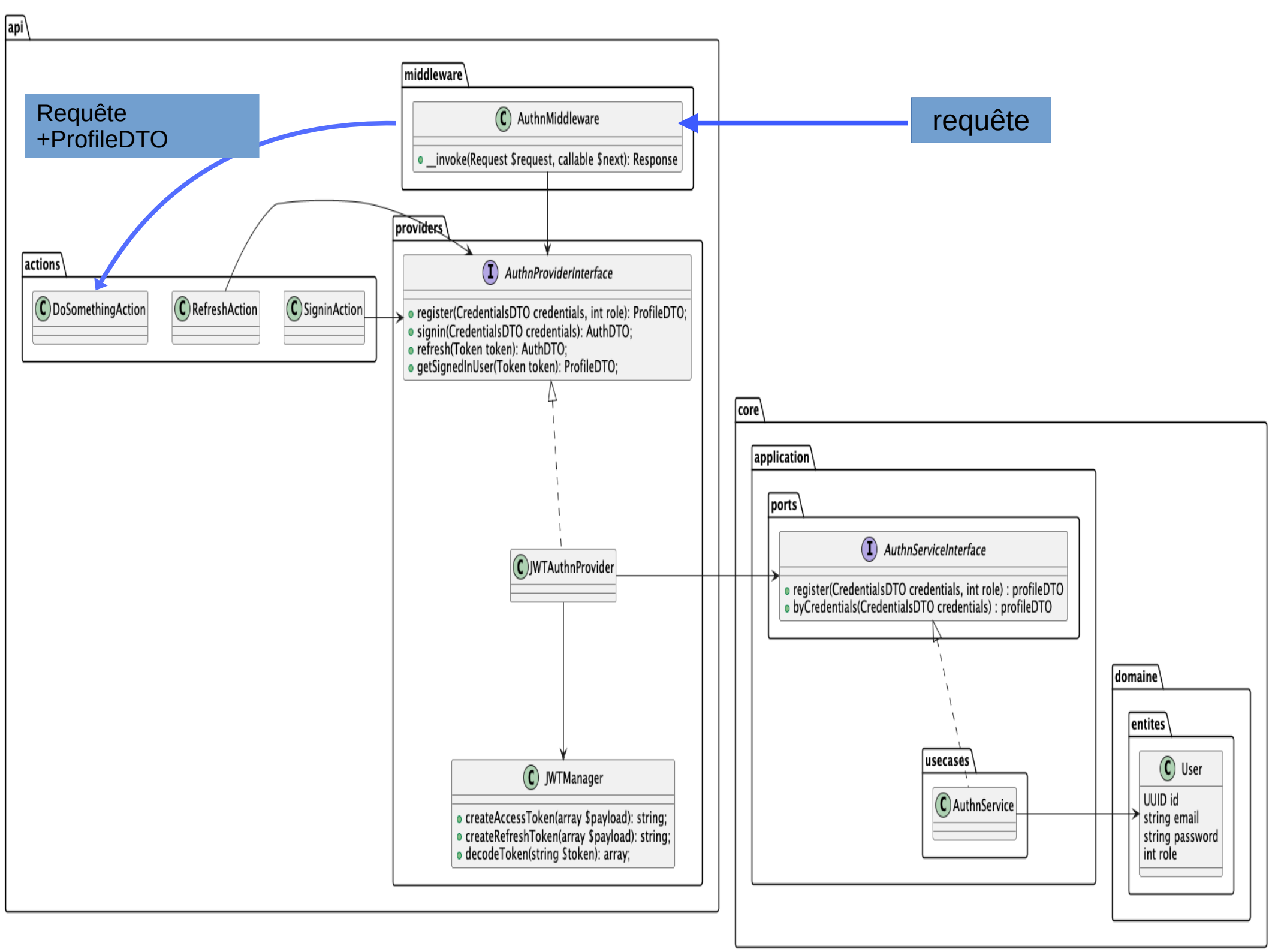
La gestion du secret

- Le secret …
  - doit rester secret !
  - et difficile à deviner :-)

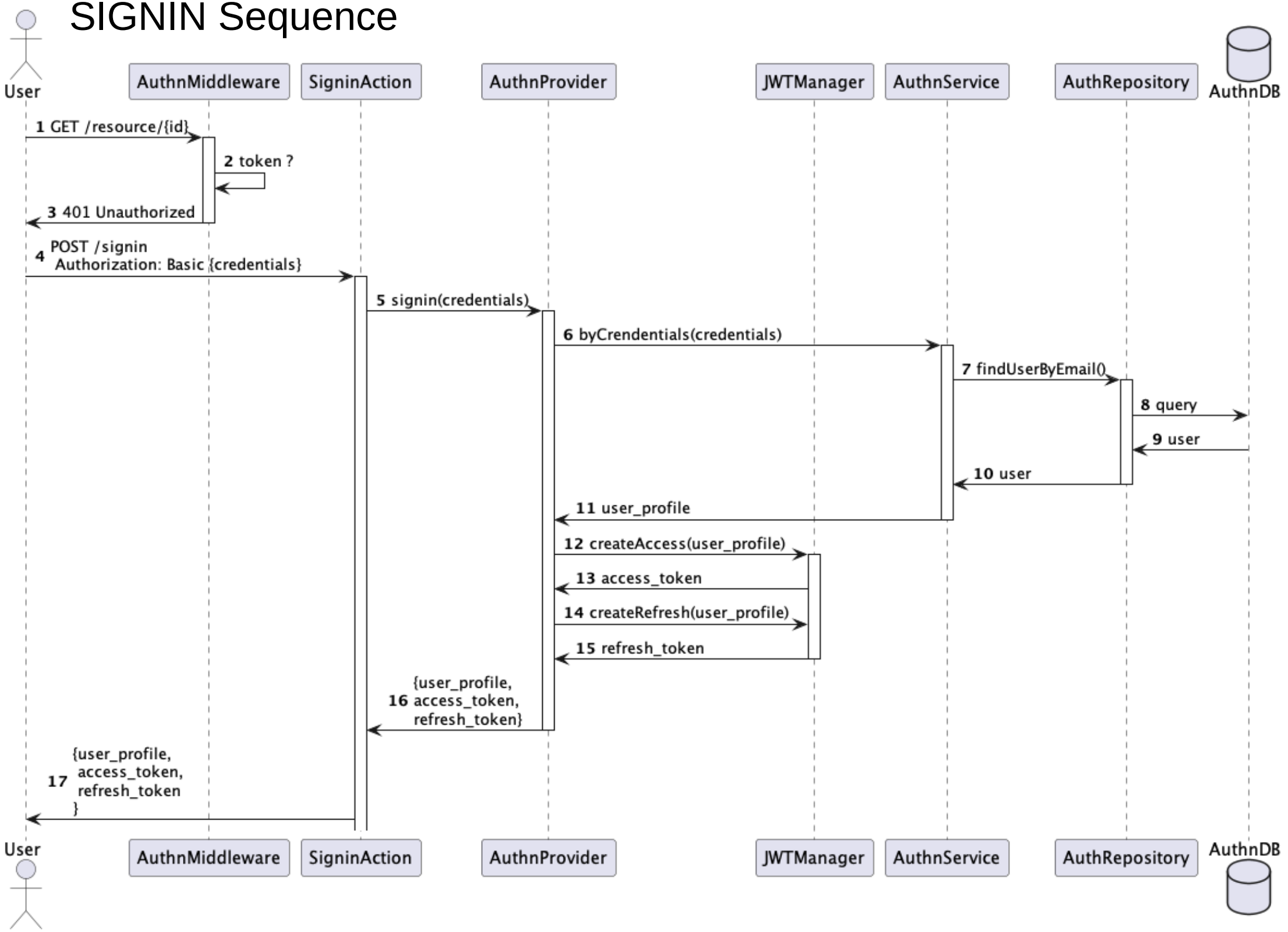
- Bonnes pratiques :
  - le secret est stocké dans une variable d'environnement
  - à définir dans le fichier docker-compose.yml
  - récupéré en PHP dans le tableau $_ENV
  - généré aléatoirement (random_bytes), suffisamment long (32)

Mise en œuvre de l'authentification à base de jetons JWT - Authn

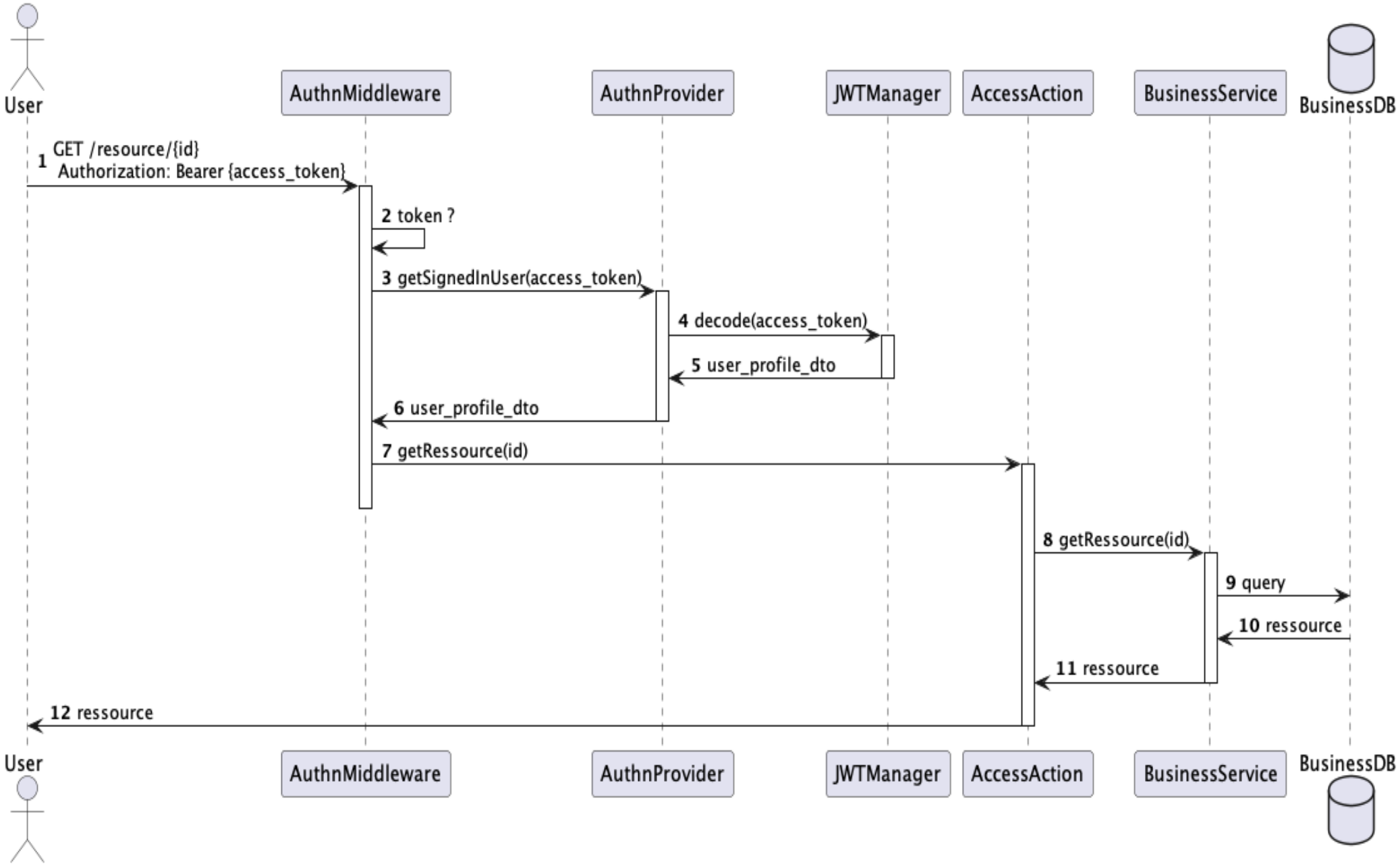
- Un **service métier** chargé de gérer contrôler les données d'authentification :
  - contrôle des credentials (ID + passwd)
  - Enregistrement des nouveaux utilisateurs
- Un manager JWT
  - création, validation, décodage de token JWT
- Un provider d'authentification
  - utilisé dans les middleware/actions de l'application
  - authentification effective
  - utilise le service d'authentification et le manager JWT


api
middleware
AuthnMiddleware
_invoke(Request $request, callable $next): Response
Requête
+ProfileDTO
requête
providers
actions
AuthnProviderInterface
register(CredentialsDTO credentials, int role): ProfileDTO;
signin(CredentialsDTO credentials): AuthDTO;
refresh(Token token): AuthDTO;
getSignedInUser(Token token): ProfileDTO;
DoSomethingAction
RefreshAction
SigninAction
JWTAuthnProvider
JWTManager
createAccessToken(array $payload): string;
createRefreshToken(array $payload): string;
decodeToken(string $token): array;
core
application
ports
AuthnServiceInterface
register(CredentialsDTO credentials, int role) : profileDTO
byCredentials(CredentialsDTO credentials) : profileDTO
usecases
AuthnService
domaine
entites
User
UUID id
string email
string password
int role

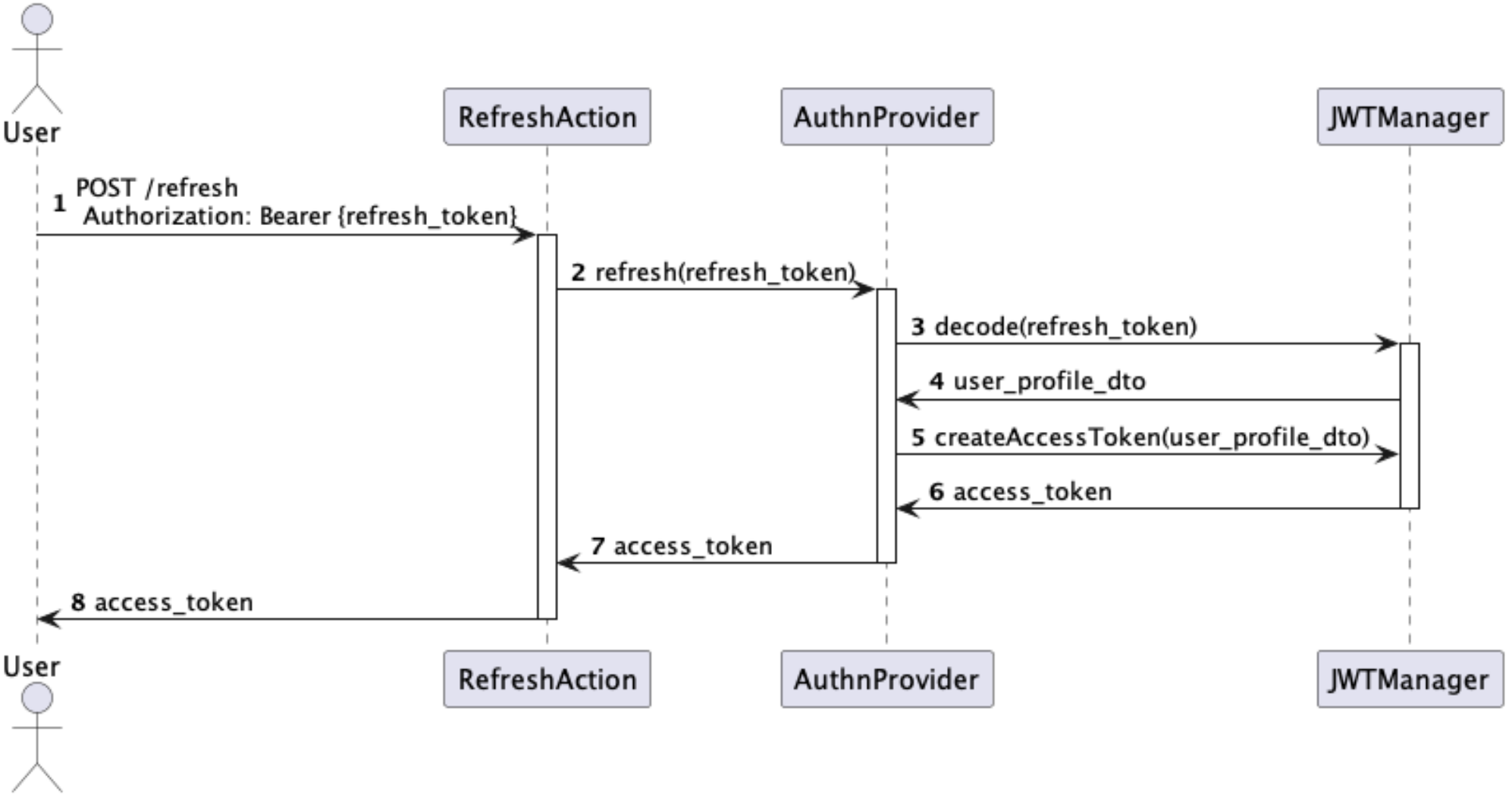
SIGNIN Sequence



ACCESS Sequence

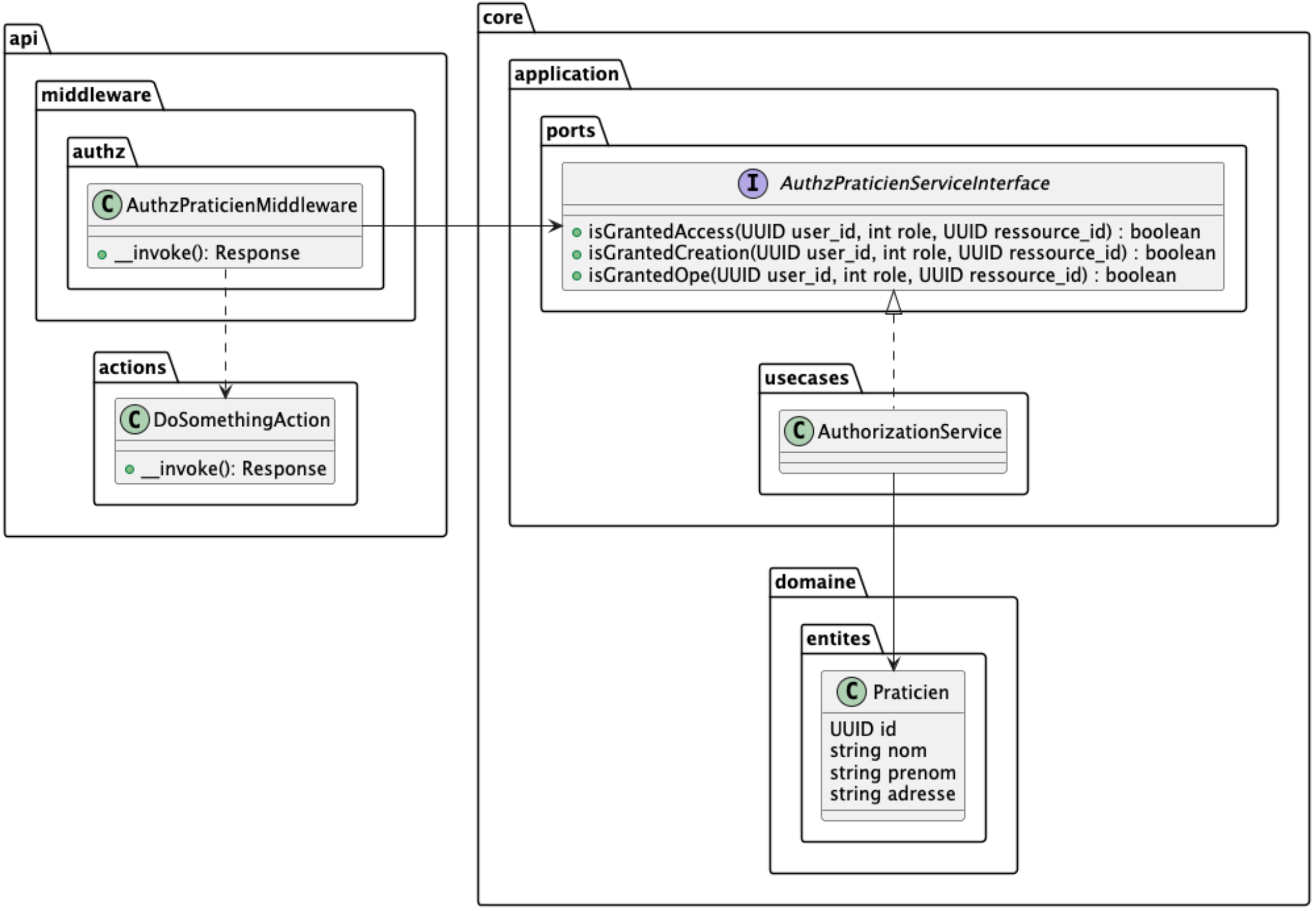


REFRESH Sequence



Autorisation (Authz)

- Autorisation : contrôler que les accès aux ressources sont conformes
- L'autorisation est fournie par des services implantés dans le noyau métier – On peut prévoir un service d'Authz par service métier
- Ce service est chargé de réaliser les politiques d'autorisations :
  - user X rôle X ressource X opération → OK | KO
- Il fournit une interface utilisée dans des middlewares
- Dans cette interface, on peut prévoir 1 méthode par opération

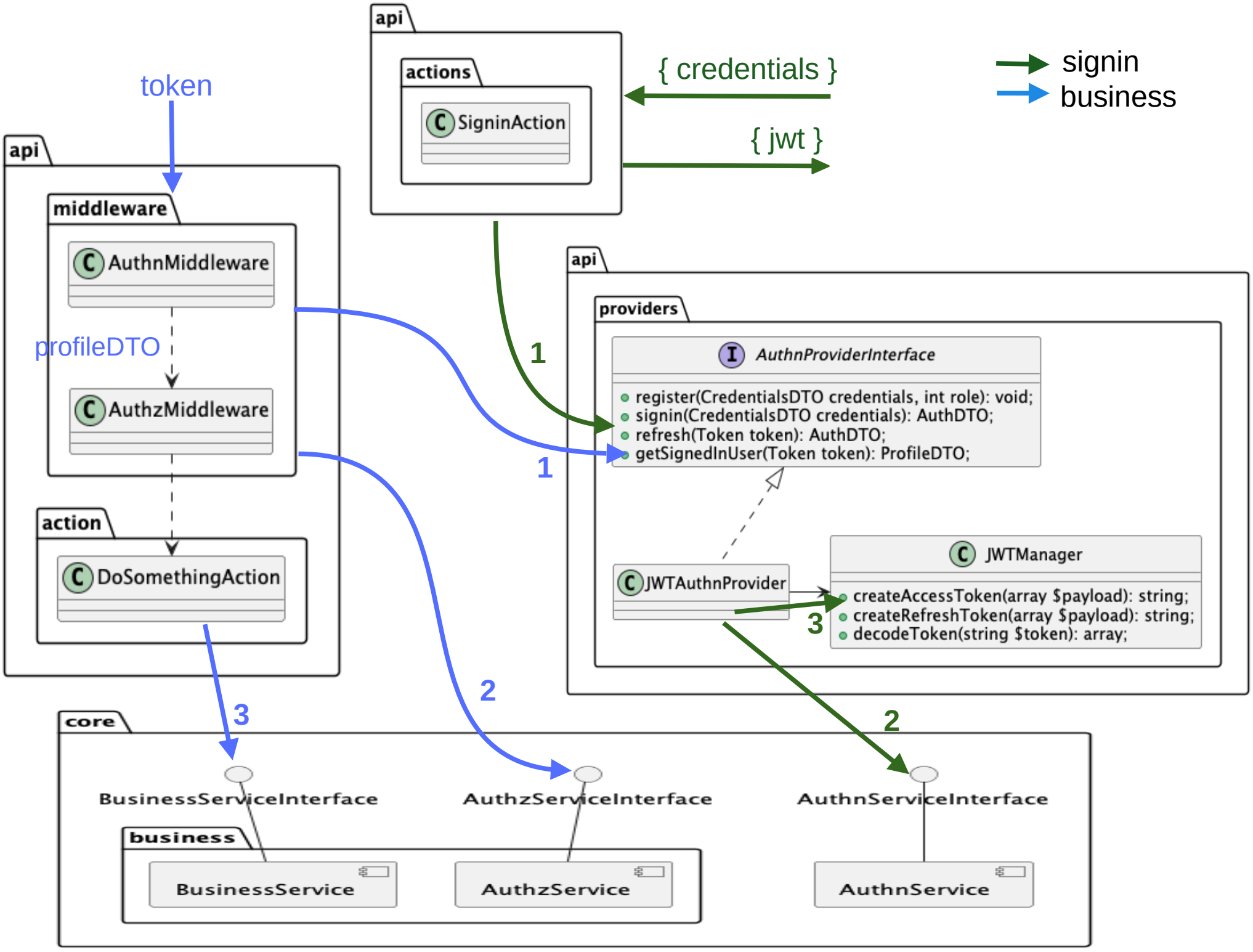

api
middleware
authz
AuthzPraticienMiddleware
__invoke(): Response
actions
DoSomethingAction
__invoke(): Response
core
application
ports
AuthzPraticienServiceInterface
isGrantedAccess(UUID user_id, int role, UUID ressource_id) : boolean
isGrantedCreation(UUID user_id, int role, UUID ressource_id) : boolean
isGrantedOpe(UUID user_id, int role, UUID ressource_id) : boolean
usecases
AuthorizationService
domaine
entites
Praticien
UUID id
string nom
string prenom
string adresse

Les services Authz

- Dédiés à chaque services métiers pour avoir accès à la base de données correspondante
- Doivent contrôler :
  1) Le rôle : le rôle de l'utilisateur authentifié permet-il d'accéder au service ?
  2) La propriété : l'identité et le rôle de l'utilisateur authentifié permettent-ils d'accéder à la ressource désignée ?
  3) La permission : l'utilisateur authentifié a-t-il le droit d'exécuter l'opération sur la ressource désignée ?

Résumé

- Des services métiers
  - authentification : création d'utilisateurs, vérification des credentials
  - autorisation : vérification des règles d'autorisation liées à la politique de droits pour chaque service métier
- Un provider d'authentification
  - fait appel au service métier
  - gère l'authentification à l'aide de token JWT
- Des middleware de contrôle basés sur l'utilisation du provider d'auth et du service d'autorisation
  - Contrôle de l'authentification
  - Contrôle d'authz


api
actions
SigninAction
{ credentials }
{ jwt }
signin
business
token
api
middleware
AuthnMiddleware
profileDTO
AuthzMiddleware
action
DoSomethingAction
1
1
api
providers
AuthnProviderInterface
register(CredentialsDTO credentials, int role): void;
signin(CredentialsDTO credentials): AuthDTO;
refresh(Token token): AuthDTO;
getSignedInUser(Token token): ProfileDTO;
JWTAuthnProvider
JWTManager
createAccessToken(array $payload): string;
createRefreshToken(array $payload): string;
decodeToken(string $token): array;
3
2
3
2
core
BusinessServiceInterface
AuthzServiceInterface
AuthnServiceInterface
business
BusinessService
AuthzService
AuthnService

Authn / Authz Sequence