Atelier 5 : Programmation Asynchrone

Objectif : L'objectif principal de ce Travail pratique et de se familiariser avec les concepts de la programmation asynchrone.

Exercice 1

1. Créer une fonction asynchrone qui simule la récupération de données utilisateur après un délai, puis traiter ces données.

```
const userData = {
        name: 'John Doe',
        email: 'johndoe@example.com',
        avatar : 'avatar.png',
        gender ; 'M'
    };
```

2. Une fois la donnée est récupérée afficher les informations dans une page selon le thème profile.
3. Ajouter plus de données : Ajoutez des informations supplémentaires à l'objet utilisateur, comme login , mot de passe, l'adresse, etc, puis afficher ces utilisateurs dans un tableau.
4. Chainer les Promises : Créez une autre fonction qui retourne une Promise et enchaînez les appels pour simuler une séquence d'actions asynchrones (par exemple, récupérer les données utilisateur puis récupérer les commandes de l'utilisateur).

Exercice 2 :

Eu utilisant le programmation asynchrone Créer un frontend en JavaScript qui interagit avec une API Laravel pour télécharger et récupérer des fichiers.

1. Créer une route et un contrôleur pour le téléchargement de fichiers : upload and file
Créer une fonction pour télécharger des fichiers :

2. Utiliser fetch pour envoyer une requête POST avec le fichier à l'API Laravel.
   - Utilisez une Promise pour gérer la réponse asynchrone.
   - Créer une fonction pour récupérer la liste des fichiers

3. Utiliser fetch pour envoyer une requête GET à l'API Laravel.
   - Utiliser une Promise pour gérer la réponse asynchrone.
   - Afficher les fichiers dans l'interface utilisateur.

4. Utiliser les Promises pour attendre les réponses et afficher les données reçues.

Exercice 3 :

Créer un frontend en JavaScript qui interagit avec une API Laravel pour gérer les salles.
1. Créer une migration pour les salles : rooms : {id, name , capacity }.
2 . Implémentez les méthodes CRUD dans `RoomController`

3. Créer des fonctions pour chaque opération CRUD :
   - Utilisez fetch pour envoyer des requêtes HTTP à l'API Laravel.
   - Utilisez des Promises pour gérer les réponses asynchrones.

4. Créer une interface utilisateur simple :
   - Ajouter des éléments HTML pour l'affichage des salles et les formulaires de création/mise à jour.

Exercice 4 :

Créer une application de suivi des stocks en temps réel où plusieurs utilisateurs peuvent voir les mises à jour des stocks en direct, visualisées avec Highcharts (https://www.highcharts.com/).

1. Utilisez `laravel-websockets` et `pusher/pusher-php-server` pour gérer les WebSockets dans Laravel.

2. Configurer `laravel-websockets` , Pusher :

```
PUSHER_APP_ID=your-app-id
PUSHER_APP_KEY=your-app-key
PUSHER_APP_SECRET=your-app-secret
PUSHER_APP_CLUSTER=mt1
```

et **broadcasting (config/broadcasting.php):**

```
'default' => env('BROADCAST_DRIVER', 'pusher'),
```

3. Créer une migration pour la table stocks : sotck {id, product_name, quantity}.
4. Créez un événement `StockUpdated` :

```
php artisan make:event StockUpdated
```

5. mplémentez les méthodes CRUD dans `StockController` et émettez (boradcast) l'événement StockUpdated. (3 méthodes/actions : store , update et destroy dans chaque action il faut utiliser).

```
broadcast(new StockUpdated($stock))->toOthers();
```

6. **Créer une page HTML pour le suivi des stocks** a travers l'api Highcharts,

- le graphe/chart doit afficher les produis et leur quantités.
- Il faut ajouter aussi une page d'ajout et suppression des produits.

Note n'oublier pas d'intégrer Pusher dans la partie front end :
<script src="https://js.pusher.com/7.0/pusher.min.js"></script>