Rapport de Projet

RE7

Site proposant des recettes de cuisines

Ismael WANE Mohamed Almahdi LDD3 MAG

Sommaire

Introduction

Le projet présenté dans ce rapport s'inscrit dans le cadre du module de **Programmation Web** de l'année universitaire 2024-2025. Il a pour objectif la conception et le développement d'une application web innovante dédiée à la gestion et au partage de **recettes culinaires multilingues**, en français et en anglais.

Dans un contexte où les pratiques culinaires se mondialisent et où les communautés en ligne échangent de plus en plus d'expériences gastronomiques, ce projet répond au besoin d'une plateforme structurée, collaborative et multirôle, permettant à différents profils d'utilisateurs — des cuisiniers amateurs aux chefs confirmés, en passant par les traducteurs spécialisés et les administrateurs — d'interagir efficacement autour de contenus culinaires.

L'application repose sur un **modèle de données initial** fourni sous forme de fichier JSON, contenant un ensemble de recettes partiellement renseignées. L'objectif majeur est non seulement d'enrichir et de structurer ces données, mais aussi de garantir leur accessibilité et leur qualité, grâce à un système de gestion des rôles finement défini :

- Les **cuisiniers** apportent un regard communautaire par des commentaires, photos et appréciations (« cœurs ») ;
- Les **chefs** enrichissent le contenu en proposant et en modifiant des recettes ;
- Les **traducteurs** assurent la complétude linguistique en traduisant les éléments manquants entre les deux langues ;
- Les **administrateurs** garantissent la cohérence, valident les recettes, modèrent les contenus et attribuent les rôles.

Ce projet s'inscrit également dans une démarche technologique spécifique, imposant une **structure mono-page (SPA)**, une gestion fine des utilisateurs et des droits, et un soin particulier apporté à l'ergonomie et au design (interface moderne, CSS lisible, multilingue).

Enfin, au-delà des aspects techniques, le projet répond à des enjeux plus larges en matière de **qualité logicielle** : chaque composant devra être rigoureusement commenté et testé, conformément aux bonnes pratiques du développement web, et les extensions ou choix technologiques avancés (patrons de conception, frameworks non vus en cours) devront être préalablement validés par l'encadrement pédagogique.

I – Cahier des charges

I.I Public cible

L'application s'adresse à :

- **Les cuisiniers amateurs** : utilisateurs souhaitant explorer des recettes, commenter, partager des photos et exprimer leur appréciation.
- **Les chefs** : utilisateurs expérimentés capables de proposer de nouvelles recettes ou de modifier leurs propres créations.
- **Les traducteurs** : utilisateurs spécialisés, chargés d'assurer la traduction des recettes entre le français et l'anglais, qui peuvent en plus être chefs ou de simples cuisiniers.
- **Les administrateurs** : gestionnaires du site, responsables de la validation, de la modération des contenus et de l'attribution des rôles.
-

I.II Fonctionnalités principales

Nous présentons les sous fonctionnalités de notre projet dans le cahier des charges, cependant, dans la partie suivante, elles seront regroupées sous forme de fonctionnalités (certaines sous fonctionnalités sont communes à travers leur conception).

Authentification et inscription

- Interface d'inscription avec rôle initial « cuisinier » ;
- Option pour demander un rôle supplémentaire (« askChef », « askTraducteur ») ;
- Interface de connexion (login, mot de passe).

Gestion des utilisateurs

- Visualisation par l'administrateur de la liste des utilisateurs ;
- Attribution, modification ou suppression des rôles par l'administrateur.
- Validation des demandes de création de recettes.
- Validation des demandes de nouveaux rôles.

Gestion des recettes

- Visualisation d'une liste filtrable et triable de recettes (sans gluten, vegan, etc.) ;
- Consultation détaillée d'une recette : ingrédients, étapes, photos, commentaires, temps total.
- Ajout de commentaires, photos, et « cœurs » par tous les utilisateurs.
- Création et modification de recettes par les chefs (avec attribution automatique comme auteur).
- Validation, publication, suppression ou modification des recettes par l'administrateur.

Traduction des recettes

- Interface spécialisée à double colonne (français ↔ anglais) pour les traducteurs, affichant les champs complétés et laissant éditables uniquement les champs vides.
- Vérification de la cohérence (même nombre d'ingrédients et d'étapes entre les deux langues).

Recherche et filtres

- Moteur de recherche textuelle sur les ingrédients, étapes, titres.

Multilinguisme

- Formulaire permettant de basculer entre les versions française et anglaise du site.

Gestion des médias

- Gestion des photos associées aux recettes, via URL globale ou upload local.
- Sélection d'une photo principale représentant la recette parmi plusieurs.

Gestion des favoris de l'utilisateur

- Chaque utilisateur peut marquer des recettes comme favoris ;
- Visualisation de la liste des recettes favorites dans le profil de l'utilisateur.

Section du profil de l'utilisateur

- Liste des **commentaires** de l'utilisateur (pour chaque recette commentée) ;
- **Recettes commentées** par l'utilisateur, avec possibilité de les visualiser et modifier ;
- **Recettes likées** par l'utilisateur, avec affichage dans le profil et possibilité de supprimer un like.

Section du compte de l'utilisateur

- Possibilité de demander un rôle supplémentaire (par exemple : « Chef » ou « Traducteur ») à travers une demande soumise à l'administrateur ;
- **Modification de l'email** de l'utilisateur dans le compte ;
- Possibilité de visualiser et modifier les informations personnelles de l'utilisateur (prénom, nom, etc.).

Système de sessions utilisateur

- **Enregistrement des données navigateurs** de l'utilisateur durant toute sa session
- **Destruction de ces données** à la fin de la session

I.III Contraintes techniques

- Application majoritairement mono-page (SPA) : navigation fluide sans rechargements complets.
- Stockage initial sous format JSON, librement extensible.

- Respect des droits utilisateurs : modification limitée selon les rôles. Interface adaptée selon le rôle, gestion sécurisée des actions sensibles.
- Interface ergonomique et moderne, assurée par un design CSS soigné.
- Code documenté, testé, et conforme aux standards établis.

I.IV Évolutions possibles

- Introduction de structures de données enrichies : unités standardisées, tables séparées pour les ingrédients.
- Amélioration et extension de la fonctionnalité de filtrage.
- Ajout d'un système de notifications ou d'un tableau de bord personnalisé.
- Mise en place d'une API RESTful pour séparer le front-end et le back-end.
- Intégration d'un système d'upload sécurisé avec gestion des formats et des tailles d'image.

II – Implémentation des fonctionnalités

Authentification et inscription :

a – Analyse

Objectif fonctionnel :
Permettre à un utilisateur de :

- Se connecter (formulaire de login) via identifiants (nom d'utilisateur et mot de passe) ;
- S'inscrire (formulaire d'inscription) pour créer un compte personnel, initialement avec le rôle « cuisinier » et, en option, demander les rôles « Chef » ou « Traducteur ».

Contraintes :

- Chaque nom d'utilisateur doit être unique.
- Chaque adresse email doit être unique.

- Le mot de passe doit satisfaire des critères de robustesse : ≥8 caractères, au moins une majuscule, une minuscule, un chiffre, un caractère spécial.
- Les données des utilisateurs doivent être persistantes dans un fichier users.json.
- Multilinguisme : les interfaces doivent être accessibles en français et en anglais.

Données manipulées :

- Fichier users.json (format JSON) contenant une liste d'utilisateurs sous forme de dictionnaires

  Ex : { "nom": "...", "prenom": "...", "username": "...", "password": "...", "mail": "...", "role": ["cuisinier", ...], "posts": [], "likes": [], "recettes": [] }

Comportement attendu côté utilisateur :

- Formulaire fluide, mono-page, permettant de basculer dynamiquement entre **connexion** et **inscription** sans rechargement.
- Enregistrement immédiat de l'utilisateur après validation des contraintes (username, email, mot de passe).
- Notification en cas d'échec (identifiant pris, email déjà utilisé, mot de passe non valide).
- Passage automatique à l'écran de login après succès de l'inscription.

b – Conception

Architecture logicielle :

- **Frontend (create_login.php + cl.js)** :
  - Interface HTML/CSS multilingue.
  - Gestion dynamique des formulaires avec jQuery :
    ↳ Passage de login → inscription (createAccountBtn) ;
    ↳ Passage d'inscription → login (backToLoginBtn).
  - Validation côté client avant soumission :
    ↳ Vérification de disponibilité (nom d'utilisateur, email) ;
    ↳ Vérification de la robustesse du mot de passe.
- **Backend (service.php)** :
  - Réception des données via GET.
  - Lecture et décodage du fichier users.json.
  - Vérification de l'unicité du nom d'utilisateur.
  - Ajout des données utilisateurs avec initialisation des champs (posts, likes, recettes) et rôles : ["cuisinier", rôle demandé].
  - Sauvegarde persistante dans le fichier JSON.
  - Réponse JSON au frontend (success ou error).

Schéma de flux :

1. L'utilisateur remplit le formulaire →
2. Validation JS locale (cl.js) →
3. Requête Ajax (GET) vers service.php →
4. Vérification et enregistrement serveur →
5. Réponse JSON →
6. Affichage des messages (alertes) et mise à jour d'interface.

Algorithmes et vérifications clés :

- Regex côté client : /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[\W_\.]).{8,}$/
- Vérification de disponibilité (client) via lecture du JSON (usernameEstDisponible, emailEstDisponible).
- Vérification d'unicité (serveur) avant écriture (parcours du tableau des utilisateurs).
- Stockage structuré des rôles sous forme de liste, garantissant toujours l'inclusion de « cuisinier ».

Cas limites gérés :

- Champ manquant ou vide → rejet.
- Nom d'utilisateur ou email déjà existant → rejet.
- Format incorrect du mot de passe → rejet.
- Fichier users.json absent ou corrompu → réinitialisation propre (tableau vide).

B – Difficultés rencontrées

1. **Sécurité des requêtes :**
Le passage des données en GET expose les informations sensibles (mot de passe) dans l'URL, ce qui constitue une faiblesse critique. Une meilleure approche consisterait à utiliser :

- Méthode POST ;
- Transmission sécurisée (HTTPS) ;
- Hashage des mots de passe avant stockage (actuellement stockés en clair, ce qui est inacceptable en production).

2. **Synchronisation des vérifications client-serveur :**
Même si le client vérifie localement la disponibilité des identifiants, une vérification serveur reste indispensable pour éviter les contournements (par ex. en forgeant manuellement des requêtes).

3. **Robustesse du fichier JSON :**
Les accès concurrents (plusieurs inscriptions simultanées) risquent de provoquer des corruptions du fichier users.json. Une solution serait d'introduire un mécanisme de verrou (file lock) ou de basculer sur un système plus robuste (base de données relationnelle ou NoSQL).

4. **Support multilingue :**
La gestion des langues repose uniquement sur la variable $_SESSION['lang'] et une sélection en début de session, sans prise en charge dynamique sur toutes les pages ni fallback complet pour les textes non traduits.

Système et gestion des Session :

a – Analyse

Objectif fonctionnel :
Garantir que l'état de connexion des utilisateurs soit maintenu entre les pages du site, permettant :

- L'identification persistante après connexion (nom d'utilisateur stocké en session) ;
- La personnalisation des contenus (accès conditionnel aux liens et fonctionnalités selon le rôle : cuisinier, chef, traducteur, admin) ;
- La récupération fluide des informations utilisateur (nom, rôles, likes) sans requêtes répétées au fichier JSON ;
- La protection des zones restreintes (ajout/édition/validation de recettes, traduction, administration) en vérifiant les sessions.

Contraintes :

- Stockage des informations sensibles côté serveur, non dans les cookies visibles.
- Fiabilité et cohérence des rôles et droits récupérés à chaque page.
- Compatibilité avec le multilinguisme de l'interface.

Données manipulées :

- Variables $_SESSION :
    - $_SESSION['username'] → identifiant de l'utilisateur connecté.
    - $_SESSION['lang'] → langue actuelle sélectionnée.
- Fichier users.json : récupération des rôles (role), des recettes aimées (likes), etc.

Comportement attendu côté utilisateur :

- Après connexion réussie, l'utilisateur navigue librement sans réauthentification manuelle.
- Les menus, filtres et actions disponibles changent dynamiquement selon les rôles (ex. voir le lien « Ajouter une recette » seulement si chef).
- La session prend fin explicitement via une action de déconnexion (logout).

b – Conception

Architecture logicielle :

- **Backend PHP** :
    - Démarrage systématique de session_start() sur les pages utilisant les sessions (check_session.php, main.php, etc.).
    - Vérification de la session active (isset($_SESSION['username'])).
    - Chargement des informations utilisateur détaillées (rôles, likes) depuis users.json au moment du rendu.
- **Fichier check_session.php** :
    - Fournit une interface minimaliste JSON pour interroger l'état de connexion par Ajax.
    - Exemples d'usage : affichage dynamique d'éléments côté client sans recharger toute la page.
- **Fichiers *.php** :
    - Lecture de la langue courante via $_SESSION['lang'] et adaptation du contenu HTML.
    - Définition de variables d'état ($isConnected, $currentUser, $currentRoles) utilisées pour afficher dynamiquement les liens et actions autorisés.

Schéma de flux :

1. L'utilisateur se connecte →
2. Le serveur enregistre $_SESSION['username'] →
3. À chaque chargement de page, le serveur vérifie cette variable pour déterminer les accès →
4. Les rôles, droits et contenus s'adaptent dynamiquement →
5. L'utilisateur se déconnecte → la session est détruite.

Modules/fichiers impliqués :

- **check_session.php** : point de vérification Ajax.
- **main.php** : page principale, rendue conditionnellement selon session et rôles.
- **logout.php** (non fourni mais attendu) : destruction propre de la session.

Cas limites gérés :

- Absence de session : l'utilisateur est redirigé ou voit une version limitée du site.
- Corruption des données en session : rechargement propre des rôles depuis le fichier JSON.
- Multi-onglets : la session PHP reste partagée entre les onglets du même navigateur.

B – Difficultés rencontrées

1. **Consistance des données utilisateur :**
Les rôles et droits sont recalculés à chaque chargement de page à partir du fichier users.json. Cela garantit

l'actualisation en cas de changement de rôle, mais peut poser des problèmes de performance si le fichier devient volumineux. Une solution optimisée impliquerait un stockage structuré (ex. en base de données) ou un cache côté serveur.

2. **Sécurité des sessions :**
La simple vérification isset($_SESSION['username']) est insuffisante pour sécuriser totalement les pages sensibles. En l'état, il n'y a pas de vérification active des jetons de session, pas de protection contre l'usurpation d'identité, ni de mécanismes avancés comme l'expiration automatique ou la régénération des identifiants de session après connexion.

3. **Langue et personnalisation persistante :**
La langue est enregistrée dans la session ($_SESSION['lang']), ce qui fonctionne bien pour l'affichage multilingue. Cependant, elle pourrait être améliorée en enregistrant ce paramètre côté utilisateur (dans users.json), pour être restaurée même après une nouvelle connexion.

4. **Synchronisation des accès dynamiques :**
Certaines actions côté client (JS) nécessitent de connaître l'état de connexion. Bien que check_session.php fournisse une interface Ajax, ce modèle pourrait être étendu pour exposer plus d'informations côté client de manière sécurisée, par exemple via des API REST protégées.

Gestion des utilisateurs :

a – Analyse

Objectif fonctionnel :
Permettre à l'administrateur du site de :

- Visualiser la liste complète des utilisateurs inscrits.
- Consulter les rôles actuels attribués à chaque utilisateur.
- Ajouter des rôles spécifiques (par exemple, transformer un utilisateur avec le rôle « askchef » en « chef », ou « asktraducteur » en « traducteur »).
- Supprimer les rôles de demande une fois le rôle validé.
- Assurer la persistance de toutes les modifications dans le fichier users.json.

Contraintes :

- Accès restreint au panneau d'administration : seul un utilisateur avec le rôle admin peut y accéder.
- Vérification sécurisée de la session avant affichage ou traitement.
- Préservation de l'intégrité des rôles (pas de doublons, suppression automatique des préfixes ask).

Données manipulées :

- Fichier users.json :
  Chaîne de caractères (exemple JSON utilisateur) :

{

"username": "exemple",

"role": ["cuisinier", "askchef"]

}

- Variable de session $_SESSION['username'] et $_SESSION['role'] pour valider l'accès.

Comportement attendu côté administrateur :

- Visualisation sous forme de tableau : liste des utilisateurs, liste des rôles actuels.
- Sélection d'un utilisateur et ajout d'un nouveau rôle via un formulaire intégré.
- Rafraîchissement automatique de la page après modification pour afficher les mises à jour.

b – Conception

Architecture logicielle :

- admin_panel.php :
    - Vérifie l'existence d'une session active et la présence du rôle admin (sécurité d'accès).
    - Charge les données utilisateurs à partir du fichier users.json.
    - Si un formulaire POST est soumis :
      → Identifie l'utilisateur ciblé ;
      → Ajoute le rôle sélectionné s'il n'est pas déjà présent ;
      → Supprime le rôle temporaire correspondant (askchef ou asktraducteur), s'il existe ;
      → Réécrit le fichier JSON avec les données mises à jour.

Algorithmes et logiques clés :

Chaîne pour éviter les doublons dans les rôles :

$user['role'][] = $newRole;

$user['role'] = array_unique($user['role']);

Chaîne pour nettoyage des anciens rôles temporaires :

if (in_array('ask' . $newRole, $user['role'])) {

$user['role'] = array_diff($user['role'], ['ask' . $newRole]);

}

Chaîne pour redirection après mise à jour réussie :

header("Location: admin_panel.php");

exit;

Chaîne pour sécurisation renforcée (perspective d'amélioration) :

$users = json_decode(file_get_contents('json/users.json'), true);

foreach ($users as $user) {

```
  if ($user['username'] === $_SESSION['username'] && in_array('admin', $user['role'])) {
    $isAdmin = true;
    break;
  }
}
if (!$isAdmin) {
  header("Location: main.php");
  exit;
}
```

Schéma de flux :

1. L'administrateur ouvre admin_panel.php →
2. Le serveur charge et affiche la liste des utilisateurs et leurs rôles →
3. L'administrateur sélectionne un utilisateur et soumet un rôle supplémentaire →
4. Le serveur traite la demande, met à jour les rôles, sauvegarde dans le JSON →
5. La page est rechargée et reflète les nouveaux rôles.

Cas limites gérés :

- Tentative d'accès sans droit admin → redirection vers main.php.
- Ajout d'un rôle déjà présent → pas de duplication (grâce à array_unique).
- Demande préexistante (askchef, asktraducteur) supprimée proprement lors de l'élévation de rôle.

B – Difficultés rencontrées

1 Sécurité d'accès :
La vérification actuelle repose uniquement sur les variables de session. Si un utilisateur malveillant manipule ses données de session côté client (par ex. via console), il pourrait obtenir l'accès admin. Une solution robuste nécessiterait :

- Une vérification systématique côté serveur (lecture de users.json pour confirmer que l'utilisateur connecté est bien admin).
- L'introduction d'un système de permissions centralisé, plutôt qu'une simple variable de session.

2 Gestion concurrente des modifications :
Deux administrateurs modifiant simultanément users.json peuvent provoquer des conflits ou des écrasements. Sans mécanisme de verrou (par ex. flock en PHP), le fichier reste vulnérable aux écritures concurrentes.

3 Expérience utilisateur :
Le rafraîchissement complet de la page après chaque modification casse la fluidité de l'interface. Un système plus moderne, basé sur Ajax, permettrait de modifier les rôles dynamiquement sans recharger l'ensemble de la page.

Gestion des recettes :

a – Analyse

Objectif fonctionnel :
Permettre aux utilisateurs d'interagir avec les recettes de cuisine en offrant :

- Un affichage filtrable et triable des recettes ;
- Une visualisation détaillée des informations (ingrédients, instructions, allergènes, auteur) ;
- La possibilité d'ajouter des commentaires (avec ou sans image) ;
- Un système d'ajout et de suppression de « like » sur les recettes ;
- Pour les chefs, la possibilité d'ajouter de nouvelles recettes ;
- Pour les administrateurs, un module de validation des recettes (statut validé/non validé).

Contraintes :

- Les utilisateurs non connectés ne peuvent pas commenter ni liker.
- Les filtres (sans gluten, vegan, végétarien, sans lait) doivent être combinables.
- Les recherches doivent être multilingues (selon la langue sélectionnée dans la session).
- La cohérence des champs traduits (français/anglais) doit être assurée.

Données manipulées :

- Fichier recipes.json (format JSON) :
  Chaîne exemple :


```
{
  "nameFR": "Tarte aux pommes",
  "name": "Apple Pie",
  "Author": "chef123",
  "ingredients": [...],
  "instructions": [...],
  "imageURL": "url_ou_path_image",
  "Without": ["NoGluten", "Vegan"],
  "validated": false,
  "likers": ["user1", "user2"]
}
```

- Fichier posts.json : stocke les commentaires liés aux recettes.

b – Conception

Architecture logicielle :

- **Dans main.php** :
    - Charge et filtre les recettes selon les critères cochés par l'utilisateur :
      Chaîne (filtrage) :

```
$filteredRecipes = array_filter($recipes, function ($recipe) {
  return !in_array("NoGluten", $recipe['Without']);
});
```

- Permet la recherche textuelle :

Chaîne (recherche) :

CopierModifier

```
$filtered = array_filter($recipes, function ($recipe) use ($query) {
  $name = ($_SESSION['lang'] == 'fr') ? strtolower($recipe['nameFR']) : strtolower($recipe['name']);
  return str_contains($name, $query);
});
```

- **details.php** :
    - Charge les informations détaillées d'une recette à partir de son ID.
    - Permet l'ajout d'un commentaire (avec ou sans image) :
      Chaîne (ajout d'un commentaire) :

```
$newComment = [
  'id' => uniqid('comment_'),
  'recipeId' => $recipe['nameFR'],
  'user' => $currentUser,
  'text' => htmlspecialchars($_POST['comment']),
  'image' => $_FILES['commentImage']['name'] ?? null,
  'timestamp' => time()
];
```

- **like.js** :
    - Met à jour dynamiquement l'apparence des boutons de like et communique avec le serveur :
      Chaîne (mise à jour) :

```
updateLikeButton(button, data.action === 'like', data.likes);
```

- **valider_recettes.php** :
    - Permet aux administrateurs et chefs de valider les recettes en changeant leur statut : Chaîne (validation) :

```
if ($recipe['nameFR'] === $recipeName) {
  $recipe['validated'] = true;
}
```

Schéma de flux :

1. L'utilisateur arrive sur la page principale et sélectionne des filtres.
2. Les recettes sont affichées dynamiquement selon les critères.
3. Un utilisateur connecté peut aimer une recette ou laisser un commentaire.
4. Un administrateur peut accéder à l'interface de validation pour officialiser une recette.

Cas limites gérés :

- Une recette sans image → affichage d'un message « Image non disponible ».
- Un utilisateur non connecté → blocage des actions sensibles (ajout, like, commentaire).
- Une recette non trouvée (détail) → message d'erreur « Recette non trouvée ».
- Une validation sur une recette déjà validée → évitée grâce au filtre préalable.

B – Difficultés rencontrées

1 **Synchronisation des filtres et des recherches :**
Combiner efficacement les multiples critères (filtres booléens + recherche textuelle) nécessite une logique d'intersection propre pour éviter les chevauchements ou les exclusions involontaires.

2 **Sécurité des commentaires et des images :**
Les fichiers images sont uploadés localement, ce qui introduit des risques (fichiers infectés, surcharge disque). Une amélioration consisterait à limiter les types MIME acceptés et à sécuriser les noms des fichiers uploadés.

3 **Gestion des likes :**
Le script actuel interroge check_session.php et like_service.php pour chaque interaction, ce qui peut générer une surcharge serveur si le nombre de likes devient important. L'usage d'une API RESTful avec optimisation côté client (par ex. debounce) améliorerait la fluidité.

4 **Validation des recettes :**
Le mécanisme simple (changement de statut binaire validated) fonctionne, mais ne permet pas de suivre un historique des validations ni de détecter les erreurs passées. Une extension pourrait intégrer un journal d'audit ou une liste de vérifications.

Traduction des recettes :

a – Analyse

Objectif fonctionnel :
Permettre aux traducteurs (ainsi qu'aux chefs et administrateurs) de :

- Compléter les champs de traduction manquants dans les recettes (titre, ingrédients, étapes) ;
- Garantir la symétrie des champs : même nombre d'ingrédients et d'étapes en français et en anglais ;
- Modifier uniquement les champs autorisés (selon leur rôle) et uniquement si les conditions sont remplies (par ex. champ vide dans une langue mais rempli dans l'autre).

Contraintes :

- L'accès à la page est strictement réservé aux utilisateurs connectés avec les rôles **traducteur**, **chef** ou **admin**.
- La structure de données des recettes doit être préalablement vérifiée : correspondance du nombre d'éléments entre les colonnes.
- Les actions de traduction doivent être enregistrées dans le fichier recipes.json de manière atomique (pour éviter les pertes de données).

Données manipulées :

- Fichier recipes.json :
  Chaîne exemple :

```
{
  "nameFR": "Tarte aux pommes",
  "nameEN": "",
  "ingredientsFR": [
    { "name": "Pommes", "quantity": "3", "type": "fruit" }
  ],
  "ingredientsEN": [ "" ],
  "stepsFR": [ "Éplucher les pommes." ],
  "stepsEN": [ "" ]
}
```

b – Conception

Architecture logicielle :

- **traduire_recette.php** :
  - Vérifie les droits d'accès via session :
    Chaîne (contrôle d'accès) :

if (!in_array('traducteur', $roles) && !in_array('admin', $roles) && !in_array('chef', $roles)) {

header("Location: main.php");

exit;

}

Présente une interface en double colonne (français / anglais) où :

- Les champs non traduits sont affichés sous forme de champs éditables (input, textarea) ;

- Les champs déjà remplis sont affichés en lecture seule, sauf pour l'auteur (chef) ou l'admin.

Garantit que les listes d'ingrédients et d'étapes soient équilibrées en longueur :

Chaîne (ajustement des tailles) :

$ingredientsEN = array_pad($ingredientsEN, count($recipe['ingredientsFR']), '');

$stepsEN = array_pad($stepsEN, count($recipe['stepsFR']), '');

Enregistre les modifications à la soumission :

Chaîne (écriture JSON) :

file_put_contents($recipesFile, json_encode($recipes, JSON_PRETTY_PRINT | JSON_UNESCAPED_UNICODE));

Schéma de flux :

1. L'utilisateur ayant le rôle Traducteur choisit d'accèder aux recettes pouvant être traduite;
2. Le serveur charge les recettes depuis recipes.json ;
3. Une interface en tableau présente les champs français et anglais côte à côte ;
4. L'utilisateur remplit les champs manquants et valide le formulaire ;
5. Les traductions sont enregistrées et l'utilisateur est redirigé avec un message de confirmation.

Cas limites gérés :

- Champ déjà traduit → verrouillé sauf si l'utilisateur est auteur (chef) ou admin.
- Recette sans traduction initiale → création des structures vides côté anglais/français.
- Incohérence dans le nombre d'ingrédients ou d'étapes → correction automatique par array_pad.
- Soumission partielle (par ex. uniquement un ingrédient) → intégration propre sans écrasement des autres champs.

B – Difficultés rencontrées

1 **Vérification des droits complexes :**
Les chefs peuvent traduire **leurs** recettes, mais uniquement les champs autorisés. Les traducteurs ne peuvent modifier que les champs vides et correspondants (pas d'accès complet). Cette logique multifiltrée a nécessité une gestion fine pour éviter les fuites de droits.

2 **Maintien de la symétrie des champs :**
Dans le fichier JSON, les listes d'ingrédients et d'étapes doivent rester équilibrées entre les langues. Sans cette vérification, on risque des décalages ou des erreurs d'affichage (par ex. 3 ingrédients FR, mais 5 ingrédients EN). L'ajout d'un array_pad assure cette cohérence.

3 **Écriture concurrente :**
Comme les traductions sont sauvegardées directement dans recipes.json, plusieurs utilisateurs modifiant simultanément différentes recettes pourraient créer des collisions. Une amélioration pourrait être l'implémentation d'un système de verrouillage ou d'un stockage en base de données.

Gestion des favoris utilisateur :

a – Analyse

Objectif fonctionnel :
Permettre aux utilisateurs connectés de :

- Visualiser toutes les recettes qu'ils ont ajoutées à leur liste de favoris (likes) ;
- Retrouver rapidement ces recettes dans une page dédiée, sans avoir besoin de rechercher manuellement dans la liste globale ;
- Accéder directement au détail des recettes à partir de la section favoris.

Contraintes :

- Fonctionnalité uniquement disponible pour les utilisateurs connectés (vérifiée via session).
- Les favoris sont identifiés par correspondance entre les identifiants des recettes (nameFR, name) et les entrées dans la liste likes de l'utilisateur dans users.json.
- Les informations des recettes doivent être mises à jour dynamiquement (compteur de likes, statut d'image).

Données manipulées :

- Fichier users.json :
  Chaîne exemple (extrait utilisateur) :

```
{
  "username": "exampleUser",
  "likes": ["Tarte aux pommes", "Apple Pie"]
}
```

- Fichier recipes.json :
  Chaîne exemple (extrait recette) :

```
{
```

```
  "nameFR": "Tarte aux pommes",
  "name": "Apple Pie",
  "likers": ["exampleUser", "anotherUser"],
  "imageURL": "path_or_url"
}
```

b – Conception

Architecture logicielle :

- **favoris.php** :
    - Vérifie si l'utilisateur est connecté, sinon redirige vers la page de connexion :
      Chaîne (contrôle d'accès) :

```
if (!isset($_SESSION["username"])) {
  header("Location: create_login.php");
  exit();
}
```

Récupère la liste des recettes aimées par l'utilisateur :

Chaîne (filtrage des favoris) :

```
foreach ($recipes as $recipe) {
  if (in_array($recipe['name'], $userLikes) || in_array($recipe['nameFR'], $userLikes)) {
    $likedRecipes[] = $recipe;
  }
}
```

- Affiche dynamiquement la liste des recettes favorites : image, auteur, compteur de likes, bouton de navigation vers les détails.

- Gère l'interface multilingue en fonction de la session.

Schéma de flux :

1. L'utilisateur clique sur l'onglet « Favoris » ;
2. Le serveur vérifie la session et charge la liste des recettes aimées ;
3. Les recettes sont affichées sous forme de cartes interactives (avec compteur et lien détaillé) ;
4. L'utilisateur peut cliquer sur une recette pour accéder à ses détails ou retirer son like (fonctionnalité complémentaire à ajouter).

Cas limites gérés :

- Utilisateur non connecté → redirection forcée vers la page de connexion.

- Aucune recette aimée → message explicite « Aucune recette n'est ajoutée à vos favoris. »
- Recette sans image → affichage d'un bloc texte « Image non disponible ».
- Compteur de likes manquant → valeur par défaut (0) affichée.

B – Difficultés rencontrées

1 **Synchronisation des données utilisateurs-recettes :**
La cohérence entre users.json (qui stocke les noms) et recipes.json (qui stocke les likers) doit être maintenue. Une désynchronisation (par ex. une recette supprimée mais encore listée comme aimée) pourrait entraîner des erreurs d'affichage ou des comportements inattendus.

2 **Performance sur de grandes listes :**
Le filtrage des favoris se fait en PHP côté serveur en parcourant toutes les recettes à chaque chargement. Pour améliorer les performances, un système de requêtes ciblées ou une base de données indexée serait préférable.

3 **Gestion multilingue :**
L'identification des recettes dans les favoris repose sur les champs name (anglais) et nameFR (français). Une amélioration pourrait consister à utiliser un identifiant unique indépendant de la langue pour éviter les collisions ou confusions.

Profil et compte utilisateur :

a – Analyse

Objectif fonctionnel :
Offrir à l'utilisateur :

- Une section *profil* (profil.php) pour visualiser ses activités : recettes likées, commentées, créées ;
- Une section *compte* (account.php) pour gérer ses informations personnelles : prénom, nom, email, statut (rôles), et faire des demandes d'évolution de rôle (chef, traducteur).

Contraintes :

- Accès strictement réservé aux utilisateurs connectés (contrôle session).
- Visualisation filtrée selon le rôle de l'utilisateur (chef, cuisinier, traducteur).
- Possibilité de modifier certaines informations (email) et d'émettre des requêtes d'évolution de rôle, sans affecter directement la base des rôles actifs.

Données manipulées :

- Fichier users.json :
  Chaîne exemple :

```
{
  "username": "jdoe",
```

"nom": "Doe",

"prenom": "John",

"mail": "john@example.com",

"role": ["cuisinier"],

"likes": ["Apple Pie"],

"posts": ["comment_123"]

}

- Fichier recipes.json :
  Liste des recettes permettant de retrouver les recettes créées par un chef ou celles aimées/commentées.
- Fichier posts.json :
  Liste des commentaires, filtrés par identifiant utilisateur.

b – Conception

Architecture logicielle :

- **profil.php** :
  - Vérifie la connexion ;
  - Charge et filtre :
    - Les recettes likées (pour les cuisiniers) ;
    - Les recettes commentées, regroupées par recette avec date et texte ;
    - Les recettes créées (pour les chefs).
  - Génère dynamiquement des cartes recettes réutilisables :
    Chaîne (fonction) :

```
function renderRecipeCard($recipe) {

  $name = $_SESSION['lang'] == 'fr' ? htmlspecialchars($recipe['nameFR']) : htmlspecialchars($recipe['name']);

  $image = !empty($recipe['imageURL']) ? '<img src="' . htmlspecialchars($recipe['imageURL']) . '" alt="' . $name . '">' : '<div class="no-image">Image non disponible</div>';

  echo "<div class='recipe-card'>$image<h3>$name</h3></div>";

}
```

- **account.php** :
  - Vérifie la connexion ;
  - Charge et affiche les informations personnelles et les rôles ;
  - Permet :

- La mise à jour de l'adresse email, avec validation :
  Chaîne (validation email) :

```
if (filter_var($newEmail, FILTER_VALIDATE_EMAIL)) {
  $currentUser["email"] = $newEmail;
  file_put_contents($usersFile, json_encode($users, JSON_PRETTY_PRINT));
}
```

La demande de rôle supplémentaire, ajoutée sous forme de `ask<role>` (par ex. `askchef`) :

Chaîne (ajout demande de rôle) :

```
if (!in_array($requestedRole, $currentUser['role']) && !in_array('ask' . $requestedRole, $currentUser['role'])) {
  $currentUser['role'][] = 'ask' . $requestedRole;
}
```

Schéma de flux :

1. L'utilisateur se connecte et accède à son profil ;
2. Il consulte ses activités passées (likes, posts, créations) ;
3. Il passe à la section *compte* pour modifier ses données ou demander une évolution de rôle ;
4. Les modifications sont enregistrées dans le fichier JSON sans redémarrage serveur.

Cas limites gérés :

- Utilisateur non trouvé → message d'erreur explicite ;
- Double demande de rôle évitée (vérification si déjà présent ou demandé) ;
- Email invalide rejeté avec message d'erreur spécifique.

B – Difficultés rencontrées

1 **Gestion des références croisées (likes, posts, recettes) :**
Il a fallu maintenir la cohérence entre les identifiants des recettes et ceux enregistrés dans les listes likes et posts des utilisateurs, en tenant compte de la langue sélectionnée.

2 **Sécurité des modifications utilisateur :**
Les actions de changement d'email et de demande de rôle ont nécessité des contrôles supplémentaires pour éviter les duplications ou les injections malveillantes.

3 **Conception multilingue uniforme :**
Chaque affichage et libellé de bouton a dû être pensé en double version (français/anglais), ce qui a augmenté le volume du code HTML et des formulaires.

4 **Gestion des fichiers volumineux :**
Avec des fichiers JSON croissants, le traitement direct côté serveur (sans base de données) devient moins performant, notamment pour les boucles de recherche sur les identifiants utilisateurs et recettes.

Conclusion

Le projet **RE7** a permis de créer une plateforme web conviviale et interactive pour partager, commenter et traduire des recettes de cuisine. En combinant plusieurs profils d'utilisateurs (cuisiniers, chefs, traducteurs, administrateurs), nous avons conçu un espace riche où chacun peut participer à enrichir le contenu.

Grâce à un système de gestion des rôles, chacun a des droits spécifiques : les utilisateurs accèdent aux panels de recette, peuvent interagir avec les utilisateurs sous des recettes notamment avec des images ,les chefs créent des recettes, les traducteurs assurent les versions bilingues, et les administrateurs veillent à la qualité et à la cohérence des données. Le site offre aussi des fonctions modernes comme le filtrage des recettes, les favoris, les likes, les commentaires avec images, et une interface multilingue accessible à tous.

Au-delà de la programmation pure, ce projet nous a confrontés à des questions importantes : comment garantir la sécurité des utilisateurs, comment gérer les accès concurrentiels aux données, comment maintenir une interface simple malgré la complexité des rôles et des interactions.

En résumé, **RE7** est un projet complet, qui va bien au-delà d'un simple exercice scolaire : c'est une application pensée pour être utile, agréable à utiliser et techniquement solide, avec encore plein de pistes d'amélioration pour l'avenir.

À terme, ce projet pourrait servir de base pour une future application mobile, une extension communautaire internationale ou même une API ouverte pour enrichir d'autres services culinaires en ligne.