Rapport de Projet – SaveurHub

Équipe:

- **Backend:** Souleimane El Qodsi
- **Frontend:** Mohammed Zeroual

Table des Matières

1. Introduction

Ce rapport détaille la conception et la réalisation du projet "SaveurHub", une application web visant à gérer des recettes de cuisine en français et en anglais.

Le but était de créer une plateforme où différents utilisateurs (Cuisiniers, Chefs, Traducteurs, Administrateur) peuvent interagir avec des recettes. Il est intéressant de noter que le projet a connu une évolution significative. Une première version, plus basique et qualifiable de "spaghetti code" (accessible à titre d'archive ici : https://midou.alwaysdata.net/), a été entièrement refactorisée pour aboutir à l'architecture actuelle basée sur une API RESTful en PHP et offrant une meilleure structure et maintenabilité.

Nous avons utilisé GitHub pour la gestion de version et la collaboration tout au long du développement : https://github.com/souleimaneelqodsi/recipes. Nous avons travaillé en intégration continue CI avec l'usage des branches et des pull requests.

Ce document présente l'architecture serveur finale, le modèle de données, l'API, les interactions client-serveur, l'interface utilisateur telle que prévue, et les aspects liés à la sécurité et au développement.

1bis. Diagrammes

Diagramme de composants réalisé

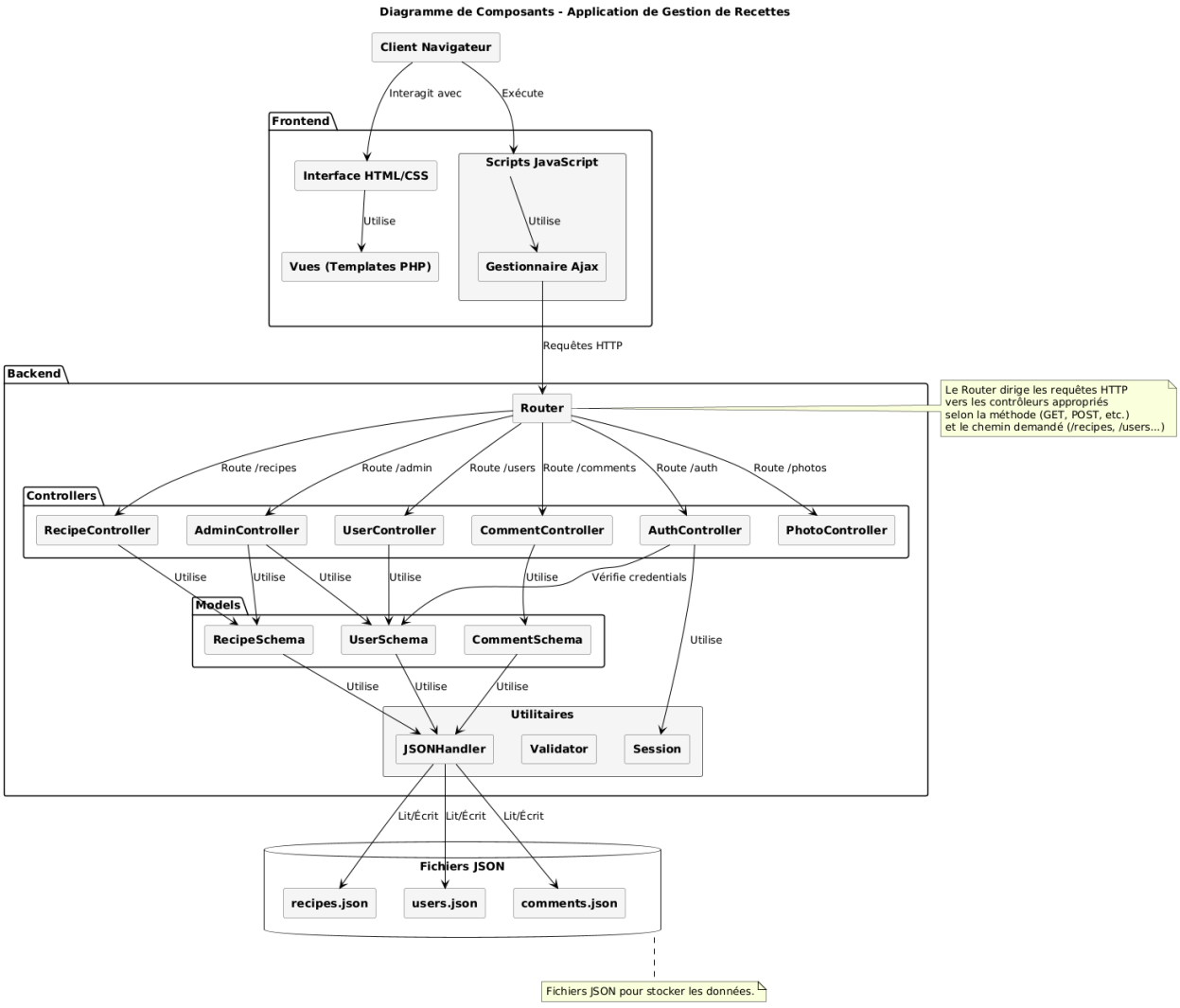
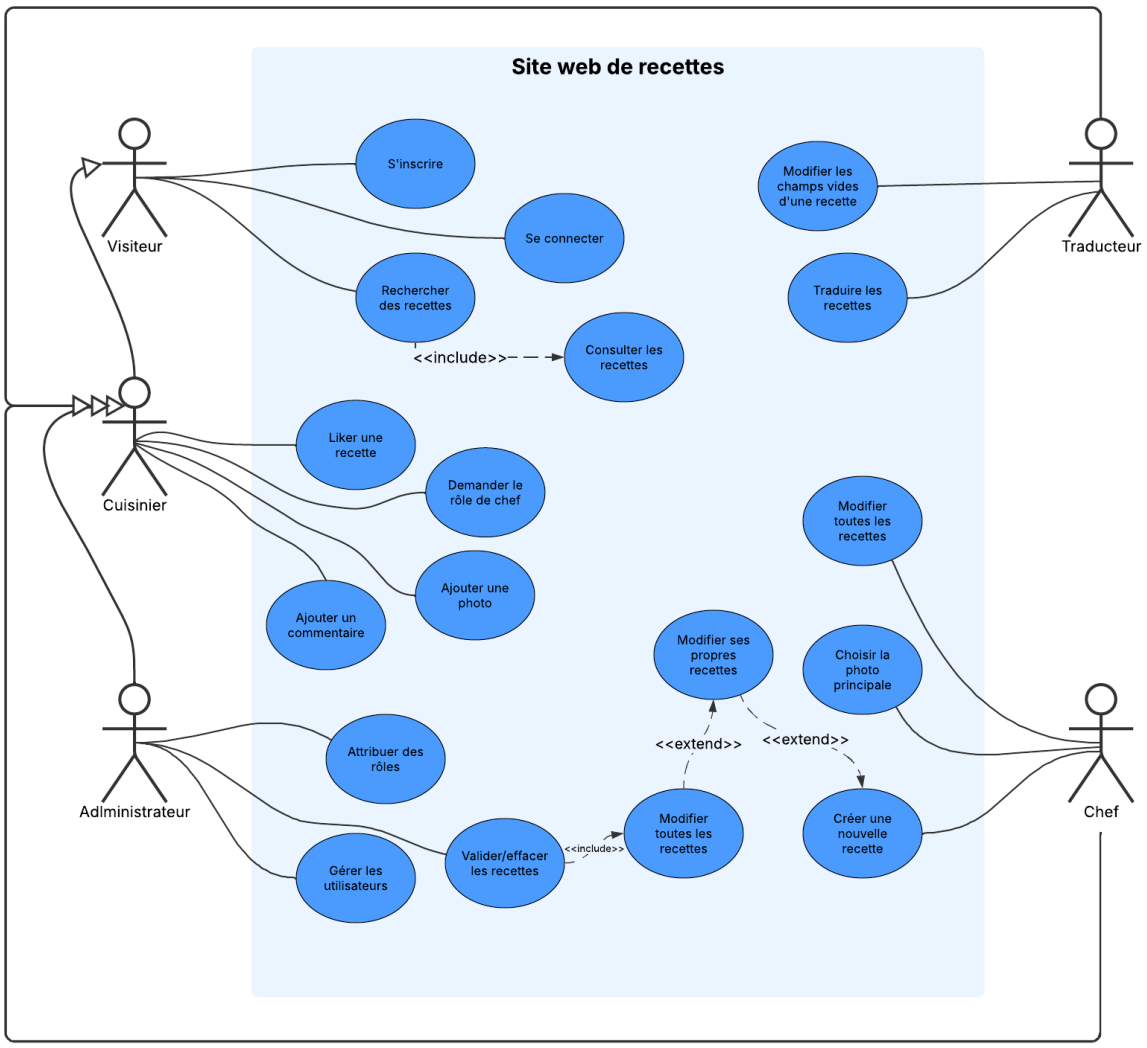


Diagramme de cas d'utilisation réalisé



2. Architecture côté serveur

L'architecture backend adopte une approche Modèle-Contrôleur en PHP (la Vue est en quelque sorte le frontend). Elle sépare la logique métier (Models/Schemas), la gestion des requêtes (Controllers), le routage (Router), et les utilitaires (dossier utils), avec les données stockées dans des fichiers JSON.

- **Router (api/router.php):** Interprète l'URI et la méthode HTTP pour appeler le bon contrôleur. Il gère les paramètres de chemin (ex: /recipes/{id}) et les query strings (ex: ?search=...).
- **Controllers (api/controllers/):** Orchestrent le traitement des requêtes, font le lien

entre le routeur et les modèles.

- **Models (Schemas) (api/models/):** Gèrent la logique des données (validation via Validator, lecture/écriture via JSONHandler).
- **Utils (api/utils/):** Fournissent des fonctionnalités de support (sessions, validation, JSON, UUID, erreurs).
- **Data (api/data/):** Stockage des données (users.json, recipes.json).

2.1. Composants serveur (Controllers)

- **AuthController (api/controllers/auth_controller.php)**: Gère inscription (register), connexion (login), déconnexion (logout).
  - register(): Valide entrées, vérifie doublons, hash mot de passe, crée user (UserSchema::create), ouvre session. Attend username, email, password (corps JSON).
  - login(): Vérifie credentials (UserSchema::getByUsername, AuthSchema::login), ouvre session. Attend username, password (corps JSON).
  - logout(): Détruit session (Session::destroy).
- **RecipeController (api/controllers/recipe_controller.php)**: CRUD et actions sur les recettes.
  - getAll(), getPublished(), getDrafts(): Listent recettes (avec contrôle d'accès Admin pour getAll/getDrafts).
  - getById(string $id): Récupère une recette.
  - search(): Recherche textuelle via RecipeSchema::search.
  - create(): Crée recette (Chef/Admin requis). Attend données recette (corps JSON).
  - update(): Met à jour recette (Auteur(Chef)/Admin requis). Attend ID et données partielles (corps JSON).
  - delete(string $recipe_id): Supprime recette (Admin requis).
  - like(string $recipe_id), unlike(string $recipe_id): Gère les likes (Login requis).
  - translate(string $recipe_id, array $translation): Traduit recette (Traducteur requis). Attend ID et traduction (corps JSON).
  - setPhoto(string $recipe_id, string $photo_id): Définit photo principale (Auteur/Admin requis). Attend ID recette et ID photo (corps JSON).
  - publish(string $recipe_id): Publie recette (Admin requis).
- **UserController (api/controllers/user_controller.php)**: Gestion utilisateurs.
  - getAll(), getById(string $user_id): Listent/récupèrent utilisateurs.
  - updateRole(string $user_id, string $role): Modifie rôle (Admin requis). Attend ID et rôle (corps JSON).

    - askRole(string $requested_role): Demande de rôle (Login requis). Attend ID (implicite via session) et rôle demandé (corps JSON).
- **CommentController (api/controllers/comment_controller.php)**: Ajout commentaires.
    - create(string $recipe_id): Ajoute commentaire (Login requis). Attend ID recette et contenu (corps JSON).
- **PhotoController (api/controllers/photo_controller.php)**: Ajout photos (via URL).
    - upload(string $recipe_id): Ajoute photo (Auteur/Chef/Admin requis). Attend ID recette et URL (corps JSON).

3. Description des routes API

Voici l'ensemble des endpoints de notre API RESTful:

- **Authentification (/auth)**
    - POST /register: AuthController::register()
    - POST /login: AuthController::login()
    - POST /logout: AuthController::logout()
- **Utilisateurs (/users)**
    - GET /: UserController::getAll()
    - GET /{userId}: UserController::getById()
    - PATCH /{userId}/role: UserController::updateRole() (Corps: {"role": "..."})
    - PATCH /{userId}/askrole: UserController::askRole() (Corps: {"requested_role": "..."})
- **Recettes (/recipes)**
    - GET /: Si ?search=... dans l'URI -> RecipeController::search(), sinon RecipeController::getAll() (Admin requis)
    - GET /published: RecipeController::getPublished()
    - GET /drafts: RecipeController::getDrafts() (Admin requis)
    - GET /{recipeId}: RecipeController::getById()
    - POST /: RecipeController::create() (Chef/Admin requis)
    - PATCH /{recipeId}: RecipeController::update() (Auteur(Chef)/Admin requis)
    - DELETE /{recipeId}: RecipeController::delete() (Admin requis)
    - POST /{recipeId}/like: RecipeController::like() (Login requis)
    - DELETE /{recipeId}/like: RecipeController::unlike() (Login requis)
    - PATCH /{recipeId}/translate: RecipeController::translate() (Traducteur requis)
    - PATCH /{recipeId}/photos: RecipeController::setPhoto() (Auteur/Admin requis, Corps: {"photo_id": "..."})
    - PATCH /{recipeId}/validate: RecipeController::publish() (Admin requis)
- **Commentaires (/recipes/{recipeId}/comments)**

    - POST /: CommentController::create() (Login requis, Corps: {"content": "..."})
- **Photos (/recipes/{recipeId}/photos)**
    - POST /: PhotoController::upload() (Auteur/Chef/Admin requis, Corps: {"url": "..."})

4. Modèle de données

Stockage dans users.json et recipes.json.

- **users.json:** Tableau d'objets utilisateur (id, username, email, password (hash), roles (array), created_at, recipes (simplifié), comments (simplifié), photos (simplifié), likes (array d'IDs recette)). Structure définie par UserSchema et Validator.
- **recipes.json:** Tableau d'objets recette (id, name, nameFR, Author, Without, ingredients/FR, steps/FR, timers, imageURL, originalURL, likes, status, comments (complet), photos (complet), total_time, created_at). Structure définie par RecipeSchema et Validator.
- **comments.json:** le code inclut également un fichier comments.json mais on l'a pas utilisé finalement
- On peut également noter qu'il y'avait des fichiers contenant les stop words ayant permis de filtrer les tokens de recherche pour la recherche de recettes.

5. Interactions HTTP frontend/backend

Le frontend (script.js) communique avec l'API via fetch.

- **Exemples clés:**
    - GET /recipes/api/recipes/published: Pour index.html.
    - GET /recipes/api/recipes/{id}: Pour recette.html.
    - GET /recipes/api/recipes?search=...: Pour la recherche.
    - POST /recipes/api/auth/register: Pour inscription.html.
    - POST /recipes/api/auth/login: Pour connexion.html.
    - POST /recipes/api/auth/logout: Déclenchée depuis le menu.
- **Réponses:** Codes HTTP (200, 201, 204, 400, 401, 403, 404, 409, 500) et corps JSON (données ou objet d'erreur {"error": "..."}) sont gérés par les contrôleurs et interprétés par le JS frontend.

6. Description des pages et formulaires frontend

- **Pages:** index.html (accueil/liste), connexion.html (login), inscription.html (register), recette.html (détail), rechercher_recette.html (résultats recherche), profil.html (infos user), deconnexion.html (page transitoire pour logout).
- **Formulaires:**

    - Connexion (connexion.html): username, password (requis).
    - Inscription (inscription.html): username, email, password, confirm_password (tous requis, validation correspondance mdp en JS).
- **Design:** Le frontend utilise Bootstrap et un style personnalisé (style/css/style.css), visant un design moderne et épuré.
- **Appels au backend:** Il faut également noter que l'on a utilisé fetch pour sa simplicité

7. Sécurité

Les mesures implémentées incluent (surtout au backend) :

- Hachage des mots de passe (password_hash/verify).
- Sessions PHP sécurisées (HttpOnly, use_only_cookies, régénération d'ID).
- Validation rigoureuse des entrées côté serveur (Validator.php).
- Nettoyage partiel des sorties (commentaires, username avec strip_tags, htmlspecialchars) pour prévenir les attaques de type XSS.
- Contrôle d'accès basé sur les rôles (Session::hasRole, Session::getRole, …etc.).
- Gestion centralisée des erreurs pour éviter fuites d'infos.
- Utilisation de flock pour l'accès aux fichiers JSON (JSONHandler) (il y'avait également l'écriture atomique càd écrire dans un fichier .temp et ensuite le renommer en le fichier voulu, mais on l'a retiré au moment où on a fait face au bug de file_put_contents()).

8. Difficultés rencontrées

- **Refactoring initial:** Le projet a débuté avec une structure moins organisée ("spaghetti code", visible sur https://midou.alwaysdata.net/ et sur les premiers commits dans le github). Une refactorisation majeure a été entreprise pour adopter une architecture API RESTful plus propre et maintenable, ce qui a représenté un effort initial important mais bénéfique.
- **Gestion du temps:** Une difficulté majeure a été la gestion du temps imparti. Bien que le backend soit fonctionnel et complet par rapport aux exigences, nous avons manqué de temps pour finaliser entièrement le frontend et intégrer toutes les fonctionnalités de l'API de manière fluide.
- **Débogage frontend:** Le développement frontend a présenté ses propres défis, notamment des erreurs JavaScript difficiles à tracer (souvent à cause d'un mauvais call au backend) qui ont consommé beaucoup de temps. Le design était pourtant bien avancé et esthétique.
- **Permissions d'écriture fichier:** Nous avons rencontré une difficulté

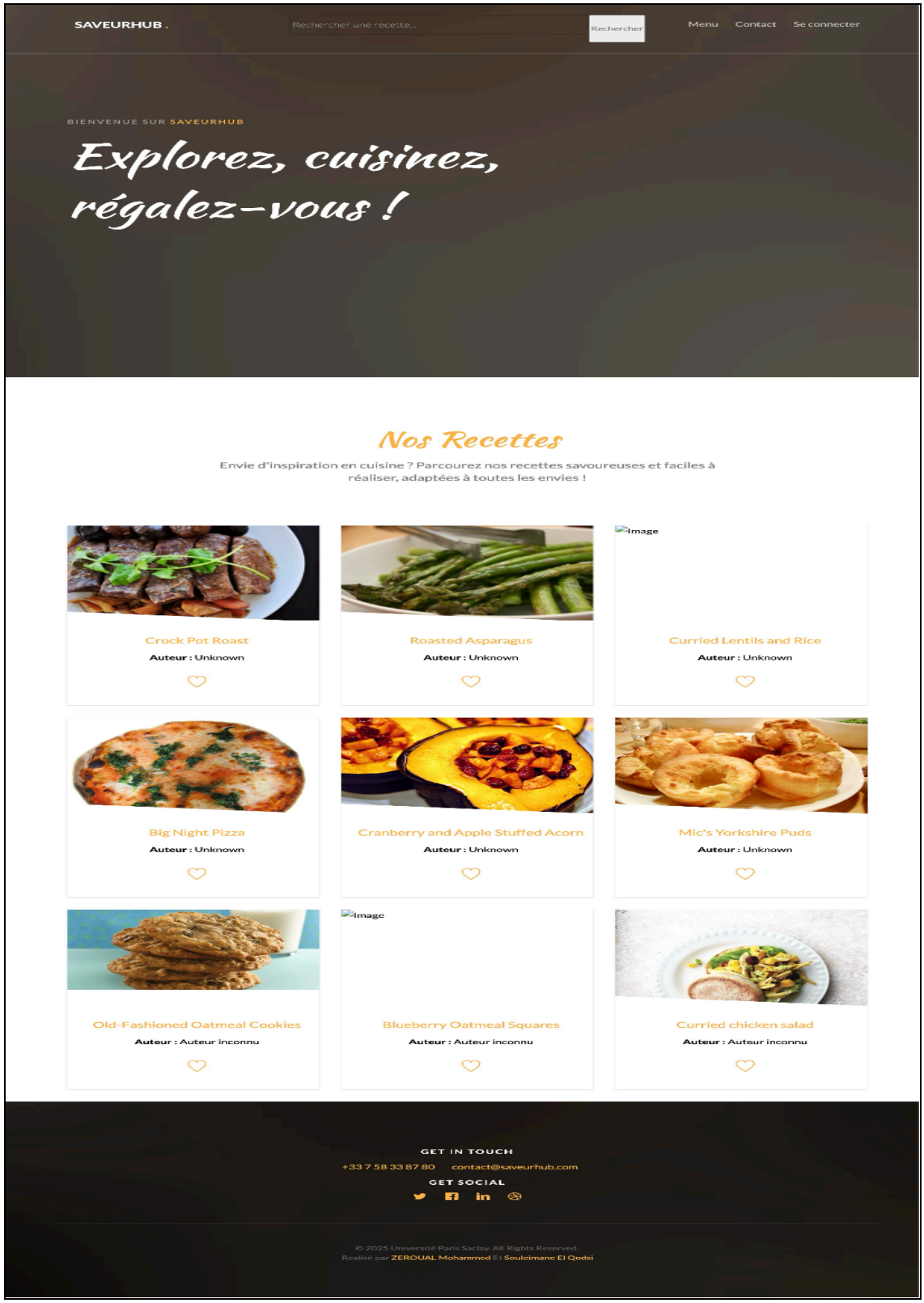
particulièrement énervante liée aux permissions d'écriture sur le serveur lors de la manipulation des fichiers JSON avec file_put_contents(). Obtenir la bonne combinaison de code et de gestion des verrous (+ aussi permissions de fichier sur windows et config XAMPP/Apache Web Server) a nécessité plusieurs essais avant de trouver une solution fiable en utilisant fopen, fwrite et flock dans notre JSONHandler.

- **Version de présentation (IA):** Face aux contraintes de temps sur le frontend, et pour pouvoir illustrer le fonctionnement potentiel avec le backend terminé, une branche séparée a été créée sur GitHub (cf. lien du repo) où une assistance IA a été utilisée pour générer rapidement une base de code frontend fonctionnelle. Cette version était destinée uniquement à la démonstration et est distincte du travail principal effectué manuellement.

9. Bilan et perspectives

- **Bilan:** Le backend PHP RESTful est robuste, sécurisé et implémente l'ensemble des fonctionnalités métier demandées. Le refactoring initial a posé des bases solides. Le frontend bien qu'ayant un design soigné, souffre d'une intégration incomplète due aux contraintes de temps et aux difficultés techniques rencontrées qui furent très chronophages. La collaboration via GitHub a été un point positif qui nous a permis de vraiment séparer le travail sur la partie frontale et la logique métier sans “casser le code”.
- **Perspectives:**
  - **Finaliser le frontend:** Priorité absolue pour connecter toutes les fonctionnalités API et corriger les bugs restants.
  - **Base de données:** Migrer du stockage JSON vers une base de données type SQL ou NoSQL, cela permettrait d’éviter des problèmes comme l’autorisation d’écriture dans les fichiers et ça serait plus sécurisé et plus facile à maintenir en cas de changement dans le modèle de données (ex: ajout d’un champs).
  - **Tests automatisés:** Mettre en place des tests unitaires et d'intégration.
  - **Recherche avancée:** Intégrer le filtrage par tags ('Without') et statut dans la recherche API.
  - **Quelques petits problèmes listés dans la section Issues du repo git**: par exemple, ajouter plus de sécurité à l’authentification en stockant sur le serveur un tableau avec le nombre de tentatives de connexions

10. Capture d’écran


SAVEURHUB .
Rechercher une recette...
Rechercher
Menu
Contact
Se connecter
BIENVENUE SUR SAVEURHUB
Explorez, cuisinez, régalez-vous !
Nos Recettes
Envie d'inspiration en cuisine ? Parcourez nos recettes savoureuses et faciles à réaliser, adaptées à toutes les envies !
Crock Pot Roast
Auteur : Unknown
Roasted Asparagus
Auteur : Unknown
Image
Curried Lentils and Rice
Auteur : Unknown
Big Night Pizza
Auteur : Unknown
Cranberry and Apple Stuffed Acorn
Auteur : Unknown
Mic's Yorkshire Puds
Auteur : Unknown
Old-Fashioned Oatmeal Cookies
Auteur : Auteur inconnu
Image
Blueberry Oatmeal Squares
Auteur : Auteur inconnu
Curried chicken salad
Auteur : Auteur inconnu
GET IN TOUCH
+33 7 58 33 87 80
contact@saveurhub.com
GET SOCIAL
© 2025 Université Paris Saclay. All Rights Reserved.
Realisé par ZEROUAL Mohammed Et Souleimane El Qodsi