EVALUATION

TP3

LabREST_03_token

Sécuriser une API REST

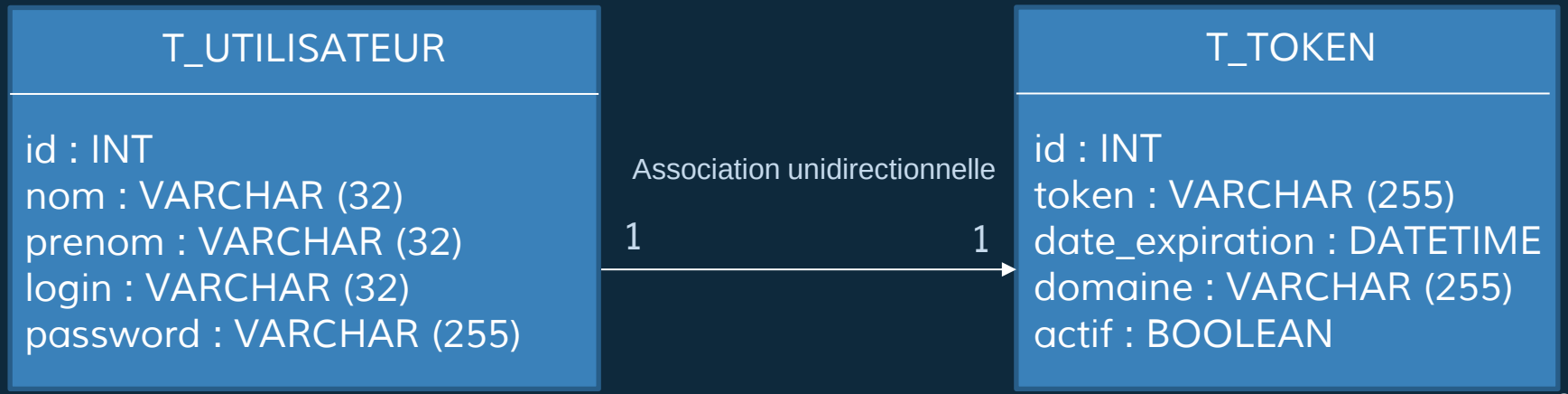
Production

- Compressez votre projet au formation zip (sans le répertoire vendor).
- Déposez le fichier dans Teams.
- Nommage du fichier : Nom_Prenom_apiRestPHP_TP3.zip.

Sécuriser une API REST

Objectif

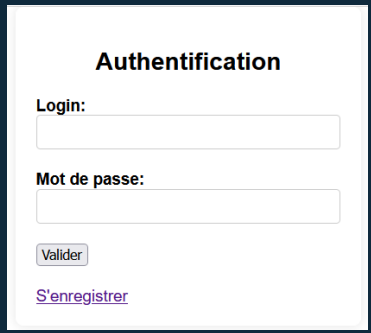
- Mettre en place une authentification basée sur un token pour sécuriser l'accès à l'API Rest du TP1.
- Base de données : création d'une table T_UTILISATEUR et d'une table T_TOKEN



Sécuriser une API REST

- Créer une page d'enregistrement (nom + prenom + login + mot de passe)
  - Appel à une API Rest avec la méthode POST
  - Créer l'utilisateur dans la base de données

- Créer une page d'authentification (login + mot de passe)
  - Appel à une API Rest avec la méthode POST



- Créer les entités suivantes : Utilisateur et Token
  - Créer les attributs en private
  - Créer les méthodes getter et setter

Sécuriser une API REST

- Créer la classe Connexion.php de connexion à la base de données

- Créer la classe dao UtilisateurDao

  Enregistrer un utilisateur dans la base de données : le mot de passe doit être encrypté.

  Rechercher un utilisateur dans la base de données à partir de son login

- Créer la classe dao TokenDao

  Récupérer le token d'un utilisateur

  Sauvegarder le token d'un utilisateur

- Créer la classe Api

  Générer un token

  Vérifier le token lors de l'appel à une API

Sécuriser une API REST

- Créer l'API login

  Appelée lors de l'authentification : méthode POST

  Vérifier si la méthode utilisée est POST

  Vérifier si le login et le mot de passe ont été envoyés

  Vérifier l'authentification : login et mot de passe corrects

  SI authentification OK

    Enregistrer le token dans la base de données

    Retourner le token au client

  SIFIN

Sécuriser une API REST

- Côté client : demande d'authentification (login + mot de passe)

  Méthode POST
- Côté serveur : création du token avec une clé secrète
- Côté serveur : stockage du token dans une base de données
- Coté serveur : envoi le token au client
- Côté client : stockage du token (ex : localStorage)

- Côté client : envoie d'une requête à l'API en incluant le token dans l'en-tête

  Méthodes GET, POST, POST ou DELETE
- Coté serveur : vérification du token avant de traiter la demande

  Si le token est valide, le serveur renvoie les données demandées en format JSON.

  Si le token est invalide ou absent, le serveur retourne un code d'erreur approprié, indiquant que l'accès est refusé.

Sécuriser une API REST

- Pour générer le token (encodage et décodage) vous pouvez :
  - Utiliser la librairie firebase/php-jwt
  - Créer votre propre classe

Sécuriser une API REST

- Utilisation de la librairie firebase/php-jwt pour la génération et la validation du token : encodage et décodage.

  https://github.com/firebase/php-jwt

Installation : fichier composer.json

```
{
   "require": {
      "firebase/php-jwt": "^6.6"
    }
  }
```

```
C:\xampp\htdocs\LabREST_03_Token
> compose update
```

Utilisation

```
<?php
use Firebase\JWT\JWT;
use Firebase\JWT\Key;

$encoded = JWT::encode($payload, $this->secretKey, 'HS256');
$decoded = JWT::decode($jwt, new Key($this->secretKey, 'HS256'));

…
```

EXEMPLE

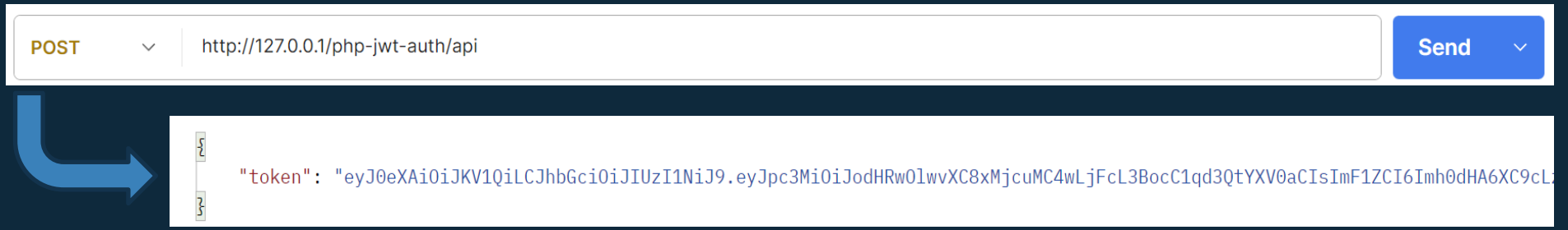
UTILISATION DE LA LIBRAIRIE firebase/php-jwt

php-jwt-auth

Sécuriser une API REST

- Exemple : application php-jwt-auth (à récupérer sur Teams)
- Méthode POST : permet de générer un token (avec Postman)
  - Dans le body de la requête, indiquez le username et la password (format JSON)

  - Exécutez la requête POST

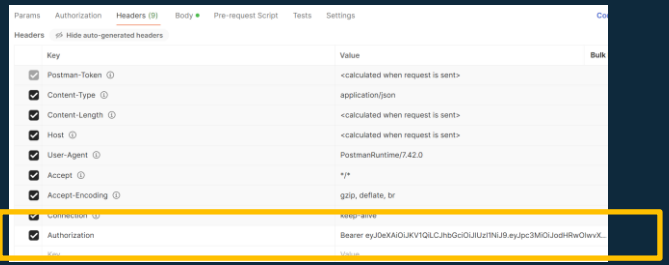


Sécuriser une API REST

- Exemple : application php-jwt-auth
- Méthode GET : permet de tester l'authentification à partir du token
  - Dans la classe api.php, recopiez le token dans la variable $secretKey

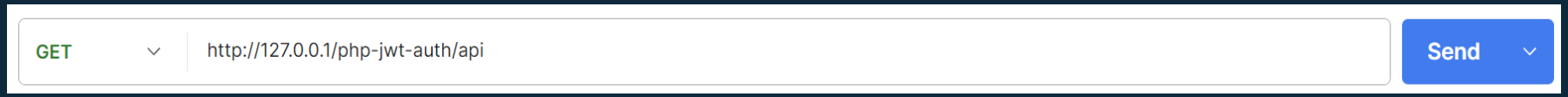
```
class api
{
    // Votre clé secrète
    private $secretKey =
'eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOlwvXC8xMjcuMC4wLjFcL3BocC1qd3QtYXV0aCIsImF1ZCI6Imh0dHA6XC9cLzEyNy4wLjAuMVwvcGhwLWp3dC1hdXRoIiwiaWF0IjoxNzI4NzQ1NDU1LCJuYmYiOjE3Mjg3NDU0NTUsInVzZXJuYW1lIjoidGVzdF91c2VyIn0.0wbsZ-Ava6PYLXmKndwDyYyOv8vqd_1uZH6EpxnQ2Uk';
```

  - Postman : dans le Header de la requête positionnez le token (key Authorization). Forme du token : Bearer valeur du token



Sécuriser une API REST

- Exemple : application php-jwt-auth



- Postman : Exécuter la requête GET avec un token correct

```
{
    "message": "Accès autorisé",
    "user": "test_user"
}
```

- Postman : Exécuter la requête GET avec un token incorrect

```
{
    "message": "Accès NON autorisé",
    "error": "Syntax error, malformed JSON"
}
```

- Postman : Exécuter la requête GET sans Header Authorization

```
{
    "message": "Authorization header non trouvé"
}
```

ENVOI DU TOKEN PAR COOKIE SECURISE

php-jwt-auth

Sécuriser une API REST

Back-end : générer le token avec firebase/php-jwt

```php
<?php
use \Firebase\JWT\JWT;
require 'vendor/autoload.php';
// Clé secrète pour signer le token (utiliser une clé plus sécurisée)
$secretKey = 'cle_secrete';
// Données utilisateur
$payload = [
    'user_id' => 123,
    'email' => 'bob@gmail.com',
    'iat' => time(),
    'exp' => time() + (60 * 60), // Expiration : 1 heure
];
// Générer le token à partir du payload et de la clé secrete
$jwt = JWT::encode($payload, $secretKey, 'HS256');
```

Le token est encrypté en HS256

Sécuriser une API REST

Back-end : configurer et envoyer le cookie sécurisé

```php
<?php
// Options du cookie sécurisé
$options = [
    'expires' => time() + (60 * 60),
    'path' => '/',
    'domain' => 'votre-domaine.com',
    'secure' => true,
    'httponly' => true,
    'samesite' => 'Strict'
];

// Envoi du cookie avec le token
setcookie('auth_token', $jwt, $options);
```

sameSite, httponly et secure permettent de contrôler le comportement des cookies, en définissant quand ces derniers peuvent être envoyés et quand ils ne le doivent pas.

expires : définit la durée de vie du cookie en secondes (ici, 1h).

path : limite le chemin où le cookie est accessible.

domain : spécifie le domaine pour lequel le cookie est valable.

secure: indique que le cookie ne sera envoyé que via HTTPS.

httponly : empêche l'accès du cookie via JavaScript pour éviter les attaques XSS

samesite définit si le cookie est envoyé dans des requêtes cross-site (Strict pour une sécurité maximale).
'Strict' = empêche les envois de cookie cross-site (protection CSRF). Le cookie ne sera envoyé que si la requête provient du même site web. Il n'est pas envoyé lors d'une 1ère visite sur une page du site, mais uniquement lors des actions qui suivent.

Sécuriser une API REST

Back-end : lire et valider le token dans le cookie

```php
<?php
// Vérification si le cookie nommé auth_token existe
if (isset($_COOKIE['auth_token'])) {
    $jwt = $_COOKIE['auth_token'];
    try {
        // Décodage et validation du token
        $decoded = JWT::decode($jwt, $secretKey, ['HS256']);

        // Accès aux données de l'utilisateur (payload du token)
        $userId = $decoded->user_id;
        $email = $decoded->email;
    } catch (Exception $ex) {
        // Le token n'est pas valide ou a expiré
        echo "Token invalide : " . $ex->getMessage();
    }
} else {
    echo "Token d'authentification manquant.";
}
```

JWT::decode lève différentes exceptions :

- InvalidArgumentException : Provided JWT was empty
- UnexpectedValueException : Provided JWT was invalid
- SignatureInvalidException : Provided JWT was invalid because the signature verification failed
- BeforeValidException : Provided JWT is trying to be used before it's eligible as defined by 'nbf'
- BeforeValidException : Provided JWT is trying to be used before it's been created as defined by 'iat'
- ExpiredException : Provided JWT has since expired, as defined by the 'exp' claim

Sécuriser une API REST

Si le token est stocké dans un cookie HttpOnly, il n’est pas accessible directement depuis JavaScript.

Le cookie est automatiquement envoyé par le navigateur dans chaque requête HTTP (si les options SameSite et Secure le permettent).

Front-end : envoyer le token dans un cookie

```
fetch('https://localhost/endpoint', {
    method: 'GET',
    credentials: 'include’
})
.then(response => response.json())
.then(data => console.log(data))
.catch(error => console.error('Erreur:', error));
```

credentials: 'include’ : inclut le cookie avec la requête