Rapport Projet Programmation Web
Gestion de Recettes de Cuisine

Bogdan GORELOV
Université Paris-Saclay

15 avril 2025

Table des matières

1 Introduction **2**

2 Architecture et Rôle des Fichiers **2**
2.1 Découpage du Code 2
2.2 Rôle des Fichiers Principaux 2

3 Fonctionnalités Clés et Mécanismes Techniques **3**
3.1 Appels AJAX 3
3.2 Gestion des Champs de Formulaire 3
3.3 Logique de Traduction Dynamique 4

4 Fonctionnalités Envisagées Non Implémentées **4**

5 Difficultés Rencontrées **4**

6 Conclusion **5**

1 Introduction

Ce rapport présente le développement d'une application web visant à gérer un catalogue de recettes de cuisine, dans le cadre d'un projet d'initiation à la programmation web. L'objectif principal était de mettre en œuvre les technologies fondamentales du web (HTML, CSS, JavaScript avec la bibliothèque **jQuery**, PHP) tout en mettant un accent particulier sur l'utilisation d'**AJAX** (Asynchronous JavaScript and XML) pour créer une interface utilisateur dynamique et réactive.

L'application permet la gestion de recettes multilingues (français/anglais) et intègre un système de rôles utilisateurs (Cuisinier, Chef, Traducteur, Administrateur) avec des permissions distinctes. Les données sont stockées et manipulées via des fichiers JSON, simulant une base de données simple. Ce projet a servi de plateforme d'apprentissage pour la manipulation du DOM avec jQuery, les interactions client-serveur asynchrones, la gestion de sessions PHP et les bases de la structuration d'un projet web.

2 Architecture et Rôle des Fichiers

L'application adopte une architecture client-serveur classique :

- **Client (Navigateur)** : Gère l'interface utilisateur (HTML/CSS) et l'interactivité (JavaScript/jQuery). Il initie des requêtes AJAX vers le serveur pour récupérer des données ou déclencher des actions sans recharger la page entière.
- **Serveur (PHP)** : Traite les requêtes reçues du client, interagit avec les fichiers de données JSON (lecture/écriture), gère les sessions utilisateur (authentification, rôles) et renvoie des réponses (souvent au format JSON) au client.

2.1 Découpage du Code

Le projet a été découpé en plusieurs fichiers afin de promouvoir la modularité, la réutilisabilité et la maintenabilité :

- **Séparation des préoccupations** : HTML pour la structure, CSS pour la présentation, JavaScript pour l'interactivité client, PHP pour la logique serveur.
- **Modularité** : Chaque fichier PHP traitant une requête AJAX (ex : `auth.php`, `comment.php`) a une responsabilité unique.
- **Réutilisabilité** : Le fichier `header.php` contient la structure commune (navigation, etc.) et le script JS global (gestion de la langue, authentification, fonction `showMessage`).
- **Lisibilité** : Un découpage logique facilite la compréhension par rapport à un fichier unique.

2.2 Rôle des Fichiers Principaux

Fichiers PHP (Interface) : `index.php`, `recipe.php`, `create_recipe.php`, `modify_recipe.php`, `translate_recipe.php`, `admin.php`, `profile.php`. Génèrent le HTML initial et incluent du JS spécifique.

Fichiers PHP (Backend/AJAX) : `auth.php`, `comment.php`, `like_recipe.php`, `update_users.php`, `update_recipes.php`, `remove_recipe.php`, `logout.php`. Reçoivent les requêtes AJAX, traitent les données JSON, renvoient des réponses JSON.

Fichiers JavaScript :

- Script dans `header.php` : Logique globale (traduction dynamique, authentification, messages, astuce du jour, appel `initializePageContent`).
- Scripts spécifiques (dans chaque page PHP interface) : Fonction `initializePageContent` pour charger le contenu dynamique de la page et attacher les écouteurs d'événements.

Implémentation simple de l'ajout/suppression de champs dans les formulaires (sans fichier externe).

Fichiers de Données : `recipes.json`, `users.json`, `data.json` (traductions).

Fichiers CSS : `styles.css` (mise en forme visuelle).

3 Fonctionnalités Clés et Mécanismes Techniques

3.1 Appels AJAX

L'utilisation intensive d'AJAX fluidifie l'expérience utilisateur. Le mécanisme est le suivant :

1. Action utilisateur → Fonction JavaScript.
2. JavaScript (jQuery `$.ajax`, `$.getJSON`) → Requête HTTP asynchrone vers un fichier PHP serveur avec des données.
3. PHP serveur reçoit, traite (session, JSON), prépare une réponse.
4. PHP serveur → Réponse (souvent JSON) au client.
5. JavaScript (callback `success`/`done`) reçoit la réponse.
6. JavaScript met à jour le DOM dynamiquement (compteurs, listes, messages via `showMessage`) sans recharger la page.

Exemples concrets :

— **Authentification** (`header.js` ↔ `auth.php`) : Login/Signup. Reload après login.
— **Likes** (`index.js`/`recipe.js` ↔ `like_recipe.php`) : Mise à jour compteur/bouton.
— **Commentaires** (`recipe.js` ↔ `comment.php`) : Ajout dynamique du commentaire.
— **Gestion Admin** (`admin.js` ↔ `update_users.php`/`update_recipes.php`) : Màj rôles/mdp, validation. Mise à jour tableau HTML.
— **Demande de Rôle** (`profile.js` ↔ `update_users.php`) : Màj rôle utilisateur.
— **Chargement Données** (`index.js`, etc. ↔ `*.json`) : Récupération initiale via `$.getJSON`.
— **Traduction** (`header.js` ↔ `data.json`) : Chargement initial et au changement de langue.

3.2 Gestion des Champs de Formulaire

Les pages de création (`create_recipe.php`) et modification (`modify_recipe.php`) de recettes permettent d'éditer des listes d'ingrédients, d'étapes et de minuteurs :

— **Affichage Statique (PHP)** : Les pages PHP génèrent un nombre fixe (ex : 10 ou 15) de champs vides ou pré-remplis pour chaque section (ingrédients EN/FR, étapes EN/FR, minuteurs).
— **Pas d'Ajout/Suppression JS** : Il n'y a pas de boutons "+" ou "x" gérés par JavaScript pour modifier le nombre de champs sur l'interface. L'utilisateur travaille avec le nombre de champs affichés.
— **Gestion Côté Serveur (PHP)** : Lors de la soumission du formulaire, le script PHP (`create_recipe.php` ou `modify_recipe.php`) utilise la fonction `array_filter` pour ignorer et donc effectivement "supprimer" les champs que l'utilisateur a laissés vides. La fonction `array_values` est ensuite utilisée pour réindexer les tableaux avant de les sauvegarder dans le fichier JSON, assurant ainsi que les données stockées sont propres et correctement indexées. La validation de la cohérence entre le nombre d'étapes et de minuteurs est également effectuée côté serveur après ce filtrage.

Cette méthode simplifie grandement le code JavaScript mais rend l'interface moins flexible pour l'utilisateur, qui doit laisser des champs vides pour simuler une suppression.

3.3 Logique de Traduction Dynamique

L'application gère l'affichage en français et en anglais sans rechargement de page lors du changement de langue.

1. Le fichier `data.json` contient toutes les chaînes de texte de l'interface (labels, boutons, messages, placeholders, astuces) organisées par langue (‘en‘, ‘fr‘) et par catégorie (ex : ‘labels‘, ‘buttons‘, ‘messages‘).
2. Au chargement initial de la page (`header.php`), JavaScript détermine la langue actuelle (depuis le `localStorage` ou 'fr' par défaut) et charge l'intégralité du fichier `data.json` via `$.getJSON`.
3. Les traductions pour la langue courante sont stockées dans une variable globale JavaScript (`currentTranslations`).
4. La fonction JavaScript `translatePage` est appelée. Elle parcourt tous les éléments HTML ayant un attribut `data-translate="cle.de.traduction"`.
5. Pour chaque élément, elle utilise la fonction `getNestedTranslation` pour trouver la valeur correspondante à la clé (ex : `currentTranslations['buttons']['login']`) dans l'objet `currentTranslations`.
6. La traduction trouvée est ensuite injectée dans l'élément HTML approprié (via `.html()`, `.text()` ou `.attr('placeholder', ...)` selon le type d'élément).
7. Lorsque l'utilisateur clique sur le bouton de changement de langue, la langue courante est mise à jour (et sauvegardée dans `localStorage`), les fonctions `translatePage` et `displayRandomTip` sont rappelées, et la fonction `initializePageContent` de la page spécifique est également réexécutée pour mettre à jour le contenu dynamique (liste de recettes, détails, etc.) dans la nouvelle langue.

Par exemple, un bouton ‘<button data-translate="buttons.login">Login</button>‘ verra son contenu remplacé par "Se connecter" si la langue sélectionnée est 'fr' et que la clé ‘buttons.login‘ existe dans la section ‘fr‘ de ‘data.json‘.

4 Fonctionnalités Envisagées Non Implémentées

Au cours du développement, l'idée d'ajouter une photo pour chaque étape de recette a été envisagée pour améliorer la visualisation. Cependant, cela n'a pas été implémenté en raison de la complexité ajoutée (upload multiple, stockage, association JSON) et de l'impact potentiel sur le stockage disque, jugés hors de portée pour ce projet d'initiation. Seuls l'upload d'une image principale (URL) et d'une image par commentaire sont possibles.

5 Difficultés Rencontrées

Le développement a présenté plusieurs défis :

Intégration de la Traduction

L'application ayant été initialement conçue en une seule langue, l'ajout du multilinguisme a nécessité une refonte importante : modification de la structure JSON, création de `data.json`, intégration du mécanisme JS de traduction dynamique (`data-translate`, `getNestedTranslation`, `translatePage`), adaptation des fonctions générant du HTML. Le débogage des clés de traduction manquantes ou des erreurs JS liées s'est avéré chronophage.

Gestion des Erreurs

Une gestion exhaustive des erreurs (réseau AJAX, JSON corrompus, permissions fichiers, données invalides, accès concurrents) n'a pas été implémentée en profondeur, car cela aurait nécessité d'analyser de nombreux cas spécifiques, ce qui dépassait l'objectif d'apprentissage des bases.

Sécurité

Seules les protections fondamentales ont été mises en place : `htmlspecialchars` contre XSS et `password_hash`/`password_verify` pour les mots de passe. Les aspects plus avancés (CSRF, validation serveur approfondie) n'ont pas été abordés.

Développement CSS "Pure"

L'absence de framework CSS (comme Bootstrap ou Tailwind) a nécessité l'écriture de tout le CSS "à la main". Bien que formateur pour comprendre les bases de Flexbox, Grid et le positionnement, cela s'est avéré chronophage pour obtenir une mise en page et un style cohérents et responsives sur l'ensemble des composants et des pages. L'organisation du fichier `styles.css` est devenue importante pour maintenir la clarté.

Évolution de la Structure du Code et Ressources

La structure du code a évolué au fil du développement, entraînant parfois des réorganisations. Pour résoudre les problèmes techniques ou comprendre certains concepts, les ressources principalement consultées ont été la documentation de **W3Schools**, divers **tutoriels YouTube** sur PHP, JS/jQuery et AJAX, ainsi que les réponses aux questions sur **Stack Overflow**.

6 Conclusion

Ce projet a constitué une expérience d'apprentissage riche sur les fondamentaux du développement web (HTML, CSS, JS/jQuery, PHP). L'utilisation intensive d'AJAX a permis de comprendre comment créer des interfaces utilisateur fluides et réactives. Le stockage via JSON, bien que simple, a été pédagogique pour la manipulation des données serveur.

Les difficultés, notamment l'ajout de la traduction et la gestion limitée des erreurs/sécurité, ont souligné l'importance de la conception initiale. L'implémentation des bases de sécurité (`htmlspecialchars`, hashage mdp) a mis en lumière l'importance de cet aspect.

Ce projet fournit une base solide pour explorer des technologies plus avancées. L'expérience acquise sur les mécanismes sous-jacents (HTTP, AJAX, DOM, sessions PHP) sera utile pour mieux appréhender des frameworks comme Laravel, déjà abordé lors de projets précédents. L'importance d'une approche structurée, sécurisée et l'utilité d'AJAX pour l'UX sont les principaux enseignements.