Rapport de Correction

Sprint 1 : pop quiz

Encadrant :
Abdelmajid Bendrif

Réalisé par :
Yahya El Omari

Table des matières

1 Présentation du Projet

Ce projet est une application web composée de deux parties distinctes communiquant via une API REST :

— **Backend** : Développé en **PHP 8** avec le micro-framework **Slim**. Il expose des endpoints REST pour gérer des données utilisateurs et des fonctionnalités de test. Il utilise `Composer` pour la gestion des dépendances.

— **Frontend** : Développé en **React (JavaScript)** buildé avec **Vite**. Il consomme l'API backend pour afficher des données dynamiques.

Ce rapport détaille les solutions techniques apportées aux dysfonctionnements identifiés lors de l'audit du code.

2 Chapitre 1 : Corrections Backend

2.1 Challenge 1 : Erreur 500 sur /broken

Problème : La route `/broken` renvoyait une erreur 500.
Cause : Une erreur de syntaxe vicieuse a été détectée dans le fichier `backend/index.php` (ligne 41). Un caractère invisible (Zero Width Space) était inséré au milieu du nom de la méthode `write`.
Solution : Réécriture manuelle de la ligne pour supprimer le caractère corrompu.

Code Avant (Bug) :

```
// Le "i" de write etait coupe par un caractere invisible (Zero Width Space)
$response->getBody()->write("Hello world!");
```

Code Après (Corrigé) :

```
$response->getBody()->write("Hello world!");
```

2.2 Challenge 2 : Crash sur /crash (Connexions simultanées)

Problème : Le serveur crashait (erreur 500) lors de tests de charge (`ab -n 200 -c 10`).
Cause :

— **Mémoire** : La fonction `str_repeat` tentait d'allouer 10Mo de mémoire, dépassant la limite fixée à 8Mo (`memory_limit`).

— **Concurrence** : L'écriture dans le fichier de log ne gérait pas les accès concurrents, provoquant des corruptions lors d'accès simultanés.

Solution : La solution utilise des flux de fichiers (`fopen`) et des verrous (`flock`) pour une gestion sûre et économe en mémoire.

Code Avant (Bug) :

```
function logRequestWithRotation($message) {
    $logFile = __DIR__ . '/request_log.txt';
    $logEntries = file($logFile); // Charge TOUT le fichier en RAM
    if (count($logEntries) > 0xA) {
        $contentClear = str_repeat('A', 0x9FFFF0); // Allocation 10Mo -> Crash
        file_put_contents($logFile, $contentClear);
```

```
    }
    file_put_contents($logFile, $logEntry, FILE_APPEND);
}
```

Code Après (Optimisé) :

```
function logRequestWithRotation($message) {
    $logFile = __DIR__ . '/request_log.txt';
    // Ouverture en mode 'c+' (lecture/écriture sans tronquer)
    $fp = fopen($logFile, 'c+');

    // Acquisition d'un verrou exclusif (bloque les autres processus)
    if (flock($fp, LOCK_EX)) {
        $stat = fstat($fp);
        // Si le fichier dépasse 10KB, on le vide (rotation simple)
        if ($stat['size'] > 10240) {
            ftruncate($fp, 0);
            rewind($fp);
        }
        fseek($fp, 0, SEEK_END);
        fwrite($fp, "[" . date("Y-m-d H:i:s") . "] " . $message .
    PHP_EOL);
        flock($fp, LOCK_UN);
    }
    fclose($fp);
}
```

3 Chapitre 2 : Corrections Frontend

3.1 Challenge 1 : Échec des appels sur /fetch

Problème : Lors de l'appel à la route `/fetch`, le serveur retournait une erreur `401 Unauthorized`. L'analyse du code backend (`index.php`) a révélé que cette route est protégée par une authentification de type "Basic Auth". Le serveur vérifie la présence d'un en-tête `Authorization` contenant les identifiants encodés en Base64. Le code frontend initial effectuait une requête simple sans aucun en-tête, ce qui provoquait le rejet immédiat par le serveur.

Code Avant (Bug) :

```
useEffect(() => {
  // Echec : Pas de header d'authentification
  fetch(`${import.meta.env.VITE_API_URL}/fetch`)
  .then(response => response.json())
},[])
```

Solution : Pour corriger cela, il fallut injecter l'en-tête standard `Authorization` avec la valeur attendue par le backend (`Basic dXNlcm5hbWU6cGFzc3dvcmQ=`). Cette chaîne correspond à `username:password` encodé en Base64.

Code Après (Corrigé) :

```
useEffect(() => {
  // Succes : Header present avec le token Base64
  fetch(`${import.meta.env.VITE_API_URL}/fetch`, {
    headers: {
      'Authorization': 'Basic dXNlcm5hbWU6cGFzc3dvcmQ='
```

```
    }
  })
  .then(response => response.json())
},[])
```

3.2 Challenge 2 : Erreur 405 sur /users

Problème : L'appel à la route `/users` retournait une erreur `405 Method Not Allowed`. Cette erreur survient lorsque la méthode HTTP utilisée par le client (ici, le navigateur) n'est pas acceptée par la route du serveur. Le code frontend forçait explicitement la méthode `POST`, alors que le backend (selon les standards REST pour récupérer des données) n'attend et n'accepte que la méthode `GET`.

Code Avant (Bug) :

```
fetch(`${import.meta.env.VITE_API_URL}/users`, {
  method: "POST", // Erreur : Le backend refuse le POST ici
})
```

Solution : La correction consiste simplement à supprimer l'option `method: "POST"`. Par défaut, la fonction `fetch()` utilise la méthode `GET`, ce qui est exactement ce que le backend attend pour renvoyer la liste des utilisateurs.

Code Après (Corrigé) :

```
fetch(`${import.meta.env.VITE_API_URL}/users`) // GET par defaut
.then(response => response.json())
```

4 Optimisation et Sécurité

4.1 Challenge 3 : Optimisation des Assets

Problème : Les fichiers ne sont pas mis en cache et leurs noms ne changent pas, empêchant l'invalidation du cache (cache busting).

Code Avant (vite.config.js) :

```
export default defineConfig({
  plugins: [react(), TanStackRouterVite()],
})
```

Solution : Configuration de noms de fichiers hashés et ajout de règles de cache Apache.

Code Après (vite.config.js) :

```
export default defineConfig({
  plugins: [react(), TanStackRouterVite()],
  build: {
    rollupOptions: {
      output: {
        // Ajout du [hash] pour le cache busting aux noms de fichiers
        entryFileNames: `assets/[name].[hash].js`,
        chunkFileNames: `assets/[name].[hash].js`,
        assetFileNames: `assets/[name].[hash].[ext]`
      }
    }
  }
})
```

Code Après (.htaccess) :

```
<IfModule mod_headers.c>
    # Cache long (1 an) pour les assets hashes
    <FilesMatch "\.(js|css|png|jpg|jpeg|gif|ico|svg)$">
        Header set Cache-Control "max-age=31536000, public, immutable"
    </FilesMatch>
    # Pas de cache pour index.html (doit toujours etre a jour pour pointer vers les bons assets)
    <FilesMatch "\.(html)$">
        Header set Cache-Control "no-cache, no-store, must-revalidate"
    </FilesMatch>
</IfModule>
```

4.2 Challenge 4 : Sécurité XSS

Problème : Une faille de sécurité de type XSS (Cross-Site Scripting) potentielle a été identifiée sur la page `/security`. Le composant React chargeait une image directement depuis une source externe non de confiance (Unsplash). Sans politique de sécurité stricte, cela permettrait à un attaquant d'injecter des ressources malveillantes ou d'exfiltrer des données si l'URL était manipulée. Le navigateur par défaut autorise le chargement de n'importe quelle ressource, ce qui est dangereux.

Code Source Vulnérable (security.lazy.jsx) :

```
function Security() {
  // L'URL externe 'images.unsplash.com' est chargée sans restriction
  return <img src="https://images.unsplash.com/photo-..." alt="" />
}
```

Code Avant (index.html) : Le fichier HTML initial ne contenait aucune balise de sécurité, laissant le champ libre à tous les domaines.

```
<head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
    <!-- Absence de Content Security Policy (CSP) -->
</head>
```

Solution : La mise en place d'une **Content Security Policy (CSP)** permet de définir une liste blanche des sources autorisées. Nous avons ajouté une balise `<meta>` pour restreindre les images (`img-src`) uniquement au domaine d'origine (`'self'`). En conséquence, le navigateur bloquera automatiquement tout chargement d'image provenant d'autres domaines (comme Unsplash), neutralisant ainsi le risque ("Broken Image" à l'écran).

Code Après (index.html) :

```
<head>
    <meta charset="UTF-8" />
    <!-- CSP Stricte : Autorise uniquement les images du MEME site ('self') -->
    <meta http-equiv="Content-Security-Policy" content="img-src 'self';" />
    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
</head>
```