Rapport de Correction des Problèmes - Pop Quiz

***Réalisé par*: Hamza Lazaar**

Chapitre 1 : Corrections Backend

⚠️ Gestion des erreurs (`/broken`)

Problème : Lors de l'accès à l'endpoint `/broken`, la requête retournait un code 500 – Internal Server Error sans message explicite. L'analyse du fichier `backend/index.php` (ligne 41) a mis en évidence la présence d'un caractère Unicode invisible (U+200B, Zero Width Space) dans l'instruction `$response->getBody()->write("Hello world!");`, ce qui provoquait une erreur de parsing PHP.

- ***L'origine de ce problème a pu être detectée en wrappant l'appel à la méthode write() dans un Try-Catch.***

Solution : Le caractère invisible a été supprimé pour corriger la faute de frappe dans l'appel de fonction `write`. Après cette correction, l'endpoint renvoie correctement un code 200 OK.

Mesures préventives : Pour éviter de tels problèmes, il est recommandé de configurer l'encodage en UTF-8 sans BOM, d'activer l'affichage des caractères invisibles dans l'éditeur et d'utiliser des outils d'analyse statique ou de linting.

🔄 Gestion des connexions simultanées (`/crash`)

Problème : Des tests de charge (`ab -n 200 -c 10 http://localhost:8888/crash`) ont révélé:

- `Allowed memory size of 8388608 bytes exhausted` : Causée par l'instruction `$contentClear = str_repeat('A', 0x9FFFF0);` qui tentait de générer une chaîne d'environ 10 Mo, dépassant la limite mémoire de 8 Mo par requête PHP.
- **Solution :** Augmenter la limite allouée à la mémoire de 8 Mo à 128 Mo par exemple.

Chapitre 2 : Corrections Frontend

🔍 Appel Fetch / XHR qui ne passent pas sur la route `/fetch`

Problème : Les requêtes AJAX vers `/fetch` échouaient avec des erreurs CORS (`No 'Access-Control-Allow-Origin' header is present`) et `401 Unauthorized`. Le frontend (http://localhost:5173) et le backend (http://localhost:8888) étant sur des origines différentes, le navigateur appliquait la politique Same-Origin Policy. La requête envoyée contenait un header `Authorization`, la transformant en une "requête non simple" qui déclenchait une requête preflight `OPTIONS`. Le backend ne gérait pas cette méthode `OPTIONS` et ne renvoyait pas les headers CORS nécessaires.

Solution : Un middleware CORS global a été ajouté au fichier `backend/index.php` (pour l'application Slim) afin de :

- Gérer les requêtes `OPTIONS` en renvoyant une réponse vide avec les headers appropriés.
- Ajouter les headers `Access-Control-Allow-Origin` (`http://localhost:5173`), `Access-Control-Allow-Headers` (`Content-Type, Authorization`) et `Access-Control-Allow-Methods` (`GET, POST, PUT, DELETE, OPTIONS`) à toutes les réponses.Cette implémentation assure que les requêtes preflight sont correctement gérées, les headers CORS sont renvoyés, et la requête `GET` subséquente (incluant l'authentification Basic que le frontend fournit) est autorisée.

🛑 Problème des appels XHR sur la Page `/users`

Problème : Les requêtes AJAX vers la page `/users` échouaient avec un statut `405 Method Not Allowed`. Le frontend (`frontend/src/routes/users.lazy.jsx`) envoyait une requête `POST`, alors que le backend (`backend/index.php`) était configuré pour bloquer

explicitement les `POST` sur cette route, attendant une requête `GET` pour récupérer la liste des utilisateurs.

Solution :

1. **Frontend :** La méthode de la requête `fetch` dans `frontend/src/routes/users.lazy.jsx` a été modifiée de `POST` à `GET` pour correspondre à l'attente du backend.
2. **Backend (Recommandation) :** Pour une meilleure clarté de l'API et pour respecter les principes REST, la définition de la route dans `backend/index.php` a été changée de `$app->any('/users', ...)` à `$app->get('/users', ...)` et la logique de blocage du `POST` a été retirée.

```
function Fetch() {
  useEffect(() => {
    fetch(`${import.meta.env.VITE_API_URL}/users`) // Changed to GET request
    .then(response => {
      if (!response.ok) {
        throw new Error(`HTTP error! status: ${response.status}`);      You, 12
      }
      return response.json();
    })
    .then(data => console.log(data))
    .catch(error => console.error("Error fetching data:", error));
  },[])

  return <div className="p-2">Hello from users!</div>
}
```

🚀 Optimisation du cache des assets statiques

Problème : Les fichiers JavaScript étaient revalidés à chaque rafraîchissement de page (status 304) ou re-téléchargés, ce qui impactait les performances. Cela était dû à une revalidation systématique des assets et à l'absence de directives `Cache-Control` adaptées sur le serveur. Bien que Vite utilise le versioning automatique (hashing) des assets pour l'invalidation du cache, les en-têtes HTTP du serveur étaient insuffisantes.

Solution : La solution principale réside dans l'ajout d'en-têtes HTTP `Cache-Control` appropriées côté serveur. Pour les fichiers statiques (JavaScript, CSS, images, polices), il est recommandé d'utiliser une directive telle que :

`Cache-Control: public, max-age=31536000, immutable`

Cette stratégie permet au navigateur de stocker ces ressources en cache pendant une très longue période (1 an) et de ne pas revalider les assets dont le contenu n'a pas changé (grâce au hash dans leur nom de fichier). L'implémentation de cette solution se ferait idéalement via la configuration du serveur web (Nginx, Apache) ou un middleware si le serveur d'application est responsable de la diffusion des assets statiques.

```php
$app->add(function (Request $request, $handler) {
    $response = $handler->handle($request);
    if ($request->getMethod() === 'OPTIONS') {
        $response = new \Slim\Psr7\Response();
    }
    return $response
        ->withHeader('Access-Control-Allow-Origin', 'http://localhost:5173')
        ->withHeader('Access-Control-Allow-Headers', 'Content-Type, Authorization')
        ->withHeader('Access-Control-Allow-Methods', 'GET, POST, PUT, DELETE, OPTIONS')
        ->withHeader('Cache-Control', 'public, max-age=31536000, immutable');
});
```

🔐 Correction de la Faille XSS sur la Page `/security`

Problème : La page `/security` contenait une image provenant d'une source externe (`https://images.unsplash.com/...`), posant un risque de Cross-Site Scripting (XSS) et ne respectant pas la consigne de bloquer les images ne provenant pas de `localhost`.

Solution : Une politique de sécurité des contenus (CSP - Content Security Policy) a été mise en place pour le serveur de développement Vite. Cette CSP est configurée dans `frontend/vite.config.js` via les en-têtes du serveur (`server.headers`) :

`Content-Security-Policy: default-src 'self'; img-src 'self' blob: data:; script-src 'self' 'unsafe-inline'; style-src 'self' 'unsafe-inline'; connect-src 'self' ws://localhost:5173 http://localhost:8888;`

Cette politique restreint les sources de contenu :

- `default-src 'self'` : Autorise uniquement les ressources provenant du même domaine.
- `img-src 'self' blob: data:` : Autorise les images locales, blob ou base64, bloquant ainsi l'image externe d'Unsplash.
- `script-src 'self' 'unsafe-inline'` et `style-src 'self' 'unsafe-inline'` : Nécessaires pour le développement avec React et Vite (HMR).
- `connect-src 'self' ws://localhost:5173 http://localhost:8888;` : Autorise les connexions WebSocket (HMR Vite) et l'API backend. Cette CSP est appliquée en

mode développement. Pour la production, il est recommandé de supprimer `'unsafe-inline'`, d'autoriser uniquement les domaines strictement nécessaires, et d'utiliser des nonces ou des hashes CSP pour les scripts et styles.

```
// https://vitejs.dev/config/
export default defineConfig({
  plugins: [react(), TanStackRouterVite()],
  server: {
    headers: {
      "Content-Security-Policy":
        "default-src 'self'; img-src 'self' blob: data:; " +
        "script-src 'self' 'unsafe-inline'; style-src 'self' 'unsafe-inline'; " +
        "connect-src 'self' ws://localhost:5173 http://localhost:8888;"
    }
  }
})
```