

Projet Neolia-devcyber

Loïc Gormand
Dorine Berton
Boris Trouche
Thomas Turti
Sabine Provo

14 juin 2024

PLAN DE LA PRÉSENTATION

- Contexte du projet
- Mise en place de l'environnement de travail
- Organisation de l'équipe
- Mission 1 : UI/UX et sécurité
- Mission 2 : RGPD
- Mission 4 : Sanitization
- Mission 5: Injection
- Mission 6 : Encryptage
- Mission 7 : Authentification
- Mission 8 : Pentest

CONTEXTE DU PROJET

Neolia, TPE de 5 personnes, développe des applications pour ses différents clients en France.

Elle nous a confié un certain nombre de missions après avoir perçu que les développements réalisés n'étaient pas forcément très sécurisés.

MISE EN PLACE DE L'ENVIRONNEMENT DE TRAVAIL

outils / autres :

- IDE VS Code, gestion du code source sur Github
- création d'un repository commun et attribution des droits d'accès et de modification
- git flow : 1 branche main, 1 branche dev, 1 branche par feature

ORGANISATION DE L'ÉQUIPE

Gestion de projet agile

- scrum
- sprint 1 semaine
- planification du sprint
- daily meeting
- collaboration
- communication

Missions

Loïc Gormand : 8
Dorine Berton : 4-5
Boris Trouche : 1-2++
Thomas Turti : 7
Sabine Provo : 6

Mission 1 - UI/UX et sécurité (3 étapes)

Boris Trouche

Avant toute chose, l'API ReactiveX (version JS) a été utilisée largement dans ce projet. Créée par Microsoft, elle est utilisée par plus de 11.5M d'utilisateurs (dont Netflix, GitHub, SoundCloud, airbnb), elle compte 541 contributeurs, elle est cross platform, elle s'utilise aussi bien pour le Front que pour le Back, dans différents langages de programmation. Elle permet de programmer de manière asynchrone. Elle utilise des observables, des observateurs, des opérateurs...

1. Affichage des erreurs sur l'interface en cours de saisie :

Dans `login.js` ligne 57:

avec la méthode `subscribe`, nous pouvons passer une fonction de gestion d'erreur qui sera appelée à chaque fois qu'une erreur est émise par l'observable. Et une fonction `"complete"` quand l'observable a fini d'émettre des objets.

la fonction `take` retourne un observable qui renvoie 1 seule valeur dans ce cas précis : `take(1)`

opérateur `take`

observable `value`

nous devons donc mettre en lien toutes les fonctions d'observation contenues les divers js :

- dans `handler.js` nous avons tous les messages d'erreur et les fonctions d'afficher / cacher les erreurs
- dans `input-handler.js` nous avons tous les événements relatifs à l'appui des touches

dans `handler.js` nous rajoutons

```
import { HtmlCompose } from './../html-composer/html-compose' // rajout manquant
```

dans `html-component.js` nous corrigeons

lig2 : `# parent! // typo 1 identifiée, enlever le !`

et nous rajoutons les fonctions `get` pour le `componentType` et le `content`

Ok ça marche ! Les erreurs de saisie sont bien affichées, malgré que ça n'affiche pas du premier coup.

Dev Cyber - Signin

Login

```
""""</////<script>alert("rer")</!</pre>
```

Password

Ce champ ne peut pas être vide

SIGNIN

2. Montrer les erreurs en provenance du backend après avoir validé le formulaire, si erreur il y a bien entendu :

Dans le backend, quand un utilisateur n'est pas trouvé (mauvais login ou mdp) nous recevons ce message :

`UserRepository.php`, lig64 :

```
throw new NotFoundException('No user were found with this credentials, using password');
```

qui est passé par `SignInService.php`, lig72 :



Nous voulons récupérer ce message et l'afficher. Etant donné que c'est seulement une exception, nous allons traiter le statut 404 dans `http-client.js` en rajoutant ces lignes :

```
81 .then((response) => {  
82     if (response.ok) {  
83         return response.json()  
84     }  
85     else if(response.status === 404) {  
86         alert('User not found !')  
87     }  
88     throw new Error('Something went wrong')  
89 })
```

3. Sanitisation en front :

login-form.js :

lig22 + 23, nous voyons que l'interface utilisateur pour s'identifier sur le site est enrichie (composée) de 2 éléments (contrôleurs) :

- username
- userpassword

```
#setFields() {  
  this  
    .addControl(new Control('username', '', [Validators.required]))  
    .addControl(new Control('userpassword', '', [Validators.required]))  
  
  // Place form handler  
  Handler.formHandler(this)  
}
```

Les 2 font appel à un validateur de saisie qui se trouve dans **validators.js** : fonction **Validators.required**

Celle-ci ne fait que vérifier si les saisies utilisateurs sont null ou pas et fait **trim** au début et à la fin de la saisie.

Nous rajoutons donc une sanitisation juste avant le renvoi du trim (lig20 + 21) :

Ajout d'une étape de sanitisation dans **validators.js**

(fonction **replace** avec une regex) juste après que

le client ait cliqué sur Sign In :

→ Cross Scripting et injections rendus inopérants.

```
export function requiredValidator(control) {  
  if (control.value === undefined) {  
    return {required: true}  
  }  
  
  control.value = control.value.replace(/^[a-z0-9-áéíóúñ \.,_]/gim,""); // Sanitization  
  return (control.value.trim().length === 0) ? {required: true} : null  
}
```

Dev Cyber - Signin

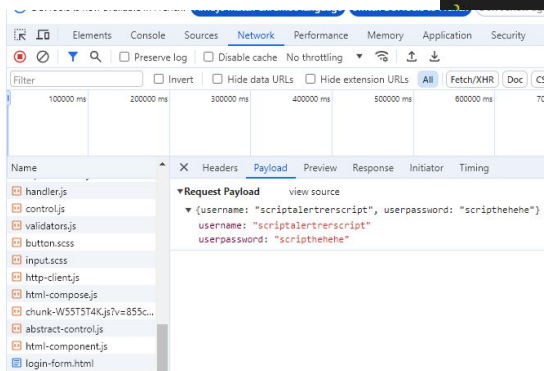
Login

=====

Password

SIGNIN

Version : 1.0.0



Mission 2 - Bandeau RGPD

Nous allons installer un module de gestion de cookies RGPD : tarteaucitron. Il est opensource, personnalisable, il existe en version gratuite et est recommandé par la CNIL car il respecte toutes les exigences RGPD.

Nous nous déplaçons dans www comme d'habitude puis nous entrons la commande :
`npm i tarteaucitronjs`

ensuite dans index.html section header nous rajoutons :

```
<script src="/tarteaucitron/tarteaucitron.js"></script>
```

et nous configurons le module avec un autre script qui contient tous les paramètres. Nous choisissons de le mettre en plein milieu. `var tarteaucitronForceLanguage = 'fr';`

puis nous rajoutons au moins 1 service juste avant le `<body>` pour activer le bandeau tarteaucitron (ici c'est google analytics) :

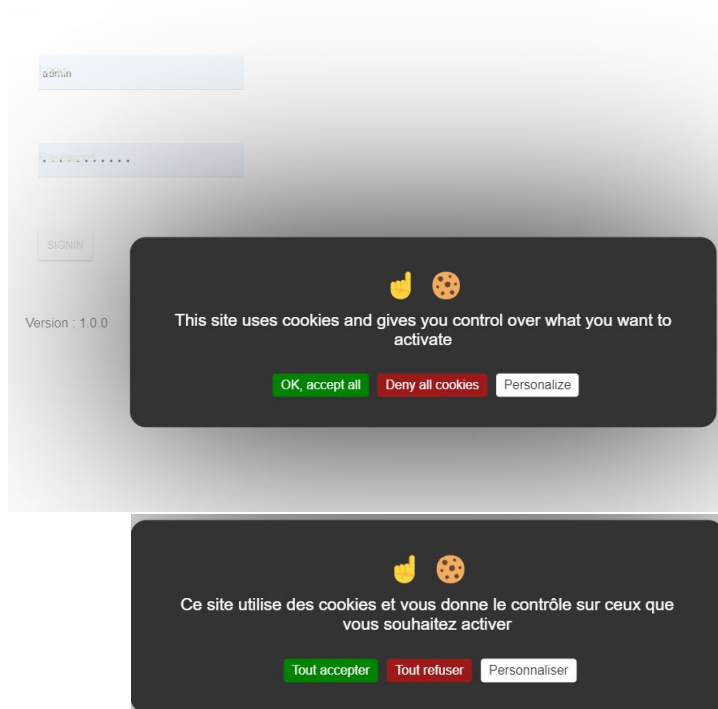
```
<script type="text/javascript">
tarteaucitron.user.gajsUa = 'UA-XXXXXXX-1';
tarteaucitron.user.gajsMore = function () { /* add here your optionnal _ga.push() */ };
(tarteaucitron.job = tarteaucitron.job || []).push('gajs');
</script>
```

et voilà le bandeau s'affiche en plein milieu.

pour le réafficher il faut delete le cookie depuis l'inspecteur et refresh.

Boris Trouche

Dev Cyber - Signin



Mission 4: sanitization

Dorine Berton

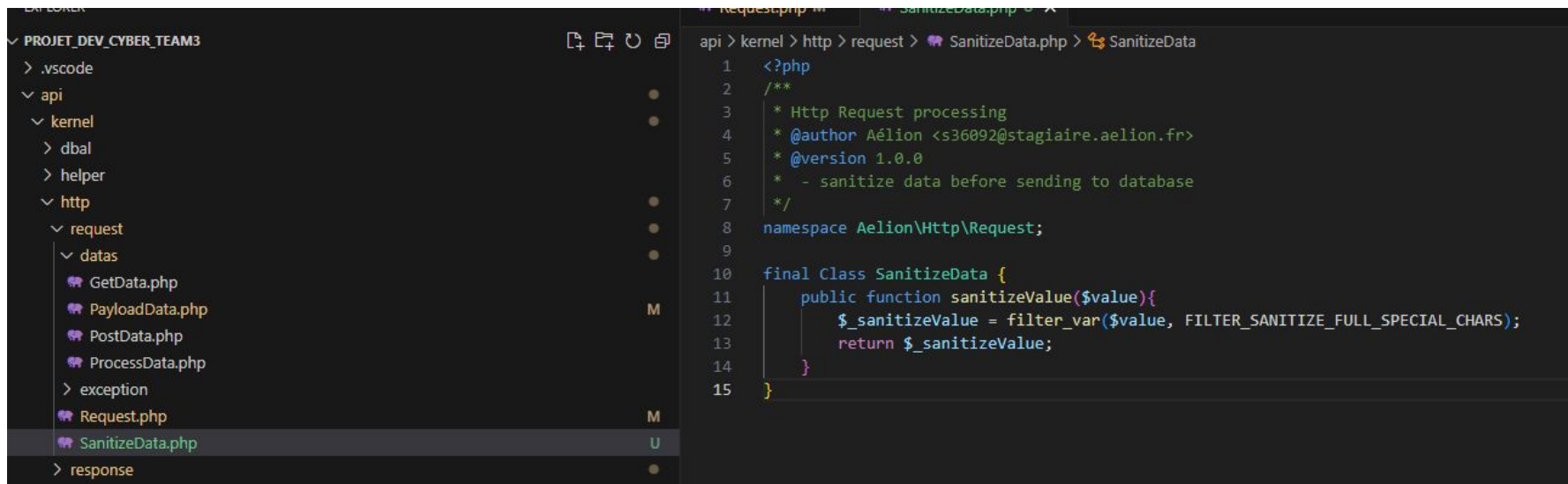
“sanitizer” les données issues des requêtes en modifiant le comportement des classes du package Aelion\Http\Request\Datas

1. Création d'une classe SanitizeData.php qui s'occupe de la sanitisation d'une donnée.

la fonction `filter_var()` permet de filtrer une variable à l'aide d'un filtre spécifique;

`FILTER_SANITIZE_FULL_SPECIAL_CHARS` sert à nettoyer les données, encode `<`, `>`, `"`, `'`, et `&` en entités HTML (`<`, `>`, `"`, `'`, `&`).

sanitization



The image shows a code editor interface. On the left is a file explorer for a project named 'PROJET_DEV_CYBER_TEAM3'. The directory structure is as follows:

- PROJET_DEV_CYBER_TEAM3
 - .vscode
 - api
 - kernel
 - dbal
 - helper
 - http
 - request
 - datas
 - GetData.php
 - PayloadData.php
 - PostData.php
 - ProcessData.php
 - exception
 - Request.php
 - SanitizeData.php
 - response

On the right, the code editor displays the contents of 'SanitizeData.php'.

```
1 <?php
2 /**
3  * Http Request processing
4  * @author Aélion <s36092@stagiaire.aelion.fr>
5  * @version 1.0.0
6  * - sanitize data before sending to database
7  */
8 namespace Aelion\Http\Request;
9
10 final Class SanitizeData {
11     public function sanitizeValue($value){
12         $_sanitizeValue = filter_var($value, FILTER_SANITIZE_FULL_SPECIAL_CHARS);
13         return $_sanitizeValue;
14     }
15 }
```

sanitization

2. Implémentation de la classe créée dans le fichier Request.php.

-> initialisation / création d'une instance de la classe SanitizeData
-> appel de la méthode sanitizeValue

```
private function sanitizeValues($value): string {  
    $sanitizeData = new SanitizeData();  
    return $sanitizeData->sanitizeValue($value);  
}
```

-> filtrage des données avant utilisation

```
public function set(string $key, string $value): void {  
    $valueSanitized = $this->sanitizeValues($value);  
    echo($valueSanitized);  
    $this->datas[$key] = $valueSanitized;  
}
```

sanitization

Dev Cyber - Signin

Login

ad<test>min

Password

.....

DevTools is now available in French! [Always match Chrome's language](#)

Elements Console Sources **Network** Performance

☐ Preserve log ☐ Disable cache No throttling

Filter ☐ Invert ☐ Hide data URLs ☐ Hide e

50000 ms 100000 ms 150000 ms 200000 ms

Name	X	Headers	Payload	Preview	Response
login-form.html	1		ad <test >minfkdsljms 		
signin	2		Warning		
signin	-		: Cannot modify header information		
signin	-		on line 16		

Mission 5: Injection

Dorine Berton

1. refactorer le repository afin d'éviter au maximum les attaques par injection
2. expliquer la stratégie choisie, tout en conservant le mode de fonctionnement du "framework"

Injection

Mesure	Pour	Contre
Requêtes préparées	impossible de modifier l'objet passé en paramètre	Plus lente / coûteuse
Validation des données utilisateurs	seules des valeurs attendues et sûres sont acceptées	une validation incorrecte ou incomplète peut laisser des failles exploitables
Procédures stockées	alternative à l'utilisation d'instructions préparées	peut être la cible des injections SQL, si elle utilise des entrées non filtrées
Echappement de caractères	Plus rapide / moins coûteuse	Moins protectrice

Injection

Choix de l'implémentation => requête préparée

La tâche a été réalisée selon 2 étapes :

1. Classe UserRepository.php directement modifiée

-> création de marqueurs nommés

```
$sqlQuery = "SELECT
    u.id userid, u.login login, u.password password,
    a.id accountid, a.lastname lastname, a.firstname
FROM
    user u
JOIN user_has_role uhr ON u.id = uhr.user_id
JOIN role r ON uhr.role_id = r.id
JOIN account a ON u.id = a.user_id
WHERE login = :username AND password = :pass;";
```

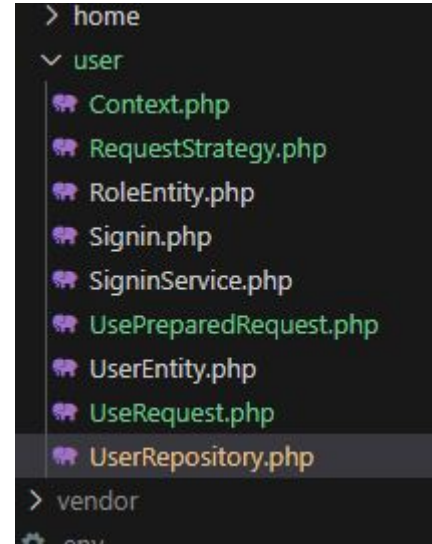
-> préparation de requête / ajout du tableau de marqueurs / exécution de la requête

Injection

2. Utilisation du design pattern Strategy pour laisser le choix à l'équipe d'utiliser une requête préparée ou un échappement de caractères

-> Création d'une interface RequestStrategy.php; des classes UseRequest.php et UsePreparedRequest.php qui implémentent l'interface

-> création de la classe Context.php qui permettra de switcher entre les différentes stratégies.



Injection

```
<?php
namespace Api\User;
interface RequestStrategy {
    public function applyStrategy(\PDO $conn, string $query, array $params = []);
}
```

```
<?php
namespace Api\User;
use Api\User\RequestStrategy;
class UseRequest implements RequestStrategy {
    function applyStrategy($conn, $query, $params = [])
    {
        foreach($params as $key => $value){
            $query = str_replace($key, $conn->quote($value), $query);
        }
        $pdoStatement = $conn -> query($query);
        return $pdoStatement;
    }
}
```

```
<?php
namespace Api\User;
use Api\User\RequestStrategy;
class UsePreparedRequest implements RequestStrategy {
    function applyStrategy(\PDO $conn, $query, $params = [])
    {
        $pdoStatement = $conn->prepare($query);
        if ($pdoStatement === false) {
            echo $this->$conn->errorCode().': '.$this->$conn->errorInfo();
        }
        foreach($params as $key => $value){
            $pdoStatement->bindValue($key, $value, \PDO::PARAM_STR);
        }

        $pdoStatement->execute();
        return $pdoStatement;
    }
}
```

Injection

-> Le Contexte

```
<?php
namespace Api\User;
class Context {
    private $strategy;
    public function __construct(RequestStrategy $requestStrategy) {
        $this->strategy = $requestStrategy;
    }
    public function setStrategy($requestStrategy){
        $this->strategy = $requestStrategy;
    }
    public function useStrategy(\PDO $conn, $query, $params = []){
        return $this->strategy->applyStrategy($conn, $query, $params);
    }
}
```

Injection

-> initialisation du contexte et choix de la stratégie dans la classe UseRepository.php

```
public function useContext($conn, $query, $params = [], $usePrepared = true){  
    $context = new Context(new UsePreparedRequest());  
    if (!$usePrepared) {  
        $context->setStrategy(new UseRequest());  
    }  
    return $context->useStrategy($conn, $query, $params);  
}
```

puis utilisation en fonction du besoin...

```
$pdoStatement = $this->useContext($this->dbInstance, $sqlQuery, $params, true);  
# $pdoStatement = $this->useContext($this->dbInstance, $sqlQuery, $params, false);
```

Mission 6 - Encryptage

Sabine Provo

Les mots de passe des utilisateurs sont stockés en clair dans la base de données. Vous allez devoir modifier la stack backend de manière à :

- stocker à l'aide d'un algorithme de cryptage le mot de passe dans la base de données,
- modifier en conséquence la stratégie d'identification de l'utilisateur.

Réflexion avant modification du code :

- Choisir le hachage pour la meilleure sécurité : MD5, SHA1 ou SHA256, password_hash
- A quel moment se fait le hachage du mot de passe et où modifier le code:

Le hachage du mot de passe dans la base de données se fait au moment de la **création ou modification** du mot de passe.

Ici il n'y a pas de formulaire pour faire cela.

Sinon c'est dans le fichier '**userEntity.php**' que la fonction est modifiée

On utiliserait la fonction "password_hash"

```
public function setPassword(string $password): void {  
    $this->password = password_hash($password, PASSWORD_BCRYPT);  
}
```

Modification manuelle des mots de passe dans la base de données pour comparaison lors de la connexion

Connexion à la base dans docker :

PS C:\Users\Aelion\Documents\neolia-devcyber\docker> docker exec -it poe_mariadb mariadb -u root -p
Enter password:

```
MariaDB [(none)]> show databases;
+-----+
| Database |
+-----+
| dev_cyber_repository |
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
```

requête SQL :

update user set password = "\$2y\$10\$Nq660PRcd44nhY/slYUmI.Nbh6DiJNyNv5yzMQ/opo0V1HV1mVLHS"
where id = 2;

Utilisation de **Bcrypt online**
pour le hachage de ces
mots de pas

```
MariaDB [dev_cyber_repository]> select * from user;
+----+-----+-----+
| id | login | password |
+----+-----+-----+
| 1 | admin | $2y$10$si0F1L.cEfswbjPx3DSzW.2irx0QToZXRrn2Uua9MsZ2CFudic4.G |
| 2 | a.dupont | $2y$10$Nq660PRcd44nhY/slYUmI.Nbh6DiJNyNv5yzMQ/opo0V1HV1mVLHS |
+----+-----+-----+
2 rows in set (0.001 sec)
```

Site Bcrypt online : <https://bcrypt.online/>

Bcrypt Hash Generator

Plain Text Input

super_admin

Cost Factor

10



[How to Choose the Right Cost Factor for Bcrypt »](#)

Output

COPY

\$2y\$10\$Qixly0tYLCH8ChcgATtIDu6k2ug1HjtBPIWW6F8fBjDo3Aecm6ila

GENERATE HASH

RESET FORM

Utilisation de crypt et de password_verify

crypt() renverra une chaîne hachée en utilisant l'algorithme standard basé sur Unix DES ou des algorithmes alternatifs.

Un grain de sel aléatoire est ajouté :

exemple : `$2y$10$0IexiTnFtZ6BeARkfVvide``mnnAuPydkcH2tEpVbnE60o97K3UZn26`

Vous pouvez distinguer trois champs, délimité par des "\$" :

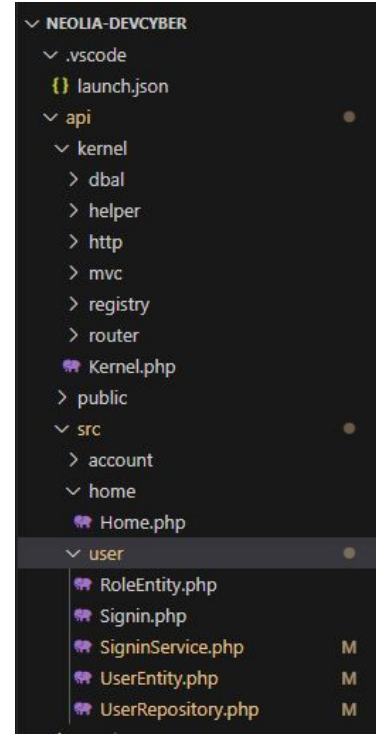
- 2y identifie la version de l'algorithme bcrypt qui a été utilisé.
- 10 est le facteur cout utilisé (par défaut en PHP).
- 22 premiers caractères décode une valeur sur 16 bits pour le grain de sel
- Les caractères restants sont le mot de passe encodé

password_verify() est compatible avec crypt() .

Par conséquent, les hachages de mots de passe créés par crypt() peuvent être utilisés avec password_verify() .

Pour pouvoir effectuer cette mission j'ai du modifier 2 fichiers situés dans api / src / user :

- SigninService.php
- User_repository.php



Modification du fichier SigninService.php

```
src > user > SigninService.php

    }

    public function signin(): Response {
        try {
            $username = $this->request->get('username');
            $userpassword = $this->request->get('userpassword');

            $userEntity = $this->repository->findByLogin($username);

            if ($userEntity && password_verify($userpassword, $userEntity->getPassword())) {
                $roles = [];
                foreach ($userEntity->getRoles() as $role) {
                    $userRole = [
                        'id' => $role->getId(),
                        'role' => $role->getRole()
                    ];
                }
            }
        } catch (Exception $e) {
            // ...
        }
    }
}
```

Vérification du Mot de Passe : Utilisation de `password_verify` pour vérifier et comparer le mot de passe en texte clair fourni par l'utilisateur avec le mot de passe haché stocké dans la base de données.

Gestion des Erreurs : Gestion des exceptions pour fournir des messages d'erreur appropriés.

Envoi du Payload : Le mot de passe n'est pas inclus dans le payload de la réponse, conformément aux bonnes pratiques de sécurité.

Suppression des hachages redondants : Le mot de passe doit être vérifié, mais il ne doit pas être renvoyé dans la réponse, et surtout pas haché à nouveau.

Modification du fichier UserRepository.php

```
public function findByLogin(string $username): ?UserEntity {  
    $sqlQuery = "SELECT  
        u.id userid, u.login login, u.password password, r.id roleid, r.role role,  
        a.id accountid, a.lastname lastname, a.firstname firstname, a.gender gender  
    FROM  
        user u  
    JOIN user_has_role uhr ON u.id = uhr.user_id  
    JOIN role r ON uhr.role_id = r.id  
    JOIN account a ON u.id = a.user_id  
    WHERE u.login = :username";
```

Modification de la méthode **findByLoginAndPassword** par **findByLogin**.

Test du côté back - le mot de passe ne s'affiche pas pour éviter d'être récupéré

The screenshot displays a REST client interface with the following components:

- Left Sidebar:** Shows a project named "projet" containing a request named "GET request_1".
- Top Bar:** Includes tabs for "Overview", "Getting sta", "GET Untitled Re", "projet", "POST request.", and "GET http://loca". It also shows "No environment" and a "Save" button.
- Request Configuration:**
 - Method:** POST
 - URL:** http://localhost:8003/signin
 - Body Type:** raw (selected), with options for none, form-data, x-www-form-urlencoded, binary, GraphQL, and JSON.
- Request Body (Raw):**

```
1 {
2   "username": "a.dupont",
3   "userpassword": "DuP8nT!"
4 }
```
- Response Section:**
 - Body:** Pretty (selected), with options for Raw, Preview, Visualize, and JSON.
 - Status:** 200 OK, 776 ms, 427 B
 - Response Body (Pretty):**

```
1 {
2   "id": 2,
3   "login": "a.dupont",
4   "account": {
5     "id": 2,
6     "lastname": "Dupont",
7     "firstname": "Antoine",
8     "gender": 1
9   },
10  "roles": [
11    {
12      "id": 2,
13      "role": "Utilisateur"
14    }
15  ]
16 }
```

Solution supplémentaire non aboutie

Création d'un endpoint, sans créer le formulaire en frontend
mais en utilisant Postman pour gérer la création de l'utilisateur

1. **Ajouter un nouveau endpoint dans le router.**
2. **Créer une nouvelle méthode dans le `UserRepository` pour hacher et stocker un mot de passe.**
3. **Créer un contrôleur pour gérer ce nouveau endpoint.**

Génération de messages d'erreurs dans de nombreux fichiers et non résolues

Mission 7 - Authentication

Thomas Turti

Le but de cette mission est de réaliser un POC pour l'ajout d'une authentification forte pour la navigation dans le site

La première étape est de choisir la méthode appropriée.

Ici, nous avons choisi l'authentification par JSON Web Token

Pourquoi choisir JSON Web Token

Simplicité et Scalabilité :

- **Stateless** : Étant auto-contenu, JWT ne nécessite pas de stockage côté serveur, simplifiant ainsi l'architecture et facilitant la scalabilité horizontale.
- **Facilité d'Intégration** : Simple à intégrer dans des applications modernes, en particulier les API RESTful et les architectures de microservices.

Performance :

- **Compact et Rapide** : Les tokens JWT sont plus compacts que les assertions SAML (Security Assertion Markup Language) et peuvent être transmis et vérifiés plus rapidement, améliorant ainsi les performances réseau et de traitement.

Interopérabilité et Adoption :

- **Standard Ouvert** : Étant un standard ouvert, JWT est largement supporté par de nombreux langages et frameworks, ce qui facilite l'intégration dans divers environnements technologiques

Objectifs

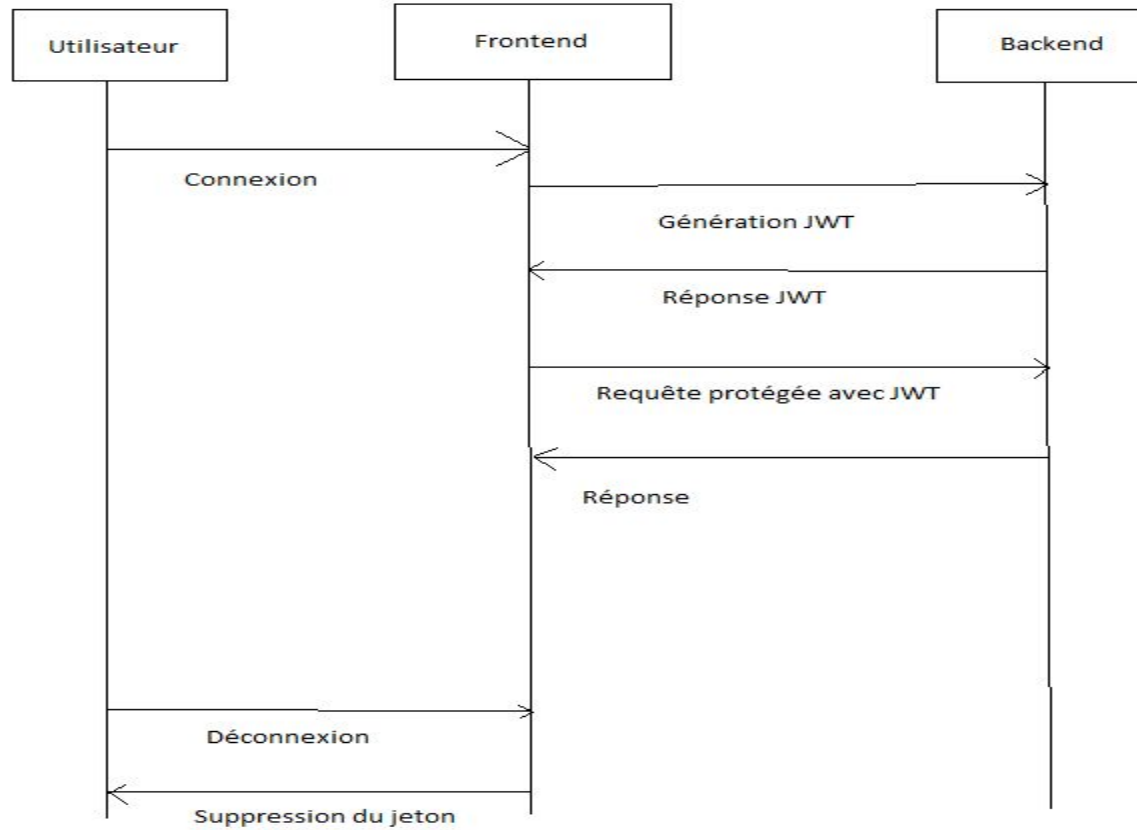
Démontrer la faisabilité de l'authentification par jeton (JWT) :

- Mettre en place un système d'authentification sécurisé utilisant des jetons.

Assurer la sécurité des données :

- Protéger les routes sensibles par des vérifications de jetons.

Workflow



Développement

Installation des dépendances

`composer require firebase/php-jwt`

On ajoute la variable d'environnement `JWT_KEY` au fichier `.env` du dossier `api`. Cette clé est nécessaire à l'encodage du jeton.

On l'ajoute dans le `.env` afin qu'elle ne soit pas accessible en clair

```
1 DSN="driver:dbname=<your_db_name>;host=<host_name_or_ip>;port=<server_port>"
2 DB_USER=<your_db_user>
3 DB_PASSWORD=<your_db_user_password>
4 JWT_KEY='BR21HA4xeejN0u46k2Nk'|
```

Backend

SignInService.php :

On inclut la bibliothèque JWT

```
// Autres inclusions et logique de l'application
use Firebase\JWT\JWT;
You, il y a 1 heure | 2 authors (DaCoDeMaNiaK and one other)
class SignInService implements Registrable {
```

On ajoute la clé de jeton comme attribut de l'objet SigninService

```
class SigninService implements Registrable {  
  
    private $repository = null;  
    private Request $request;  
    private $secretKey;  
  
    private function __construct(Request $request) {  
        $this->request = $request;  
        $this->repository = new UserRepository();  
        $this->secretKey = $_ENV['JWT_KEY'];  
    }  
  
    /**  
        DaCoDeMaNiaK, il y a 5 jours • Database Abstraction Layer
```

On crée une méthode de génération du jeton

```
/**
 * Generate JSON Web Token
 */
protected function generateJWT(array $payload, string $secretKey): string{
    return JWT::encode($payload, $secretKey, 'HS256');
}
```

Dans l'array \$payload, on doit rajouter les valeurs suivantes :

- 'iss' = Url de l'émetteur du jeton
- 'aud' = Url du receveur du jeton
- 'iat' = Heure d'émission du jeton
- 'nbf' = Heure de validation du jeton

```
public function signin(): Response {
    try {
        $userEntity = $this->repository->findByLoginAndPassword($this->request->get('username'), $this->request->get('userpassword'));
        $roles = [];
        foreach ($userEntity->getRoles() as $role) {
            $userRole = [
                'id' => $role->getId(),
                'role' => $role->getRole()
            ];
            array_push($roles, $userRole);
        }

        $payload = [
            'iss' => "http://localhost:8003/signin", // Emetteur du jeton
            'aud' => "http://localhost:8003/signin", // Receveur
            'iat' => time(), // Heure émission du jeton
            'nbf' => time(), // Heure validation
            'data' => [
                'id' => $userEntity->getId(),
                'login' => $userEntity->getLogin(),
                'password' => $userEntity->getPassword(),
                'account' => [
                    'id' => $userEntity->getAccount()->getId(),
                    'lastname' => $userEntity->getAccount()->getLastName(),
                    'firstname' => $userEntity->getAccount()->getFirstname(),
                    'gender' => $userEntity->getAccount()->getGender()
                ],
                // You, il y a 2 heures • AbstractResponse: added abstract method generat...
                'roles' => $roles,
            ],
        ];
    }
}
```


On génère et on ajoute le jeton au payload

```
        'firstname' => $userEntity->getAccount()->getFirstname(),  
        'gender' => $userEntity->getAccount()->getGender()  
    ],  
    'roles' => $roles,  
];  
  
// Génération du JWT  
$payload['data']['token'] = $this->generateJWT($payload['data'], $this->secretKey);  
$response = new JsonResponse();  
$response->setPayload($payload);
```

Réponse du backend après authentification

[illegible]

Token Middleware

Créer une classe Middleware contenant la vérification et le décodage du token.

Ajouter cette classe dans le router. Chaque route ayant besoin d'une vérification passera par cette classe et la vérification se fera automatiquement

```

<?php

namespace Api\Middleware;

use Firebase\JWT\JWT;
use Firebase\JWT\Key;
use Aelion\Http\Response\JsonResponse;
use Aelion\Http\Response\HttpResponseStatus;
...
class Middleware {
    private $secretKey;

    public function __construct() {
        $this->secretKey = $_ENV['JWT_KEY'];
    }

    public function handle($request, $next) {
        $authHeader = $request->getHeader('Authorization');

        if (!$authHeader)
            return $this->unauthorizedResponse();

        $token = str_replace('Bearer ', '', $authHeader);

        try {
            $decoded = JWT::decode($token, new Key($this->secretKey, 'HS256'));
            $request->user = $decoded;
            return $next($request);
        } catch (\Exception $e) {
            return $this->unauthorizedResponse();
        }
    }

    private function unauthorizedResponse() {
        $response = new JsonResponse();
        $response->setStatus(HttpResponseStatus::Unauthorized);
        $response->setPayload(['message' => 'Unauthorized']);
        return $response;
    }
}

```

Frontend

Login-service.js

On crée la méthode storeToken qui enregistre le jeton dans le stockage local

```
import { of } from 'rxjs'
import { HttpClient } from '../core/http-client/http-client'
You, il y a 1 heure | 3 authors (DaCoDeMaNiaK and others)
export class LoginService {
  #protocol = 'http'
  #apiHost = 'localhost'
  #apiPort = 8003
  #apiEndpoint = 'signin'

  #uri = ''

  #httpClient = null

  constructor() {
    this.#httpClient = new HttpClient()
    this.#uri = `${this.#protocol}://${this.#apiHost}:${this.#apiPort}/`
  }

  signin(value) {
    const uri = `${this.#uri}${this.#apiEndpoint}`
    return this.#httpClient.post(
      uri,
      value
    )
  }

  storeToken(token) {
    localStorage.setItem('jwt', token)
  }
}
```

DaCoDeMaNiaK, il y a 6 jours • Login processing stack ...

login.js

On récupère et stocke le jeton dans le stockage local si il existe

```
send() {  
  const value = this.#form.value  
  //console.log(JSON.stringify(value))  
  this.#service.signin(value)  
    .pipe(  
      take(1)  
    )  
    .subscribe({  
      next: (response) => {  
        if (response.data.token) {  
          this.#service.storeToken(response.data.token)  
        }  
      },  
      error: (error) => {  
        /**  
         * Your logic here  
         */  
      },  
      complete: () => {  
        this.#form.unsubscribe()  
        //document.querySelector('form').remove()  
      }  
    })  
}
```

http-client.js

On ajoute le jeton dans les headers des requêtes (pour la requête post on vérifie d'abord si le jeton existe car lors de la première connexion le jeton n'est pas encore généré)

```
you, il y a 1 heure · 4 authors (DaCoDeMaNiaK and others)
export class HttpClient {
  #_method = 'GET'
  #_uri = ''
  #_contentType = 'application/json'
  #_body = ''
  #_fetchOptions = {}
  #_token = localStorage.getItem('token');
  /**
   *
   * @param {string} uri
   * @returns Observable<any>
   */
  get(uri) {
    this.#_uri = uri

    this.#_fetchOptions = {
      method: 'get',
      mode: 'cors',
      headers: {
        "Content-Type": this.#_contentType,
        "Authorization" : `Bearer ${this.#_token}`
      }
    }
    return this.#send()
  }
}
```

DaCoDeMaNiaK, il y a 6 jours · HttpClient service _

```
post(uri, body) {  
  this.#_method = 'post'  
  this.#_uri = uri  
  this.#_body = JSON.stringify(body)  
  
  this.#_fetchOptions = {  
    method: 'post',  
    mode: 'cors',  
    headers: {  
      "Content-Type": this.#_contentType,  
    },  
    body: this.#_body  
  }  
  if (this.#_token) {  
    this.#_fetchOptions.headers["Authorization"] = `Bearer ${this.#_token}`;  
  }  
  return this.#send()  
}
```



```
put(uri, body) {  
  this.#_method = 'put'  
  this.#_uri = uri  
  
  this.#_body = JSON.stringify(body)  
  
  this.#_fetchOptions = {  
    method: 'put',  
    mode: 'cors',  
    headers: {  
      "Content-Type": this.#_contentType,  
      "Authorization" : `Bearer ${this.#_token}`  
    },  
    body: this.#_body  
  }  
  
  return this.#send()  
}
```

Conclusion

Grâce à cette méthode, chaque requête que l'on voudra sécuriser ne pourra aboutir sans avoir accès au token.

Il reste toujours un risque que le token soit récupéré dans les fichier locaux ou via Postman comme montré plus haut

De nouvelles couches de sécurité pourront être ajoutées dans une prochaine version en ajoutant par exemple une période de validité pour le token. Cela permettra d'éviter l'exécution de script automatique pouvant récupérer le token et exécuter des requêtes. Ou encore en évitant de stocker le token dans le stockage local

Lien du POC

https://docs.google.com/document/d/1_btxSfcYygivg_tjo3UU89NvrYzs0TvFBmVYRnF1BkQ/edit

Mission 8 - Pentest

Loïc Gormand

Identification des vulnérabilités

The screenshot shows a web browser at `localhost:5173` displaying a login page titled "Dev Cyber - Signin". The page has two input fields labeled "Login" and "Password", and a "SIGNIN" button. The version "Version : 1.0.0" is visible at the bottom left. The browser's developer tools are open to the "Network" tab, showing a request to `http://localhost:5173`. The "Headers" sub-tab is selected, displaying the following information:

Header	Value
Content-Type	application/json
Date	Wed, 12 Jun 2024 12:45:19 GMT
Keep-Alive	timeout=5, max=99
Server	Apache/2.4.59 (Debian)
Transfer-Encoding	chunked

The "Request Headers" sub-tab is also visible, showing:

Header	Value
Accept	*/
Accept-Encoding	gzip, deflate, br, zstd
Accept-Language	fr-FR;fr;q=0.9,en-US;q=0.8,en;q=0.7
Connection	keep-alive
Content-Length	47
Content-Type	application/json
Host	localhost:8003
Origin	http://localhost:5173

- Login
- Password
- URL formulaire : <http://localhost:5173>
- URL du back : <http://localhost:8003>

Formulaire formulaire chargé par du JavaScript

→ Exécution avec le package Selenium

```
driver = webdriver.Chrome()
```

→ Extraction du formulaire

```
soup = BeautifulSoup(html_out, 'html.parser')  
form = soup.find('form')
```

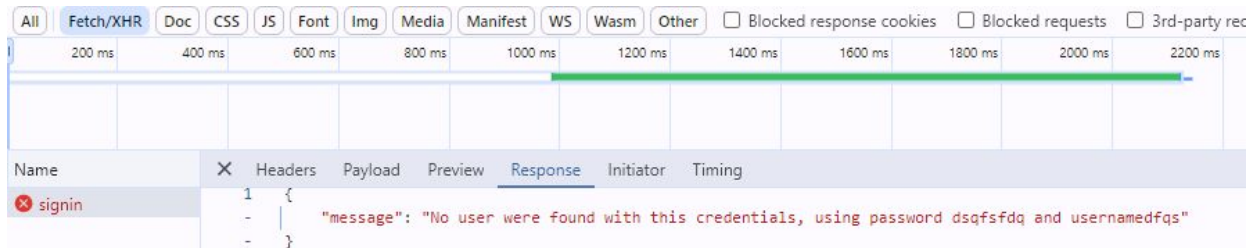
→ Analyse du formulaire

```
<input data-rel="username" required="" type="text"/>
```

```
<input data-rel="userpassword" required="" type="password"/>
```

Initialisation des données

- URL du formulaire : <http://localhost:5173>
- URL utilisé par le backend : <http://localhost:8003/singin>



- Message d'erreur renvoyé : "No user were found with this credentials, using password"
- Liste des exploits à tester (SQL Injection, XSS)

```
exploit_name = ["Correct login/password", "False login/password",  
               "SQL Injection Login 1", "SQL Injection Login 2",  
               "SQL Injection Password 1", "SQL Injection Password 2",  
               "XSS 1", "XSS 2"]  
  
login_list = ["admin", "admin",  
             "' OR '1'='1'; --", "or 1-- -' or 1 or '1'or 1 or'",  
             "false_login", "false_login",  
             "<script>alert(document.domain)</script>", "<img src/onerror=alert(document.cookie)>"]  
  
password_list = ["super_admin", "false_password",  
                "false_password", "false_paswd",  
                "' OR '1'='1'; --", "or 1-- -' or 1 or '1'or 1 or'",  
                "<script>alert(document.domain)</script>", "<img src/onerror=alert(document.cookie)>"]
```

Tests de pénétration

- Réalisé avec la librairie “requests”

```
import requests
```

- Requête HTTP POST ⇒ Récupération de la réponse

```
response = requests.post(url_back, data=payloads[i])  
response_text = response.text
```

Payloads de la forme d'une liste de dictionnaire

```
{0: {'username': 'admin', 'userpassword': 'super_admin'}.}
```

- Tests sur les **injections SQL**

```
if error_message in response_text:  
    resultat = "Access Denied"  
    print(resultat+ "\n")  
else:  
    resultat = "Access accepted"  
    print(resultat + "\n")
```

Tests de pénétration

- **Attaque XSS**

Aucune réponse du back lisible par le navigateur

⇒ Modification de la réponse du back du fichier ./api/user/UserRepository.php

```
} else {  
    throw new NotFoundException('No user were found with this credentials, using password '.$password.' and username '.$username);  
}
```

Comparaison après traitement du back

```
if "<" in response_text:  
    if ">" in response_text:  
        resultat = "success XSS attack"  
        print(resultat+ "\n")  
    else:  
        resultat = "failed XSS attack"  
        print(resultat+ "\n")  
        print(login_list[i])
```

Résultats

- Ecriture d'un rapport dans un fichier csv avec la librairie csv

```
import csv
```

```
writer.writerow([exploit_name[i],resultat, login_list[i], password_list[i]])
```

- Exploitation des résultats (Cas du projet initial)

	A	B	C	D
1	EXPLOIT NAME	RESULT	LOGIN	PASSWORD
2				
3	Correct login/password	Access accepted	admin	super_admin
4				
5	False login/password	Access Denied	admin	false_password
6				
7	SQL Injection Login 1	Access accepted	' OR '1'='1'; --	false_password
8				
9	SQL Injection Login 2	Access accepted	or 1-- '-' or 1 or '1'or 1 or'	false_paswd
10				
11	SQL Injection Password 1	Access accepted	false_login	' OR '1'='1'; --
12				
13	SQL Injection Password 2	Access accepted	false_login	or 1-- '-' or 1 or '1'or 1 or'
14				
15	XSS 1	success XSS attack	<script>alert(document.domain)</script>	<script>alert(document.domain)</script>
16				
17	XSS 2	success XSS attack		

Résultats

- Exploitation des résultats (Cas du projet avec sanitisation)

	A	B	C	D
1	EXPLOIT NAME	RESULT	LOGIN	PASSWORD
2				
3	Correct login/password	Access accepted	admin	super_admin
4				
5	False login/password	Access Denied	admin	false_password
6				
7	SQL Injection Login 1	Access Denied	' OR '1'='1'; --	false_password
8				
9	SQL Injection Login 2	Access Denied	or 1-- -' or 1 or '1'or 1 or'	false_paswd
10				
11	SQL Injection Password 1	Access Denied	false_login	' OR '1'='1'; --
12				
13	SQL Injection Password 2	Access Denied	false_login	or 1-- -' or 1 or '1'or 1 or'
14				
15	XSS 1	failed XSS attack	<script>alert(document.domain)</script>	<script>alert(document.domain)</script>
16				
17	XSS 2	failed XSS attack		

FIN

Merci pour votre attention