Travail pratique 2: Single Page Application (SPA) avec PHP, MySQL et JavaScript

Objectifs

Vous devrez créer une application web de livraison de pizzas en utilisant une architecture SPA (Single Page Application).

L'application doit permettre, sans recharger la page, aux utilisateurs:

- de parcourir le menu et le trier par prix ou par nom
- de voir le détail de chaque pizza au clic d'une pizza du menu

Elle doit permettre également à l'administrateur:

- d'ajouter des pizzas au menu.

Vous n'avez pas à implémenter le côté serveur. Les scripts PHP pour interagir avec la base de données MySQL sont déjà fournis. Vous devez vous concentrer sur le développement du front-end en JavaScript. Vous devez par contre installer la base de données MySQL en utilisant les scripts SQL fournis. Vous n'avez pas à modifier les scripts PHP, sauf le fichier de configuration de la base de données pour y mettre vos informations de connexion à votre serveur MySQL.

Architecture de l'application

L'architecture de l'application est organisée comme suit :

- index.js : Le point d'entrée de l'application qui initialise l'application.
- Application.js : Le gestionnaire d'état global qui coordonne les services et l'état partagé.
- Router.js : Le gestionnaire des routes qui écoute les changements d'URL et affiche les vues appropriées.
- vues/ : Le répertoire contenant les différentes vues/pages de l'application (Accueil, PizzaDetail, PizzaAjout).
- composants/ : Le répertoire contenant les composants réutilisables (Pizza, Formulaire, Filtre, etc.).
- api/ : Le répertoire contenant les scripts PHP pour interagir avec la base de données MySQL.
- api/sql/ : Le répertoire contenant le script SQL pour créer et peupler la base de données. À installer avant de lancer l'application.
- api/config/database.php : La classe de configuration de la base de données. À modifier avec vos informations de connexion.

Tâches à réaliser

1. Installation du projet

1. Installer la base de données MySQL en utilisant le script SQL fourni dans `api/sql/donnees.sql`.

2. Configurer la connexion à la base de données dans `api/config/database.php`.
3. Tester les scripts PHP individuellement pour s'assurer qu'ils fonctionnent correctement en utilisant un outil comme Postman ou directement via le navigateur.
4. Si vous développez avec XAMPP, WAMP ou MAMP, assurez-vous que le fichier `.htaccess` est correctement configuré pour rediriger les requêtes vers les scripts PHP en modifiant la règle `RewriteBase` si nécessaire.
5. Dans le fichier `index.php`, modifier la balise `<base href="/">` si l'application n'est pas hébergée à la racine du serveur web.

2. Application et routage

1. Implémenter la classe `Application.js` pour gérer l'état global de l'application et instancier le Router.
    - Créer les méthodes pour accéder aux actions des pizzas (récupérer le menu, récupérer une pizza par ID, ajouter une pizza).
2. Implémenter le gestionnaire de routes dans `Router.js` pour écouter les changements d'URL. Vous avez trois routes principales à gérer :
    - `/` : Affiche la vue d'accueil avec le menu des pizzas.
    - `/pizza/:id` : Affiche le détail d'une pizza spécifique.
    - `/admin` : Affiche le formulaire pour ajouter une nouvelle pizza.
    - Créer également une route par défaut pour afficher une page 404 si la route n'existe pas.

3. Vues

1. Créer les vues suivantes dans le répertoire `views/` :
    - `AccueilView.js` : Affiche le menu des pizzas en utilisant le composant Pizza pour chaque pizza.
    - `PizzaDetailView.js` : Affiche les détails d'une pizza spécifique en utilisant le composant Pizza.
    - `PizzaAjoutView.js` : Affiche un formulaire pour ajouter une nouvelle pizza en utilisant le composant Formulaire.
    - `Pizza404.js` : Affiche un message qui indique que la page demandée n'existe pas avec un lien pour retourner à l'accueil.

4. Composants

1. Créer un composant `Filtre.js` pour trier les pizzas par prix ou par nom sur la vue d'accueil.
2. Créer un composant `Formulaire.js` pour gérer l'ajout de nouvelles pizzas dans la vue d'administration.
3. Créer un composant `Pizza.js` pour gérer les informations d'une pizza (nom, description, prix, image, ingrédients, options sans gluten et vegan).
4. Créer un composant `Toast.js` pour afficher des messages de succès ou d'erreur lors de l'ajout d'une pizza.

Informations

Au niveau de l'image des pizzas, vous n'avez pas à gérer le téléchargement de fichiers. Dans le formulaire d'ajout de pizza, l'utilisateur doit simplement fournir le nom du fichier image (par exemple, "margherita.jpg").

Vous pouvez utiliser des images fictives ou des images existantes dans le répertoire `images/` pour afficher les pizzas.

Vous pouvez réutiliser et **ADAPTER** le code des travaux pratiques précédents. L'objectif est de bien comprendre l'architecture et le flux de données dans une application SPA sans avoir à tout réécrire de zéro.

Au niveau du design, vous avez la liberté de choisir votre propre style CSS. Il peut être simple mais doit être fonctionnel et agréable à utiliser.

Vous n'avez **pas** à déployer l'application sur un serveur web public. Il suffit de la faire fonctionner en local sur votre machine. Si vous la déployez, assurez-vous que le lien est accessible.

Bonus

Si vous implémentez la modification et la suppression de pizzas dans la vue d'administration, j'ajouterai 5% à la note de l'évaluation. Cependant, ce n'est pas obligatoire.

Critères d'évaluation

Le travail sera évalué selon les critères suivants :

- Juste compréhension des fonctionnalités demandées à l'intérieur du devis (10 %): Est-ce que toutes les fonctionnalités demandées sont présentes et fonctionnelles ?
- Utilisation correcte d'un langage de programmation orienté objet côté client (30 %) : Est-ce que le code est bien structuré en classes et objets, avec une bonne séparation des responsabilités ? Est-ce que les concepts de base de la programmation orientée objet sont bien appliqués (encapsulation, héritage, polymorphisme) ?
- Gestion adéquate de l'affichage dynamique (20 %): Est-ce que l'application met à jour l'interface utilisateur de manière fluide et réactive en fonction des interactions de l'utilisateur et des changements d'état ?
- Traitement et exécution adéquates des réponses synchrones et asynchrones (25 %): Est-ce que l'application gère correctement les appels API et les mises à jour de l'interface utilisateur en fonction des réponses reçues ? Affichage des messages d'erreur ou de succès appropriés ?
- Structure et qualité optimale de la programmation (15 %): Est-ce que le code est bien organisé, lisible et maintenable ? Est-ce que les bonnes pratiques de développement sont respectées ? Est-ce que le code est bien commenté et documenté ?

Remise

Le travail compte pour 25 % de la note finale du cours et doit être remis avant le cours 19.

Vous devez remettre un fichier ZIP contenant l'ensemble du projet, y compris les scripts PHP et SQL. Assurez-vous que le code est bien commenté et structuré.

Vous devez déployer votre code sur un dépôt GitHub et fournir le lien dans votre remise. Si vous voulez garder votre code privé, vous pouvez créer un dépôt privé et ajouter m'ajouter comme collaborateur: "mlacassegermain@cmaisonneuve.qc.ca".

Plagiat

Le plagiat est strictement interdit. Toute forme de plagiat entraînera une note de zéro pour le travail tel qu'indiqué dans la politique institutionnelle d'évaluation des apprentissages (PIEA). L'utilisation de code provenant de sources externes doit être correctement citée et référencée.

L'utilisation de génération de code par IA n'est pas autorisée et sera considérée comme du plagiat.