

EatFit

Rapport TPI et documentation technique

Avril – Mai 2023

Almeida Costa Lucas

Maitre d'apprentissage : Katia Mota Stroppolo

Table des matières

Introduction	4
Rappel de l'énoncé	4
Description de l'application	4
Organisation	5
Livrables	5
Matériel et logiciels à disposition	5
Méthodologie	5
Planification en 6 étapes	5
Planification	7
Planification prévue	7
Planification effective	7
Product backlog	8
Analyse organique	11
Front-end	11
Api (Interface de Programmation d'Applications)	11
Back-end	12
Architecture de l'api	13
Classes	13
Architecture Front	16
Classes	16
Base de données	18
Informations techniques	18
MLD	19
Tables	19
Généralités concernant l'implémentation	23
Langages (versions)	29
Environnement (local et/ou distant)	29
Plan de test	30
Périmètre	30
Environnement	30
Scénarios (les tests, les données et leurs évolution)	30
Evolution des tests	37
Conclusion	38
Difficultés rencontrées	38

Variantes de solutions et choix.....	39
Améliorations possibles	39
Bilan personnel	39
Remerciements.....	39
Bibliographie.....	40
Annexes.....	40
Conversations ChatGPT	40
Graphique en js	40
Glossaire.....	52

Introduction

Ce document est un rapport présentant différents aspects de la conception du projet Eatfit. Ce projet a été réalisé dans le cadre du *Travail pratique individuel* (TPI) durant la session de mai 2023. Il a pour but de valider mes compétences acquises pendant la formation *Informaticien CFC* dispensée à l'école d'informatique du CFPT au Petit-Lancy.

EatFit est un site web de gestion de recettes culinaires permettant aux utilisateurs de rechercher, créer, modifier et partager des recettes en fonction de leur rôle (visiteur non-authentifié, membre authentifié ou administrateur technique). Les utilisateurs peuvent également gérer leur profil, consulter l'historique de leurs recettes consommées et visualiser un graphique montrant la répartition des types d'aliments consommés.

Rappel de l'énoncé

Les informations suivantes sont extraites du cahier de charges du TPI.

Description de l'application

L'application est un site web de gestion de recettes culinaires qui offre aux utilisateurs une plateforme interactive pour explorer, créer et partager des recettes. Elle est conçue pour être utilisée par trois types d'utilisateurs : les visiteurs non-authentifiés, les membres authentifiés et les administrateurs techniques.

Fonctionnalités pour les visiteurs non-authentifiés :

- Rechercher des recettes par nom, catégorie ou date d'ajout.
- Filtrer les recettes par catégorie ou type d'aliments.
- Lire la liste des recettes triées par ordre de publication, du plus récent au plus ancien.
- Sélectionner une recette et afficher les détails, incluant le titre, la catégorie, l'auteur, la durée de préparation, la difficulté, l'évaluation moyenne, les commentaires et les images.
- Lire les informations sur le fonctionnement du site.
- Créer un nouveau compte ou se connecter à un compte existant.

Fonctionnalités pour les membres authentifiés :

- Accéder à tous les droits des visiteurs non-authentifiés.
- Afficher des détails supplémentaires sur les recettes, tels que les instructions de préparation, le nombre de calories et la répartition des types d'aliments.
- Gérer un historique de recettes consommées (ajouter, lire, modifier, supprimer) avec des informations sur la date et le moment de la consommation.
- Afficher un graphique montrant la répartition des types d'aliments consommés sur différentes périodes (1 semaine, 1 mois, 1 an ou toute la période enregistrée).
- Ajouter de nouveaux types d'aliments.
- Gérer ses propres recettes (créer, lire, modifier, supprimer).
- Évaluer et commenter les recettes.
- Gérer son propre profil (lire, modifier, supprimer).

Fonctionnalités pour les administrateurs techniques :

- Gérer la base de données via l'interface d'administration avec un rôle administrateur.
- Maintenir le code source de l'application.

Le site web est structuré de manière logique, avec un découpage en user stories pour faciliter la compréhension et l'organisation du travail. La méthodologie suivie comporte six étapes : s'informer, planifier, décider, réaliser, contrôler et évaluer. Les tests fonctionnels et de sécurité sont réalisés pour s'assurer que les fonctionnalités et les droits des utilisateurs sont correctement implémentés et gérés.

Organisation

Élève	Maîtresse d'apprentissage
Lucas Almeida Costa <lucas.almdc@eduge.ch>	Katia Mota Stroppolo <katia.motastroppolo@edu.ge.ch>

Experts	
Brian Nydegger <briannydegger@gmail.com>	Borys Folomietow <borys@folomietow.ch>

Livrables

- Résumé du TPI sur une page A4
- Rapport de projet (réalisé en continu ; livraison intermédiaire à la fin de la première semaine, livraison finale à la fin du projet)
- Code source de l'application en annexe
- Manuel utilisateur
- Journal de travail (disponible pour consultation à tout moment par les experts et l'enseignante)

Matériel et logiciels à disposition

- Un PC standard école, 2 écrans
- Windows 10
- Visual Studio code
- MySQLWorkbench
- DBeaver
- Suite Office
- Serveur Web, avec PHP

Méthodologie

Planification en 6 étapes

1. S'informer

Lors de la première étape de mon projet, j'ai effectué une analyse approfondie du cahier des charges afin de comprendre toutes les fonctionnalités à implémenter pour l'application de gestion de recettes culinaires. J'ai également consulté diverses ressources en ligne et des forums de discussion pour m'informer sur les meilleures pratiques et les technologies à utiliser. Lorsque nécessaire, j'ai sollicité l'aide de ma formatrice pour obtenir des éclaircissements sur certains aspects du projet.

2. Planifier

J'ai préparé un planning de travail détaillé pour organiser les différentes tâches à réaliser et déterminer les délais associés. J'ai divisé le projet en plusieurs user stories, en tenant compte des différents types d'utilisateurs et des fonctionnalités à implémenter pour chacun

d'eux. Pour chaque user story, j'ai attribué un niveau de priorité, allant de P1 (bloquant) à P4 (secondaire), afin de déterminer l'ordre d'exécution des tâches.

3. Décider

Tout au long du projet, j'ai dû prendre des décisions concernant l'architecture de l'application, les technologies à utiliser et les solutions à adopter pour résoudre des problèmes spécifiques. J'ai consigné ces décisions dans mon journal de bord, en expliquant les raisons qui m'ont conduit à opter pour certaines options plutôt que d'autres.

4. Réaliser

Après avoir pris les décisions appropriées, j'ai procédé à l'implémentation des fonctionnalités dans le code source de l'application. J'ai travaillé de manière itérative, en développant et en testant chaque fonctionnalité individuellement avant de passer à la suivante.

5. Contrôler

À chaque étape de réalisation, j'ai effectué des tests rigoureux pour m'assurer que chaque fonctionnalité fonctionnait correctement et répondait aux exigences du cahier des charges. J'ai documenté ces tests dans mon rapport de projet, en décrivant les scénarios de test et les résultats obtenus. J'ai également réalisé des tests de compatibilité sur différents navigateurs pour m'assurer que l'application fonctionnait correctement.

6. Evaluer

Enfin, j'ai procédé à une évaluation globale du projet, en examinant les réussites et les points à améliorer. J'ai rédigé des bilans journaliers dans mon journal de bord pour analyser les progrès réalisés et identifier les éventuels obstacles rencontrés. Dans la partie Conclusion de mon rapport de projet, j'ai présenté une auto-évaluation complète, en abordant les aspects positifs et négatifs du travail accompli et en proposant des pistes d'amélioration pour l'avenir.

Planification

Planification prévue

		Jour 1	Jour 2	Jour 3	Jour 4	Jour 5	Jour 6	Jour 7	Jour 8	Jour 9	Jour 10	Jour 11
Tâches à réaliser	Temps nécessaire	27 avr 2023	2 mai 2023	3 mai 2023	4 mai 2023	8 mai 2023	9 mai 2023	10 mai 2023	11 mai 2023	15 mai 2023	16 mai 2023	17 mai 2023
S'informer												
Lecture de l'énoncé	04:30	00:30	00:30	00:30	00:30	00:30	00:30	00:30	00:30	00:30		
Recherche web	01:00		01:00									
Planifier												
Réalisation du planning	03:30	03:30										
Rédaction du produit backlog	03:00	03:00										
Mise en place d'un dépôt git	00:10	00:10										
Réalisation des scénarios de tests	01:30	01:30										
Réalisation du projet												
S2 : Installation de la base de données	01:30		01:30									
S3 : Création du style du site	02:30				00:30	00:30	00:30	00:30	00:30			
S4 : Gestion du profil utilisateur	04:00			02:30	01:30							
S5 : Création d'un Virtual Host	00:10		00:10									
S6 : Déconnexion	00:30					00:30						
S7 : Gestion des recettes	04:00					02:00	02:00					
S8 : Affichage des recettes sur la page d'accueil	01:30				01:30							
S9 : Gestion des évaluations des recettes	02:00					02:00						
S10 : Recherche de recettes	01:30								01:30			
S11 : Filtrage des recettes	02:00							02:00				
S12 : Historique des recettes consommées	02:00						02:00					
S13 : Affichage des statistiques	02:00								02:00			
S14 : Ajout de nouveaux types d'aliments	01:00							01:00				
S15 : Sécurité et protection des données	06:00		00:30	00:30	00:30	00:30	00:30	00:30	00:30	02:30		
S16 : Affichage de la page de la recette	00:00											
S17 : Faire la structure de l'API	01:00		01:00									
S18 : Adapter le modèle de l'API à l'application	03:00		01:00	02:00								
Test												
Rapport de test	00:40		00:05	00:05	00:05	00:05	00:05	00:05	00:05	00:05		
Tests en profondeur et corrections de bugs	11:50		01:30	01:30	01:30	01:30	01:30	01:30	01:30	01:20		
Documentation												
Rédaction de la documentation	23:40	01:00	01:00	01:00	02:00	01:00	01:00	02:10	02:00	03:00	04:45	04:45
Tenue du journal de bord	03:30	00:10	00:20	00:20	00:20	00:20	00:20	00:20	00:20	00:20	00:20	00:20
Durée	88:00	09:50	08:35	08:25	08:25	08:55	08:25	08:35	08:55	07:45	05:05	05:05

Planification effective

EatFit

Planning effectif

Almeida Costa Lucas

		Jour 1	Jour 2	Jour 3	Jour 4	Jour 5	Jour 6	Jour 7	Jour 8	Jour 9	Jour 10	Jour 11
Tâches à réaliser	Temps nécessaire	27 avr 2023	2 mai 2023	3 mai 2023	4 mai 2023	8 mai 2023	9 mai 2023	10 mai 2023	11 mai 2023	15 mai 2023	16 mai 2023	17 mai 2023
S5 : Création d'un Virtual Host	00:10		00:10									
S6 : Déconnexion	00:00											
S7 : Gestion des recettes	03:00					02:00			01:00			
S8 : Affichage des recettes sur la page d'accueil	01:30				01:30							
S9 : Gestion des évaluations des recettes	02:00					02:00						
S10 : Recherche de recettes	00:30							00:30				
S11 : Filtrage des recettes	00:30							00:30				
S12 : Historique des recettes consommées	03:00						03:00					
S13 : Affichage des statistiques	01:30								01:30			
S14 : Ajout de nouveaux types d'aliments	02:00							02:00				
S15 : Sécurité et protection des données	04:00		00:30	00:30	00:30					02:30		
S16 : Affichage de la page de la recette	00:00											
S17 : Faire la structure de l'API	01:00		01:00									
S18 : Adapter le modèle de l'API à l'application	02:00		01:00	01:00								
Test												
Rapport de test	00:35		00:05	00:05	00:05	00:05	00:05	00:05	00:05			
Tests en profondeur et corrections de bugs	09:30		01:30	01:30	01:30	01:00	01:00	01:30	01:30			
Documentation												
Rédaction de la documentation	30:10	01:00	01:00	01:00	00:30	02:00	00:40	04:00	02:00	06:00	06:00	06:00
Tenue du journal de bord	02:15	00:10	00:20	00:20	00:20	00:20	00:10	00:10	00:10	00:05	00:05	00:05
Durée	89:10	09:50	08:35	07:25	08:15	08:25	08:25	08:45	08:45	08:35	06:05	06:05

Product backlog

Nom	S1 : Utilisation d'un dépôt Git
Description	En tant que développeur, je dois utiliser un dépôt Git afin d'enregistrer et accéder à toutes les versions de mon projet.
Critère d'acceptation	Un dépôt Git public est créé, et les modifications du projet sont régulièrement poussées sur ce dépôt.
Priorité	P1 : Bloquant

Nom	S2 : Installation de la base de données
Description	En tant que développeur, je dois installer une base de données. Pour ce faire, j'utilise le script SQL mis à disposition par mon formateur.
Critère d'acceptation	La base de données est créée ainsi qu'un utilisateur qui a les droits que sur cette base de données.
Priorité	P1 : Bloquant

Nom	S3 : Création du style du site
Description	En tant que développeur, je dois avoir un site visuel pour me concentrer seulement sur le back-end de celui-ci.
Critère d'acceptation	Les pages mentionnées dans les consignes du TPI sont créées et disposent des éléments requis.
Priorité	P1 : Bloquant

Nom	S4 : Gestion du profil utilisateur
Description	En tant qu'utilisateur identifié, je peux afficher et gérer mon profil (lire, modifier, supprimer).
Critère d'acceptation	Les modifications apportées au profil sont enregistrées et appliquées. Tests :
Priorité	P1 : Bloquant

Nom	S5 : Création d'un Virtual Host
Description	En tant que développeur, je dois créer un Virtual Host qui me permettra d'utiliser un « router ».
Critère d'acceptation	Un Virtual Host est créé, permettant d'utiliser un "router" pour naviguer entre les différentes pages du site. Test :
Priorité	P1 : Bloquant

Nom	S6 : Déconnexion
Description	En tant qu'utilisateur identifié, je peux me déconnecter de mon compte.
Critère d'acceptation	L'utilisateur identifié est déconnecté, et la session est terminée. Test :
Priorité	P3 : Important

Nom	S7 : Gestion des recettes
Description	En tant qu'utilisateur identifié, je peux ajouter, modifier ou supprimer des recettes.
Critère d'acceptation	
Priorité	P1 : Bloquant

Nom	S8 : Affichage des recettes sur la page d'accueil
Description	En tant qu'utilisateur identifié, je dois voir les recettes par catégories (petit déjeuner, entrée, etc...)
Critère d'acceptation	Les recettes sont affichées sur la page d'accueil, classées par catégorie (petit déjeuner, entrée, etc.).
Priorité	P2 : Critique

Nom	S9 : Gestion des évaluations des recettes
Description	En tant qu'utilisateur identifié, je peux évaluer de 1 à 5 et commenter, je peux aussi modifier et supprimer l'évaluation
Critère d'acceptation	
Priorité	P3 : Important

Nom	S10 : Recherche de recettes
Description	En tant qu'utilisateur, je peux rechercher des recettes en fonction de leur nom, catégorie ou date d'ajout.
Critère d'acceptation	La recherche retourne les résultats correspondants aux critères de recherche. Tests :
Priorité	P2 : Critique

Nom	S11 : Filtrage des recettes
Description	En tant qu'utilisateur, je peux filtrer les recettes par catégorie ou type d'aliments.
Critère d'acceptation	Les recettes sont filtrées en fonction des critères sélectionnés. Tests :
Priorité	P2 : Critique

Nom	S12 : Historique des recettes consommées
Description	En tant qu'utilisateur identifié, je peux consulter et gérer l'historique de mes recettes consommées.
Critère d'acceptation	L'historique affiche correctement les recettes consommées et permet leur gestion (ajouter, lire, modifier, supprimer). Tests :
Priorité	P2 : Critique

Nom	S13 : Affichage des statistiques
Description	En tant qu'utilisateur identifié, je peux afficher un graphique basé sur les données de l'historique des recettes consommées, montrant la répartition des proportions de types d'aliments consommés sur différentes périodes.
Critère d'acceptation	Le graphique est généré en fonction des données de l'historique et affiche les répartitions correctes pour les périodes sélectionnées. Tests :
Priorité	P3 : Important

Nom	S14 : Ajout de nouveaux types d'aliments
Description	En tant qu'utilisateur identifié, je peux ajouter de nouveaux types d'aliments.
Critère d'acceptation	Les nouveaux types d'aliments sont ajoutés et peuvent être utilisés pour filtrer les recettes Tests :
Priorité	P3 : Important

Nom	S15 : Sécurité et protection des données
Description	En tant que développeur, je dois m'assurer que le site est sécurisé et protège les données des utilisateurs.
Critère d'acceptation	Les données sont stockées et transmises de manière sécurisée, et les vulnérabilités courantes (injections SQL, XSS, CSRF, etc.) sont traitées. Tests :
Priorité	P4 : Secondaire

Nom	S16 : Affichage de la page de la recette
Description	En tant qu'utilisateur, je veux regarder une recette détaillée

Critère d'acceptation	Tests :
Priorité	P1 : Bloquant

Nom	S17 : Faire la structure de l'API
Description	En tant que développeur, je dois créer la structure de base de l'API pour gérer les requêtes entre l'application et la base de données.
Critère d'acceptation	L'API est correctement structurée avec les routes, les contrôleurs et les modèles nécessaires pour gérer les requêtes. Tests :
Priorité	P1 : Bloquant

Nom	S18 : Adapter le modèle de l'API à l'application
Description	En tant que développeur, je dois adapter le modèle de l'API pour qu'il corresponde aux besoins de l'application et aux relations entre les différentes entités (utilisateurs, recettes, évaluations, etc.).
Critère d'acceptation	Le modèle de l'API est adapté à l'application, et les relations entre les entités sont correctement gérées. Tests :
Priorité	P1 : Bloquant

Analyse organique

L'application EatFit se compose de trois principales composantes : le front-end, l'API et le back-end.

Front-end

Le front-end est l'interface utilisateur de l'application EatFit. Il s'agit de la partie de l'application qui est directement visible et utilisable par les utilisateurs. Le front-end peut être une interface web, une application mobile, ou tout autre client qui communique avec l'API.

Il est conçu pour fournir une expérience utilisateur fluide et intuitive, permettant aux utilisateurs de naviguer facilement à travers les différentes recettes, de les noter et de les ajouter à leur historique personnel.

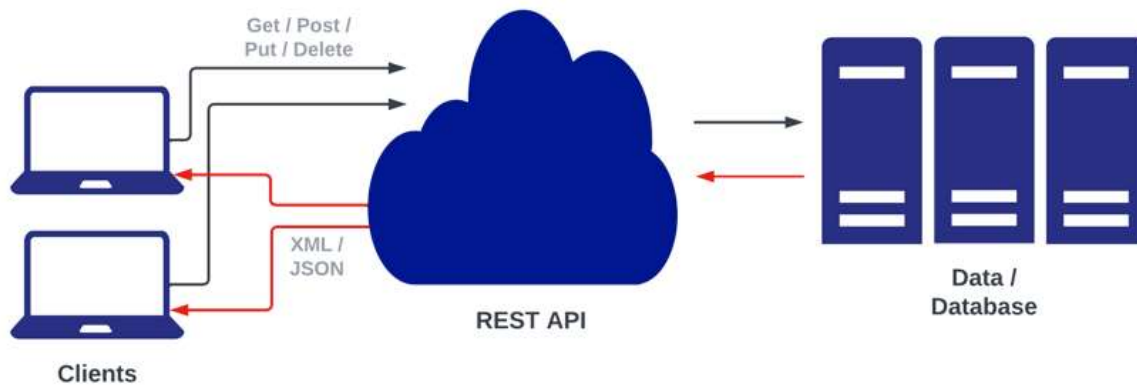
Api (Interface de Programmation d'Applications)

L'API est le pont entre le front-end et le back-end. Elle est conçue pour accepter les requêtes provenant du front-end, traiter ces requêtes en fonction de la logique métier définie dans le back-end, puis renvoyer les réponses appropriées au front-end.

L'API d'EatFit est basée sur le protocole HTTP et suit les principes de l'architecture REST. Elle comprend un ensemble de points d'accès (ou "routes") qui correspondent à différentes

actions que le front-end peut demander, comme la création d'une nouvelle recette ou la récupération d'une liste de recettes.

Image : <https://player.support.brightcove.com/getting-started/learning-guide-using-rest-apis.html>



Back-end

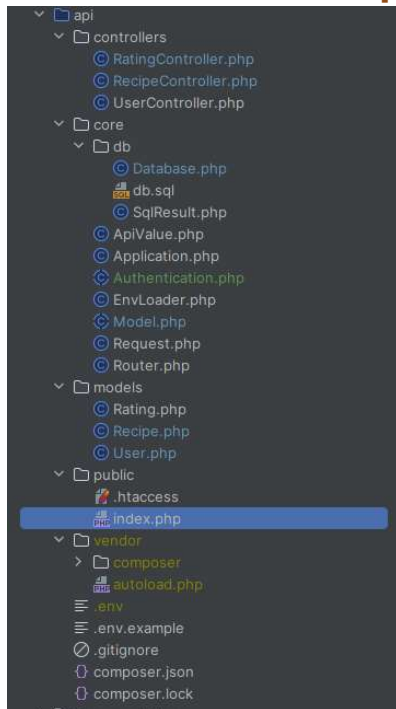
Le back-end est le cœur de l'application EatFit, où se trouve la logique métier et où les données sont stockées et récupérées. Il est composé de plusieurs composants, y compris les contrôleurs, les modèles et la base de données.

Les contrôleurs reçoivent les requêtes de l'API, effectuent les actions appropriées en utilisant les modèles, puis renvoient les réponses à l'API. Les modèles sont des représentations des structures de données de l'application, et ils contiennent les méthodes pour interagir avec la base de données.

La base de données est le dépôt où toutes les données de l'application sont stockées. Cela inclut les recettes, les utilisateurs, les notations et autres données pertinentes.

Cette architecture permet à EatFit d'être à la fois robuste et flexible, capable de s'adapter à divers clients front-end et de gérer une large gamme de requêtes d'utilisateurs.

Architecture de l'api



Classes

Pour implémenter les diverses fonctionnalités de l'api j'ai créé les classes suivantes :

Eatfit\Api\Controllers\UserController

Gère les opérations CRUD (Create, Read, Update, Delete) pour les utilisateurs, ainsi que les fonctionnalités d'authentification.

Eatfit\Api\Controllers\RecipeController

Responsable des opérations CRUD pour les recettes, y compris les fonctionnalités de recherche et de filtrage.

Eatfit\Api\Controllers\RatingController

Gère le CRUD pour les évaluations des recettes.

Eatfit\Api\Controllers\HistoryController

Eatfit\Api\Core\Db\Database

Fournit une interface pour se connecter à la base de données et effectuer des requêtes.

Eatfit\Api\Core\Db\SqlResult

Fournit une interface pour traiter les résultats des requêtes SQL.

Eatfit\Api\Core\ApiValue

Gère les réponses de l'API, y compris la codification des erreurs et la fourniture de messages d'erreur.

Eatfit\Api\Core**Application**

Classe principale de l'application qui gère l'initialisation, le routage, et la gestion des erreurs.

Eatfit\Api\Core\EnvLoader

Charge les variables d'environnement à partir d'un fichier .env pour une utilisation dans l'application.

Eatfit\Api\Core\Model

Classe de base pour tous les modèles, fournit des fonctionnalités communes pour la gestion des données d'entités comme les utilisateurs et les recettes.

Eatfit\Api\Core\Request

Gère les requêtes entrantes, extrait les informations à partir des en-têtes, des paramètres de requête et des données POST.

Eatfit\Api\Core\Router

Gère le routage des requêtes vers les contrôleurs appropriés en fonction de l'URL et de la méthode HTTP.

Eatfit\Api\Models\User

Modèle pour les utilisateurs, gère les interactions avec la base de données pour les informations des utilisateurs.

Eatfit\Api\Models\Recipe

Modèle pour les recettes, gère les interactions avec la base de données pour les informations des recettes.

Toutes ces classes travaillent ensemble pour fournir les fonctionnalités de l'application EatFit. Elles définissent la structure de l'API, permettant une séparation claire des préoccupations et une organisation logique du code.

Routes

GET '/' : Cette route renvoie une liste de toutes les routes disponibles dans l'application.

GET '/history' : Cette route renvoie l'historique de l'utilisateur connecté. Elle utilise la méthode 'read' du contrôleur 'HistoryController'.

POST '/history' : Cette route ajoute une entrée à l'historique de l'utilisateur connecté. Elle utilise la méthode 'create' du contrôleur 'HistoryController'.

PUT '/history' : Cette route met à jour une entrée de l'historique de l'utilisateur connecté. Elle utilise la méthode 'update' du contrôleur 'HistoryController'.

DELETE '/history' : Cette route supprime une entrée de l'historique de l'utilisateur connecté. Elle utilise la méthode 'delete' du contrôleur 'HistoryController'.

POST '/food_type' : Cette route crée un nouveau type de nourriture. Elle utilise la méthode 'addFoodType' du contrôleur 'RecipeController'.

GET '/food_types' : Cette route renvoie tous les types de nourriture disponibles. Elle utilise la méthode 'getFoodTypes' du contrôleur 'RecipeController'.

GET '/categories' : Cette route renvoie toutes les catégories de recettes disponibles. Elle utilise la méthode 'getCategories' du contrôleur 'RecipeController'.

PUT '/login' : Cette route est utilisée pour authentifier un utilisateur. Elle utilise la méthode login du contrôleur UserController, qui prend un email et un mot de passe en entrée et renvoie un jeton d'accès si les informations sont correctes.

GET '/userById' : Cette route récupère les détails d'un utilisateur à partir de son jeton d'accès. Elle utilise la méthode getUserByToken du contrôleur UserController.

POST '/history' : Cette route ajoute une recette à l'historique d'un utilisateur. Elle utilise la méthode addHistory du contrôleur RecipeController, qui prend l'ID de la recette comme argument.

GET '/user' : Cette route renvoie les détails de l'utilisateur actuellement connecté. Elle utilise la méthode read du contrôleur UserController, qui prend l'email et le mot de passe de l'utilisateur comme arguments.

POST '/user' : Cette route crée un nouvel utilisateur. Elle utilise la méthode create du contrôleur UserController, qui prend l'email, le mot de passe, la confirmation du mot de passe et le nom d'utilisateur comme arguments.

PUT '/user' : Cette route met à jour les détails de l'utilisateur actuellement connecté. Elle utilise la méthode update du contrôleur UserController, qui prend l'email, le mot de passe et le nom d'utilisateur comme arguments.

DELETE '/user' : Cette route supprime l'utilisateur actuellement connecté. Elle utilise la méthode delete du contrôleur UserController.

GET '/recipe' : Cette route renvoie les détails de toutes les recettes disponibles. Elle utilise la méthode read du contrôleur RecipeController, qui peut prendre des filtres de recherche comme arguments.

POST '/recipe' : Cette route crée une nouvelle recette. Elle utilise la méthode create du contrôleur RecipeController, qui prend les détails de la recette comme arguments.

PUT '/recipe' : Cette route met à jour une recette existante. Elle utilise la méthode update du contrôleur RecipeController, qui prend l'ID de la recette et les nouveaux détails comme arguments.

DELETE '/recipe' : Cette route supprime une recette existante. Elle utilise la méthode delete du contrôleur RecipeController, qui prend l'ID de la recette comme argument.

GET '/rating' : Cette route renvoie les évaluations de toutes les recettes. Elle utilise la méthode read du contrôleur RatingController.

POST '/rating' : Cette route crée une nouvelle évaluation pour une recette. Elle utilise la méthode create du contrôleur RatingController, qui prend le score de l'évaluation, le commentaire et l'ID de la recette comme arguments.

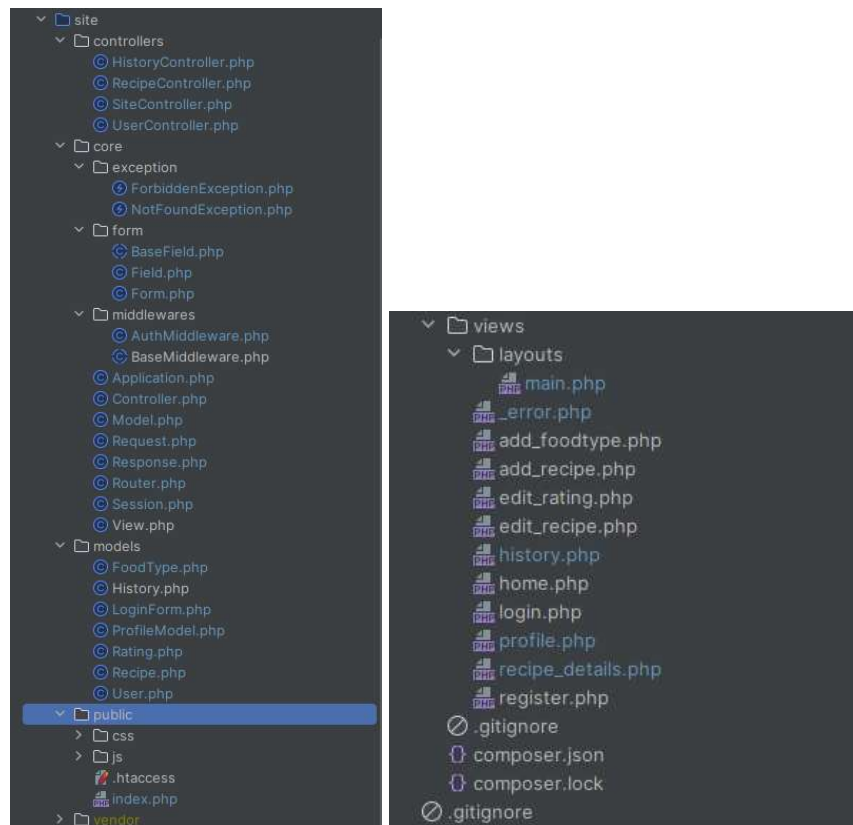
PUT '/rating' : Cette route met à jour une évaluation existante. Elle utilise la méthode update du contrôleur RatingController, qui prend l'ID de l'évaluation, le nouveau score et le nouveau commentaire comme arguments.

DELETE '/rating' : Cette route supprime une évaluation existante. Elle utilise la méthode delete du contrôleur RatingController, qui prend l'ID de l'évaluation comme argument.

POST '/food_type' : Cette route ajoute un nouveau type de nourriture. Elle utilise la méthode addFoodType du contrôleur RecipeController, qui prend le nom du nouveau type de nourriture comme argument.

Chaque route est liée à une fonction spécifique dans un contrôleur spécifique, qui effectue une action spécifique en fonction de la demande de l'utilisateur. Cette architecture permet de garder votre code organisé et facile à comprendre.

Architecture Front



Classes

Eatfit\Site\Controllers\HistoryController : Contrôleur pour gérer l'historique des recettes consultées/utilisées par un utilisateur.

Eatfit\Site\Controllers\RecipeController : Contrôleur pour gérer les opérations relatives aux recettes (par exemple, création, détails, mise à jour, suppression).

Eatfit\Site\Controllers\SiteController : Contrôleur principal de l'application, probablement pour gérer les vues et les opérations générales du site.

Eatfit\Site\Controllers**UserController** : Contrôleur pour gérer les opérations relatives à l'utilisateur (par exemple, connexion, déconnexion, inscription, profil).

Eatfit\Site\Core\Exception**ForbiddenException** : Une exception personnalisée pour gérer les scénarios d'accès interdit.

Eatfit\Site\Core\Exception**NotFoundException** : Une exception personnalisée pour gérer les scénarios où la page n'existe pas.

Eatfit\Site\Core\Form**BaseField** : Classe de base pour les champs de formulaire.

Eatfit\Site\Core\Form**Field** : Classe pour représenter un champ de formulaire spécifique.

Eatfit\Site\Core\Form**Form** : Permet de construire une balise <form> et de créer des champs.

Eatfit\Site\Core\Middlewares**AuthMiddleware** : Middleware pour gérer l'authentification des utilisateurs.

Eatfit\Site\Core\Middlewares**BaseMiddleware** : Classe de base pour tous les middlewares.

Eatfit\Site\Core**Application** : Classe principale de l'application qui pourrait initialiser et configurer l'application.

Eatfit\Site\Core**Controller** : Classe de base pour tous les contrôleurs.

Eatfit\Site\Core**Model** : Classe de base pour tous les modèles.

Eatfit\Site\Core**Request** : Classe pour gérer les requêtes entrantes.

Eatfit\Site\Core**Response** : Classe pour gérer les réponses sortantes.

Eatfit\Site\Core**Router** : Classe pour gérer le routage de l'application.

Eatfit\Site\Core**Session** : Classe pour gérer les sessions des utilisateurs.

Eatfit\Site\Core**View** : Classe pour gérer les vues de l'application.

Eatfit\Site\Core\Models**FoodType** : Modèle pour représenter un type d'aliment.

Eatfit\Site\Core\Models**History** : Modèle pour représenter l'historique de consultation/utilisation des recettes par un utilisateur.

Eatfit\Site\Core\Models**LoginForm** : Modèle pour représenter le formulaire de connexion.

Eatfit\Site\Core\Models**ProfileModel** : Modèle pour représenter le profil d'un utilisateur.

Eatfit\Site\Core\Models**Rating** : Modèle pour représenter une évaluation.

Eatfit\Site\Core\Models**Recipe** : Modèle pour représenter une recette.

Eatfit\Site\Core\Models**User** : Modèle pour représenter un utilisateur.

Routes :

'/' : La page d'accueil du site.

'/recipe' : Page pour créer une nouvelle recette.

'/recipe/detail/{idRecipe}' : Page de détail d'une recette spécifique.

'recipe/edit/{idRecipe}' : Page pour modifier une recette spécifique.

'recipe/delete/{idRecipe}' : Route pour supprimer une recette spécifique.

'rating/delete/{idRating}' et 'rating/update/{idRating}' : Routes pour supprimer et mettre à jour une évaluation spécifique.

'/food_type' : Page pour ajouter un nouveau type d'aliment.

'/login' et '/logout' : Pages pour se connecter et se déconnecter.

'/register' : Page pour s'enregistrer en tant que nouvel utilisateur.

'/profile' et '/profile/{method}' : Pages pour afficher et mettre à jour le profil d'un utilisateur.

'/addToHistory/{idRecipe}' and '/history' : Pages pour ajouter une recette à l'historique d'un utilisateur et pour afficher l'historique.

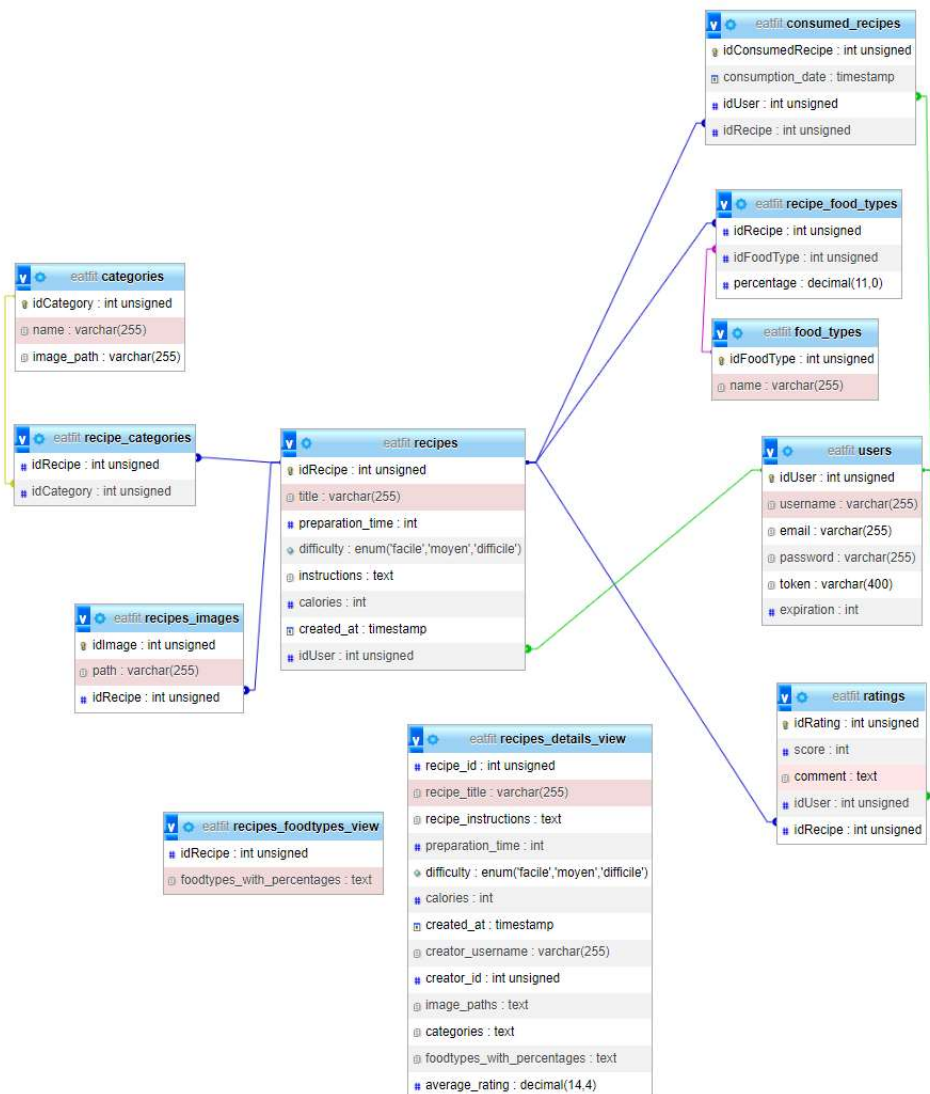
'/history/delete' et '/history/delete/{idConsumedRecipe}' : Routes pour supprimer tout l'historique ou une recette spécifique de l'historique d'un utilisateur.

Base de données

Informations techniques

EatFit utilise une base de données MySQL pour stocker les informations des recettes, les utilisateurs et l'historique de chaque recette consommée. J'ai utilisé InnoDB comme moteur de stockage et utf8_general_ci comme interclassement.

MLD



Tables categories

Colonne	Type	Null	Défaut
idCategory	Int	Non	
name	Varchar (255)	Non	
Image_path	Varchar (255)	Non	

Dans cette table :

- Le champ « idCategory » est la clé primaire
- Le champ name permet de stocker le nom de la catégorie (ex : Petit déjeuner)
- Le champ image_path permet de stocker l'image par défaut de la catégorie

consumed_recipes

Colonne	Type	Null	Défaut
idConsumedRecipe	Int	Non	
consumption_date	Date	Non	
idUser	Int	Non	
idRecipe	Int	Non	
idCategory	Int	Non	

Dans cette table :

- Le champ "idConsumedRecipe" est la clé primaire qui identifie chaque enregistrement de recette consommée.
- Le champ "consumption_date" permet de stocker la date à laquelle la recette a été consommée.
- Le champ "idUser" est une clé étrangère faisant référence à l'utilisateur qui a consommé la recette.
- Le champ "idRecipe" est une clé étrangère faisant référence à la recette consommée.
- Le champ "idCategory" est une clé étrangère faisant référence à la catégorie à laquelle la recette appartient.

food_types

Colonne	Type	Null	Défaut
idFoodType	Int	Non	
name	Varchar (255)	Non	

Dans cette table :

- Le champ "idFoodType" est la clé primaire qui identifie chaque type de nourriture.
- Le champ "name" permet de stocker le nom du type de nourriture.

ratings

Colonne	Type	Null	Défaut
idRating	Int	Non	
score	Int	Non	
comment	Text	Oui	NULL
idUser	Int	Non	
idRecipe	Int	Oui	

Dans cette table :

- Le champ "idRating" est la clé primaire qui identifie chaque enregistrement de notation.
- Le champ "score" permet de stocker la note attribuée à une recette.
- Le champ "comment" permet de stocker un commentaire facultatif associé à la notation.
- Le champ "idUser" est une clé étrangère faisant référence à l'utilisateur qui a donné la note.
- Le champ "idRecipe" est une clé étrangère faisant référence à la recette notée.

recipes

Colonne	Type	Null	Défaut
idRecipe	Int	Non	
title	Int	Non	
preparation_time	Text	Oui	NULL
difficulty	Int	Non	
instructions	Int	Oui	
calories	Int	Non	
created_at	timestamp	Oui	CURRENT_TIMESTAMP
idImage	Int		
idUser	Int	Non	

Dans cette table :

- Le champ "idRecipe" est la clé primaire qui identifie chaque recette.
- Le champ "title" permet de stocker le titre de la recette.
- Le champ "preparation_time" permet de stocker le temps de préparation de la recette.
- Le champ "difficulty" permet de stocker le niveau de difficulté de la recette.
- Le champ "instructions" permet de stocker les instructions de préparation de la recette.
- Le champ "calories" permet de stocker le nombre de calories de la recette.
- Le champ "created_at" permet de stocker la date de création de la recette.
- Le champ "idImage" est une clé étrangère faisant référence à l'image associée à la recette.
- Le champ "idUser" est une clé étrangère faisant référence à l'utilisateur qui a créé la recette.

recipe_categories

Colonne	Type	Null	Défaut
idRecipe	Int	Non	
idCategory	Int	Non	

Dans cette table :

- Le champ "idRecipe" est une clé étrangère faisant référence à une recette.
- Le champ "idCategory" est une clé étrangère faisant référence à une catégorie à laquelle la recette appartient.

recipe_food_types

Colonne	Type	Null	Défaut
idRecipe	Int	Non	
idFoodType	Int	Non	
percentage	Decimal	Non	

Dans cette table :

- Le champ "idRecipe" est une clé étrangère faisant référence à une recette.
- Le champ "idFoodType" est une clé étrangère faisant référence à un type de nourriture associé à la recette.
- Le champ "percentage" permet de stocker le pourcentage de ce type de nourriture dans la recette.

users

Colonne	Type	Null	Défaut
idUser	Int	Non	
username	Varchar (255)	Non	
email	Varchar (255)	Non	
password	Varchar (255)	Non	
token	Varchar(400)	Non	
expiration	Int	Non	

Dans cette table :

- Le champ "idUser" est la clé primaire qui identifie chaque utilisateur.
- Le champ "username" permet de stocker le nom d'utilisateur.
- Le champ "email" permet de stocker adresse e-mail de l'utilisateur.
- Le champ "password" permet de stocker le mot de passe de l'utilisateur de manière sécurisée.
- Le champ "token" permet de stocker le jeton d'authentification de l'utilisateur.
- Le champ "expiration" permet de stocker la durée de validité du jeton d'authentification.

recipes_images

Colonne	Type	Null	Défaut
idImage	Int	Non	
path	Varchar (255)	Non	
idRecipe	Int	Non	

Dans cette table :

- Le champ "idImage" est la clé primaire qui identifie chaque image associée à une recette.
- Le champ "path" permet de stocker le chemin d'accès à l'image de la recette.
- Le champ "idRecipe" est une clé étrangère faisant référence à la recette notée.

recipes_details_view

Colonne	Type	Null	Défaut
recipe_id	Int	Non	
recipe_title	Varchar (255)	Non	
recipe_instructions	Int	Non	

difficulty	Enum	Non	
calories	Int	Non	
created_at	Timestamp	Non	
preparation_time	Int	Non	
creator_user	Varchar	Non	
creator_id	Int	Non	
image_paths	Text	Non	
categories	Text	Non	
foodtype_with_percentages	Text	Non	
average_rating	Decimal	Non	

Dans cette view:

- Le champ "recipe_id" est la clé unique qui identifie chaque recette.
- Le champ "recipe_title" stocke le titre de chaque recette.
- Le champ "recipe_instructions" stocke les instructions pour la recette.
- Le champ "difficulty" représente le niveau de difficulté de la recette.
- Le champ "calories" donne le nombre de calories de la recette.
- Le champ "created_at" enregistre la date et l'heure de création de la recette.
- Le champ "preparation_time" donne le temps de préparation de la recette.
- Le champ "creator_user" stocke le nom d'utilisateur du créateur de la recette.
- Le champ "creator_id" est une clé étrangère qui fait référence à l'utilisateur qui a créé la recette.
- Le champ "image_paths" stocke les chemins d'accès aux images associées à la recette.
- Le champ "categories" pourrait stocker les catégories associées à la recette.
- Le champ "foodtype_with_percentages" stocke les types d'aliments utilisés dans la recette avec leurs pourcentages respectifs.
- Le champ "average_rating" donne la note moyenne de la recette.

recipes_foodtypes_view

Colonne	Type	Null	Défaut
idRecipe	Int	Non	
foodtypes_with_percentage	Varchar (255)	Non	

Dans cette table :

- Le champ "idRecipe" est la clé primaire qui identifie chaque recette.
- Le champ "foodtypes_with_percentage" stocke les types d'aliments utilisés dans la recette avec leurs pourcentages respectifs.

Analyse des fonctionnalités majeures

Recherche de recettes

Afin de permettre une navigation optimale et personnalisée à travers le large éventail de recettes disponibles, un système de recherche sophistiqué a été implémenté. Lorsqu'une

recherche est effectuée, une requête est envoyée à l'API pour récupérer toutes les recettes correspondant à une catégorie spécifiée, ainsi qu'à tous les filtres sélectionnés par l'utilisateur. Ainsi, si une recette correspond aux critères choisis, elle est ajoutée à la variable « `recipesToDisplay` ». Ce processus est facilité par une fonction qui génère un carousel HTML contenant les recettes filtrées selon plusieurs critères.

Le code PHP ci-dessous illustre cette fonctionnalité. Il filtre d'abord les recettes en fonction de plusieurs critères, tels que le titre, la catégorie, la date de la recette, ainsi que des filtres spécifiques tels que le type de nourriture et le nom d'utilisateur. Ensuite, il génère un carousel HTML avec les recettes qui répondent à tous ces critères. Si aucune recette ne répond aux critères de recherche et aux filtres spécifiés, le carousel n'est pas généré.

```
function renderCarouselWithSearch($category,
$recipesByCategory, $carouselId, $searchQuery,
$categoryFilter, $foodtypeFilter, $usernameFilter): void
{
    // On filtre les recettes en fonction des critères de
    recherche et des filtres spécifiés.
    $recipesToDisplay = array_filter($recipesByCategory,
function ($recipe) use ($searchQuery, $categoryFilter,
$foodtypeFilter, $usernameFilter) {
        // On vérifie si le titre, la catégorie, ou la date de
        la recette contiennent la requête de recherche.
        $titleContainsSearch = empty($searchQuery) ||
stripos($recipe->recipe_title, $searchQuery) !== false;
        $categoryContainsSearch = empty($searchQuery) ||
stripos($recipe->categories, $searchQuery) !== false;
        $dateContainsSearch = empty($searchQuery) ||
stripos($recipe->created_at, $searchQuery) !== false;
        // On vérifie si la catégorie, le type de nourriture,
        ou le nom d'utilisateur de la recette correspondent aux
        filtres spécifiés.
        $categoryMatchesFilter = empty($categoryFilter) ||
$recipe->categories == $categoryFilter;
        $foodtypeMatchesFilter = empty($foodtypeFilter) ||
stripos($recipe->foodtypes_with_percentages, $foodtypeFilter)
!== false;
        $usernameMatchesFilter = empty($usernameFilter) ||
$recipe->creator_username == $usernameFilter;
        // On retourne vrai si la recette satisfait tous les
        critères.
        return ($titleContainsSearch ||
$categoryContainsSearch || $dateContainsSearch) &&
$categoryMatchesFilter && $foodtypeMatchesFilter &&
$usernameMatchesFilter;
    });
    // Si aucune recette ne satisfait les critères, on ne
    génère pas le carousel.
    if (count($recipesToDisplay) === 0) return;
    ?>
    <div class="carousel-container">
        <h1><?= $category->name ?></h1>
```



```

        <div class="inner-carousel">
            <div class="track" id="track<?= $carouselId ?>">
                <?php
                    // Pour chaque recette à afficher, on génère
une carte avec les détails de la recette.
                    foreach ($recipesToDisplay as $recipe) {
                        // On récupère les chemins des images de
la recette.
                        $recipe->image_paths = !empty($recipe-
>image_paths) && str_contains($recipe->image_paths, ',') ?
array_map('trim', explode(',', $recipe->image_paths)) :
array($recipe->image_paths);
                        if (str_contains($recipe->image_paths[0],
'default')) $recipe->image_paths = [explode('_', $recipe-
>image_paths[0])[1]];
                        ?>
                        <div class="card-container">
                            <div class="container">
                                <div class="row">
                                    <div class="menu-item">
                                        <!-- On génère un lien
vers la page de détails de la recette, avec une image de la
recette. -->
                                        <a
href="/recipe/detail/<?= $recipe->recipe_id ?>">image_paths[0]
?>"
                                                alt="<?= $recipe-
>recipe_id ?>" class="rounded"></a>
                                        <div>
                                            <!-- On affiche le
titre et les instructions de la recette. -->
                                            <h3 style="text-align:
left;"><?= $recipe->recipe_title ?></h3>
                                            <p style="text-align:
left;"><?= $recipe->recipe_instructions ?></p>
                                        </div>
                                    </div>
                                </div>
                            </div>
                        </div>
                    <?php
                }
            ?>
        </div>
        <!-- On génère les boutons de navigation du
carrousel (précédent et suivant). -->
        <div class="nav" <?php if
(count($recipesToDisplay) < 5) echo 'style="display: none;"'
?>>

```

```

        <button class="prev" data-index="" id="prev<?=
$carouselId ?>" onclick="Prev(<?= $carouselId ?>)"><i
            class="fas fa-arrow-left fa-
2x"></i></button>
        <button class="next" data-index="0"
id="next<?= $carouselId ?>" onclick="Next(<?= $carouselId
?>)"><i
            class="fas fa-arrow-right fa-
2x"></i></button>
    </div>
</div>
</div>
<?php
}

```

Affichage des recettes

Lorsqu'un utilisateur se connecte sur le site Eatfit, toutes les recettes sont automatiquement récupérées depuis l'API. La vue affichant les recettes présente une mise en page soignée, incluant les images, le titre et un bref aperçu des instructions. Cette présentation permet aux utilisateurs d'avoir une vision claire et attrayante de chaque recette avant de décider d'en explorer davantage les détails.

```

$j = 0;
$searchQuery = $_POST['search'] ?? '';
$categoryFilter = $_POST['category-filter'] ?? '';
$foodtypeFilter = $_POST['foodtype-filter'] ?? '';
$usernameFilter = $_POST['username-filter'] ?? '';
foreach ($model->getCategories()->value as $category) {
    $recipesByCategory = $model->getRecipeByFilter("category",
$category->name)->value;
    if ($recipesByCategory === null) continue;
    renderCarouselWithSearch($category, $recipesByCategory,
$j, $searchQuery, $categoryFilter, $foodtypeFilter,
$usernameFilter);
    $j++;
}

```

Création de recettes

La fonctionnalité de création de recettes sur le site EatFit est assurée par un ensemble de fonctions en PHP qui travaillent ensemble pour intégrer l'ensemble des informations relatives à une recette dans la base de données.

Lorsqu'un utilisateur soumet un formulaire de création de recette, une requête est envoyée à l'API. Cette dernière récupère d'abord le token de l'utilisateur pour accéder à ses informations. L'API récupère ensuite tous les champs du formulaire, vérifie leur validité et commence à les insérer dans la base de données.

Le processus de création de la recette se déroule en plusieurs étapes :

```

public static function create(array $data): array|Exception
{
    $user = self::getUserByToken();
    if (!$user) throw new Exception("Non autorisé", 401);
    self::validateRecipeData($data);
    $data = [
        'title' => $data['title'],
        'preparation_time' => $data['preparation_time'],
        'difficulty' => $data['difficulty'],
        'instructions' => $data['instructions'],
        'calories' => $data['calories'],
        'created_at' => date("Y-m-d H:i:s"),
        'image' => $data['image'],
        'category' => $data['category'],
        'idUser' => $user['idUser'],
        'food_type' => $data['food_type']
    ];
    try {
        Application::$app->db->beginTransaction();
        self::insertRecipe($data);
        $idRecipe = Application::$app->db->getLastInsertId();
        self::insertRecipeCategories($data, $idRecipe);
        self::insertRecipeFoodType($data, $idRecipe);
        self::insertRecipeImages($data, $idRecipe);
        Application::$app->db->commit();
    } catch (Exception $e) {
        Application::$app->db->rollBack();
        throw new Exception("Erreur lors de la création de la
recette : " . $e->getMessage(), 500);
    }
    return self::getRecipe($data['title'])->getFirstRow();
}

```

Insertion de la recette : Les informations générales de la recette (titre, temps de préparation, difficulté, instructions, calories, date de création et l'identifiant de l'utilisateur) sont insérées dans la table "recipes" de la base de données.

```

private static function insertRecipe(array $data): void
{
    Application::$app->db->execute("INSERT INTO recipes
(title, preparation_time, difficulty, instructions,
calories,created_at,idUser)
VALUES (:title, :preparation_time, :difficulty,
:instructions, :calories,:created_at,:idUser)",
[
    ":title" => $data['title'],
    ":preparation_time" => $data['preparation_time'],
    ":difficulty" => $data['difficulty'],
    ":instructions" => $data['instructions'],
    ":calories" => $data['calories'],
    ":created_at" => $data['created_at'],

```

```

        ":idUser" => $data['idUser']
    });
}

```

Insertion des catégories de recettes : L'identifiant de la recette et l'identifiant de la catégorie sont insérés dans la table "recipe_categories".

```

private static function insertRecipeCategories(array $data,
$idRecipe): void
{
    Application::$app->db->execute(
        "INSERT INTO recipe_categories (idRecipe, idCategory)
VALUES (:idRecipe, (SELECT idCategory FROM categories WHERE
name = :name))",
        [
            ":idRecipe" => $idRecipe,
            ":name" => $data['category']
        ]
    );
}

```

Insertion des types d'aliments : L'identifiant de la recette, l'identifiant du type d'aliment et le pourcentage de ce type d'aliment dans la recette sont insérés dans la table "recipe_food_types".

```

private static function insertRecipeFoodType(array $data,
$idRecipe): void
{
    $values = [];
    $parameters = [];
    foreach ($data['food_type'] as $index => $food_type) {
        $values[] = "(SELECT idRecipe FROM recipes WHERE
idRecipe = :idRecipe), (SELECT idFoodType FROM food_types
WHERE name = :name{$index}), :percentage{$index})";
        $parameters[":name{$index}"] = $food_type[0];
        $parameters[":percentage{$index}"] = $food_type[1];
    }
    $query = "INSERT INTO recipe_food_types (idRecipe,
idFoodType, percentage) VALUES " . implode(", ", $values);
    $parameters[":idRecipe"] = $idRecipe;
    Application::$app->db->execute($query, $parameters);
}

```

Insertion des images de recettes : Si une ou plusieurs images sont fournies, elles sont insérées dans la base de données à l'aide de la fonction SQL spécifique qui permet d'insérer une ou plusieurs images et de récupérer les identifiant et chemin unique (« IdImage_Path ») pour garantir un nom unique à chaque image. Ces images sont ensuite stockées dans les fichiers de l'Api. Si aucune image n'est fournie, une image par défaut est utilisée.

```
private static function insertRecipeImages(array $data,
$idRecipe): void
{
    if ($data['image'] == "") {
        $image = Application::$app->db->execute(
            "SELECT image_path FROM categories WHERE name =
:name",
            [":name" => $data['category']] -
>getFirstRow()["image_path"];
        Application::$app->db->execute(
            "SELECT insert_unique_image_name(:path, (SELECT
idRecipe FROM recipes WHERE idRecipe = :idRecipe));",
            [":path" => $image, ":idRecipe" => $idRecipe]
        );
    } else {
        foreach ($data['image'] as $images) {
            $image = explode(",", $images);
            $imageSql = Application::$app->db->execute(
                "SELECT insert_unique_image_name(:path,
(SELECT idRecipe FROM recipes WHERE title = :title)) as
path;",
                [":path" => trim($image[0]), ":title" =>
$data['title']]
            );
            file_put_contents(Application::$UPLOAD_PATH .
$imageSql->getFirstRow()['path'],
base64_decode(trim($image[1])));
        }
    }
}
```

Toutes ces opérations sont effectuées dans le cadre d'une transaction pour garantir l'intégrité des données. Si une erreur se produit à n'importe quelle étape, la transaction est annulée et une exception est levée. Si toutes les étapes se déroulent sans erreur, la transaction est validée et la recette est créée.

Généralités concernant l'implémentation

Langages (versions)

- PHP (version 8.2)
- MySQL (8.0.31)
- Apache (2.4.54.2)
- HTML5
- CSS3

Environnement (local et/ou distant)

J'utilise l'environnement de développement local Wampserver - 3.3.0 – 64bit.

Plan de test

Périmètre

Le plan de test pour le projet EatFit couvrira toutes les fonctionnalités clés de l'application, y compris la gestion des profils d'utilisateurs, la gestion des recettes, les évaluations de recettes, la recherche de recettes et le filtrage des recettes. Les tests fonctionnels seront réalisés pour vérifier le bon fonctionnement et la convivialité de l'interface utilisateur. En outre, la sécurité de l'application sera testée pour garantir la protection adéquate des données personnelles des utilisateurs.

Environnement

Les tests seront effectués dans divers environnements pour assurer la compatibilité et la performance de l'application. Cela comprend les environnements de développement et de test. Les tests seront également réalisés sur différents navigateurs (Chrome, Firefox) pour assurer une expérience utilisateur optimale sur tous les navigateurs.

Scénarios (les tests, les données et leurs évolution)

Nom	1.1 Création d'un Virtual Host
User story	S5 : Création d'un Virtual Host
Situation	Le développeur a créé un Virtual Host pour permettre l'utilisation d'un "router" pour naviguer entre les différentes pages du site.
Résultats obtenus	Virtual host créée, peut être utilisé pour le site
Statut	OK

Nom	2.1 Structure de l'API
User story	S17 : Faire la structure de l'API
Situation	Le développeur crée la structure de base de l'API pour gérer les requêtes entre l'application et la base de données. Par exemple, il pourrait mettre en place des endpoints pour les opérations CRUD (Création, Lecture, Mise à jour, Suppression) sur les utilisateurs, les recettes et les évaluations.
Résultats obtenus	La structure de base de l'API est mise en place et prête à être utilisée par l'application.
Statut	OK

Nom	2.2 Adapter le modèle
User story	S18 : Adapter le modèle de l'API à l'application
Situation	Le développeur adapte le modèle de l'API pour qu'il corresponde aux besoins de l'application et aux relations entre les différentes entités (utilisateurs, recettes, évaluations, etc.).
Résultats obtenus	Le modèle de l'API est adapté pour correspondre aux besoins de l'application.

Statut	OK
--------	----

Nom	3.1 Création d'un profil utilisateur
User story	S4 : Gestion du profil utilisateur
Situation	L'utilisateur non identifié crée un nouveau compte en entrant «Username, email@gmail.com , test1234.» le nouveau compte est enregistré dans la base de données
Résultats obtenus	Retourne les informations de l'utilisateur avec un message de succès
Statut	OK

Nom	3.1.2 Création d'un profil utilisateur - Echec
User story	S4 : Gestion du profil utilisateur
Situation	L'utilisateur non identifié tente de créer un nouveau compte avec une adresse e-mail ou un pseudo qui est déjà utilisé(e).
Résultats obtenus	Un message d'erreur est affiché indiquant que l'adresse e-mail est déjà utilisée.
Statut	OK

Nom	3.2 Modification d'un profil utilisateur
User story	S4 : Gestion du profil utilisateur
Situation	Un utilisateur existant modifie les détails de son profil, comme son pseudo, son adresse e-mail ou son mot de passe.
Résultats obtenus	Les modifications sont enregistrées dans la base de données.
Statut	OK

Nom	3.2.2 Modification d'un profil utilisateur - Echec
User story	S4 : Gestion du profil utilisateur
Situation	Un utilisateur existant tente de modifier son adresse e-mail en une qui est déjà utilisée.
Résultats obtenus	Un message d'erreur est affiché indiquant que l'adresse e-mail est déjà utilisée.
Statut	OK

Nom	3.3 Suppression d'un profil utilisateur
User story	S4 : Gestion du profil utilisateur
Situation	L'utilisateur identifié peut supprimer son compte.
Résultats obtenus	Retourne un message de succès
Statut	OK

Nom	3.3.2 Suppression d'un profil utilisateur -
-----	---

	Echec
User story	S4 : Gestion du profil utilisateur
Situation	L'utilisateur non identifié tente de supprimer un compte.
Résultats obtenus	Un message d'erreur est affiché indiquant que l'utilisateur doit être identifié pour supprimer un compte.
Statut	OK

Nom	3.4 Lecture d'un profil utilisateur
User story	S4 : Gestion du profil utilisateur
Situation	L'utilisateur identifié peut accéder à ses informations.
Résultats obtenus	Retourne les informations de l'utilisateur
Statut	OK

Nom	4.1 Création d'une recette
User story	S7 : Gestion des recettes
Situation	L'utilisateur identifié peut créer une recette. Par exemple, un utilisateur peut créer une recette "Poulet à la moutarde" en renseignant le titre, le temps nécessaire, les instructions, la difficulté, les calories, la date de création et le créateur.
Résultats obtenus	La recette est créée avec succès et est visible dans la liste des recettes.
Statut	OK

Nom	4.1.2 Création d'une recette - Echec
User story	S7 : Gestion des recettes
Situation	L'utilisateur non identifié tente de créer une recette.
Résultats obtenus	Un message d'erreur est affiché indiquant que l'utilisateur doit être identifié pour créer une recette.
Statut	OK

Nom	4.2 Modification d'une recette
User story	S7 : Gestion des recettes
Situation	L'utilisateur identifié peut modifier sa recette. Par exemple, l'utilisateur pourrait changer le temps de préparation de sa recette "Poulet à la moutarde" de 45 minutes à 60 minutes.
Résultats obtenus	La recette est mise à jour avec succès et les modifications sont visibles dans la description de la recette.
Statut	OK

Nom	4.2.2 Modification d'une recette - Echec
User story	S7 : Gestion des recettes
Situation	L'utilisateur non identifié tente de modifier une recette.
Résultats obtenus	Un message d'erreur est affiché indiquant que l'utilisateur doit être identifié pour modifier une recette.
Statut	OK

Nom	4.3 Suppression d'une recette
User story	S7 : Gestion des recettes
Situation	L'utilisateur identifié peut supprimer sa recette. Par exemple, l'utilisateur pourrait décider de supprimer sa recette "Poulet à la moutarde" de sa liste de recettes.
Résultats obtenus	La recette est supprimée avec succès et n'est plus visible dans la liste des recettes.
Statut	OK

Nom	4.3.2 Suppression d'une recette - Echec
User story	S7 : Gestion des recettes
Situation	L'utilisateur non identifié tente de supprimer une recette.
Résultats obtenus	Un message d'erreur est affiché indiquant que l'utilisateur doit être identifié pour supprimer une recette.
Statut	OK

Nom	4.4 Lecture d'une recette
User story	S7 : Gestion des recettes
Situation	L'utilisateur identifié peut accéder à sa recette. Par exemple, l'utilisateur pourrait consulter les détails de sa recette "Poulet à la moutarde".
Résultats obtenus	L'utilisateur peut consulter les détails de sa recette avec succès.
Statut	OK

Nom	4.5 Recherche d'une recette
User story	S10 : Recherche de recettes
Situation	L'utilisateur peut rechercher des recettes en fonction de leur nom, catégorie ou date d'ajout. Par exemple, l'utilisateur pourrait rechercher toutes les recettes qui contiennent le mot "Poulet" dans leur nom.
Résultats obtenus	Les recettes correspondant aux critères de recherche de l'utilisateur sont affichées avec succès.
Statut	OK

Nom	4.6 Filtrage des recettes
User story	S11 : Filtrage des recettes
Situation	L'utilisateur identifié peut filtrer les recettes par catégorie ou type d'aliments. Par exemple, l'utilisateur pourrait choisir de n'afficher que les recettes avec protéines.
Résultats obtenus	Les recettes correspondant aux critères de filtrage de l'utilisateur sont affichées avec succès.
Statut	OK

Nom	5.1 Création d'une évaluation de recette
User story	S9 : Gestion des évaluations des recettes
Situation	L'utilisateur identifié peut évaluer de 1 à 5 et commenter(facultatif). Par exemple, après avoir cuisiné et goûté le "Poulet à la moutarde", l'utilisateur pourrait donner une note de 4 et laisser un commentaire comme "Très bon, mais un peu trop épicé pour moi".
Résultats obtenus	L'évaluation est enregistrée dans la base de données.
Statut	OK

Nom	5.1.2 Création d'une évaluation de recette - Echec
User story	S9 : Gestion des évaluations des recettes
Situation	L'utilisateur non identifié tente de créer une évaluation de recette.
Résultats obtenus	Un message d'erreur est affiché indiquant que l'utilisateur doit être identifié pour créer une évaluation.
Statut	OK

Nom	5.2 Modification d'une évaluation de recette
User story	S9 : Gestion des évaluations des recettes
Situation	L'utilisateur peut modifier la note ou le commentaire s'il n'y en avait pas avant il peut en ajouter. Par exemple, après une seconde préparation de la recette "Poulet à la moutarde", l'utilisateur décide de modifier sa note de 4 à 5 et d'ajouter un commentaire "Parfait avec moins de piment".
Résultats obtenus	L'évaluation de la recette est modifiée avec succès et les changements sont visibles sur la page de la recette.
Statut	OK

Nom	5.2.2 Modification d'une évaluation de recette - Echec
-----	--

User story	S9 : Gestion des évaluations des recettes
Situation	L'utilisateur non identifié tente de modifier une évaluation de recette. Un utilisateur identifié tente de modifier le titre de la recette et une recette avec ce nom existe déjà.
Résultats obtenus	Un message d'erreur est affiché indiquant que l'utilisateur doit être identifié pour modifier une évaluation. Un message d'erreur est affiché indiquant que le titre est déjà utilisé.
Statut	OK

Nom	5.3 Suppression d'une évaluation de recette
User story	S9 : Gestion des évaluations des recettes
Situation	L'utilisateur identifié peut supprimer l'évaluation. Par exemple, l'utilisateur décide de supprimer son évaluation sur la recette "Poulet à la moutarde" pour des raisons personnelles.
Résultats obtenus	L'évaluation est supprimée avec succès et n'est plus visible sur la page de la recette.
Statut	OK

Nom	5.3.2 Suppression d'une évaluation de recette - Echec
User story	S9 : Gestion des évaluations des recettes
Situation	L'utilisateur non identifié tente de supprimer une évaluation de recette.
Résultats obtenus	Un message d'erreur est affiché indiquant que l'utilisateur doit être identifié pour supprimer une évaluation.
Statut	OK

Nom	5.4 Lecture d'une évaluation de recette
User story	S9 : Gestion des évaluations des recettes
Situation	L'utilisateur identifié peut voir les commentaires d'une recette. Par exemple, l'utilisateur peut lire les commentaires laissés par d'autres utilisateurs sur la recette "Poulet à la moutarde".
Résultats obtenus	L'utilisateur peut lire avec succès les évaluations et commentaires d'autres utilisateurs sur la recette.
Statut	OK

Nom	6.1 Historique des recettes consommées
User story	S12 : Historique des recettes consommées
Situation	L'utilisateur identifié peut accéder à son

	historique et voir les dernières recettes consommées. Par exemple, l'utilisateur peut voir consommé la recette "Poulet à la moutarde" il y a deux jours.
Résultats obtenus	L'utilisateur peut accéder avec succès à son historique et voir les recettes qu'il a consommées récemment.
Statut	OK

Nom	6.1.2 Historique des recettes consommées - Echec
User story	S12 : Historique des recettes consommées
Situation	Un utilisateur non identifié tente d'accéder à l'historique des recettes consommées.
Résultats obtenus	Un message d'erreur est affiché indiquant que l'utilisateur doit être identifié pour accéder à l'historique des recettes consommées.
Statut	OK

Nom	7.1 Création du style du site
User story	S3 : Création du style du site
Situation	Le développeur a créé le style visuel du site, en se concentrant sur l'aspect esthétique et la convivialité de l'interface utilisateur.
Résultats obtenus	Le site a un aspect esthétiquement agréable et une interface utilisateur intuitive.
Statut	OK

Nom	8.1 Connexion à un compte utilisateur
User story	S4 : Gestion du profil utilisateur
Situation	L'utilisateur tente de se connecter avec un nom d'utilisateur et un mot de passe corrects.
Résultats obtenus	L'utilisateur est connecté avec succès et est redirigé vers la page d'accueil.
Statut	OK

Nom	8.2 Échec de la connexion à un compte
User story	S4 : Gestion du profil utilisateur
Situation	L'utilisateur tente de se connecter avec un nom d'utilisateur et un mot de passe incorrects.
Résultats obtenus	Un message d'erreur est affiché indiquant que les informations d'identification sont incorrectes.
Statut	OK

Nom	9.1 Ajout d'un nouveau type de nourriture
User story	S14 : Ajout de nouveaux types d'aliments
Situation	L'utilisateur rentre un type de nourriture dans le champ.
Résultats obtenus	Le type est enregistré et un message de succès est affiché.
Statut	OK

Nom	9.1.2 Ajout d'un nouveau type de nourriture - Echec
User story	S14 : Ajout de nouveaux types d'aliments
Situation	Un utilisateur non identifié tente d'ajouter un nouveau type de nourriture. Un utilisateur identifié tente d'ajouter un type existant.
Résultats obtenus	Un message d'erreur est affiché indiquant que l'utilisateur doit être identifié pour ajouter un nouveau type de nourriture. Un message d'erreur est affiché indiquant que le type existe déjà.
Statut	OK

Evolution des tests

N° test	J1	J2	J3	J4	J5	J6	J7	J8	J9	J10	J11
1.1	X	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
2.1	X	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
2.2	X	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
3.1	X	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
3.1.2	X	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
3.2	X	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
3.2.2	X	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
3.3	X	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
3.3.2	X	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

3.4	X	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
4.1	X	X	X	X	✓	✓	✓	✓	✓	✓	✓
4.1.2	X	X	X	X	✓	✓	✓	✓	✓	✓	✓
4.2	X	X	X	X	✓	✓	✓	✓	✓	✓	✓
4.2.2	X	X	X	X	✓	✓	✓	✓	✓	✓	✓
4.3	X	X	X	X	✓	✓	✓	✓	✓	✓	✓
4.3.2	X	X	X	X	✓	✓	✓	✓	✓	✓	✓
4.4	X	X	X	X	✓	✓	✓	✓	✓	✓	✓
4.5	X	X	X	X	X	X	X	✓	✓	✓	✓
4.6	X	X	X	X	X	X	✓	✓	✓	✓	✓
5.1	X	X	X	X	✓	✓	✓	✓	✓	✓	✓
5.1.2	X	X	X	X	✓	✓	✓	✓	✓	✓	✓
5.2	X	X	X	X	✓	✓	✓	✓	✓	✓	✓
5.2.2	X	X	X	X	✓	✓	✓	✓	✓	✓	✓
5.3	X	X	X	X	✓	✓	✓	✓	✓	✓	✓
5.3.2	X	X	X	X	✓	✓	✓	✓	✓	✓	✓
5.4	X	X	X	X	✓	✓	✓	✓	✓	✓	✓
6.1	X	X	X	X	X	✓	✓	✓	✓	✓	✓
6.1.2	X	X	X	X	X	✓	✓	✓	✓	✓	✓
7.1	X	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
8.1	X	X	X	X	X	X	X	✓	✓	✓	✓
8.2	X	X	X	X	X	X	X	✓	✓	✓	✓
9.1	X	X	X	X	X	X	✓	✓	✓	✓	✓
9.1.2	X	X	X	X	X	X	✓	✓	✓	✓	✓

Conclusion

Difficultés rencontrées

Bien que je n'aie pas rencontré de problèmes majeurs durant mon projet, une difficulté que j'ai rencontrée concernait la planification des tâches. En rétrospective, chaque tâche aurait dû bénéficier de plus de temps alloué. Par exemple, lorsque je progressais rapidement sur la gestion des utilisateurs, je n'avais pas initialement envisagé le temps nécessaire pour

développer le front-end correspondant à cette gestion. Ainsi, plutôt que de prévoir 4 heures pour une tâche, j'aurais peut-être dû prévoir 6 heures (par exemple). Cette expérience a souligné l'importance d'une planification précise et réaliste dans le cadre de la gestion de projet.

Variantes de solutions et choix

Choix et variantes			
Sujet	+	-	Conclusion
API ou Site classique			
API	Facilité de réutilisation pour d'autres plateformes (ex : app mobile)	Moins de familiarité avec la création d'API	Choix pour le projet actuel : API pour EatFit
	Meilleure séparation entre la logique back-end et front-end	Temps de développement potentiellement plus long	
	Facilite l'intégration avec d'autres services et applications	Nécessite une documentation plus complète	
Site classique	Plus familier avec la création de sites classiques	Limité aux navigateurs Web	
	Moins de documentation nécessaire	Difficulté d'intégration avec d'autres services	
	Temps de développement potentiellement plus court	Moins modulaire et moins évolutif	

Améliorations possibles

Le code actuel pourrait bénéficier de certaines améliorations, notamment en matière de refactoring pour réduire la duplication de code et optimiser la conception et l'architecture. Cela est dû au temps limité dont je disposais, qui a parfois conduit à copier-coller du code au lieu de bien le structurer en fonctions/classes distinctes.

Bilan personnel

Ce projet m'a permis de renforcer mes compétences en matière de gestion de projet et de planification, tout en m'offrant l'occasion d'explorer de nouvelles approches techniques. J'ai acquis une expérience précieuse en matière de développement d'API et de séparation des préoccupations entre le back-end et le front-end.

Remerciements

Je tiens à exprimer ma profonde gratitude à toutes les personnes qui ont contribué de près ou de loin à la réalisation de ce projet. Je remercie tout particulièrement Mme Mota, mes formateurs et mes collègues pour leurs précieux conseils et leur soutien tout au long de ce projet.

Bibliographie

Les liens que j'ai le plus consulté pendant mon TPI sont :

- <https://www.php.net/>
- <https://unsplash.com/> (pour les images)
- <https://stackoverflow.com>
- <https://www.w3schools.com>
- <https://codepen.io/bensassi/pen/dWBdwe>
- Avant de consulter, [ChatGPT](#) j'ai utilisé ce site pour rechercher des exemples de graphiques et comprendre comment les implémenter dans mon projet.
<https://www.chartjs.org/docs/latest/>
- <https://sql.sh/>
- https://www.youtube.com/playlist?list=PLLQuc_7jk_Uk_QnJMPndbdKECcTEwTA1
- <https://github.com/thecodeholic/tc-php-mvc-core>
- <https://github.com/thecodeholic/php-mvc-framework>

Annexes

Conversations ChatGPT

Graphique en js

Peux tu me faire un graphique camembert en js

ChatGPT

Malheureusement, en tant que modèle de texte AI, je ne peux pas créer des graphiques directement. Cependant, je peux vous montrer comment vous pouvez créer un graphique à secteurs (ou graphique camembert) en utilisant JavaScript et une bibliothèque populaire appelée Chart.js.

Voici un exemple de code que vous pouvez utiliser :

html

Copy code

```
<!DOCTYPE html>

<html>

<head>

  <title>Graphique Camembert en JavaScript</title>

  <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>

</head>

<body>
```



```
<canvas id="myChart"></canvas>

<script>
  var ctx = document.getElementById('myChart').getContext('2d');
  var myChart = new Chart(ctx, {
    type: 'pie',
    data: {
      labels: ['Rouge', 'Bleu', 'Jaune', 'Vert', 'Violet', 'Orange'],
      datasets: [{
        label: '# de votes',
        data: [12, 19, 3, 5, 2, 3],
        backgroundColor: [
          'rgba(255, 99, 132, 0.2)',
          'rgba(54, 162, 235, 0.2)',
          'rgba(255, 206, 86, 0.2)',
          'rgba(75, 192, 192, 0.2)',
          'rgba(153, 102, 255, 0.2)',
          'rgba(255, 159, 64, 0.2)'
        ],
        borderColor: [
          'rgba(255, 99, 132, 1)',
          'rgba(54, 162, 235, 1)',
          'rgba(255, 206, 86, 1)',
          'rgba(75, 192, 192, 1)',
          'rgba(153, 102, 255, 1)',
          'rgba(255, 159, 64, 1)'
        ],
        borderWidth: 1
      }]
    },
    options: {
      responsive: true,
    }
  }
}
```

```
});  
</script>  
</body>  
</html>
```

Dans cet exemple, nous créons un graphique à secteurs avec six sections différentes, chacune ayant une couleur différente. Les valeurs de chaque section sont spécifiées dans le tableau data et les étiquettes correspondantes sont spécifiées dans le tableau labels.

N'oubliez pas d'inclure Chart.js dans votre fichier HTML en utilisant la balise `<script>` dans l'en-tête de votre document. Vous pouvez télécharger Chart.js et l'inclure localement, ou vous pouvez utiliser un CDN comme dans l'exemple ci-dessus.

User

peux tu le faire plus petit

ChatGPT

Bien sûr, pour rendre le graphique plus petit, vous pouvez simplement définir la largeur et la hauteur du canvas. Par exemple, vous pouvez définir la largeur et la hauteur à 200 pixels comme ceci :

html

Copy code

```
<!DOCTYPE html>  
<html>  
<head>  
  <title>Graphique Camembert en JavaScript</title>  
  <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>  
</head>  
<body>  
  <canvas id="myChart" width="200" height="200"></canvas>  
  <script>
```

```
var ctx = document.getElementById('myChart').getContext('2d');
var myChart = new Chart(ctx, {
  type: 'pie',
  data: {
    labels: ['Rouge', 'Bleu', 'Jaune', 'Vert', 'Violet', 'Orange'],
    datasets: [{
      label: '# de votes',
      data: [12, 19, 3, 5, 2, 3],
      backgroundColor: [
        'rgba(255, 99, 132, 0.2)',
        'rgba(54, 162, 235, 0.2)',
        'rgba(255, 206, 86, 0.2)',
        'rgba(75, 192, 192, 0.2)',
        'rgba(153, 102, 255, 0.2)',
        'rgba(255, 159, 64, 0.2)'
      ],
      borderColor: [
        'rgba(255, 99, 132, 1)',
        'rgba(54, 162, 235, 1)',
        'rgba(255, 206, 86, 1)',
        'rgba(75, 192, 192, 1)',
        'rgba(153, 102, 255, 1)',
        'rgba(255, 159, 64, 1)'
      ],
      borderWidth: 1
    }]
  },
  options: {
    responsive: false, // pour désactiver le redimensionnement automatique
  }
});
</script>
```

</body>

</html>

Notez l'ajout des attributs `width="200"` et `height="200"` à la balise `<canvas>`. J'ai également modifié l'option `responsive` à `false` pour désactiver le redimensionnement automatique en fonction de la taille de la fenêtre.

User

J'aimerais récupérer les données d'une api

ChatGPT

Bien sûr, vous pouvez récupérer les données d'une API en utilisant la fonction `fetch()` en JavaScript. Voici comment vous pouvez le faire :

javascript

Copy code

```
fetch('URL_DE_VOTRE_API')
  .then(response => response.json())
  .then(data => {
    // Utilisez vos données ici
    console.log(data);
  })
  .catch(error => console.error('Erreur:', error));
```

Vous pouvez remplacer `'URL_DE_VOTRE_API'` par l'URL de l'API que vous souhaitez utiliser.

Cela dit, pour mettre à jour votre graphique avec les données récupérées, vous pouvez remplacer la partie de la récupération des données dans le script du graphique `camembert` comme suit :

html

Copy code

```
<!DOCTYPE html>
<html>
<head>
  <title>Graphique Camembert en JavaScript</title>
  <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
</head>
<body>
  <canvas id="myChart" width="200" height="200"></canvas>
  <script>
    fetch('URL_DE_VOTRE_API')
      .then(response => response.json())
      .then(data => {
        // Ici, je suppose que 'data' est un objet avec deux propriétés : 'labels' et 'datasets'
        var ctx = document.getElementById('myChart').getContext('2d');
        var myChart = new Chart(ctx, {
          type: 'pie',
          data: data,
          options: {
            responsive: false,
          }
        });
      })
      .catch(error => console.error('Erreur:', error));
  </script>
</body>
</html>
```

Dans cet exemple, je suppose que l'API renvoie un objet JSON avec les propriétés labels et datasets conformément à la structure attendue par Chart.js. Vous devrez adapter cette partie du code en fonction de la structure réelle des données renvoyées par votre API.

User

je peux récupérer des données du php ?

ChatGPT

Oui, vous pouvez certainement récupérer des données d'un script PHP. Le PHP peut générer des données dynamiques qui peuvent être consommées par votre application JavaScript.

Pour ce faire, vous pouvez effectuer une requête HTTP à votre script PHP en utilisant la fonction `fetch()` de JavaScript, puis utiliser la réponse pour alimenter votre graphique.

Voici comment vous pouvez le faire :

html

Copy code

```
<!DOCTYPE html>

<html>

<head>

  <title>Graphique Camembert en JavaScript</title>

  <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>

</head>

<body>

  <canvas id="myChart" width="200" height="200"></canvas>

  <script>

    fetch('votre_script.php')

    .then(response => response.json())

    .then(data => {

      // Ici, je suppose que 'data' est un objet avec deux propriétés : 'labels' et 'datasets'

      var ctx = document.getElementById('myChart').getContext('2d');

      var myChart = new Chart(ctx, {

        type: 'pie',

        data: data,

        options: {

          responsive: false,

        }

      });

    })

  </script>

</body>

</html>
```

```
.catch(error => console.error('Erreur:', error));  
</script>  
</body>  
</html>
```

Dans votre script PHP, assurez-vous que vous générez une réponse JSON valide. Voici un exemple simple de ce que pourrait être votre script PHP :

php

Copy code

```
<?php  
header('Content-Type: application/json');  
  
$data = [  
    'labels' => ['Rouge', 'Bleu', 'Jaune', 'Vert', 'Violet', 'Orange'],  
    'datasets' => [  
        [  
            'label' => '# de votes',  
            'data' => [12, 19, 3, 5, 2, 3],  
            'backgroundColor' => [  
                'rgba(255, 99, 132, 0.2)',  
                'rgba(54, 162, 235, 0.2)',  
                'rgba(255, 206, 86, 0.2)',  
                'rgba(75, 192, 192, 0.2)',  
                'rgba(153, 102, 255, 0.2)',  
                'rgba(255, 159, 64, 0.2)'  
            ],  
            'borderColor' => [  
                'rgba(255, 99, 132, 1)',  
                'rgba(54, 162, 235, 1)',  
                'rgba(255, 206, 86, 1)',  
                'rgba(75, 192, 192, 1)',  
                'rgba(153, 102, 255, 1)',  
                'rgba(255, 159, 64, 1)'  
            ]  
        ]  
    ]  
];
```

```
    ],  
    'borderWidth' => 1  
  ]  
]  
];
```

```
echo json_encode($data);
```

```
?>
```

Veuillez remplacer 'votre_script.php' par l'URL de votre script PHP. Aussi, n'oubliez pas de configurer correctement votre serveur pour permettre les requêtes CORS si votre script PHP et votre page HTML ne sont pas sur le même domaine.

User

JE peux pas faire plus simple j'ai une variable qui contient le pourcentage et une qui contient le nom

ChatGPT

D'accord, si vous avez déjà les variables en PHP, vous pouvez les incorporer directement dans votre script JavaScript de la manière suivante:

Supposons que vous ayez une variable \$nom pour le nom et une variable \$pourcentage pour le pourcentage en PHP. Vous pouvez les utiliser dans votre script JavaScript comme ceci :

php

Copy code

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
  <title>Graphique Camembert en JavaScript</title>
```

```
  <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
```

```
</head>
```

```
<body>
```

```
  <canvas id="myChart" width="200" height="200"></canvas>
```



```
<script>
  <?php
    $nom = 'Exemple'; // Remplacez par votre vraie variable
    $pourcentage = 75; // Remplacez par votre vraie variable
  ?>

  var ctx = document.getElementById('myChart').getContext('2d');
  var myChart = new Chart(ctx, {
    type: 'pie',
    data: {
      labels: ['<?php echo $nom; ?>', 'Autres'],
      datasets: [{
        data: [<?php echo $pourcentage; ?>, <?php echo (100 - $pourcentage); ?>],
        backgroundColor: [
          'rgba(255, 99, 132, 0.2)',
          'rgba(54, 162, 235, 0.2)'
        ],
        borderColor: [
          'rgba(255, 99, 132, 1)',
          'rgba(54, 162, 235, 1)'
        ],
        borderWidth: 1
      }]
    },
    options: {
      responsive: false,
    }
  });
</script>
</body>
</html>
```

Ici, nous utilisons les valeurs PHP directement dans le script JavaScript en les intégrant dans le code source avec `<?php echo $variable; ?>`.

Notez que ce code doit être exécuté sur un serveur PHP pour fonctionner correctement, car le PHP est un langage de script côté serveur. Si vous ouvrez ce fichier directement dans un navigateur sans passer par un serveur PHP, le code PHP ne sera pas exécuté.

User

Fais aussi une fonction qui génère des couleurs aléatoires pour automatiser le graphique

ChatGPT

Bien sûr, voici comment vous pouvez générer des couleurs aléatoires en JavaScript :

javascript

Copy code

```
function generateRandomColor() {  
    var letters = '0123456789ABCDEF';  
    var color = '#';  
    for (var i = 0; i < 6; i++) {  
        color += letters[Math.floor(Math.random() * 16)];  
    }  
    return color;  
}
```

Cette fonction génère une couleur hexadécimale aléatoire.

Et voici comment vous pouvez l'incorporer dans votre script de graphique :

php

Copy code

```
<!DOCTYPE html>  
  
<html>  
  
<head>
```

```

<title>Graphique Camembert en JavaScript</title>
<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
</head>
<body>
  <canvas id="myChart" width="200" height="200"></canvas>
  <script>
    function generateRandomColor() {
      var letters = '0123456789ABCDEF';
      var color = '#';
      for (var i = 0; i < 6; i++) {
        color += letters[Math.floor(Math.random() * 16)];
      }
      return color;
    }

    <?php
    $nom = 'Exemple'; // Remplacez par votre vraie variable
    $pourcentage = 75; // Remplacez par votre vraie variable
    ?>

    var ctx = document.getElementById('myChart').getContext('2d');
    var myChart = new Chart(ctx, {
      type: 'pie',
      data: {
        labels: ['<?php echo $nom; ?>', 'Autres'],
        datasets: [{
          data: [<?php echo $pourcentage; ?>, <?php echo (100 - $pourcentage); ?>],
          backgroundColor: [
            generateRandomColor(),
            generateRandomColor()
          ],
          border: 1,
          borderColor: [

```

```

        generateRandomColor(),
        generateRandomColor()
    ],
    borderWidth: 1
  }]
},
options: {
  responsive: false,
}
});
</script>
</body>
</html>

```

Dans ce code, `generateRandomColor()` est appelé à chaque fois que vous avez besoin d'une nouvelle couleur pour `backgroundColor` et `borderColor`.

Glossaire

Terme métier	
Recette (Recipe)	Instructions étape par étape pour préparer un plat spécifique.
Catégorie de recette (Category)	Classification des recettes en fonction de leur type, de leur moment de consommation ou de leurs ingrédients principaux (par exemple, petit déjeuner, déjeuner, dîner, dessert).
Difficulté (Difficulty)	Niveau de complexité ou d'effort requis pour préparer une recette, généralement classé en facile, moyen ou difficile.
Évaluation (Rating)	Appréciation ou jugement porté sur une recette par les utilisateurs, sous forme de note sur une échelle de 1 à 5.
Commentaire (Comment)	Retour d'information textuel d'un utilisateur sur une recette, généralement lié à l'évaluation donnée.
Historique des recettes consommées (History)	Liste des recettes réalisées et consommées par un utilisateur, avec des informations sur les dates et les moments de consommation.

Terme technique	
Front-end	Partie d'un système informatique ou d'une application avec laquelle l'utilisateur interagit directement.
Back-end	Partie d'un système informatique ou d'une application qui gère et traite les opérations en arrière-plan. Le back-end est responsable du fonctionnement et du traitement des données, qui sont ensuite envoyées au front-end pour être affichées à l'utilisateur.
Utilisateur (User)	Personne qui utilise l'application ou le système informatique. Dans le contexte de ce projet, cela peut se référer à toute personne qui utilise l'application de recettes, que ce soit pour parcourir les recettes, les noter, les commenter, ou les ajouter à leur historique de recettes consommées.
API	Une API est un ensemble de règles et de protocoles pour la construction et l'interaction avec des logiciels. Dans le contexte de ce projet, l'API permet à différentes parties de l'application (par exemple, le front-end et le back-end) de communiquer entre elles de manière efficace et sécurisée. Une API peut également permettre à l'application de recettes d'interagir avec d'autres logiciels ou services externes.
JSON	JSON est un format de données léger utilisé pour le stockage et le transport de données.
MLD	Le MLD est un modèle de données qui représente la structure d'une base de données de manière logique et indépendante du système de gestion de base de données utilisé. Il décrit la structure de la base de données en termes de tables, de champs et de relations entre les tables. Dans le contexte de ce projet, le MLD a été utilisé pour concevoir la structure de la base de données pour l'application de recettes.

Code source

History Controller Api

```
<?php
```

```
namespace Eatfit\Api\Controllers;
```

```
use Eatfit\Api\Core\ApiValue;
```

```
use Eatfit\Api\Core\Request;
```

```
use Eatfit\Api\Models\History;
use Exception;

class HistoryController
{
    /**
     * Crée une nouvelle entrée d'historique.
     *
     * @param Request $request La requête HTTP.
     * @return ApiValue L'objet ApiValue contenant la réponse
de l'API.
     * @throws Exception
     */
    public function create(Request $request): ApiValue
    {
        return new ApiValue(History::create($request->getData(['idRecipe'])), 200);
    }

    /**
     * Supprime une entrée d'historique spécifiée.
     *
     * @param Request $request La requête HTTP.
     * @return ApiValue L'objet ApiValue contenant la réponse
de l'API.
     * @throws Exception
     */
    public function delete(Request $request): ApiValue
    {
        return new ApiValue(History::delete($request->getData(['idConsumedRecipe'], false)), 200);
    }

    /**
     * Met à jour une entrée d'historique spécifiée.
     *
     * @param Request $request La requête HTTP.
     * @return ApiValue L'objet ApiValue contenant la réponse
de l'API.
     * @throws Exception
     */
    public function update(Request $request): ApiValue
    {
        return new ApiValue(History::update($request->getData(['idConsumedRecipe', 'consumption_date'])), 200);
    }

    /**
     * Récupère les entrées d'historique pour une recette
spécifiée.
```

```

    *
    * @param Request $request La requête HTTP.
    * @return ApiValue L'objet ApiValue contenant la réponse
    de l'API.
    * @throws Exception
    */
    public function read(Request $request): ApiValue
    {
        return new ApiValue(History::read($request-
>getData(['idRecipe'], false)), 200);
    }

}

```

Rating Controller API

```
<?php
```

```

namespace Eatfit\Api\Controllers;

use Eatfit\Api\Core\ApiValue;
use Eatfit\Api\Core\Request;
use Eatfit\Api\Models\Rating;
use Exception;

class RatingController
{
    /**
     * Récupère une évaluation spécifiée pour une recette
     spécifiée.
     *
     * @param Request $request La requête HTTP.
     * @return ApiValue L'objet ApiValue contenant la réponse
     de l'API.
     * @throws Exception
     */
    public function read(Request $request): ApiValue
    {
        return new ApiValue(Rating::read($request-
>getData(['idRecipe', 'idRating'], false)), 200);
    }

    /**
     * Crée une nouvelle évaluation pour une recette
     spécifiée.
     *
     * @param Request $request La requête HTTP.
     * @return ApiValue L'objet ApiValue contenant la réponse
     de l'API.
     * @throws Exception
     */
    public function create(Request $request): ApiValue

```

```

    {
        return new ApiValue(Rating::create($request-
>getData(['score', 'comment', 'idRecipe'], false)), 201);
    }

    /**
     * Met à jour une évaluation spécifiée.
     *
     * @param Request $request La requête HTTP.
     * @return ApiValue L'objet ApiValue contenant la réponse
de l'API.
     * @throws Exception
     */
    public function update(Request $request): ApiValue
    {
        return new ApiValue(Rating::update($request-
>getData(["idRating", "score", "comment"], false)), 200);
    }

    /**
     * Supprime une évaluation spécifiée.
     *
     * @param Request $request La requête HTTP.
     * @return ApiValue L'objet ApiValue contenant la réponse
de l'API.
     * @throws Exception
     */
    public function delete(Request $request): ApiValue
    {
        return new ApiValue(Rating::delete($request-
>getData(["idRating"])), 200);
    }
}

```

Recipe Controller API

```
<?php
```

```

namespace Eatfit\Api\Controllers;

use Eatfit\Api\Core\ApiValue;
use Eatfit\Api\Core\Request;
use Eatfit\Api\Models\Recipe;
use Exception;

class RecipeController
{
    /**
     * Récupère les recettes en fonction des filtres de

```



```
recherche spécifiés.
*
* @param Request $request La requête contenant les
données des filtres de recherche.
* @return ApiValue Les recettes trouvées en fonction des
filtres, avec un code de statut HTTP 200.
* @throws Exception
*/
public function read(Request $request): ApiValue
{
    return new ApiValue(Recipe::read($request->getData([
        'search',
        'filter',
        'idRecipe'
    ]), false)), 200);
}

/**
* Crée une nouvelle recette en utilisant les données
fournies.
*
* @param Request $request La requête contenant les
données de la nouvelle recette.
* @return ApiValue La recette nouvellement créée, avec un
code de statut HTTP 201.
* @throws Exception
*/
public function create(Request $request): ApiValue
{
    return new ApiValue(Recipe::create($request->getData([
        "title",
        "preparation_time",
        "difficulty",
        "instructions",
        "calories",
        "image",
        "category",
        "food_type"
    ])), 201);
}

/**
* Met à jour une recette existante en utilisant les
données fournies.
*
* @param Request $request La requête contenant les
données de mise à jour pour la recette.
* @return ApiValue La recette mise à jour, avec un code
de statut HTTP 200.
* @throws Exception
*/
```

```
public function update(Request $request): ApiValue
{
    return new ApiValue(Recipe::update($request-
>getData(["idRecipe", "category", "title", "preparation_time",
"difficulty", "instructions", "calories", "image",
"food_type"], false)), 200);
}

/**
 * Supprime une recette en fonction du titre spécifié.
 *
 * @param Request $request La requête contenant le titre
de la recette à supprimer.
 * @return ApiValue Un message indiquant que la recette a
été supprimée, avec un code de statut HTTP 200.
 * @throws Exception
 */
public function delete(Request $request): ApiValue
{
    return new ApiValue(Recipe::delete($request-
>getData(["idRecipe"])), 200);
}

/**
 * Ajoute un nouveau type de nourriture.
 *
 * @param Request $request La requête contenant les
données du nouveau type de nourriture.
 * @return ApiValue Le nouveau type de nourriture créé,
avec un code de statut HTTP 201.
 * @throws Exception
 */
public function addFoodType(Request $request): ApiValue
{
    return new ApiValue(Recipe::addFoodType($request-
>getData(['name'])), 201);
}

/**
 * Récupère les catégories de recettes disponibles.
 *
 * @return ApiValue Les catégories de recettes, avec un
code de statut HTTP 200.
 * @throws Exception
 */
public function getCategories(): ApiValue
{
    return new ApiValue(Recipe::getCategories(), 200);
}
```

```
/**
 * Récupère les types de nourriture disponibles.
 *
 * @return ApiValue Les types de nourriture, avec un code
de statut HTTP 200.
 * @throws Exception
 */
public function getFoodTypes(): ApiValue
{
    return new ApiValue(Recipe::getFoodTypes(), 200);
}
```

User Controller API

<?php

```
namespace Eatfit\Api\Controllers;

use Eatfit\Api\Core\ApiValue;
use Eatfit\Api\Core\Request;
use Eatfit\Api\Models\User;
use Exception;

class UserController
{
    /**
     * Récupère les informations de l'utilisateur en fonction
du jeton d'authentification.
     *
     * @return ApiValue Les informations de l'utilisateur,
avec un code de statut HTTP 200.
     * @throws Exception
     */
    public static function getUserByToken(): ApiValue
    {
        return new ApiValue(User::getUserByToken(), 200);
    }

    /**
     * Récupère les informations de l'utilisateur en fonction
de l'email et du mot de passe.
     *
     * @param Request $request La requête contenant l'email et
le mot de passe de l'utilisateur.
     * @return ApiValue Les informations de l'utilisateur,
avec un code de statut HTTP 200.
     * @throws Exception
     */
    public function read(Request $request): ApiValue
    {

```

```
        return new ApiValue(User::read($request->getData(['email', "password"])), 200);
    }

    /**
     * Crée un nouvel utilisateur en utilisant les données
     * fournies.
     *
     * @param Request $request La requête contenant les
     * données du nouvel utilisateur.
     * @return ApiValue Les informations de l'utilisateur
     * nouvellement créé, avec un code de statut HTTP 201.
     * @throws Exception
     */
    public function create(Request $request): ApiValue
    {
        return new ApiValue(User::create($request->getData(['email', "password", "confirm_password",
        "username"])), 201);
    }

    /**
     * Met à jour les informations de l'utilisateur en
     * utilisant les données fournies.
     *
     * @param Request $request La requête contenant les
     * données de mise à jour pour l'utilisateur.
     * @return ApiValue Les informations de l'utilisateur mise
     * à jour, avec un code de statut HTTP 200.
     * @throws Exception
     */
    public function update(Request $request): ApiValue
    {
        return new ApiValue(User::update($request->getData(['email', "password", "confirm_password",
        "username"], false)), 200);
    }

    /**
     * Supprime l'utilisateur actuellement authentifié.
     *
     * @return ApiValue Un message indiquant que l'utilisateur
     * a été supprimé, avec un code de statut HTTP 200.
     * @throws Exception
     */
    public function delete(): ApiValue
    {
        return new ApiValue(User::delete(), 200);
    }

    /**
```

```

        * Authentifie l'utilisateur en utilisant l'email et le
        mot de passe fournis.
        *
        * @param Request $request La requête contenant l'email et
        le mot de passe de l'utilisateur.
        * @return ApiValue Les informations de l'utilisateur
        authentifié, avec un code de statut HTTP 200.
        * @throws Exception
        */
        public function login(Request $request): ApiValue
        {
            return new ApiValue(User::authenticate($request-
>getData(['email', 'password'])), 200);
        }
    }
}

```

Database Api

```
<?php
```

```
namespace Eatfit\Api\Core\Db;
```

```
use Exception;
use PDO;
use PDOException;
```

```
class Database
```

```

{
    private static ?PDO $pdo = null;

    public function __construct(array $config)
    {
        if (self::$pdo === null) self::connect($config);
    }

    /**
     * Établit une connexion à la base de données en utilisant
     la configuration fournie.
     *
     * @param array $config La configuration de connexion à la
     base de données.
     * @throws PDOException Si une erreur survient lors de la
     connexion à la base de données.
     */
    private static function connect(array $config): void
    {
        try {
            self::$pdo = new PDO($config['dsn'],
            $config['user'], $config['password']);
            self::$pdo->setAttribute(PDO::ATTR_ERRMODE,
            PDO::ERRMODE_EXCEPTION);
        }
    }
}

```

```
        } catch (PDOException $e) {
            throw new PDOException("La connexion à la base de
données a échoué", 400);
        }
    }

    /**
     * Exécute une requête SQL préparée avec des paramètres.
     *
     * @param string $query La requête SQL préparée.
     * @param array $params Un tableau associatif contenant
les valeurs des paramètres.
     * @return SqlResult|false Un objet SqlResult contenant le
résultat de la requête ou false si une erreur survient.
     * @throws PDOException Si la requête est vide ou si une
erreur survient lors de l'exécution de la requête.
     * @throws Exception
    */
    // $query = execute("SELECT * FROM users WHERE id = :id
AND name = :name", [":id"=>1, ":name" => "John"]);
    public static function execute(string $query, array
$params = []): SqlResult|false
    {
        if (empty($query)) throw new Exception("Query cannot
be empty.", 400);
        if (self::$pdo === null) throw new Exception("Database
connection not initialized.", 500);
        $stmt = self::$pdo->prepare($query);
        $stmt->execute($params);
        return new SqlResult($stmt);
    }

    /**
     * Récupère le dernier identifiant inséré dans la base de
données.
     *
     * @return false|string L'identifiant inséré ou false si
aucun identifiant n'est disponible.
    */
    public static function getLastInsertId(): false|string
    {
        return self::$pdo->lastInsertId();
    }

    /**
     * Démarre une transaction.
    */
    public function beginTransaction(): void
    {
        self::$pdo->beginTransaction();
    }
}
```

```
/**
 * Valide une transaction.
 */
public function commit(): void
{
    self::$pdo->commit();
}

/**
 * Annule une transaction.
 */
public function rollBack(): void
{
    self::$pdo->rollBack();
}
}
```

SqlResult.php Api

```
<?php
```

```
namespace Eatfit\Api\Core\Db;
```

```
use PDOStatement;
```

```
class SqlResult
```

```
{
    private array $values = [];

    public function __construct(PDOStatement $statement)
    {
        $rows = $statement->fetchAll();
        foreach ($rows as $row) {
            $tempRow = $row;
            for ($i = 0; $i < count($row); $i++) {
                unset($tempRow[$i]);
            }
            $this->values[] = $tempRow;
        }
    }

    /**
     * Récupère le nombre total de lignes dans le résultat.
     *
     * @return int Le nombre total de lignes.
     */
    public function getRowCount(): int
    {
        return count($this->values);
    }
}
```

```
}

/**
 * Vérifie si le résultat est vide.
 *
 * @return bool True si le résultat est vide, sinon False.
 */
public function isEmpty(): bool
{
    return empty($this->values);
}

/**
 * Récupère toutes les valeurs du résultat.
 *
 * @return array Les valeurs du résultat.
 */
public function getValues(): array
{
    return $this->values;
}

/**
 * Récupère la première ligne du résultat.
 *
 * @return array|null La première ligne du résultat, ou
null si le résultat est vide.
 */
public function getFirstRow(): ?array
{
    return $this->values[0] ?? null;
}

/**
 * Récupère une ligne spécifique du résultat en fonction
de l'indice de ligne.
 *
 * @param int $rowIndex L'indice de ligne.
 * @return array|null La ligne spécifique du résultat, ou
null si l'indice de ligne est invalide.
 */
public function getRow(int $rowIndex): ?array
{
    return $this->values[$rowIndex] ?? null;
}

/**
 * Récupère une colonne spécifique du résultat en fonction
de l'étiquette de colonne.
 *
 * @param string $columnLabel L'étiquette de colonne.
```



```

        * @return array Les valeurs de la colonne spécifique.
        */
    public function getColumn(string $columnLabel): array
    {
        return array_column($this->values, $columnLabel);
    }
}

```

ApiValue Api

```
<?php
```

```

namespace Eatfit\Api\Core;
class ApiValue
{
    private const STATUS_CODES = [
        200 => 'OK',
        201 => 'Created',
        204 => 'No Content',
        400 => 'Bad Request',
        401 => 'Unauthorized',
        403 => 'Forbidden',
        404 => 'Not Found',
        405 => 'Method Not Allowed',
        409 => 'Conflict',
        422 => 'Unprocessable Entity',
        498 => 'Token expired/invalid',
        500 => 'Internal Server Error',
    ];

    public mixed $value;
    public string $message;
    public int $code;

    /**
     * Constructeur de la classe ApiValue
     *
     * @param mixed $value La valeur à renvoyer au client.
     * @param int $code Le code HTTP correspondant à la
     réponse.
     * @param string $message (optionnel) Le message de la
     réponse (si non fourni, le message sera généré automatiquement
     en fonction du code).
     */
    public function __construct(mixed $value, int $code,
    string $message = "")
    {
        $this->message = $message !== "" ? $message :
        (self::STATUS_CODES[$code] ?? self::STATUS_CODES[500]);
        $this->value = $value;
        $this->setCode($code);
    }
}

```

```

        http_response_code($this->code);
    }

    /**
     * Fonction setCode() - définit le code HTTP de la
     réponse.
     *
     * @param int $code Le code HTTP à définir.
     * @return void
     */
    private function setCode(int $code): void
    {
        $this->code = self::$STATUS_CODES[$code] !== null ?
$code : 400;
    }

    /**
     * Fonction magique __toString() - retourne l'objet
     ApiValue sous forme de JSON.
     *
     * @return string L'objet ApiValue sous forme de JSON.
     */
    public function __toString(): string
    {
        header('Content-Type: application/json; charset=UTF-
8');
        return json_encode($this);
    }
}

```

Application Api

<?php

```

namespace Eatfit\Api\Core;

use Eatfit\Api\Core\Db\Database;
use Exception;

class Application
{
    public static Application $app;
    public static string $ROOT_DIR;
    public static string $UPLOAD_PATH;
    public Router $router;
    public Request $request;
    public Database $db;
    private array $config;

    /**
     * Constructeur de l'application.

```

```

*
* @param string $rootDir Le répertoire racine de
l'application.
* @param array $config La configuration de l'application.
* @throws Exception
*/
public function __construct(string $rootDir, array
$config)
{
    self::$ROOT_DIR = $rootDir;
    self::$app = $this;
    $this->request = new Request();
    $this->router = new Router($this->request);
    $this->config = $config;
    self::$UPLOAD_PATH = $this->config['UPLOAD_PATH'];
}

/**
* Lance l'application.
* Résout la requête et renvoie la réponse correspondante.
* En cas d'exception, renvoie une réponse d'erreur.
*/
public function run(): void
{
    try {
        if (!is_dir(self::$UPLOAD_PATH)) {
            if (!mkdir(self::$UPLOAD_PATH, 0777, true)) {
                throw new Exception(sprintf('Échec de la
création du répertoire "%s"', self::$UPLOAD_PATH));
            }
        }
        $this->connectToDatabase();
        echo $this->router->resolve();
    } catch (Exception $e) {
        echo new ApiValue(null, $e->getCode(), $e-
>getMessage());
    }
}

/**
* Établit la connexion à la base de données.
*
* @throws Exception
*/
private function connectToDatabase(): void
{
    if (!isset($this->db)) {
        $this->db = new Database($this->config['db']);
    }
}
}

```

EnvLoader Api

```

<?php

namespace Eatfit\Api\Core;

use RuntimeException;

class EnvLoader
{
    private string $envFilePath;

    /**
     * Constructeur EnvLoader.
     *
     * @param string $envFilePath Le chemin du fichier .env
     */
    public function __construct(string $envFilePath)
    {
        $this->envFilePath = $envFilePath;
    }

    /**
     * Charge les variables d'environnement depuis le fichier
     * .env.
     *
     * @throws RuntimeException Si le fichier .env n'existe
     * pas
     */
    public function load(): void
    {
        if (!file_exists($this->envFilePath)) throw new
        RuntimeException("Le fichier .env n'existe pas : {$this-
        >envFilePath}", 404);
        $fileContents = file_get_contents($this->envFilePath);
        $lines = preg_split('/\r\n|\r|\n/', $fileContents);

        foreach ($lines as $line) {
            if (preg_match('/^\s*([\w\.\-
            ]+)\s*=\s*(.*)?\s*$/ ', $line, $matches)) {
                list(, $name, $value) = $matches;
                $value = trim($value);
                if (preg_match('/^"(.*)"$/s', $value,
                $matches)) $value = $matches[1];
                $_ENV[$name] = $value;
            }
        }
    }
}

```

Model Api

```
<?php
```

```
namespace Eatfit\Api\Core;

use Eatfit\Api\Models\User;
use Exception;

abstract class Model
{
    /**
     * Filtre un tableau en appliquant un filtre aux valeurs.
     *
     * @param array $array Le tableau à filtrer
     * @param int $filter Le filtre à appliquer aux valeurs
     * (par défaut FILTER_SANITIZE_SPECIAL_CHARS)
     * @return array Le tableau filtré
     */
    protected static function filterArray($array, $filter =
    FILTER_SANITIZE_SPECIAL_CHARS): array
    {
        $filteredArray = array();
        foreach ($array as $key => $value) {
            if (is_array($value)) {
                $filteredArray[$key] =
self::filterArray($value, $filter);
            } else {
                $filteredArray[$key] = filter_var($value,
$filter);
            }
        }
        return $filteredArray;
    }

    /**
     * Vérifie si le jeton d'authentification est valide.
     *
     * @param bool $expiration Vérifie l'expiration du jeton
     si true
     * @throws Exception Si le jeton est invalide ou expiré
     * @return array Les données décodées du jeton
     */
    protected static function getDataToken(bool $expiration =
true): array
    {
        $data = self::decodeJWT(self::getToken());
        if (!User::getUser($data['payload']['email'],
$data['payload']['username'])) throw new
Exception("Utilisateur introuvable", 404);
        if (self::isTokenExpired($data['payload']['exp'])) {
            Application::$app->db->execute("UPDATE users SET
```

```

token = NULL, expiration = NULL WHERE email = :email",
[":email" => $data['payload']['email']]);
        if ($expiration) {
            throw new Exception("Jeton expiré ou
invalide", 498);
        }
    }
    return $data;
}

/**
 * Décode un JWT et retourne ses composants.
 *
 * @param string $jwt Le JWT à décoder
 * @return array Les composants du JWT
 * @throws Exception Si le JWT est invalide
 */
private static function decodeJWT(string $jwt): array
{
    $parts = explode('.', $jwt);
    if (count($parts) !== 3) {
        throw new Exception('Format JWT invalide', 498);
    }
    $headers = json_decode(self::urlDecode($parts[0]),
true);
    $payload = json_decode(self::urlDecode($parts[1]),
true);
    $signature = self::urlDecode($parts[2]);
    return ['headers' => $headers, 'payload' => $payload,
'signature' => $signature];
}

/**
 * Décode une chaîne encodée en base64 URL-safe.
 *
 * @param string $data La chaîne à décoder
 * @return false|string La chaîne décodée
 */
private static function urlDecode(string $data):
false|string
{
    return base64_decode(strtr($data, '-_', '+/'));
}

/**
 * Récupère le jeton d'authentification à partir des en-
têtes de la requête.
 *
 * @return string Le jeton d'authentification
 * @throws Exception Si le jeton est invalide ou manquant
 */

```

```

private static function getToken(): string
{
    $headers = getAllheaders();
    if (!isset($headers['Authorization'])) {
        throw new Exception("Jeton expiré ou invalide",
498);
    }
    return explode(" ", $headers['Authorization'])[1];
}

/**
 * Vérifie si le jeton est expiré.
 *
 * @param int $expiration L'horodatage de l'expiration du
jeton
 * @return bool True si le jeton est expiré, sinon False
 * @throws Exception
 */
protected static function isTokenExpired($expiration):
bool
{
    return $expiration < time() || $expiration ===
'expiré';
}

/**
 * Génère un JSON Web Token (JWT) à partir des données
fournies.
 *
 * @param array $data Les données à inclure dans le JWT
 * @param int|null $expiration Expiration du JWT (par
défaut 2 heures après la génération)
 * @return string Le JWT généré
 */
protected static function generateJWT(array $data,
$expiration = null): string
{
    if ($expiration == null) $expiration = time() + (2 *
3600);
    $salt = $_ENV['SALT'] ?? "";
    $header = ['alg' => 'HS256', 'typ' => 'JWT'];
    $payload = [
        'email' => $data['email'],
        'username' => $data['username'],
        'exp' => $expiration,
    ];
    $encodedHeader =
self::urlencode(json_encode($header));
    $encodedPayload =
self::urlencode(json_encode($payload));
    $encodedParts = $encodedHeader . '.' .

```

```

$encodedPayload;
    $encodedSignature =
self::urlEncode(hash_hmac('sha256', $encodedParts, $salt,
true));
    return $encodedHeader . '.' . $encodedPayload . '.' .
$encodedSignature;
}

/**
 * Encode une chaîne en base64 URL-safe.
 *
 * @param false|string $json_encode La chaîne à encoder
 * @return array|string La chaîne encodée en base64 URL-
safe
 */
private static function urlEncode(false|string
$json_encode): array|string
{
    return str_replace(['+', '/', '='], ['-', '_', ''],
base64_encode($json_encode));
}

/**
 * Récupère les informations de l'utilisateur à partir du
jeton d'authentification.
 *
 * @param bool $expiration Vérifie l'expiration du jeton
si true
 * @return array|null Les informations de l'utilisateur,
ou null si l'utilisateur n'est pas trouvé
 * @throws Exception Si le jeton est invalide
 */
protected static function getUserByToken(bool $expiration
= true): ?array
{
    $data = self::decodeJWT(self::getToken());
    if (self::isTokenExpired($data['payload']['exp'])) {
        Application::$app->db->execute("UPDATE users SET
token = NULL, expiration = NULL WHERE email = :email",
["email" => $data['payload']['email']]);
        if ($expiration) {
            throw new Exception("Jeton expiré ou
invalide", 498);
        }
    }
    $user = User::getUser($data['payload']['email'],
$data['payload']['username']);
    if ($user->isEmpty()) throw new Exception("Utilisateur
introuvable", 404);
    unset($user->getFirstRow()['password']);
    return $user->getFirstRow();
}

```



```
}  
}
```

Request Api

```
<?php
```

```
namespace Eatfit\Api\Core;
```

```
use Exception;
```

```
class Request
```

```
{  
    private array $routeParams = [];  
  
    /**  
     * Récupère la méthode HTTP utilisée pour la requête.  
     *  
     * @return string La méthode HTTP utilisée.  
     */  
    public function getMethod(): string  
    {  
        return strtolower($_SERVER['REQUEST_METHOD']);  
    }  
  
    /**  
     * Récupère l'URL de la requête.  
     *  
     * @return string L'URL de la requête.  
     */  
    public function getUrl(): string  
    {  
        // Récupérer l'URI de la requête  
        $path = $_SERVER['REQUEST_URI'];  
        // Vérifier si l'URI contient un '?', ce qui signifie  
        // des paramètres de requête  
        $position = strpos($path, '?');  
        // Si des paramètres de requête sont présents, ne  
        // conserver que la partie de l'URI avant les paramètres de  
        // requête  
        if ($position !== false) {  
            $path = substr($path, 0, $position);  
        }  
        return $path;  
    }  
  
    /**  
     * Récupère tous les paramètres de la route associée à la  
     * requête.  
     *  
     * @return array Les paramètres de la route.
```

```
*/
public function getRouteParams(): array
{
    return $this->routeParams;
}

/**
 * Définit les paramètres de la route associée à la
 requête.
 *
 * @param array $params Les paramètres de la route.
 * @return self Une référence à l'objet courant,
 permettant le chaînage des méthodes.
 */
public function setRouteParams(array $params): self
{
    $this->routeParams = $params;
    return $this;
}

/**
 * Récupère la valeur d'un paramètre de la route associée
 à la requête.
 *
 * @param string $param Le nom du paramètre de la route.
 * @param mixed|null $default La valeur par défaut à
 retourner si le paramètre n'existe pas.
 * @return mixed La valeur du paramètre de la route, ou la
 valeur par défaut si le paramètre n'existe pas.
 */
public function getRouteParam(string $param, mixed
$default = null): mixed
{
    return $this->routeParams[$param] ?? $default;
}

/**
 * Récupère les données envoyées dans le corps de la
 requête HTTP en tant que JSON et les retourne sous forme
 d'array associatif.
 * Si $requiredFields est spécifié, cette fonction valide
 que tous les champs obligatoires sont présents dans le corps
 de la requête.
 * Si un champ obligatoire est manquant, une exception est
 levée avec un code HTTP 400 (Bad Request).
 *
 * @param array|null $requiredFields Un tableau contenant
 les noms des champs obligatoires dans le corps de la requête.
 * @param bool $requireAllFields Un booléen indiquant si
 tous les champs sont requis.
 * @return array Les données du corps de la requête sous
```

```

forme d'array associatif.
    * @throws Exception Si les données sont manquantes,
    invalides, ou si un champ obligatoire est manquant.
    */
    public function getData(array $requiredFields = null, bool
    $requireAllFields = true): array
    {
        $body = $this->getBody();
        if (!$requireAllFields && $body == null) return [];
        if (!is_array($body)) throw new Exception("Corps de la
    requête invalide", 400);
        $this->validateData($body, $requiredFields,
    $requireAllFields);
        return $body;
    }

    /**
     * Récupère le corps de la requête HTTP.
     *
     * @return mixed Le corps de la requête HTTP.
     */
    private function getBody(): mixed
    {
        return json_decode(file_get_contents('php://input'),
    true);
    }

    /**
     * Valide les données reçues dans le corps de la requête
    HTTP.
     *
     * @param array $fields Les champs à valider.
     * @param array|null $requiredFields Les champs requis.
     * @param bool $requireAllFields Un booléen indiquant si
    tous les champs sont requis.
     * @param string $parentKey La clé parente pour les champs
    imbriqués.
     * @throws Exception Si les données sont invalides.
     */
    private function validateData(array $fields, array
    $requiredFields = null, bool $requireAllFields = true,
    $parentKey = ''): void
    {
        if (empty($fields)) throw new Exception("Pas de
    données dans le corps de la requête", 400);
        if ($requiredFields !== null) {
            if (!is_array($requiredFields) ||
    count($requiredFields) == 0) throw new Exception("Champs
    requis invalides", 400);
            $missingFields = [];
            $extraFields = array_diff(array_keys($fields),

```

```

array_keys($requiredFields));

        foreach ($requiredFields as $field =>
$nestedFields) {
            if (is_array($nestedFields)) {
                if (array_key_exists($field, $fields) &&
is_array($fields[$field])) {
                    $this->validateData($fields[$field],
$nestedFields, $requireAllFields, $parentKey . $field . '.');
                    $extraFields =
array_diff($extraFields, [$field]);
                } else if ($requireAllFields) {
                    $missingFields[] = $parentKey .
$field;
                }
            } else {
                if (!array_key_exists($nestedFields,
$fields)) {
                    if ($requireAllFields)
$missingFields[] = $parentKey . $nestedFields;
                } else {
                    $extraFields =
array_diff($extraFields, [$nestedFields]);
                }
            }
        }

        if (!empty($missingFields)) {
            $allowedFields = array_map(function ($key,
$value) {
                return is_array($value) ? $key : $value;
            }, array_keys($requiredFields),
$requiredFields);

            $missingFieldsFormatted = array_map(function
($field) {
                return str_replace(explode('.', $field)[0]
. ". ", '', $field);
            }, $missingFields);
            throw new Exception("Champs manquants : " .
implode(", ", $missingFieldsFormatted) . ". Champs autorisés"
. ($parentKey ? " dans " . rtrim($parentKey, '.') : "") . "
sont : " . implode(", ", $allowedFields), 422);
        }

        if (!empty($extraFields)) {
            $allowedFields = array_map(function ($key,
$value) {
                return is_array($value) ? $key : $value;
            }, array_keys($requiredFields),
$requiredFields);

```

```

        throw new Exception("Les champs : " .
implode(", ", $extraFields) . " n'existent pas. Les champs
autorisés" . ($parentKey ? " dans " . rtrim($parentKey, '.') :
"") . " sont : " . implode(", ", $allowedFields), 422);
    }
}
}
}

```

Router Api

<?php

```
namespace Eatfit\Api\Core;
```

```
use Exception;
```

```
class Router
```

```

{
    private Request $request;
    private array $routeMap = [];

    public function __construct(Request $request)
    {
        $this->request = $request;
    }

    public function get(string $url, mixed $callback): void
    {
        $this->addRoute('get', $url, $callback);
    }

    /**
     * Ajoute une route pour une méthode HTTP spécifiée
     *
     * @param string $method La méthode HTTP (ex: GET, POST,
etc.)
     * @param string $url L'URL à associer à la route
     * @param mixed $callback Le callback à appeler lorsque la
route est atteinte
     * @return void
     */
    private function addRoute(string $method, string $url,
mixed $callback): void
    {
        $this->routeMap[strtolower($method)][$url] =
$callback;
    }

    public function post(string $url, mixed $callback): void
    {

```

```

        $this->addRoute('post', $url, $callback);
    }

    public function put(string $url, mixed $callback): void
    {
        $this->addRoute('put', $url, $callback);
    }

    public function delete(string $url, mixed $callback): void
    {
        $this->addRoute('delete', $url, $callback);
    }

    public function getRoutes(): array
    {
        return $this->routeMap;
    }

    /**
     * Résout la requête HTTP en appelant le bon callback
     *
     * @throws Exception Si la méthode HTTP n'est pas
    autorisée ou si aucune route n'a été trouvée pour l'URL de la
    requête courante
     * @return mixed La réponse retournée par le callback
    appelé
     */
    public function resolve(): mixed
    {
        $method = $this->request->getMethod();
        $url = $this->request->getUrl();

        $callback = $this->routeMap[$method][$url] ?? false;
        if (!$callback) {
            $callback = $this->getCallback();
            $routes = [];
            foreach ($this->routeMap as $key => $value) {
                foreach ($value as $k => $v) {
                    if (in_array($k, $routes)) continue;
                    $routes[] = $k;
                }
            }
            if (in_array($url, $routes)) throw new
Exception("La méthode $method n'est pas autorisée pour la
route : '$url'", 405);
            if ($callback === false) throw new
Exception("Aucune route trouvée pour la route : $url", 404);
        }
        if (is_array($callback)) $callback = [new
$callback[0], $callback[1]];
        return call_user_func($callback, $this->request);
    }

```

```

    }

    /**
     * Récupère le callback associé à l'URL de la requête
     courante
     *
     * @return mixed Le callback associé à l'URL de la requête
     courante, ou false si aucun callback trouvé
     */
    public function getCallback(): mixed
    {
        $method = $this->request->getMethod();
        $url = $this->request->getUrl();
        // Trim slashes
        $url = trim($url, '/');
        // Get all routes for current request method
        $routes = $this->getRouteMap($method);
        $routeParams = false;
        // Start iterating registered routes
        foreach ($routes as $route => $callback) {
            // Trim slashes
            $route = trim($route, '/');
            $routeNames = [];
            if (!$route) continue;
            // Find all route names from route and save in
            $routeNames
            if (preg_match_all('/\{(\w+)(:[^}]+)?\}/', $route,
            $matches)) {
                $routeNames = $matches[1];
            }
            // Convert route name into regex pattern
            $routeRegex = "@^" .
            preg_replace_callback('/\{(\w+)(:[^}]+)?\}/', fn($m) =>
            isset($m[2]) ? "({$m[2]})" : '(\w+)', $route) . "$@";
            // Test and match current route against
            $routeRegex
            if (preg_match_all($routeRegex, $url,
            $valueMatches)) {
                $values = [];
                for ($i = 1; $i < count($valueMatches); $i++)
                {
                    $values[] = $valueMatches[$i][0];
                }
                $routeParams = array_combine($routeNames,
            $values);
                $this->request->setRouteParams($routeParams);
                return $callback;
            }
        }
        return false;
    }
}

```

```

    public function getRouteMap(string $method): array
    {
        return $this->routeMap[strtolower($method)] ?? [];
    }
}

```

History Api

```
<?php
```

```

namespace Eatfit\Api\Models;

use Eatfit\Api\Core\Application;
use Eatfit\Api\Core\Model;
use Exception;

class History extends Model
{
    /**
     *
     * Ajoute une recette à l'historique de l'utilisateur.
     * @param array $data Les données de la recette à ajouter
     * à l'historique
     * @return string Message de succès
     * @throws Exception En cas d'erreur ou d'authentification
     * échouée
     * @throws \Exception
     */
    public
    static function create(array $data): string
    {
        $user = self::getUserByToken();
        if (!$user) throw new Exception("Non autorisé", 401);
        if (!is_numeric($data['idRecipe'])) throw new
Exception("L'ID de la recette doit être un nombre", 400);
        $recipe = Recipe::getRecipeById($data['idRecipe']);
        if (!$recipe) throw new Exception("Recette non
trouvée", 404);
        try {
            Application::$app->db->execute("INSERT INTO
consumed_recipes(idUser, idRecipe, consumption_date)
VALUES(:idUser, :idRecipe, :consumption_date)", [
                ":idUser" => $user['idUser'],
                ":idRecipe" => $data['idRecipe'],
                ":consumption_date" => date("Y-m-d H:i:s")
            ]);
        } catch (Exception $e) {
            throw new Exception("Erreur lors de l'ajout de la
recette à l'historique", 500);
        }
    }
}

```



```

        return "Recette ajoutée à l'historique avec succès";
    }

    /**
     * Supprime une recette de l'historique de l'utilisateur.
     *
     * @param array $data Les données de la recette à
supprimer de l'historique
     * @return string Message de succès
     * @throws Exception En cas d'erreur ou d'authentification
échouée
     */
    public
    static function delete(array $data): string
    {
        $user = self::getUserByToken();
        if (!$user) throw new Exception("Non autorisé", 401);
        if (isset($data['idConsumedRecipe'])) {
            if (!is_numeric($data['idConsumedRecipe'])) throw
new Exception("L'ID de l'historique doit être un nombre",
400);

            if (self::getHistory($data['idConsumedRecipe'])-
>isEmpty()) throw new Exception("Historique non trouvé", 404);
            Application::$app->db->execute("DELETE FROM
consumed_recipes WHERE idUser = :idUser AND idConsumedRecipe =
:idConsumedRecipe", [
                ":idUser" => $user['idUser'],
                ":idConsumedRecipe" =>
$data['idConsumedRecipe']
            ]);
            return "Recette supprimée de l'historique avec
succès";
        }
        Application::$app->db->execute("DELETE FROM
consumed_recipes WHERE idUser = :idUser", [
            ":idUser" => $user['idUser']
        ]);
        return "Historique supprimé avec succès";
    }

    private static function getHistory($idConsumedRecipe)
    {
        return Application::$app->db->execute("SELECT * FROM
consumed_recipes WHERE idConsumedRecipe = :idConsumedRecipe",
[
            ":idConsumedRecipe" => $idConsumedRecipe
        ]);
    }

    /**

```

```

    * Met à jour une recette dans l'historique de
    l'utilisateur.
    *
    * @param array $data Les données de la recette à mettre à
    jour dans l'historique
    * @return string Message de succès
    * @throws Exception En cas d'erreur ou d'authentification
    échouée
    */
    public static function update(array $data): string
    {
        $user = self::getUserByToken();
        if (!$user) throw new Exception("Non autorisé", 401);
        if (!is_numeric($data['idConsumedRecipe'])) throw new
Exception("L'ID de l'historique doit être un nombre", 400);
        if (self::getHistory($data['idConsumedRecipe'])-
>isEmpty()) throw new Exception("Historique non trouvé", 404);
        if (!isset($data['consumption_date'])) throw new
Exception("La date de consommation doit être renseignée",
400);
        if (!strtotime($data['consumption_date'])) throw new
Exception("La date de consommation doit être au format YYYY-
MM-DD HH:MM:SS", 400);
        try {
            Application::$app->db->execute("UPDATE
consumed_recipes SET consumption_date = :consumption_date
WHERE idUser = :idUser AND idConsumedRecipe =
:idConsumedRecipe", [
                ":idUser" => $user['idUser'],
                ":idConsumedRecipe" =>
$data['idConsumedRecipe'],
                ":consumption_date" => date("Y-m-d H:i:s")
            ]);
        } catch (Exception $e) {
            throw new Exception("Erreur lors de la mise à jour
de l'historique", 500);
        }

        return "Historique mis à jour avec succès";
    }

    /**
    * Lit l'historique de l'utilisateur.
    *
    * @return array L'historique de l'utilisateur
    * @throws Exception En cas d'erreur ou d'authentification
    échouée
    */
    public static function read($data): array
    {
        $user = self::getUserByToken();

```

```

        if (!$user) throw new Exception("Non autorisé", 401);
        $history = null;
        $data = self::filterArray($data);
        if (isset($data['idRecipe'])) {
            if (!is_numeric($data['idRecipe'])) throw new
Exception("L'ID de la recette doit être un nombre", 400);
            $history = Application::$app->db->execute("SELECT
consumed_recipes.*, recipes.title,recipes.created_at FROM
consumed_recipes JOIN recipes ON consumed_recipes.idRecipe =
recipes.id WHERE consumed_recipes.idUser = :idUser AND
consumed_recipes.idRecipe = :idRecipe", [
                ":idUser" => $user['idUser'],
                ":idRecipe" => $data['idRecipe']
            ]);
        } else {
            $history = Application::$app->db->execute("SELECT
consumed_recipes.*, recipes.title,recipes.created_at FROM
consumed_recipes JOIN recipes ON consumed_recipes.idRecipe =
recipes.idRecipe WHERE consumed_recipes.idUser = :idUser ORDER
BY consumed_recipes.consumption_date DESC", [":idUser" =>
$user['idUser']]);
        }
        if ($history->isEmpty()) throw new Exception("Aucun
historique trouvé", 404);
        return $history->getValues();
    }
}

```

Rating Api

<?php

```

namespace Eatfit\Api\Models;

use Eatfit\Api\Core\Application;
use Eatfit\Api\Core\Db\SqlResult;
use Eatfit\Api\Core\Model;
use Exception;

class Rating extends Model
{
    /**
     * Lit toutes les évaluations de la base de données pour
     une recette spécifiée.
     *
     * @param array $data Les données de la requête.
     * @return array La liste de toutes les évaluations pour
     la recette spécifiée.
     * @throws Exception Si une erreur se produit lors de
     l'exécution de la requête.
     */
}

```

```

public static function read($data): array
{
    if (isset($data['idRecipe'])) {
        $query = "SELECT
idRating,ratings.idUser,score,comment,users.username FROM
`ratings` JOIN users ON ratings.idUser = users.idUser ";
        $query .= " WHERE idRecipe = :idRecipe";
        $params = [":idRecipe" => $data['idRecipe']];
        $result = Application::$app->db->execute($query,
$params);
        if ($result->isEmpty()) throw new
Exception("Aucune évaluation trouvée", 404);
        return $result->getValues();
    }
    if (!isset($data['idRating'])) throw new
Exception("L'ID de l'évaluation est requis", 400);
    $rating = self::getRatingById($data['idRating']);
    if ($rating->isEmpty()) throw new Exception("Aucune
évaluation trouvée", 404);
    return $rating->getValues();
}

/**
 * Méthode pour obtenir une évaluation spécifique par son
identifiant.
 *
 * @param $idRating L'identifiant de l'évaluation à
obtenir.
 * @return SqlResult Le résultat de la requête SQL.
 * @throws Exception Si une erreur se produit lors de
l'exécution de la requête.
 */
public static function getRatingById($idRating): SqlResult
{
    return Application::$app->db->execute("SELECT * FROM
ratings WHERE idRating = :idRating", [":idRating" =>
$idRating]);
}

/**
 * Crée une nouvelle évaluation pour une recette
spécifiée.
 *
 * @param array $data Les données de l'évaluation à créer.
 * @return array Les informations sur l'évaluation créée.
 * @throws Exception Si une erreur se produit lors de la
validation des données ou de l'exécution de la requête.
 */
public static function create(array $data): array
{
    $query = "INSERT INTO ratings (score, comment, idUser,

```

```

idRecipe) VALUES (:score, :comment, :idUser, :idRecipe)";
    $data = self::filterArray($data);
    $user = self::getUserByToken();
    if (!isset($data['score'])) throw new Exception("Le
score est requis", 400);
    if (!is_numeric($data['score'])) throw new
Exception("Le score doit être un nombre", 400);
    if ($data['score'] < 1 || $data['score'] > 5) throw
new Exception("Le score doit être entre 1 et 5", 400);
    if (!isset($data['idRecipe'])) throw new
Exception("L'ID de la recette est requis", 400);
    if (!is_numeric($data['idRecipe'])) throw new
Exception("L'ID de la recette doit être un nombre", 400);
    $data['idUser'] = $user['idUser'];
    if (isset($data['comment']) && $data['comment'] == "")
    $data['comment'] = null;
    try {
        Application::$app->db->execute($query, [
            ":score" => $data['score'],
            ":comment" => $data['comment'],
            ":idUser" => $data['idUser'],
            ":idRecipe" => $data['idRecipe']
        ]);
    } catch (Exception $e) {
        throw new Exception("Erreur lors de la création de
l'évaluation", 500);
    }
    return Application::$app->db->execute("SELECT * FROM
ratings WHERE idRating = :idRating", [":idRating" =>
Application::$app->db->getLastInsertId()])->getFirstRow();
}

/**
 * Méthode pour mettre à jour une évaluation existante.
 *
 * @param array $data Les données de l'évaluation à mettre
à jour.
 * @return string Un message indiquant que l'évaluation a
été mise à jour avec succès.
 * @throws Exception Si une erreur se produit lors de la
validation des données ou de l'exécution de la requête.
 */
public static function update(array $data): string
{
    $data = self::filterArray($data);
    if (!isset($data['idRating'])) throw new
Exception("L'ID de l'évaluation est requis", 400);
    elseif (!is_numeric($data['idRating'])) throw new
Exception("L'ID de l'évaluation doit être un nombre", 400);
    $rating = self::getRatingById($data['idRating']);
    if ($rating->isEmpty()) throw new

```

```

Exception("Évaluation introuvable", 404);
    if ($rating->getFirstRow()['idUser'] !=
self::getUserByToken()['idUser']) throw new Exception("Vous ne
pouvez pas mettre à jour cette évaluation", 403);
    if (isset($data['score'])) if ($data['score'] < 1 ||
$data['score'] > 5) throw new Exception("Le score doit être
entre 1 et 5", 400);
    $updates = [];
    foreach ($data as $key => $value) {
        if ($key == "idRating") continue;
        if (isset($value) && $value != "") {
            $updates[$key] = $value;
        }
    }
    $sb = "L'évaluation a été mise à jour avec succès :";
    try {
        $query = "UPDATE ratings SET " . implode(", ",
array_map(fn($key) => "$key = :$key", array_keys($updates))) .
" WHERE idRating = :idRating";
        Application::$app->db->execute($query,
array_merge($updates, [":idRating" => $data['idRating']]));
        $sb .= implode(", ", array_map(fn($key, $value) =>
"$key = $value", array_keys($updates),
array_values($updates)));
    } catch (Exception $e) {
        throw new Exception("Erreur lors de la mise à jour
de l'évaluation", 500);
    }
    return $sb;
}

/**
 * Méthode pour supprimer une évaluation existante.
 *
 * @param array $data Les données de l'évaluation à
supprimer.
 * @return string Un message indiquant que l'évaluation a
été supprimée avec succès.
 * @throws Exception Si une erreur se produit lors de la
validation des données ou de l'exécution de la requête.
 */
public static function delete(array $data): string
{
    $data = self::filterArray($data);
    if (!isset($data['idRating'])) throw new
Exception("L'ID de l'évaluation est requis", 400);
    elseif (!is_numeric($data['idRating'])) throw new
Exception("L'ID de l'évaluation doit être un nombre", 400);
    $rating = self::getRatingById($data['idRating']);
    if ($rating->isEmpty()) throw new
Exception("Évaluation introuvable", 404);

```

```

        if ($rating->getFirstRow()['idUser'] !=
self::getUserByToken()['idUser']) throw new Exception("Vous ne
pouvez pas supprimer cette évaluation", 403);
        try {
            Application::$app->db->execute("DELETE FROM
ratings WHERE idRating = :idRating", [":idRating" =>
$data['idRating']]);
        } catch (Exception $e) {
            throw new Exception("Erreur lors de la suppression
de l'évaluation", 500);
        }
        return "Évaluation supprimée avec succès";
    }
}

```

Recipe Api

```
<?php
```

```
namespace Eatfit\Api\Models;
```

```

use DateTime;
use Eatfit\Api\Core\Application;
use Eatfit\Api\Core\Db\SqlResult;
use Eatfit\Api\Core\Model;
use Exception;

```

```

class Recipe extends Model
{

```

```

    /**
     * Récupère une recette ou toutes les recettes en fonction
des données fournies.
     * Si aucun titre de recette spécifique n'est fourni dans
les filtres de recherche, toutes les recettes sont renvoyées.
     * Sinon, elle renvoie la recette spécifique correspondant
au titre.
     * Lance une exception si aucune recette ne correspond au
titre.
     *

```

```

     * @param array $data Les données de recherche
     * @return array|null Les recettes correspondantes
     * @throws Exception En cas d'erreur ou de recette non
trouvée
     */

```

```

    public static function read(array $data): ?array
    {
        if (isset($data['idRecipe'])) {
            $recipe = self::getRecipeById($data['idRecipe']);
            if ($recipe == null) throw new Exception("Recette
non trouvée", 404);

```

```

        return $recipe;
    }
    if (!isset($data['search'])) {
        if (isset($data['filter'])) {
            foreach ($data['filter'] as $filter => $value)
            {
                if ($filter == 'category') return
self::filter($value)->getValues();
                if ($filter == 'food_type') return
self::filter(null, $value)->getValues();
                if ($filter == 'username') return
self::filter(null, null, $value)->getValues();
            }
        }
        return self::getAllRecipes();
    }

    try {
        $recipe = self::search($data['search']);
    } catch (Exception $e) {
        throw new Exception("Erreur lors de la recherche
de recette : " . $e->getMessage(), 500);
    }
    if ($recipe->isEmpty()) throw new Exception("Recette
non trouvée", 404);
    return $recipe->getValues();
}

/**
 * Récupère une recette par son ID.
 *
 * @param int $idRecipe L'ID de la recette
 * @return array|null Les données de la recette ou null si
non trouvée
 * @throws Exception En cas d'erreur
 */
public static function getRecipeById(int $idRecipe):
?array
{
    $query = "SELECT * FROM recipes_details_view WHERE
recipe_id = :idRecipe";
    $statement = Application::$app->db->execute($query,
["idRecipe" => $idRecipe]);
    return $statement->isEmpty() ? null : $statement-
>getFirstRow();
}

/**
 *
 * Filtre les recettes par catégorie ou type d'aliment.
 * @param null $category

```



```

    * @param null $foodType
    * @param null $username
    * @return SqlResult|Exception Les résultats du filtrage
    * @throws Exception En cas d'erreur
    */
    public static function filter($category = null, $foodType
= null, $username = null): SqlResult|Exception
    {
        $sql = "SELECT recipes_details_view.* FROM
recipes_details_view ";
        $where = [];
        $params = [];
        if ($category !== null) {
            $where[] = 'FIND_IN_SET(:category,
recipes_details_view.categories) > 0';
            $params[':category'] = $category;
        }
        if ($foodType !== null) {
            $where[] = 'FIND_IN_SET(:foodType,
recipes_details_view.foodtypes_with_percentages) > 0';
            $params[':foodType'] = $foodType;
        }
        if ($username !== null) {
            $where[] = 'recipes_details_view.creator_username
= :username';
            $params[':username'] = $username;
        }
        if (!empty($where)) {
            $sql .= 'WHERE ' . implode(' AND ', $where) . ' ';
        }
        $sql .= 'ORDER BY `recipes_details_view`.`created_at`
DESC';

        try {
            return Application::$app->db->execute($sql,
$params);
        } catch (Exception $e) {
            throw new Exception("Erreur lors du filtrage des
recettes", 500);
        }
    }

    /**
     * Récupère toutes les recettes.
     *
     * @return array Les recettes
     * @throws Exception En cas d'erreur ou si aucune recette
n'est trouvée
     */
    private static function getAllRecipes(): array

```

```

    {
        $statement = Application::$app->db->execute("SELECT *
FROM recipes_details_view");
        if ($statement->isEmpty()) throw new Exception("Aucune
recette trouvée", 404);
        return $statement->getValues();
    }

/**
 * Recherche des recettes en fonction des filtres fournis.
 *
 * @param string $search La valeur de recherche
 * @return SqlResult Les résultats de la recherche
 * @throws Exception En cas d'erreur
 */
private static function search(string $search): SqlResult
{
    $date = DateTime::createFromFormat('Y-m-d', $search);
    $isDate = $date && $date->format('Y-m-d') === $search;
    $searchForText = "%{$search}%";
    $searchForDate = $isDate ? $search : '';
    $sql = "SELECT recipes_details_view.* FROM
recipes_details_view
WHERE recipes_details_view.recipe_title LIKE
:searchText
OR recipes_details_view.categories LIKE
:searchText
OR DATE_FORMAT(recipes_details_view.created_at,
'%Y-%m-%d') = :searchDate";
    return Application::$app->db->execute($sql,
[':searchText' => $searchForText, ':searchDate' =>
$searchForDate]);
}

/**
 * Crée une nouvelle recette.
 *
 * @param array $data Les données de la recette
 * @return array|Exception Les informations de la recette
créée ou une exception en cas d'erreur
 * @throws Exception En cas d'erreur ou d'authentification
échouée
 */
public static function create(array $data):
array|Exception
{
    $user = self::getUserByToken();
    if (!$user) throw new Exception("Non autorisé", 401);
    self::validateRecipeData($data);
    $data = [
        'title' => $data['title'],

```

```

        'preparation_time' => $data['preparation_time'],
        'difficulty' => $data['difficulty'],
        'instructions' => $data['instructions'],
        'calories' => $data['calories'],
        'created_at' => date("Y-m-d H:i:s"),
        'image' => $data['image'],
        'category' => $data['category'],
        'idUser' => $user['idUser'],
        'food_type' => $data['food_type']
    ];
    try {
        Application::$app->db->beginTransaction();
        self::insertRecipe($data);
        $idRecipe = Application::$app->db->getLastInsertId();
        self::insertRecipeCategories($data, $idRecipe);
        self::insertRecipeFoodType($data, $idRecipe);
        self::insertRecipeImages($data, $idRecipe);
        Application::$app->db->commit();
    } catch (Exception $e) {
        Application::$app->db->rollBack();
        throw new Exception("Erreur lors de la création de
la recette : " . $e->getMessage(), 500);
    }
    return self::getRecipe($data['title'])->getFirstRow();
}

/**
 * Valide les données de la recette.
 *
 * @param array $data Les données de la recette
 * @throws Exception En cas de données invalides ou en cas
d'erreur
 */
private static function validateRecipeData(array $data):
void
{
    $data = self::filterArray($data);
    $categories = Application::$app->db->execute("SELECT *
FROM categories")->getColumn("name");
    $food_types = Application::$app->db->execute("SELECT *
FROM food_types")->getColumn("name");
    $difficulties = ["facile", "moyen", "difficile"];
    if (!is_numeric($data['preparation_time'])) throw new
Exception("Temps de préparation invalide : '" .
$data['preparation_time'] . "'. Le temps de préparation doit
être un nombre", 400);
    if (!in_array($data['difficulty'], $difficulties))
throw new Exception("Difficulté invalide : '" .
$data['difficulty'] . "'. Les difficultés autorisées sont :
facile, moyen, difficile", 400);
}

```

```

        if (!in_array($data['category'], $categories)) throw
new Exception("Catégorie invalide : " . $data['category'] .
". Les catégories autorisées sont : " . implode(", ",
$categories), 400);
        if (!is_array($data['food_type'])) $data['food_type']
= [$data['food_type']]; //A revoir
        $percentage = 0;
        foreach ($data['food_type'] as $food_type) {
            $percentage += $food_type[1];
            if ($food_type[1] < 0 || $food_type[1] > 100)
throw new Exception("Pourcentage invalide : " . $food_type[1]
. " ". Le pourcentage doit être compris entre 0 et 100", 400);
            if (!in_array($food_type[0], $food_types)) throw
new Exception("Type d'aliment invalide : '$food_type[0]'. Les
types d'aliments autorisés sont : " . implode(", ",
$food_types), 400);
        }
        if ($percentage != 100) throw new Exception("La somme
des pourcentages doit être égale à 100", 400);
    }

/**
 * Récupère une recette par son titre.
 *
 * @param string $title Le titre de la recette
 * @return SqlResult Les résultats de la requête
 * @throws Exception En cas d'erreur
 */
private
static function getRecipe(string $title): SqlResult
{
    $query = "SELECT * FROM recipes WHERE title = :title";
    return Application::$app->db->execute($query,
[" :title" => $title]);
}

/**
 * Insère une recette dans la base de données.
 *
 * @param array $data Les données de la recette
 * @param array $user Utilisation
 * @throws Exception En cas d'erreur lors de l'insertion
de la recette
 */
private static function insertRecipe(array $data): void
{
    Application::$app->db->execute("INSERT INTO recipes
(title, preparation_time, difficulty,
instructions, calories, created_at, idUser)
VALUES (:title, :preparation_time, :difficulty,
:instructions, :calories, :created_at, :idUser)",

```

```

        [
            ":title" => $data['title'],
            ":preparation_time" =>
$data['preparation_time'],
            ":difficulty" => $data['difficulty'],
            ":instructions" => $data['instructions'],
            ":calories" => $data['calories'],
            ":created_at" => $data['created_at'],
            ":idUser" => $data['idUser']
        ]);
    }

    /**
     * Insère les catégories de la recette dans la base de
    données.
     *
     * @param array $data Les données de la recette
     * @throws Exception En cas d'erreur lors de l'insertion
    des catégories
     */
    private static function insertRecipeCategories(array
    $data, $idRecipe): void
    {
        Application::$app->db->execute(
            "INSERT INTO recipe_categories (idRecipe,
            idCategory) VALUES (:idRecipe, (SELECT idCategory FROM
            categories WHERE name = :name))",
            [
                ":idRecipe" => $idRecipe,
                ":name" => $data['category']
            ]
        );
    }

    /**
     * Insère les types d'aliments de la recette dans la base
    de données.
     *
     * @param array $data Les données de la recette
     * @throws Exception En cas d'erreur lors de l'insertion
    des types d'aliments
     */
    private static function insertRecipeFoodType(array $data,
    $idRecipe): void
    {
        $values = [];
        $parameters = [];
        foreach ($data['food_type'] as $index => $food_type) {
            $values[] = "((SELECT idRecipe FROM recipes WHERE
            idRecipe = :idRecipe), (SELECT idFoodType FROM food_types
            WHERE name = :name{$index}), :percentage{$index})";
        }
    }

```

```

        $parameters[":name{$index}"] = $food_type[0];
        $parameters[":percentage{$index}"] =
$food_type[1];
    }
    $query = "INSERT INTO recipe_food_types (idRecipe,
idFoodType, percentage) VALUES " . implode(", ", $values);
    $parameters[":idRecipe"] = $idRecipe;
    Application::$app->db->execute($query, $parameters);

}

/**
 * Insère les images de la recette dans la base de
données.
 *
 * @param array $data Les données de la recette
 * @throws Exception En cas d'erreur lors de l'insertion
des images
 */
private static function insertRecipeImages(array $data,
$idRecipe): void
{
    if ($data['image'] == "") {
        $image = Application::$app->db->execute(
            "SELECT image_path FROM categories WHERE name
= :name",
            [":name" => $data['category']] -
>getFirstRow() ['image_path'];
        Application::$app->db->execute(
            "SELECT insert_unique_image_name(:path,
(SELECT idRecipe FROM recipes WHERE idRecipe = :idRecipe));",
            [":path" => $image, ":idRecipe" => $idRecipe]
        );
    } else {
        foreach ($data['image'] as $images) {
            $image = explode(",", $images);
            $imageSql = Application::$app->db->execute(
                "SELECT insert_unique_image_name(:path,
(SELECT idRecipe FROM recipes WHERE title = :title)) as
path;",
                [":path" => trim($image[0]), ":title" =>
$data['title']]
            );
            file_put_contents(Application::$UPLOAD_PATH .
$imageSql->getFirstRow() ['path'],
base64_decode(trim($image[1])));
        }
    }
}

/**

```

```

    * Met à jour une recette existante.
    *
    * @param array $data Les données de mise à jour
    * @return string Message de succès
    * @throws Exception En cas d'erreur ou d'authentification
    échouée
    */
    public
    static function update(array $data): string
    {
        $data = self::filterArray($data);
        if (!isset($data['idRecipe'])) throw new
Exception("L'idRecipe n'a pas été renseigné", 400);
        $recipeResult =
self::getRecipeById($data['idRecipe']);
        if ($recipeResult == null) throw new
Exception("Recette non trouvée", 404);
        if ($recipeResult['creator_id'] !=
self::getUserByToken()['idUser']) throw new Exception("Non
autorisé", 401);

        $updates = [];
        if (isset($data['difficulty'])) {
            $difficulties = ["facile", "moyen", "difficile"];
            if (!in_array($data['difficulty'], $difficulties))
throw new Exception("Difficulté invalide : '" .
$data['difficulty'] . "'. Les difficultés autorisées sont :
facile, moyen, difficile", 400);
        }
        if (isset($data['food_type'])) {
            if (!is_array($data['food_type']))
$data['food_type'] = [$data['food_type']]; //A revoir
        }
        if (isset($data['preparation_time'])) {
            if (!is_numeric($data['preparation_time'])) throw
new Exception("Temps de préparation invalide : '" .
$data['preparation_time'] . "'. Le temps de préparation doit
être un nombre", 400);
        }
        if (isset($data['category'])) {
            $categories = Application::$app->db-
>execute("SELECT * FROM categories")->getColumn("name");
            if (!in_array($data['category'], $categories))
throw new Exception("Catégorie invalide : '" .
$data['category'] . "'. Les catégories autorisées sont : " .
implode(", ", $categories), 400);
        }
        foreach ($data as $key => $value) {
            if ($key == "idRecipe") continue;
            if ($key == "food_type") {
                $updates[$key] = $value;
            }
        }
    }

```

```

        continue;
    }
    if (isset($value) && $value != "") $updates[$key]
= $value;
    }
    $sb = "Recette mise à jour avec succès";
    try {

        if (isset($updates['food_type'])) {
            self::updateFoodtype((int)$data['idRecipe'],
$updates);
            unset($updates['food_type']);
        }
        if (isset($data['category'])) {
            Application::$app->db->execute("DELETE FROM
recipe_categories WHERE idRecipe = :idRecipe", [":idRecipe" =>
$data['idRecipe']]);
            self::insertRecipeCategories($data,
$data['idRecipe']);
            unset($updates['category']);
        }
        if (isset($data['image'])) {
            $paths = Application::$app->db->execute("CALL
DeleteImages(:idRecipe)", [":idRecipe" => $data['idRecipe']]) -
>getColumn("ImageID_Path");
            foreach ($paths as $path)
unlink(Application::$UPLOAD_PATH . $path);
            if (!isset($data['category']))
$data['category'] = $recipeResult['categories'];
            self::insertRecipeImages($data,
$data['idRecipe']);
            unset($updates['image']);
        }
        if (count($updates) > 0) {
            $query = "UPDATE recipes SET " . implode(", ",
array_map(fn($key) => "$key = :$key", array_keys($updates))) .
" WHERE idRecipe = :idRecipe";
            Application::$app->db->execute($query,
array_merge($updates, [":idRecipe" => $data['idRecipe']]));
        }
    } catch (Exception $e) {
        throw new Exception($e->getMessage(), 500);
    }
    return $sb;
}

/**
 * @throws Exception
 */
private static function updateFoodtype($idRecipe,
$updates): void

```



```

    {
        Application::$app->db->execute("DELETE FROM
recipe_food_types WHERE idRecipe = :idRecipe", [":idRecipe" =>
$idRecipe]);
        self::insertRecipeFoodType($updates, $idRecipe);
    }

/**
 * Supprime une recette.
 *
 * @param array $data Les données de la recette à
supprimer
 * @return string Message de succès
 * @throws Exception En cas d'erreur ou d'authentification
échouée
 */
public static function delete(array $data): string
{
    $recipe = self::getRecipeById($data['idRecipe']);
    if ($recipe == null) throw new Exception("Recette non
trouvée", 404);
    if ($recipe['creator_id'] !=
self::getUserByToken()['idUser']) throw new Exception("Non
autorisé", 401);
    try {
        //Call procedure
        $delete = Application::$app->db->execute("CALL
delete_recipe(:idRecipe)", [":idRecipe" =>
$data['idRecipe']]);
        if ($delete->isEmpty()) throw new
Exception("Erreur lors de la suppression de la recette", 500);
        foreach ($delete->getValues() as $key => $value) {
            if (str_contains($value["CONCAT(idImage, '_',
path)"], "default")) continue;
            unlink(Application::$UPLOAD_PATH .
$value["CONCAT(idImage, '_', path)"]);
        }
    } catch (Exception $e) {
        throw new Exception("Erreur lors de la suppression
de la recette", 500);
    }
    // Retournez un message de succès
    return "Recette supprimée avec succès";
}

/**
 * Ajoute un type d'aliment.
 *
 * @param array $data Les données du type d'aliment
 * @return string Message de succès
 * @throws Exception En cas d'erreur ou d'authentification

```

```

échouée
    */
    public
    static function addFoodType(array $data)
    {
        $user = self::getUserByToken();
        if (!$user) throw new Exception("Unauthorized", 401);
        if (!Application::$app->db->execute("SELECT * FROM
food_types WHERE name = :name", [":name" => $data['name']])->isEmpty()) throw new Exception("Le type d'aliment existe déjà", 400);
        Application::$app->db->execute("INSERT INTO
food_types(name) VALUES(:name)", [":name" => $data['name']]);
        return "Type d'aliment ajouté avec succès";
    }

    /**
     * Récupère les catégories de recettes.
     *
     * @return array|null Les catégories de recettes ou null
    si non trouvées
     * @throws Exception En cas d'erreur
    */
    public
    static function getCategories(): ?array
    {
        $query = "SELECT * FROM categories";
        $statement = Application::$app->db->execute($query);
        return $statement->isEmpty() ? throw new
Exception("Aucune catégorie trouvée", 404) : $statement->
getValues();
    }

    /**
     * Récupère les types d'aliments.
     *
     * @return array|null Les types d'aliments ou null si non
    trouvés
     * @throws Exception En cas d'erreur
    */
    public static function getFoodTypes(): ?array
    {
        $query = "SELECT * FROM food_types";
        $statement = Application::$app->db->execute($query);
        return $statement->isEmpty() ? throw new
Exception("Aucun type d'aliment trouvé", 404) : $statement->
getValues();
    }
}

```

User Api

<?php

```
namespace Eatfit\Api\Models;

use Eatfit\Api\Core\Application;
use Eatfit\Api\Core\Db\SqlResult;
use Eatfit\Api\Core\Model;
use Exception;

class User extends Model
{
    /**
     * Récupère les informations d'un utilisateur existant
     * dans la base de données.
     *
     * @param array $data Un tableau contenant l'email et le
     * mot de passe de l'utilisateur.
     * @return array|Exception Les informations de
     * l'utilisateur ou une exception en cas d'erreur.
     * @throws Exception
     */
    public static function read(array $data): array|Exception
    {
        return self::getUserInfo($data['email'],
            $data['password']);
    }

    /**
     * Vérifie si un utilisateur existe dans la base de
     * données.
     *
     * @param $email
     * @param string $password
     * @return array|Exception Les informations de
     * l'utilisateur si l'utilisateur existe, une exception sinon.
     * @throws Exception
     */
    private static function getUserInfo($email, string
        $password): array|Exception
    {
        $user = Application::$app->db->execute("SELECT * FROM
            users WHERE email = :email", [":email" => $email]);

        if ($user->isEmpty()) throw new Exception("Utilisateur
            ou mot de passe invalide", 400);
        $user = $user->getFirstRow();
        if (password_verify($password, $user['password'])) {
            unset($user['password']);
            if ($user['expiration'] < time())

```

```

$user['expiration'] = 'expiré';
    return $user;
}
    throw new Exception("Utilisateur ou mot de passe
    invalide", 400);
}

/**
 * Crée un nouvel utilisateur dans la base de données.
 *
 * @param array $data Un tableau contenant les
    informations de l'utilisateur.
 * @return array|Exception Les informations de
    l'utilisateur créé ou une exception en cas d'erreur.
 * @throws Exception
 */
    public static function create(array $data):
    array|Exception
    {
        $email = filter_var($data['email'],
        FILTER_VALIDATE_EMAIL);
        if (!$email) throw new Exception("Email invalide",
        400);

        $data = self::filterArray($data);
        if (strlen($data['password']) < 8) throw new
        Exception("Le mot de passe doit comporter au moins 8
        caractères", 400);
        if (!self::getUser($email, $data['username'])-
        >isEmpty()) throw new Exception("L'utilisateur existe déjà",
        409);

        if ($data['password'] != $data['confirm_password'])
        throw new Exception("Les mots de passe ne correspondent pas",
        400);

        $data = [
            'username' => $data['username'],
            'email' => $email,
            'password' => password_hash($data['password'],
        PASSWORD_DEFAULT),
            'token' => self::generateJWT($data),
            'expiration' => time() + (2 * 3600)
        ];
        try {
            Application::$app->db->execute("INSERT INTO users
            (username, email, password, token, expiration) VALUES
            (:username, :email, :password, :token, :expiration)",
            [":username" => $data['username'], ":email" => $data['email'],
            ":password" => $data['password'], ":token" => $data['token'],
            ":expiration" => $data['expiration']]);
        } catch (Exception $e) {
            throw new Exception("Erreur lors de la création de
            l'utilisateur", 500);
        }
    }
}

```

```

    }
    return self::getUser($data['email'],
    $data['username'])->getFirstRow();
}

/**
 * Récupère les informations d'un utilisateur à partir de
son adresse e-mail.
 *
 * @param string|null $email L'adresse e-mail de
l'utilisateur.
 * @param string|null $username
 * @return SqlResult Les informations de l'utilisateur.
 * @throws Exception
 */
public static function getUser(string $email = null,
string $username = null): SqlResult
{
    $query = "SELECT * FROM users ";
    $params = [];
    if ($email != null && $username == null) {
        $query .= " WHERE email = :email";
        $params = [":email" => $email];
    } elseif ($username != null && $email == null) {
        $query .= " WHERE username = :username";
        $params = [":username" => $username];
    } else {
        $query .= "WHERE email = :email OR username =
:username";
        $params = [":email" => $email, ":username" =>
$username];
    }
    try {
        return Application::$app->db->execute($query,
$params);
    } catch (Exception $e) {
        throw new Exception("Un problème est survenu",
500);
    }
}

/**
 * Met à jour les informations d'un utilisateur existant.
 *
 * @param array $data Un tableau contenant les
informations mises à jour de l'utilisateur.
 * @return array|string Un message indiquant le succès de
la mise à jour.
 * @throws Exception
 */
public static function update(array $data): array|string

```

```

{
    $token = null;
    $user = parent::getUserByToken(false);
    $idUser = $user['idUser'];
    $updates = [];
    if ($data == null) throw new Exception("Aucune donnée
à mettre à jour", 400);
    $data = self::filterArray($data);
    if (isset($data['email']) && $data['email'] != "") {
        if (!filter_var($data['email'],
FILTER_VALIDATE_EMAIL)) throw new Exception("Email invalide",
400);
        if (self::getUser($data['email'])->isEmpty())
$updates['email'] = $data['email'];
        else throw new Exception("Cet utilisateur existe
déjà", 409);
    }
    if (isset($data['username']) && $data['username'] !=
"") {
        if (self::getUser(null, $data['username'])-
>isEmpty()) $updates['username'] = $data['username'];
        else throw new Exception("Cet utilisateur existe
déjà", 409);
    }
    if (isset($data['password']) && $data['password'] !=
"") {
        if (strlen($data['password']) < 8) throw new
Exception("Le mot de passe doit comporter au moins 8
caractères", 400);
        if (!isset($data['confirm_password']) ||
$data['password'] != $data['confirm_password']) throw new
Exception("Les mots de passe ne correspondent pas", 400);
        $updates['password'] =
password_hash($data['password'], PASSWORD_DEFAULT);
    }
    $sb = "";
    if (count($updates) > 0) {
        $sb = "Utilisateur mis à jour avec succès :";
        try {
            unset($user['idUser'], $user['token']);
            $updatedUserData = array_merge($user,
$updates);
            $newToken =
self::generateJWT($updatedUserData,
$updatedUserData['expiration']);
            foreach ($updates as $key => $value) $sb .= "
$key = $value";
            $token = $newToken;
            $updates['token'] = $newToken;
            Application::$app->db->execute("UPDATE users
SET " . implode(", ", array_map(fn($key) => "$key = :$key",

```

```

array_keys($updates))) . " WHERE idUser = :idUser",
array_merge($updates, [":idUser" => $idUser]));
        } catch (Exception $e) {
            throw new Exception("Erreur lors de la mise à
jour de l'utilisateur", 500);
        }
    }
    if ($token != null) return ["Mises à jour" => $sb,
"token" => $token];
    return $sb;
}

/**
 * Récupère les informations d'un utilisateur à partir du
jeton d'authentification.
 *
 * @param bool $expiration Vérifie l'expiration du jeton
si true.
 * @return array|null Les informations de l'utilisateur,
ou null si l'utilisateur n'est pas trouvé.
 * @throws Exception Si le jeton est invalide.
 */
public static function getUserByToken(bool $expiration =
true): ?array
{
    return parent::getUserByToken($expiration);
}

/**
 * Supprime un utilisateur existant dans la base de
données.
 *
 * @return string Un message indiquant que la suppression
a été effectuée avec succès.
 * @throws Exception
 */
public static function delete(): string
{
    $user = parent::getUserByToken();
    try {
        Application::$app->db->execute("DELETE FROM users
WHERE idUser = :idUser", [":idUser" => $user['idUser']]);
    } catch (Exception $e) {
        throw new Exception("Erreur lors de la suppression
de l'utilisateur", 500);
    }
    return "Utilisateur supprimé avec succès";
}

/**
 * Authentifie un utilisateur et génère un nouveau jeton

```

```

d'authentification si nécessaire.
    *
    * @param array $data Un tableau contenant l'email et le
mot de passe de l'utilisateur.
    * @return array|Exception Les informations de
l'utilisateur authentifié ou une exception en cas d'erreur.
    * @throws Exception
    */
    public static function authenticate(array $data):
array|Exception
    {
        $email = filter_var($data['email'],
FILTER_VALIDATE_EMAIL);
        $password = filter_var($data['password'],
FILTER_SANITIZE_SPECIAL_CHARS);
        if (!$email || !$password) throw new Exception("Email
ou mot de passe invalide", 400);
        $user = self::getUserInfo($email, $password);
        try {
            if (self::isTokenExpired($user['expiration'])) {
                $user['token'] = self::generateJWT($user);
                $user['expiration'] = time() + (2 * 3600);
                Application::$app->db->execute("UPDATE users
SET token = :token, expiration = :expiration WHERE idUser =
:idUser", [":token" => $user['token'], ":expiration" =>
$user['expiration'], ":idUser" => $user['idUser']]);
            }
        } catch (Exception $e) {
            throw new Exception("Erreur lors de la mise à jour
de l'utilisateur", 500);
        }
        unset($user['password']);
        return $user;
    }
}

```

Htaccess Api

Options -MultiViews

RewriteEngine On

RewriteBase /

Ignore les fichiers et les dossiers réels

RewriteCond %{REQUEST_FILENAME} !-f

RewriteCond %{REQUEST_FILENAME} !-d

Redirige toutes les requêtes vers index.php

RewriteRule ^(.*)\$ index.php/\$1 [L,QSA]

Index Api

<?php

```
use Eatfit\Api\Controllers\HistoryController;
use Eatfit\Api\Controllers\RatingController;
use Eatfit\Api\Controllers\RecipeController;
use Eatfit\Api\Controllers\UserController;
use Eatfit\Api\Core\ApiValue;
use Eatfit\Api\Core\Application;
use Eatfit\Api\Core\EnvLoader;

require_once __DIR__ . '/../vendor/autoload.php';

// Chargement des variables d'environnement à partir du
fichier .env
$dotenv = new EnvLoader(dirname(__DIR__) . '/.env');
try {
    $dotenv->load();
} catch (Exception $e) {
    echo new ApiValue(null, $e->getCode(), $e->getMessage());
    return;
}

// Configuration de l'application
$config = [
    'db' => [
        'dsn' => $_ENV['DB_DSN'],
        'user' => $_ENV['DB_USER'],
        'password' => $_ENV['DB_PASSWORD'],
    ],
    'UPLOAD_PATH' => $_ENV['UPLOAD_PATH'],
];

// Création de l'instance de l'application
$app = new Application(dirname(__DIR__), $config);

// Définition des routes et des contrôleurs associés

// Route racine
$app->router->get('/', function (): ApiValue {
    return new ApiValue(Application::$app->router-
>getRoutes(), 200);
});

// Authentification de l'utilisateur
$app->router->put('/login', [UserController::class, 'login']);

// Récupère les informations de l'utilisateur actuel
$app->router->get('/userById', [UserController::class,
'getUserByToken']);
```

```
// Routes pour le contrôleur HistoryController
$app->router->get('history', [HistoryController::class,
'read']);
$app->router->post('history', [HistoryController::class,
'create']);
$app->router->put('history', [HistoryController::class,
'update']);
$app->router->delete('history', [HistoryController::class,
'delete']);

// Routes pour le contrôleur UserController
$app->router->get('/user', [UserController::class, 'read']);
$app->router->post('/user', [UserController::class,
'create']);
$app->router->put('/user', [UserController::class, 'update']);
$app->router->delete('/user', [UserController::class,
'delete']);

// Routes pour le contrôleur RecipeController
$app->router->get('/recipe', [RecipeController::class,
'read']);
$app->router->post('/recipe', [RecipeController::class,
'create']);
$app->router->put('/recipe', [RecipeController::class,
'update']);
$app->router->delete('/recipe', [RecipeController::class,
'delete']);

// Routes pour le contrôleur RatingController
$app->router->get('/rating', [RatingController::class,
'read']);
$app->router->post('/rating', [RatingController::class,
'create']);
$app->router->put('/rating', [RatingController::class,
'update']);
$app->router->delete('/rating', [RatingController::class,
'delete']);

// Autres routes
$app->router->post('/food_type', [RecipeController::class,
'addFoodType']);
$app->router->get('/food_types', [RecipeController::class,
'getFoodTypes']);
$app->router->get('/categories', [RecipeController::class,
'getCategories']);

// Lancement de l'application
$app->run();
```

Composer.json Api

```
{
    "name": "eatfit/api",
    "autoload": {
        "psr-4": {
            "Eatfit\\Api\\": "./"
        }
    },
    "authors": [
        {
            "name": "LAAC172003",
            "email": "61194486+LAAC172003@users.noreply.github.com"
        }
    ],
    "require": {
        "ext-pdo": "*"
    }
}
```

HistoryController Site

```
<?php
```

```
namespace Eatfit\Site\Controllers;

use Eatfit\Site\Core\Application;
use Eatfit\Site\Core\Controller;
use Eatfit\Site\Core\Middlewares\AuthMiddleware;
use Eatfit\Site\Core\Request;
use Eatfit\Site\Models\History;

class HistoryController extends Controller
{
    public function __construct()
    {
        $this->registerMiddleware(new
AuthMiddleware(['history', 'deleteHistory', 'addToHistory']));
    }

    /**
     * Affiche l'historique des recettes consultées par
     l'utilisateur.
     *
     * @return string Le contenu HTML de la page d'historique.
     */
    public function history(): string
    {
        $historyModel = new History();
        return $this->render('history', ['model' =>
$historyModel]);
    }
}
```

```

    }

    /**
     * Supprime une recette de l'historique.
     *
     * @param Request $request La requête HTTP.
     */
    public function deleteHistory(Request $request): void
    {
        $historyModel = new History();
        $historyModel->idUser = Application::$app->user->idUser;
        if (isset($request->getRouteParams()['idConsumedRecipe'])) $historyModel->deleteHistory($request->getRouteParams()['idConsumedRecipe']);
        else $historyModel->deleteAllHistory();
        Application::$app->session->setFlash("success", "L'historique a bien été supprimé");
        Application::$app->response->redirect('/history');
    }

    /**
     * Ajoute une recette à l'historique.
     *
     * @param Request $request La requête HTTP.
     */
    public function addToHistory(Request $request): void
    {
        if (Application::$isGuest()) {
            Application::$app->session->setFlash("error", "Vous devez être connecté pour accéder à cette page");
            Application::$app->response->redirect('/login');
        }
        $historyModel = new History();
        $historyModel->idRecipe = $request->getRouteParams()['idRecipe'];
        $historyModel->save();
        Application::$app->session->setFlash("success", "La recette a bien été ajoutée à l'historique");
        Application::$app->response->redirect('/recipe/detail/' . $request->getRouteParams()['idRecipe']);
    }
}

```

RecipeController Site

```
<?php
```

```
namespace Eatfit\Site\Controllers;
```

```

use Eatfit\Site\Core\Application;
use Eatfit\Site\Core\Controller;
use Eatfit\Site\Core\Exception\ForbiddenException;
use Eatfit\Site\Core\Exception\NotFoundException;
use Eatfit\Site\Core\Middlewares\AuthMiddleware;
use Eatfit\Site\Core\Request;
use Eatfit\Site\Models\FoodType;
use Eatfit\Site\Models\Rating;
use Eatfit\Site\Models\Recipe;

class RecipeController extends Controller
{
    public function __construct()
    {
        $this->registerMiddleware(new
AuthMiddleware(['create', 'update', 'delete', 'deleteRating',
'addFoodType', 'updateRating']));
    }

    /**
     * Affiche les détails d'une recette.
     *
     * @param Request $request La requête HTTP.
     * @return string Le contenu HTML de la page de détails de
recette.
     */
    public function detail(Request $request): string
    {
        $ratings = new Rating();
        $model = new Recipe();
        $recipe = $model->getRecipe("idRecipe", $request-
>getRouteParams()['idRecipe'], false)->value;
        $ratings->idRecipe = $request-
>getRouteParams()['idRecipe'];
        $recipe->image_paths = !empty($recipe->image_paths) &&
str_contains($recipe->image_paths, ',') ? array_map('trim',
explode(',', $recipe->image_paths)) : array($recipe-
>image_paths);
        $recipe->categories = !empty($recipe->categories) &&
str_contains($recipe->categories, ',') ? array_map('trim',
explode(',', $recipe->categories)) : array($recipe-
>categories);
        $recipe->foodtypes_with_percentages = !empty($recipe-
>foodtypes_with_percentages) && str_contains($recipe-
>foodtypes_with_percentages, ',') ? array_map('trim',
explode(',', $recipe->foodtypes_with_percentages)) :
array($recipe->foodtypes_with_percentages);
        if (Application::isGuest()) {
            return $this->render(

```

```

        'recipe_details_guest', [
            'recipe' => $recipe, 'ratings' => $ratings
        ]);
    }
    $ratings->idUser = Application::$app->user->idUser;
    if ($request->isPost()) {
        if (!isset($_POST['score'])) {
            Application::$app->session->setFlash('error',
                'Veuillez sélectionner une note');
            return $this->render(
                'recipe_details', [
                    'recipe' => $recipe, 'ratings' => $ratings
                ]);
        }
        $ratings->score = $_POST['score'];
        if (!isset($_POST['comment'])) $ratings->comment =
null;
        else $ratings->comment = $_POST['comment'];
        if ($ratings->validate()) {
            if ($ratings->create()->value == null)
Application::$app->session->setFlash('error', $ratings-
>create()->message);
            else Application::$app->session-
>setFlash('success', 'Votre évaluation a été enregistrée');
            Application::$app->response-
>redirect('/recipe/detail/' . $request-
>getRouteParams()['idRecipe']);
        }
    }
    return $this->render(
        'recipe_details', [
            'recipe' => $recipe, 'ratings' => $ratings
        ]);
}

/**
 * Crée une nouvelle recette.
 *
 * @param Request $request La requête HTTP.
 * @return string Le contenu HTML de la page de création
de recette.
 */
public function create(Request $request): string
{
    $model = new Recipe();
    if (Application::$app->request->isPost()) {
        $model->image = $_FILES;
        $model->idUser = Application::$app->user->idUser;
        if (!isset($_POST['foodtype'])) ||
!isset($_POST['percentage']) Application::$app->session-
>setFlash('error', 'Veuillez sélectionner au moins un type de

```

```

nourriture');
        else {
            $foodTypes = $_POST['foodtype'];
            $percentages = $_POST['percentage'];
            $totalPercentage = array_sum($percentages);
            if ($totalPercentage != 100) {
                Application::$app->session->setFlash('error', 'La somme des pourcentages doit être de 100');
                return $this->render('add_recipe', ['model' => $model]);
            }
            if (count($foodTypes) !== count(array_unique($foodTypes))) {
                Application::$app->session->setFlash('error', 'Les types de nourriture ne peuvent pas se répéter');
                return $this->render('add_recipe', ['model' => $model]);
            }
            foreach ($foodTypes as $key => $value) {
                $model->foodType[] = [$value, $percentages[$key]];
            }

            $data = $request->getBody();
            $data = array_merge($data, [['image' => $model->image], ['idUser' => $model->idUser], [['food_type' => $model->foodType]]);
            $model->categories = [$data['categories']];
            $model->loadData($data);
            if ($model->validate()) {
                $model->create();
                Application::$app->session->setFlash('success', 'Recette ajoutée avec succès');
                Application::$app->response->redirect('/');
            }
        }
        return $this->render('add_recipe', ['model' => $model]);
    }

    /**
     * Supprime une évaluation d'une recette.
     *
     * @param Request $request La requête HTTP.
     * @throws ForbiddenException
     */
    public function deleteRating(Request $request): void

```

```

{
    $idRating = $request->getRouteParams()['idRating'];
    $model = new Rating();
    $model->idRating = $idRating;
    $rating = $model->getRatingById();
    $model->idRecipe = $rating->value[0]->idRecipe;
    if ($rating->value[0]->idUser != Application::$app->user->idUser) throw new ForbiddenException();
    $apiResponse = $model->delete();
    Application::$app->session->setFlash('success',
    $apiResponse->value);
    Application::$app->response->redirect('/recipe/detail/' . $model->idRecipe);
}

/**
 * Supprime une recette.
 *
 * @param Request $request La requête HTTP.
 * @throws ForbiddenException
 * @throws NotFoundException
 */
public function delete(Request $request): void
{
    $model = new Recipe();
    $model->idRecipe = $request->getRouteParams()['idRecipe'];
    $recipe = $model->getRecipe("idRecipe", $request->getRouteParams()['idRecipe']);
    $model->idUser = $recipe->value->creator_id;
    if (!$recipe) throw new NotFoundException();
    if (Application::$app->user->idUser != $model->idUser) throw new ForbiddenException();
    $model->delete();
    Application::$app->session->setFlash('success', 'La recette a été supprimée');
    Application::$app->response->redirect('/');
}

/**
 * Met à jour une évaluation d'une recette.
 *
 * @param Request $request La requête HTTP.
 * @return string Le contenu HTML de la page de modification d'évaluation.
 * @throws ForbiddenException
 */
public function updateRating(Request $request): string
{
    $idRating = $request->getRouteParams()['idRating'];
    $model = new Rating();

```



```

$model->idRating = $idRating;
$test = $model->getRatingById();

$model->idRecipe = $test->value[0]->idRecipe;
$model->score = $test->value[0]->score;
$model->comment = $test->value[0]->comment;
if ($test->value[0]->idUser != Application::$app->user->idUser) throw new ForbiddenException();
if ($request->isPost()) {
    if (isset($_POST['score'])) $model->score =
$_POST['score'];
    if (isset($_POST['comment'])) $model->comment =
$_POST['comment'];
    $apiResponse = $model->update();
    if ($apiResponse->value == null)
Application::$app->session->setFlash('error', $apiResponse->message);
    else Application::$app->session->setFlash('success', "Votre évaluation a été modifiée");
    Application::$app->response->redirect('/recipe/detail/' . $model->idRecipe);
}
return $this->render('edit_rating', ['model' =>
$model]);
}

/**
 * Met à jour une recette existante.
 *
 * @param Request $request La requête HTTP.
 * @return string Le contenu HTML de la page de
modification de recette.
 * @throws ForbiddenException
 */
public function update(Request $request): string
{
    $model = new Recipe();
    $recipe = $model->getRecipe("idRecipe", $request->getRouteParams()['idRecipe']->value;
    $recipe->image_paths = !empty($recipe->image_paths) &&
str_contains($recipe->image_paths, ',') ? array_map('trim',
explode(',', $recipe->image_paths)) : array($recipe->image_paths);
    $recipe->categories = !empty($recipe->categories) &&
str_contains($recipe->categories, ',') ? array_map('trim',
explode(',', $recipe->categories)) : array($recipe->categories);
    $recipe->foodtypes_with_percentages = !empty($recipe->foodtypes_with_percentages) &&
str_contains($recipe->foodtypes_with_percentages, ',') ? array_map('trim',
explode(',', $recipe->foodtypes_with_percentages)) :

```

```

array($recipe->foodtypes_with_percentages);
    $foodtypes = [];
    $foodtypesArray = [];
    foreach ($recipe->foodtypes_with_percentages as $item)
    {
        if (empty($item)) continue;
        list($type, $percentage) = explode('(',
rtrim($item, ')'));
        $percentage = (int)$percentage;
        if ($percentage > 0) $foodtypesArray[trim($type)]
= $percentage;
    }
    $model->idRecipe = $recipe->recipe_id;
    $model->image = $recipe->image_paths;
    $model->idUser = $recipe->creator_id;
    $model->categories = $recipe->categories;
    $model->title = $recipe->recipe_title;
    $model->instructions = $recipe->recipe_instructions;
    $model->calories = $recipe->calories;
    $model->date = $recipe->created_at;
    $model->preparation_time = $recipe->preparation_time;
    $model->foodType = $foodtypesArray;
    if ($model->idUser != Application::$app->user->idUser)
throw new ForbiddenException();

    if (Application::$app->request->isPost()) {
        $data = $request->getBody();
        $data = array_filter($data);
        $_POST = array_filter($_POST);

        if (isset($_POST['percentage'])) {
            $percentageMax = 0;
            foreach ($_POST['percentage'] as $key =>
$value) {
                if (empty($value)) continue;
                $percentageMax += (int)$value;
                $foodtypes[] = [$key, (int)$value];
            }
            if ($percentageMax != 100) {
                Application::$app->session->setFlash('error', 'La somme des pourcentages doit être égale à 100');
                Application::$app->response->redirect('/recipe/edit/' . $model->idRecipe);
            }
            $data = array_merge($data, ['food_type' =>
$foodtypes]);
        }
        $data['idRecipe'] = $request->getRouteParams()['idRecipe'];
        $validated = $model->

```

```

>validateAndPrepareImages($_FILES);
    $images = [];
    foreach ($validated as $name => $base64) $images[]
= $name . "," . $base64;
    if (!empty($validated)) $data['image'] = $images;
    else {
        if (isset($_POST['default'])) {
            if ($_POST['default'] == "on") {
                unset($data['default']);
                $data['image'] = "";
            }
        } elseif (str_contains($recipe-
>image_paths[0], 'default')) {
            $data['image'] = "";
        }
    }
    if (isset($data['categories'])) {
        $data['category'] = $data['categories'];
        unset($data['categories']);
    }
    $model->update($data);
    Application::$app->response-
>redirect('/recipe/detail/' . $model->idRecipe);
}
return $this->render('edit_recipe', [
    'model' => $model, 'recipe' => $recipe
]);
}

/**
 * Ajoute un nouveau type de nourriture.
 *
 * @param Request $request La requête HTTP.
 * @return string Le contenu HTML de la page d'ajout de
type de nourriture.
 */
public function addFoodType(Request $request): string
{
    $model = new FoodType();
    if ($request->isPost()) {
        $model->loadData($request->getBody());
        if ($model->validate()) {
            $apiResponse = $model->save();
            if ($apiResponse->value == null)
Application::$app->session->setFlash('error', $apiResponse-
>message);
            else Application::$app->session-
>setFlash('success', "Votre évaluation a été modifiée");
            Application::$app->response->redirect('/');
        }
    }
}

```

```
        return $this->render('add_foodtype', ['model' =>
$model]);
    }
}
```

SiteController Site

<?php

```
namespace Eatfit\Site\Controllers;

use Eatfit\Site\Core\Controller;
use Eatfit\Site\Models\Recipe;

class SiteController extends Controller
{
    /**
     * Affiche la page d'accueil.
     *
     * @return string Le contenu HTML de la page d'accueil.
     */
    public function home(): string
    {
        $recipeModel = new Recipe();
        return $this->render('home', [
            'model' => $recipeModel
        ]);
    }
}
```

UserController Site

<?php

```
namespace Eatfit\Site\Controllers;

use Eatfit\Site\Core\Application;
use Eatfit\Site\Core\Controller;
use Eatfit\Site\Core\Middlewares\AuthMiddleware;
use Eatfit\Site\Core\Request;
use Eatfit\Site\Models\LoginForm;
use Eatfit\Site\models\ProfileModel;
use Eatfit\Site\Models\User;
use Exception;

class UserController extends Controller
{
    public function __construct()
    {
        $this->registerMiddleware(new
AuthMiddleware(['profile']));
    }
}
```

```

    }

    /**
     * Affiche la page de profil de l'utilisateur.
     *
     * @param Request $request La requête HTTP.
     * @return string Le contenu HTML de la page de profil.
     * @throws Exception En cas d'erreur lors de la mise à
    jour du profil.
    */
    public function profile(Request $request): string
    {
        $profileModel = new ProfileModel();
        if (isset($request->getRouteParams()['method'])) {
            if ($request->getRouteParams()['method'] ==
'update') {

                $profileModel->loadData($request->getBody());
                $apiResponse = $profileModel->update();
                if ($apiResponse->message == "OK" &&
$apiResponse->code == 200) {
                    Application::$app->session-
>setFlash('success', 'Le profil a bien été mis à jour');
                    Application::$app->session->set("user",
$apiResponse->value->token);
                } else Application::$app->session-
>setFlash('error', $apiResponse->message);
                Application::$app->response-
>statusCode($apiResponse->code);
                Application::$app->response-
>redirect('/profile');
            }
            if ($request->getRouteParams()['method'] ==
"delete") {
                $apiResponse = $profileModel->delete();
                Application::$app->session-
>setFlash('success', $apiResponse->value);
                Application::$app->response-
>statusCode($apiResponse->code);
                Application::$app->response->redirect('/');
                Application::$app->logout();
            }
        }
        return $this->render('profile', ['model' =>
$profileModel, 'user' => Application::$app->user]);
    }

    /**
     * Déconnecte l'utilisateur et le redirige vers la page
    d'accueil.
    */

```

```

public function logout(): void
{
    Application::$app->logout();
    Application::$app->response->redirect('/');
}

/**
 * Affiche la page d'inscription.
 *
 * @param Request $request La requête HTTP.
 * @return string Le contenu HTML de la page
d'inscription.
 */
public function register(Request $request): string
{
    if (!Application::$isGuest()) Application::$app->response->redirect('/');
    $registerModel = new User();
    if ($request->getMethod() === 'post') {
        $registerModel->loadData($request->getBody());
        $apiResponse = $registerModel->save();
        if ($registerModel->validate()) {
            if (!$apiResponse) {
                Application::$app->session->setFlash('error', 'Une erreur est survenue');
                Application::$app->response->redirect('/register');
            }
            if ($apiResponse->code == 201 && $apiResponse->value != null) {
                Application::$app->session->setFlash('success', $apiResponse->message);
                Application::$app->login($apiResponse);
                Application::$app->response->redirect('/');
            } else {
                Application::$app->session->setFlash('error', $apiResponse->message);
                Application::$app->response->statusCode($apiResponse->code);
                Application::$app->response->redirect('/register');
            }
        }
    }
    return $this->render('register', [
        'model' => $registerModel
    ]);
}

/**

```

```

    * Affiche la page de connexion.
    *
    * @param Request $request La requête HTTP.
    * @return string Le contenu HTML de la page de connexion.
    */
    public function login(Request $request): string
    {
        if (!Application::isGuest()) Application::$app->
>response->redirect('/');
        $loginForm = new LoginForm();
        if ($request->getMethod() === 'post') {
            $loginForm->loadData($request->getBody());
            if ($loginForm->validate() && $loginForm->login())
        {
            Application::$app->session->
>setFlash('success', 'Vous êtes maintenant connecté');
            Application::$app->response->redirect('/');
        }
    }

    return $this->render('login', [
        'model' => $loginForm
    ]);
}
}

```

ForbiddenException Site

```
<?php
```

```
namespace Eatfit\Site\Core\Exception;
```

```
use Exception;
```

```
class ForbiddenException extends Exception
{
    protected $message = "Vous n'avez pas la permission
d'accéder à cette page";
    protected $code = 403;
}

```

NotFoundException Site

```
<?php
```

```
namespace Eatfit\Site\Core\Exception;
```

```
use Exception;
```

```
class NotFoundException extends Exception

```

```
{
    protected $message = 'Cette page n\'existe pas';
    protected $code = 404;
}
```

BaseField Site

```
<?php
```

```
namespace Eatfit\Site\Core\Form;
```

```
use Eatfit\Site\Core\Model;
```

```
abstract class BaseField
```

```
{
```

```
    protected Model $model;
    protected string $attribute;
    protected string $type = 'text';
    protected string $placeholder = '';
```

```
    /**
```

```
     * Constructeur de la classe BaseField.
```

```
     *
```

```
     * @param Model $model Le modèle associé au champ.
```

```
     * @param string $attribute L'attribut du modèle
    correspondant au champ.
```

```
     */
```

```
    public function __construct(Model $model, string
    $attribute)
```

```
    {
```

```
        $this->model = $model;
        $this->attribute = $attribute;
    }
```

```
    /**
```

```
     * Méthode magique __toString() - renvoie le champ sous
    forme de chaîne de caractères.
```

```
     *
```

```
     * @return string Le champ HTML rendu.
```

```
     */
```

```
    public function __toString(): string
```

```
    {
```

```
        return sprintf('<div class="form-label label-dark">
            <label>%s</label>
            %s
            <div class="invalid-feedback">
                %s
            </div>
        </div>', $this->model->getLabel($this->attribute),
    $this->renderInput(), $this->getErrorMessage());
    }
```



```

    /**
     * Méthode abstraite renderInput() - rendu de l'élément de
    champ.
     *
     * @return string Le rendu de l'élément de champ
    spécifique.
     */
    abstract protected function renderInput(): string;

    /**
     * Obtient le message d'erreur associé au champ.
     *
     * @return string Le message d'erreur du champ.
     */
    protected function getErrorMessage(): string
    {
        return $this->model->getFirstError($this->attribute);
    }

    /**
     * Définit le type du champ.
     *
     * @param string $type Le type du champ.
     * @return BaseField L'instance de BaseField.
     */
    public function setType(string $type): self
    {
        $this->type = $type;
        return $this;
    }
}

```

Field Site

```
<?php
```

```

namespace Eatfit\Site\Core\Form;

use Eatfit\Site\Core\Model;

class Field extends BaseField
{
    const TYPE_TEXT = 'text';
    const TYPE_TEXTAREA = 'textarea';
    const TYPE_NUMBER = 'number';
    const TYPE_PASSWORD = 'password';
    const TYPE_FILE = 'file';
    const TYPE_CHECKBOX = 'checkbox';
}

```

```
const TYPE_RADIO = 'radio';
protected array $attributes = [];
private array $options;

/**
 * Constructeur de la classe Field.
 *
 * @param Model $model Le modèle associé au champ.
 * @param string $attribute L'attribut du modèle
correspondant au champ.
 */
public function __construct(Model $model, string
$attribute)
{
    $this->type = self::TYPE_TEXT;
    parent::__construct($model, $attribute);
}

/**
 * Définit le champ comme un champ textarea.
 *
 * @return Field L'instance du champ Field.
 */
public function textarea(): self
{
    return $this->setType(self::TYPE_TEXTAREA);
}

/**
 * Définit le champ comme un champ checkbox.
 *
 * @return Field L'instance du champ Field.
 */
public function checkbox(): self
{
    return $this->setType(self::TYPE_CHECKBOX);
}

/**
 * Définit le champ comme un champ radio.
 *
 * @return Field L'instance du champ Field.
 */
public function radio(): self
{
    return $this->setType(self::TYPE_RADIO);
}

/**
 * Définit le champ comme un champ de mot de passe.
```

```
*
* @return Field L'instance du champ Field.
*/
public function passwordField(): self
{
    return $this->setType(self::TYPE_PASSWORD);
}

/**
 * Définit le champ comme un champ de nombre.
 *
 * @return Field L'instance du champ Field.
 */
public function numberField(): self
{
    return $this->setType(self::TYPE_NUMBER);
}

/**
 * Définit le champ comme un champ select avec les options
spécifiées.
 *
 * @param array $options Les options du champ select.
 * @return Field L'instance du champ Field.
 */
public function selectField(array $options): self
{
    $this->type = 'select';
    $this->options = $options;
    return $this;
}

/**
 * Définit les attributs supplémentaires du champ.
 *
 * @param array $attributes Les attributs du champ.
 * @return Field L'instance du champ Field.
 */
public function setAttributes(array $attributes): self
{
    $this->attributes = $attributes;
    return $this;
}

/**
 * Rendu de l'élément de champ.
 *
 * @return string Le rendu de l'élément de champ
spécifique.
 */
public function renderInput(): string
```

```

{
    $placeholderText = !empty($this->placeholder) ?
    sprintf('placeholder="%s"', $this->placeholder) : '';
    $additionalAttributes = '';
    foreach ($this->attributes as $attribute => $value) {
        $additionalAttributes .= sprintf(' %s="%s"',
        $attribute, $value);
    }

    if ($this->type === self::TYPE_TEXTAREA) {
        return sprintf('<textarea class="form-control%s"
name="%s" %s %s>%s</textarea>',
        $this->model->hasError($this->attribute) ? '
is-invalid' : '',
        $this->attribute,
        $placeholderText,
        $additionalAttributes,
        $this->model->{$this->attribute}
    );
    }
    if ($this->type === 'select') {
        $optionsString = '<option value="">Choisissez une
option</option>'; // option par défaut
        foreach ($this->options as $value => $label) {
            $selected = ($this->model->{$this->attribute}
=== $value) ? ' selected' : '';
            $optionsString .= sprintf('<option
value="%s"%s>%s</option>', $value, $selected, $label);
        }

        return sprintf('<select class="form-control%s"
name="%s" %s %s>%s</select>',
        $this->model->hasError($this->attribute) ? '
is-invalid' : '',
        $this->attribute,
        $placeholderText,
        $additionalAttributes,
        $optionsString
    );
    }

    return sprintf('<input type="%s" class="form-
control%s" name="%s" value="%s" %s %s>',
        $this->type,
        $this->model->hasError($this->attribute) ? ' is-
invalid' : '',
        $this->attribute,
        $this->model->{$this->attribute},
        $placeholderText,
        $additionalAttributes
    );
}

```

```

    }

    /**
     * Définit le texte d'espace réservé pour le champ.
     *
     * @param string $placeholder Le texte d'espace réservé.
     * @return Field L'instance du champ Field.
     */
    public
    function setPlaceholder(string $placeholder): self
    {
        $this->placeholder = $placeholder;
        return $this;
    }

    public
    function fileField(): self
    {
        return $this->setType(self::TYPE_FILE);
    }
}

```

Form Site

```
<?php
```

```

namespace Eatfit\Site\Core\Form;

use Eatfit\Site\Core\Model;

class Form
{
    /**
     * Démarre un formulaire HTML.
     *
     * @param string $action L'action du formulaire (URL de
     destination des données).
     * @param string $method La méthode HTTP à utiliser (GET,
     POST, etc.).
     * @param array $options Options supplémentaires pour les
     attributs du formulaire.
     * @return Form Une instance de la classe Form.
     */
    public static function begin($action, $method, $options =
    [])
    {
        $attributes = [];
        foreach ($options as $key => $value) {

```

```

        $attributes[] = "$key=\"$value\"";
    }
    echo sprintf('<form action="%s" method="%s" %s>',
$action, $method, implode(" ", $attributes));
    return new Form();
}

/**
 * Ferme le formulaire HTML.
 *
 * @return void
 */
public static function end()
{
    echo '</form>';
}

/**
 * Crée un champ de formulaire pour un modèle et un
attribut spécifiés.
 *
 * @param Model $model Le modèle associé au champ de
formulaire.
 * @param string $attribute L'attribut du modèle
correspondant au champ de formulaire.
 * @return Field Une instance de la classe Field pour le
champ de formulaire créé.
 */
public function field(Model $model, $attribute): Field
{
    return new Field($model, $attribute);
}
}

```

AuthMiddleware Site

```
<?php
```

```

namespace Eatfit\Site\Core\Middlewares;

use Eatfit\Site\Core\Application;
use Eatfit\Site\Core\Exception\ForbiddenException;

class AuthMiddleware extends BaseMiddleware
{
    protected array $actions = [];

    /**
     * AuthMiddleware constructor.
     *
     * @param array $actions Les actions qui nécessitent une

```

```

authentication.
    */
    public function __construct($actions = [])
    {
        $this->actions = $actions;
    }

    /**
     * Exécute le middleware.
     *
     * @throws ForbiddenException Si l'utilisateur n'est pas
    authentifié.
    */
    public function execute(): void
    {
        if (Application::isGuest()) {
            if (empty($this->actions) ||
in_array(Application::$app->controller->action, $this->actions)) {
                throw new ForbiddenException();
            }
        }
    }
}

```

BaseMiddleware Site

```

<?php

namespace Eatfit\Site\Core\Middlewares;

abstract class BaseMiddleware
{
    abstract public function execute();
}

```

Application Site

```

<?php

namespace Eatfit\Site\Core;

use Eatfit\Site\Core\Db\Database;
use Eatfit\Site\Core\Db\DbModel;
use Eatfit\Site\Models\User;
use Exception;

class Application
{
    public static Application $app;
}

```

```

public static string $ROOT_DIR;
public static array $ALLOWED_IMAGE_EXTENSIONS;
public static string $API_URL;
public string $layout = 'main';
public Router $router;
public Request $request;
public Response $response;
public ?Controller $controller = null;
public Session $session;
public View $view;
public $user;

/**
 * Constructeur de l'application.
 *
 * @param string $rootDir Le répertoire racine de
l'application.
 * @param array $config La configuration de l'application.
 */
public function __construct(string $rootDir, array
$config)
{
    $this->user = null;
    self::$API_URL = $config['API_URL'];
    self::$ALLOWED_IMAGE_EXTENSIONS =
$config['ALLOWED_IMAGE_EXTENSIONS'];
    self::$ROOT_DIR = $rootDir;
    self::$app = $this;
    $this->request = new Request();
    $this->response = new Response();
    $this->router = new Router($this->request, $this-
>response);
    $this->session = new Session();
    $this->view = new View();
    $token = Application::$app->session->get('user');

    if ($token != null) {
        $user = User::getUserByToken($token);
        if (isset($user->value->token) == null &&
isset($user->value->expiration) == null) {
            $this->session->setFlash("error", "Votre
session a expiré, veuillez vous reconnecter");
            $this->logout();
        } else $this->user = $user->value;
    }
}

/**
 * Déconnecte l'utilisateur en supprimant son token de la
session.
 */

```



```
public function logout(): void
{
    $this->user = null;
    self::$app->session->remove('user');
}

/**
 * Vérifie si l'utilisateur est un invité (non
authentifié).
 *
 * @return bool True si l'utilisateur est un invité, False
sinon.
 */
public static function isGuest(): bool
{
    return !self::$app->user;
}

/**
 * Connecte l'utilisateur en enregistrant son token dans
la session et effectue une redirection.
 *
 * @param mixed $user Les informations de l'utilisateur.
 * @return bool True si la connexion réussit, False sinon.
 */
public function login(mixed $user): bool
{
    Application::$app->session->set('user', $user->value-
>token);
    $this->response->statusCode($user->code);
    Application::$app->response->redirect('/');
    return true;
}

/**
 * Lance l'application.
 */
public function run(): void
{
    try {
        echo $this->router->resolve();
    } catch (Exception $e) {
        echo $this->router->renderView('_error', [
            'exception' => $e,
        ]);
    }
}
}
```

Controller Site

<?php

```
namespace Eatfit\Site\Core;

use Eatfit\Site\Core\Middlewares\BaseMiddleware;

/**
 * La classe Controller est la classe de base pour tous les
 * contrôleurs de l'application.
 * Elle fournit des fonctionnalités communes aux contrôleurs
 * tels que la gestion du layout, le rendu des vues
 * et la gestion des middlewares.
 */
class Controller
{
    public string $layout = 'main';
    public string $action = '';

    protected array $middlewares = [];

    /**
     * Rend la vue spécifiée avec les paramètres donnés.
     *
     * @param string $view Le nom de la vue à rendre.
     * @param array $params Les paramètres à passer à la vue.
     * @return string Le contenu de la vue rendue.
     */
    public function render(string $view, array $params = []):
string
    {
        return Application::$app->router->renderView($view,
$params);
    }

    /**
     * Enregistre un middleware à appliquer aux actions du
     * contrôleur.
     *
     * @param BaseMiddleware $middleware Le middleware à
     enregistrer.
     * @return void
     */
    public function registerMiddleware(BaseMiddleware
$params)
    {
        $this->middlewares[] = $middleware;
    }

    /**
     * Récupère les middlewares enregistrés pour le
```

```

contrôleur.
    *
    * @return array Les middlewares enregistrés.
    */
    public function getMiddlewares(): array
    {
        return $this->middlewares;
    }
}

```

Model Site

<?php

```

namespace Eatfit\Site\Core;

/**
 * La classe Model est une classe de base pour tous les
 * modèles de l'application.
 * Elle fournit des fonctionnalités communes liées à la
 * validation des données et à la gestion des erreurs.
 */
class Model
{
    const RULE_REQUIRED = 'required';
    const RULE_EMAIL = 'email';
    const RULE_MIN = 'min';
    const RULE_MAX = 'max';
    const RULE_MATCH = 'match';

    public array $errors = [];

    /**
     * Effectue une requête API avec les données JSON
     * fournies.
     *
     * @param array $data Les données JSON de la requête.
     * @param bool $addBearer Indique si le jeton
     * d'authentification doit être ajouté à la requête.
     * @param bool $returnArray Indique si la réponse doit
     * être renvoyée sous forme de tableau.
     * @return mixed Le résultat de la requête ou false en cas
     * d'erreur.
     */
    protected static function getJsonResult(array $data, bool
    $addBearer = false, bool $returnArray = false): mixed
    {
        if (!isset($data['data']) || !isset($data['method']))
        || !isset($data['url'])) return false;
        $http_header[] = 'Content-Type: application/json';
        if ($addBearer) $http_header[] = 'Authorization:

```

```

Bearer ' . Application::$app->user->token;
    $curl = curl_init();
    curl_setopt_array($curl, array(
        CURLOPT_URL => Application::$API_URL .
$data['url'],
        CURLOPT_RETURNTRANSFER => true,
        CURLOPT_ENCODING => '',
        CURLOPT_MAXREDIRS => 10,
        CURLOPT_TIMEOUT => 0,
        CURLOPT_FOLLOWLOCATION => true,
        CURLOPT_HTTP_VERSION => CURL_HTTP_VERSION_1_1,
        CURLOPT_CUSTOMREQUEST => $data['method'],
        CURLOPT_POSTFIELDS => json_encode($data['data']),
        CURLOPT_HTTPHEADER => $http_header,
    ));
    $response = curl_exec($curl);
    if (curl_errno($curl)) $error_msg = curl_error($curl);
    curl_close($curl);
    return $error_msg ?? json_decode($response,
$returnArray);
}

/**
 * Charge les données dans le modèle.
 *
 * @param array $data Les données à charger dans le
modèle.
 * @return void
 */
public function loadData($data): void
{
    foreach ($data as $key => $value) {
        if (property_exists($this, $key)) {
            $this->{$key} = $value;
        }
    }
}

/**
 * Retourne les attributs du modèle.
 *
 * @return array Les attributs du modèle.
 */
public function attributes()
{
    return [];
}

/**
 * Retourne le label d'un attribut donné.
 *

```

```

    * @param string $attribute Le nom de l'attribut.
    * @return string Le label de l'attribut.
    */
    public function getLabel(string $attribute): string
    {
        return $this->labels()[$attribute] ?? $attribute;
    }

    /**
     * Retourne les labels des attributs du modèle.
     *
     * @return array Les labels des attributs du modèle.
     */
    public function labels(): array
    {
        return [];
    }

    /**
     * Valide les données du modèle en fonction des règles de
    validation.
     *
     * @return bool True si les données sont valides, False
    sinon.
     */
    public function validate(): bool
    {
        foreach ($this->rules() as $attribute => $rules) {
            $value = $this->{$attribute};
            foreach ($rules as $rule) {
                $ruleName = $rule;
                if (!is_string($rule)) $ruleName = $rule[0];
                if ($ruleName === self::RULE_REQUIRED &&
                !$value) $this->addErrorByRule($attribute,
                self::RULE_REQUIRED);
                if ($ruleName === self::RULE_EMAIL &&
                !filter_var($value, FILTER_VALIDATE_EMAIL)) $this-
                >addErrorByRule($attribute, self::RULE_EMAIL);
                if ($ruleName === self::RULE_MIN &&
                strlen($value) < $rule['min']) $this-
                >addErrorByRule($attribute, self::RULE_MIN, ['min' =>
                $rule['min']]);
                if ($ruleName === self::RULE_MAX &&
                strlen($value) > $rule['max']) $this-
                >addErrorByRule($attribute, self::RULE_MAX);
                if ($ruleName === self::RULE_MATCH && $value
                !== $this->{$rule['match']}) $this->addErrorByRule($attribute,
                self::RULE_MATCH, ['match' => $rule['match']]);
            }
        }
        return empty($this->errors);
    }

```

```
}

/**
 * Retourne les règles de validation du modèle.
 *
 * @return array Les règles de validation du modèle.
 */
public function rules()
{
    return [];
}

/**
 * Ajoute une erreur de validation basée sur une règle
 donnée.
 *
 * @param string $attribute L'attribut concerné.
 * @param string $rule La règle de validation.
 * @param array $params Les paramètres à substituer dans
 le message d'erreur.
 * @return void
 */
protected function addErrorByRule(string $attribute,
string $rule, $params = [])
{
    $params['field'] ??= $attribute;
    $errorMessage = $this->errorMessage($rule);
    foreach ($params as $key => $value) {
        $errorMessage = str_replace("{{$key}}", $value,
$errorMessage);
    }
    $this->errors[$attribute][] = $errorMessage;
}

/**
 * Retourne le message d'erreur correspondant à une règle
 de validation donnée.
 *
 * @param string $rule La règle de validation.
 * @return string Le message d'erreur.
 */
public function errorMessage($rule): string
{
    return $this->errorMessages()[$rule];
}

/**
 * Retourne les messages d'erreur associés aux règles de
 validation.
 *
 * @return array Les messages d'erreur associés aux règles
```

```
de validation.
    */
    public function errorMessages(): array
    {
        return [
            self::RULE_REQUIRED => 'Le champ {field} est
requis',
            self::RULE_EMAIL => 'Le champ {field} doit être
une adresse email valide',
            self::RULE_MIN => 'Le champ {field} doit contenir
au moins {min} caractères',
            self::RULE_MAX => 'Le champ {field} doit contenir
au maximum {max} caractères',
            self::RULE_MATCH => 'Le champ {field} doit être
identique au champ {match}'
        ];
    }

    /**
     * Ajoute une erreur personnalisée.
     *
     * @param string $attribute L'attribut concerné.
     * @param string $message Le message d'erreur.
     * @return void
     */
    public function addError(string $attribute, string
$message)
    {
        $this->errors[$attribute][] = $message;
    }

    /**
     * Vérifie si une erreur existe pour un attribut donné.
     *
     * @param string $attribute L'attribut concerné.
     * @return bool True si une erreur existe, False sinon.
     */
    public function hasError($attribute)
    {
        return $this->errors[$attribute] ?? false;
    }

    /**
     * Retourne la première erreur pour un attribut donné.
     *
     * @param string $attribute L'attribut concerné.
     * @return string|null Le premier message d'erreur, ou
null s'il n'y a pas d'erreur.
     */
    public function getFirstError($attribute)
    {

```

```

        $errors = $this->errors[$attribute] ?? [];
        return $errors[0] ?? '';
    }
}

```

Request Site

```
<?php
```

```
namespace Eatfit\Site\Core;
```

```

/**
 * La classe Request représente une requête HTTP entrante.
 * Elle fournit des méthodes pour récupérer des informations
 * sur la requête.
 */
class Request
{
    private array $routeParams = [];

    /**
     * Retourne l'URL de la requête.
     *
     * @return string L'URL de la requête.
     */
    public function getUrl(): string
    {
        $path = $_SERVER['REQUEST_URI'];
        $position = strpos($path, '?');
        if ($position !== false) {
            $path = substr($path, 0, $position);
        }
        return $path;
    }

    /**
     * Récupère les données de la requête.
     *
     * @return array Les données de la requête.
     */
    public function getBody(): array
    {
        $data = [];
        if ($this->isGet()) {
            foreach ($_GET as $key => $value) {
                $data[$key] = filter_input(INPUT_GET, $key,
FILTER_SANITIZE_SPECIAL_CHARS);
            }
        }
        if ($this->isPost()) {

```



```
        foreach ($_POST as $key => $value) {
            $data[$key] = filter_input(INPUT_POST, $key,
FILTER_SANITIZE_SPECIAL_CHARS);
        }
    }
    return $data;
}

/**
 * Vérifie si la méthode de la requête est GET.
 *
 * @return bool True si la méthode est GET, False sinon.
 */
public function isGet(): bool
{
    return $this->getMethod() === 'get';
}

/**
 * Retourne la méthode de la requête (GET, POST, etc.).
 *
 * @return string La méthode de la requête.
 */
public function getMethod(): string
{
    return strtolower($_SERVER['REQUEST_METHOD']);
}

/**
 * Vérifie si la méthode de la requête est POST.
 *
 * @return bool True si la méthode est POST, False sinon.
 */
public function isPost(): bool
{
    return $this->getMethod() === 'post';
}

/**
 * Retourne les paramètres de la route.
 *
 * @return array Les paramètres de la route.
 */
public function getRouteParams(): array
{
    return $this->routeParams;
}

/**
 * Définit les paramètres de la route.
 *
```

```

    * @param array $params Les paramètres de la route.
    * @return self
    */
    public function setRouteParams(array $params): static
    {
        $this->routeParams = $params;
        return $this;
    }

    /**
     * Retourne la valeur d'un paramètre de la route donné.
     *
     * @param string $param Le nom du paramètre.
     * @param mixed|null $default La valeur par défaut à
    retourner si le paramètre n'est pas trouvé.
     * @return mixed La valeur du paramètre ou la valeur par
    défaut.
     */
    public function getRouteParam(string $param, mixed
    $default = null): mixed
    {
        return $this->routeParams[$param] ?? $default;
    }
}

```

Response Site

```
<?php
```

```
namespace Eatfit\Site\Core;
```

```

/**
 * La classe Response représente la réponse HTTP à renvoyer au
    client.
 * Elle fournit des méthodes pour gérer le code de statut de
    la réponse et les redirections.
 */
class Response
{
    /**
     * Définit le code de statut de la réponse.
     *
     * @param int $code Le code de statut HTTP.
     */
    public function statusCode(int $code): void
    {
        http_response_code($code);
    }

    /**

```

```

    * Effectue une redirection vers l'URL spécifiée.
    *
    * @param string $url L'URL de destination.
    */
    public function redirect(string $url): void
    {
        header("Location: $url");
    }
}

```

Router Site

<?php

```

namespace Eatfit\Site\Core;

use Eatfit\Site\Core\Exception\NotFoundException;

/**
 * La classe Router gère le routage des requêtes HTTP vers les
 * actions correspondantes.
 */
class Router
{
    private Request $request;
    private Response $response;
    private array $routeMap = [];

    /**
     * Constructeur de la classe Router.
     *
     * @param Request $request L'objet Request pour récupérer
     * les informations de la requête.
     * @param Response $response L'objet Response pour gérer
     * la réponse HTTP.
     */
    public function __construct(Request $request, Response
    $response)
    {
        $this->request = $request;
        $this->response = $response;
    }

    /**
     * Enregistre une route pour la méthode GET.
     *
     * @param string $url L'URL de la route.
     * @param mixed $callback Le callback à exécuter lorsque
     * la route est atteinte.

```

```
*/
public function get(string $url, mixed $callback): void
{
    $this->routeMap['get'][$url] = $callback;
}

/**
 * Enregistre une route pour la méthode POST.
 *
 * @param string $url L'URL de la route.
 * @param mixed $callback Le callback à exécuter lorsque
la route est atteinte.
 */
public function post(string $url, mixed $callback): void
{
    $this->routeMap['post'][$url] = $callback;
}

/**
 * Résout la route actuelle de la requête et exécute le
callback correspondant.
 *
 * @return mixed Le résultat du callback exécuté.
 * @throws NotFoundException Si aucune route ne correspond
à la route actuelle de la requête.
 */
public function resolve(): mixed
{
    $method = $this->request->getMethod();
    $url = $this->request->getUrl();
    $callback = $this->routeMap[$method][$url] ?? false;
    if (!$callback) {
        $callback = $this->getCallback();

        if ($callback === false) {
            throw new NotFoundException();
        }
    }
    if (is_string($callback)) {
        return $this->renderView($callback);
    }
    if (is_array($callback)) {
        /**
         * @var $controller Controller
         */
        $controller = new $callback[0];
        $controller->action = $callback[1];

        Application::$app->controller = $controller;
        $middlewares = $controller->getMiddlewares();
    }
}
```

```

        foreach ($middlewares as $middleware) {
            $middleware->execute();
        }
        $callback[0] = $controller;
    }
    return call_user_func($callback, $this->request,
$this->response);
}

/**
 * Récupère le callback correspondant à la route actuelle
de la requête.
 *
 * @return mixed Le callback correspondant à la route
actuelle, ou false si aucune route ne correspond.
 */
public function getCallback(): mixed
{
    $method = $this->request->getMethod();
    $url = $this->request->getUrl();
    // Trim slashes
    $url = trim($url, '/');

    // Get all routes for current request method
    $routes = $this->getRouteMap($method);

    $routeParams = false;

    // Start iterating registered routes
    foreach ($routes as $route => $callback) {
        // Trim slashes
        $route = trim($route, '/');
        $routeNames = [];

        if (!$route) {
            continue;
        }

        // Find all route names from route and save in
$routeNames
        if (preg_match_all('/\{(\w+)(:[^}]+)?\}/', $route,
$matches)) {
            $routeNames = $matches[1];
        }

        // Convert route name into regex pattern
        $routeRegex = "@^" .
preg_replace_callback('/\{(\w+)(:[^}]+)?\}/', fn($m) =>
isset($m[2]) ? "({$m[2]})" : '(\w+)', $route) . "$@";

        // Test and match current route against

```

```

$routeRegex
    if (preg_match_all($routeRegex, $url,
$valueMatches)) {
        $values = [];
        for ($i = 1; $i < count($valueMatches); $i++)
        {
            $values[] = $valueMatches[$i][0];
        }
        $routeParams = array_combine($routeNames,
$value);

        $this->request->setRouteParams($routeParams);
        return $callback;
    }

    return false;
}

/**
 * Récupère la liste des routes enregistrées pour une
méthode donnée.
 *
 * @param string $method La méthode HTTP (get, post,
etc.).
 * @return array Les routes enregistrées pour la méthode
donnée.
 */
public function getRouteMap(string $method): array
{
    return $this->routeMap[$method] ?? [];
}

/**
 * Rend la vue spécifiée avec les paramètres donnés.
 *
 * @param string $view Le nom de la vue.
 * @param array $params Les paramètres à passer à la vue.
 * @return string Le contenu de la vue rendue.
 */
public function renderView($view, $params = []):
false|array|string
{
    return Application::$app->view->renderView($view,
$params);
}

```

Session Site

```
<?php
```

```
namespace Eatfit\Site\Core;
```

```
class Session
{
    protected const FLASH_KEY = 'flash_messages';

    /**
     * Constructeur de la classe Session.
     * Initialise la session et les messages flash.
     */
    public function __construct()
    {
        if (session_status() === PHP_SESSION_NONE)
            session_start();

        $_SESSION[self::FLASH_KEY] =
            $_SESSION[self::FLASH_KEY] ?? [];
        foreach ($_SESSION[self::FLASH_KEY] as $key =>
            &$flashMessage) {
            $flashMessage['remove'] = true;
        }
    }

    /**
     * Définit un message flash avec une clé spécifiée.
     *
     * @param string $key La clé du message flash
     * @param string $message Le contenu du message flash
     */
    public function setFlash(string $key, string $message):
        void
    {
        $_SESSION[self::FLASH_KEY][$key] = [
            'remove' => false,
            'value' => $message
        ];
    }

    /**
     * Récupère un message flash associé à une clé spécifiée.
     * Le message flash est ensuite supprimé de la session.
     *
     * @param string $key La clé du message flash
     * @return string|null Le contenu du message flash ou null
     * s'il n'existe pas
     */
    public function getFlash(string $key): ?string
```

```
{
    return $_SESSION[self::FLASH_KEY][$key]['value'] ??
null;
}

/**
 * Définit une valeur dans la session avec une clé
spécifiée.
 *
 * @param string $key La clé de la valeur
 * @param mixed $value La valeur à enregistrer
 */
public function set(string $key, mixed $value): void
{
    $_SESSION[$key] = $value;
}

/**
 * Récupère une valeur de la session en utilisant la clé
spécifiée.
 *
 * @param string $key La clé de la valeur
 * @return mixed|null La valeur correspondante ou null si
elle n'existe pas
 */
public function get(string $key)
{
    return $_SESSION[$key] ?? null;
}

/**
 * Supprime une valeur de la session en utilisant la clé
spécifiée.
 *
 * @param string $key La clé de la valeur à supprimer
 */
public function remove(string $key): void
{
    unset($_SESSION[$key]);
}

public function __destruct()
{
    $this->removeFlashMessages();
}

/**
 * Supprime les messages flash de la session qui ont été
marqués pour suppression.
 */
private function removeFlashMessages(): void
```



```

    {
        $flashMessages = $_SESSION[self::FLASH_KEY] ?? [];
        foreach ($flashMessages as $key => $flashMessage) {
            if ($flashMessage['remove'])
unset($flashMessages[$key]);
        }
        $_SESSION[self::FLASH_KEY] = $flashMessages;
    }
}

```

View Site

```
<?php
```

```
namespace Eatfit\Site\Core;
```

```
class View
```

```

{
    public string $title = '';

    public function renderView($view, array $params):
array|false|string
    {
        $layoutName = Application::$app->layout;
        if (Application::$app->controller) {
            $layoutName = Application::$app->controller-
>layout;
        }
        $viewContent = $this->renderViewOnly($view, $params);
        ob_start();
        include_once Application::$ROOT_DIR .
"/views/layouts/$layoutName.php";
        $layoutContent = ob_get_clean();
        return str_replace('{{content}}', $viewContent,
$layoutContent);
    }

    private function renderViewOnly($view, array $params):
false|string
    {
        foreach ($params as $key => $value) $$key = $value;
        ob_start();
        include_once Application::$ROOT_DIR .
"/views/$view.php";
        return ob_get_clean();
    }
}

```

FoodType Site

```
<?php
```

```
namespace Eatfit\Site\Models;

use Eatfit\Site\Core\Model;

class FoodType extends Model
{
    public int $idFoodType = 0;
    public string $name = '';

    public static function getFoodTypes()
    {
        return self::getJsonResult([
            'url' => 'food_types',
            'method' => 'GET',
            'data' => []
        ]);
    }

    public function rules(): array
    {
        return [
            'name' => [self::RULE_REQUIRED]
        ];
    }

    public function labels(): array
    {
        return [
            'name' => "Nom: "
        ];
    }

    public function save()
    {
        return self::getJsonResult([
            'url' => 'food_type',
            'method' => 'POST',
            'data' => [
                'name' => $this->name
            ]
        ], true);
    }
}
```

History Site

```
<?php
```

```
namespace Eatfit\Site\Models;
```

```
use Eatfit\Site\Core\Model;
```

```
class History extends Model
```

```
{
    public int $idRecipe = 0;
    public string $title = '';
    public string $consumption_date = '';
    public int $idUser = 0;
    public int $idConsumedRecipe = 0;

    public function rules(): array
    {
        return [
            'idRecipe' => [self::RULE_REQUIRED],
            'title' => [self::RULE_REQUIRED],
            'consumption_date' => [self::RULE_REQUIRED],
            'meal_time' => [self::RULE_REQUIRED],
        ];
    }

    public function labels(): array
    {
        return [
            'idRecipe' => "Identifiant de la recette: ",
            'title' => "Titre de la recette: ",
            'consumption_date' => "Date de consommation: ",
            'meal_time' => "Moment de la consommation: ",
        ];
    }

    public function save()
    {
        $data = [
            'idRecipe' => $this->idRecipe
        ];
        return self::getJsonResult([
            'url' => 'history',
            'method' => 'POST',
            'data' => $data
        ], true);
    }

    public function getHistory(int $idRecipe = 0, $addBearer = true)
    {
        if ($idRecipe == 0 || $idRecipe == null) {
```

```
        return self::getJsonResult([
            'url' => 'history',
            'method' => 'GET',
            'data' => []
        ], $addBearer);
    }
    return self::getJsonResult([
        'url' => 'history',
        'method' => 'GET',
        'data' => [
            'idRecipe' => $idRecipe
        ]
    ], $addBearer);
}

public function updateHistory($idConsumedRecipe,
    $addBearer = true)
{
    $data = [
        'idConsumedRecipe' => $this->idConsumedRecipe,
        'consumption_date' => $this->consumption_date,
    ];

    return self::getJsonResult([
        'url' => 'history',
        'method' => 'PUT',
        'data' => $data
    ], $addBearer);
}

public function deleteHistory($idConsumedRecipe,
    $addBearer = true)
{
    return self::getJsonResult([
        'url' => 'history',
        'method' => 'DELETE',
        'data' => [
            'idConsumedRecipe' => $idConsumedRecipe
        ]
    ], $addBearer);
}

public function deleteAllHistory($addBearer = true)
{
    return self::getJsonResult([
        'url' => 'history',
        'method' => 'DELETE',
        'data' => []
    ], $addBearer);
}
```

```
}  
}
```

LoginForm Site

```
<?php
```

```
namespace Eatfit\Site\Models;  
  
use Eatfit\Site\Core\Application;  
use Eatfit\Site\Core\Model;  
  
class LoginForm extends Model  
{  
    public string $email = '';  
    public string $password = '';  
  
    public function rules()  
    {  
        return [  
            'email' => [self::RULE_REQUIRED,  
self::RULE_EMAIL],  
            'password' => [self::RULE_REQUIRED],  
        ];  
    }  
  
    public function labels(): array  
    {  
        return [  
            'email' => 'Email:',  
            'password' => 'Password:'  
        ];  
    }  
  
    /**  
     * Connecte l'utilisateur  
     */  
    public function login()  
    {  
        $user = User::getUser($this->email, $this->password);  
        if ($user->value == null && $user->code != 200) {  
            Application::$app->session->setFlash('error',  
$user->message);  
            return false;  
        }  
        return Application::$app->login($user);  
    }  
}
```

ProfileModel Site

```
<?php
```

```
namespace Eatfit\Site\models;
```

```
use Eatfit\Site\Core\Model;
```

```
class ProfileModel extends Model
```

```
{
    public int $idUser = 0;
    public ?string $username = null;
    public ?string $email = null;
    public ?string $password = null;
    public ?string $confirm_password = null;

    public function rules()
    {
        return [
            'username' => [[self::RULE_REQUIRED]],
            'email' => [[self::RULE_EMAIL],
[self::RULE_REQUIRED]],
            'password' => [[self::RULE_MIN, 'min' => 8]],
            'confirm_password' => [self::RULE_REQUIRED,
[self::RULE_MATCH, 'match' => 'password']]
        ];
    }

    public function labels(): array
    {
        return [
            'username' => 'Name:',
            'email' => 'Email:',
            'password' => 'Password:',
            'confirm_password' => 'Confirm Password:',
        ];
    }

    /**
     *
     * Met a jour l'utilisateur
     */
    public function update()
    {
        $updates = [];
        foreach ($this->attributes() as $attribute) {
            if ($this->$attribute == null || empty($this->$attribute)) continue;
            $updates[$attribute] = $this->$attribute;
        }
        return self::getJsonResult([
            'url' => 'user',

```

```

        'method' => 'PUT',
        'data' => $updates
    ], true);
}

public function attributes(): array
{
    return ['username', 'email', 'password',
'confirm_password'];
}

/**
 * supprime l'utilisateur
 */
public function delete()
{
    return self::getJsonResult([
        'url' => 'user',
        'method' => 'DELETE',
        'data' => []
    ], true);
}
}

```

Rating Site

<?php

```

namespace Eatfit\Site\Models;

use Eatfit\Site\Core\Model;

class Rating extends Model
{
    public int $idRating = 0;
    public int $score = 0;
    public string $comment = '';
    public int $idRecipe = 0;
    public int $idUser = 0;

    public function rules(): array
    {
        return [
            'score' => [self::RULE_REQUIRED]
        ];
    }

    public function labels(): array
    {
        return [
            'score' => "Note (1-5): ",

```

```
        'comment' => "Commentaire: ",
    ];
}

public function getRatingByIdRecipe($addBearer = true)
{
    return self::getJsonResult([
        'url' => 'rating',
        'method' => 'GET',
        'data' => [
            'idRecipe' => $this->idRecipe
        ]
    ], $addBearer);
}

public function getRatingById()
{
    return self::getJsonResult([
        'url' => 'rating',
        'method' => 'GET',
        'data' => [
            'idRating' => $this->idRating
        ]
    ], true);
}

public function create()
{
    return self::getJsonResult([
        'url' => 'rating',
        'method' => 'POST',
        'data' => [
            'score' => $this->score,
            'comment' => $this->comment,
            'idRecipe' => $this->idRecipe
        ]
    ], true);
}

public function update()
{
    $updates = [];
    foreach ($this->attributes() as $attribute) {
        if (!isset($this->$attribute)) continue;
        $updates[$attribute] = $this->$attribute;
    }
    var_dump($updates);
    return self::getJsonResult([
        'url' => 'rating',
        'method' => 'PUT',
        'data' => $updates
    ])
```



```

        ], true);
    }

    public function attributes(): array
    {
        return ['idRating', 'score', 'comment'];
    }

    public function delete()
    {
        return self::getJsonResult([
            'url' => 'rating',
            'method' => 'DELETE',
            'data' => [
                'idRating' => $this->idRating
            ]
        ], true);
    }
}

```

Recipe Site

```
<?php
```

```

namespace Eatfit\Site\Models;

use Eatfit\Site\Core\Model;

class Recipe extends Model
{
    public int $idRecipe = 0;
    public string $title = '';
    public int $preparation_time = 0;
    public string $difficulty = '';
    public string $instructions = '';
    public int $calories = 0;
    public $date = '';

    public array $categories = [];
    public array $image = [];
    public array $foodType = [];
    public int $idUser = 0;

    public function loadData($data): void
    {
        if (isset($data['categories'])) $data['categories'] =
explode(",", $data['categories']);
        parent::loadData($data); // TODO: Change the
autogenerated stub
    }
}

```

```

public function rules(): array
{
    return [
        'title' => [self::RULE_REQUIRED],
        'preparation_time' => [self::RULE_REQUIRED],
        'difficulty' => [self::RULE_REQUIRED],
        'instructions' => [self::RULE_REQUIRED],
        'calories' => [self::RULE_REQUIRED],
        'categories' => [self::RULE_REQUIRED],
        'foodType' => [self::RULE_REQUIRED]
    ];
}

public function labels(): array
{
    return [
        'title' => "Titre: ",
        'preparation_time' => "Temps de préparation: ",
        'difficulty' => "Difficulté: ",
        'instructions' => "Instructions: ",
        'calories' => "Calories: ",
        'categories' => "Catégorie: ",
        'foodType' => "Type de nourriture: "
    ];
}

public function create()
{
    $validatedImages = $this->validateAndPrepareImages($_FILES);
    if (isset($validatedImages['error'])) {
        return false;
    } else {
        $images = [];
        //https://stackoverflow.com/questions/3967515/how-to-convert-an-image-to-base64-encoding
        foreach ($validatedImages as $name => $base64)
            $images[] = $name . "," . $base64;
        if ($validatedImages == []) $images = "";
        $data = [
            'title' => $this->title,
            'preparation_time' => $this->preparation_time,
            'difficulty' => $this->difficulty,
            'instructions' => $this->instructions,
            'calories' => $this->calories,
            'image' => $images,
            'category' => $this->categories,
            'food_type' => $this->foodType
        ];
        $data['category'] = $data['category'][0];
        return self::getJsonResult([

```

```

        'url' => 'recipe',
        'method' => 'POST',
        'data' => $data
    ], true);
}
}

function validateAndPrepareImages($files): array
{
    $images = $files['file-input'];
    $numberOfFiles = count($images['name']);
    $imageContents = [];
    if ($images['error'][0] == 4) return [];
    for ($i = 0; $i < $numberOfFiles; $i++) {
        if ($images['error'][$i] !== UPLOAD_ERR_OK) return
['error' => 'Erreur pour le fichier ' . $images['name'][$i]];
        if (!getimagesize($images['tmp_name'][$i])) return
['error' => 'Le fichier ' . $images['name'][$i] . ' n\'est pas
une image'];
        $imageContent =
file_get_contents($images['tmp_name'][$i]);
        $imageContents[$images['name'][$i]] =
base64_encode($imageContent);
    }

    return $imageContents;
}

public function delete()
{
    return self::getJsonResult([
        'url' => 'recipe',
        'method' => 'DELETE',
        'data' => [
            'idRecipe' => $this->idRecipe
        ]
    ], true);
}

public function getRecipe($field, $search, $addBearer =
true)
{
    return self::getJsonResult([
        'url' => 'recipe',
        'method' => 'GET',
        'data' => [
            $field => $search
        ]
    ], $addBearer);
}

```

```
public function getRecipeByFilter($filter, $search)
{
    return self::getJsonResult([
        'url' => 'recipe',
        'method' => 'GET',
        'data' => [
            'filter' => [
                $filter => $search
            ]
        ]
    ]);
}

public function getCategories()
{
    return self::getJsonResult([
        'url' => 'categories',
        'method' => 'GET',
        'data' => []
    ]);
}

public function update($data)
{
    return self::getJsonResult([
        'url' => 'recipe',
        'method' => 'PUT',
        'data' => $data
    ], true);
}
}
```

User Site

```
<?php
```

```
namespace Eatfit\Site\Models;

use Eatfit\Site\Core\Application;
use Eatfit\Site\Core\Model;

class User extends Model
{
    public int $idUser = 0;
    public string $username = '';
    public string $email = '';
    public string $password = '';
    public string $password_confirm = '';
```

```

public string $token = '';
public int $expiration = 0;

public static function getUser($email, $password)
{
    return self::getJsonResult([
        'url' => 'login',
        'method' => 'PUT',
        'data' => [
            'email' => $email,
            'password' => $password
        ]
    ]);
}

public static function getUserByToken($token)
{
    $curl = curl_init();
    curl_setopt_array($curl, array(
        CURLOPT_URL => Application::$API_URL .
'/userById',
        CURLOPT_RETURNTRANSFER => true,
        CURLOPT_ENCODING => '',
        CURLOPT_MAXREDIRS => 10,
        CURLOPT_TIMEOUT => 0,
        CURLOPT_FOLLOWLOCATION => true,
        CURLOPT_HTTP_VERSION => CURL_HTTP_VERSION_1_1,
        CURLOPT_CUSTOMREQUEST => 'GET',
        CURLOPT_HTTPHEADER => array(
            "Authorization: Bearer $token"
        ),
    ));

    $response = curl_exec($curl);
    curl_close($curl);
    return json_decode($response);
}

public function rules(): array
{
    return [
        'username' => [self::RULE_REQUIRED],
        'email' => [self::RULE_REQUIRED,
self::RULE_EMAIL],
        'password' => [self::RULE_REQUIRED,
[self::RULE_MIN, 'min' => 8]],
        'password_confirm' => [self::RULE_REQUIRED,
[self::RULE_MATCH, 'match' => 'password']],
    ];
}

```

```

public function labels(): array
{
    return [
        'username' => 'Username: ',
        'email' => 'Email: ',
        'password' => 'Password: ',
        'password_confirm' => 'Password Confirm:'
    ];
}

public function save()
{
    return self::getJsonResult([
        'url' => 'user',
        'method' => 'POST',
        'data' => [
            'username' => $this->username,
            'email' => $this->email,
            'password' => $this->password,
            'confirm_password' => $this->password_confirm
        ]
    ]);
}
}

```

Home.js

```

const prev = document.querySelector(".prev");
const next = document.querySelector(".next");
const next2 = document.getElementById("deuxiemeNext")
const carousel = document.querySelector(".carousel-
container");
const track = document.querySelector(".track");
const track2 = document.getElementById("track");
let width = carousel.offsetWidth;
let index = 0;
window.addEventListener("resize", function () {
    width = carousel.offsetWidth;
});

function Next(id) {
    let next = document.getElementById("next" + id);
    let track = document.getElementById("track" + id);
    let prev = document.getElementById("prev" + id);
    index = parseInt(next.dataset.index) + 1;
    next.setAttribute("data-index", parseInt(index));
    prev.setAttribute("data-index", parseInt(index));
    console.log(index);
    prev.classList.add("show");
    track.style.transform = "translateX(" + index * -width +
"px)";
    if (track.offsetWidth - index * width < index * width) {
        next.classList.add("hide");
    }
}

```

```

    }
}

function Prev(id) {
    let next = document.getElementById("next" + id);
    let track = document.getElementById("track" + id);
    let prev = document.getElementById("prev" + id);
    index = parseInt(next.dataset.index) - 1;
    prev.setAttribute("data-index", parseInt(index));
    next.setAttribute("data-index", parseInt(index));

    console.log(index);
    next.classList.remove("hide");
    if (index <= 0) {
        prev.classList.remove("show");
    }
    track.style.transform = "translateX(" + index * -width +
"px)";

    console.log(index * -width + "px");
}

```

Htaccess Site

```
Options -MultiViews
```

```
RewriteEngine On
```

```
RewriteBase /
```

```
# Ignore les fichiers et les dossiers réels
```

```
RewriteCond %{REQUEST_FILENAME} !-f
```

```
RewriteCond %{REQUEST_FILENAME} !-d
```

```
# Redirige toutes les requêtes vers index.php
```

```
RewriteRule ^(.*)$ index.php/$1 [L,QSA]
```

Index Site

```
<?php
```

```

use Eatfit\Site\Controllers\AboutController;
use Eatfit\Site\Controllers\HistoryController;
use Eatfit\Site\Controllers\RecipeController;
use Eatfit\Site\Controllers\SiteController;
use Eatfit\Site\Controllers\UserController;
use Eatfit\Site\Core\Application;

```

```
require_once __DIR__ . '/../vendor/autoload.php';
```

```

$config = [
    'API_URL' => "http://eatfittapi2023api/",
    'UPLOAD_PATH' => 'uploads/',

```

```
'MAX_FILE_SIZE' => 3 * 1024 * 1024, //(3mb)
'MAX_FILES_SIZE' => 70 * 1024 * 1024, //(70mb)
'ALLOWED_IMAGE_EXTENSIONS' => array('png', 'jpg', 'jpeg'),
];

$app = new Application(dirname(__DIR__), $config);

$app->router->get('/', [SiteController::class, 'home']);
$app->router->post('/', [SiteController::class, 'home']);

$app->router->get('/recipe', [RecipeController::class,
'create']);
$app->router->post('/recipe', [RecipeController::class,
'create']);

$app->router->get('/recipe/detail/{idRecipe}',
[RecipeController::class, 'detail']);
$app->router->post('/recipe/detail/{idRecipe}',
[RecipeController::class, 'detail']);

$app->router->get('recipe/edit/{idRecipe}',
[RecipeController::class, 'update']);
$app->router->post('recipe/edit/{idRecipe}',
[RecipeController::class, 'update']);
$app->router->get('recipe/delete/{idRecipe}',
[RecipeController::class, 'delete']);

$app->router->get('rating/delete/{idRating}',
[RecipeController::class, 'deleteRating']);
$app->router->get('rating/update/{idRating}',
[RecipeController::class, 'updateRating']);
$app->router->post('rating/update/{idRating}',
[RecipeController::class, 'updateRating']);

$app->router->get('/food_type', [RecipeController::class,
'addFoodType']);
$app->router->post('/food_type', [RecipeController::class,
'addFoodType']);

$app->router->get('/login', [UserController::class, 'login']);
$app->router->post('/login', [UserController::class,
'login']);
$app->router->get('/logout', [UserController::class,
'logout']);

$app->router->get('/register', [UserController::class,
'register']);
$app->router->post('/register', [UserController::class,
'register']);

$app->router->get('/profile', [UserController::class,
```



```
'profile']]);
$app->router->post('/profile/{method}/',
[UserController::class, 'profile']);

$app->router->get('/addToHistory/{idRecipe}',
[HistoryController::class, 'addToHistory']);
$app->router->get('/history', [HistoryController::class,
'history']);
$app->router->get('/history/delete',
[HistoryController::class, 'deleteHistory']);
$app->router->get('/history/delete/{idConsumedRecipe}',
[HistoryController::class, 'deleteHistory']);

// /{id}
$app->run();
```

main site

```
<?php
/** @var $this View */

use Eatfit\Site\Core\Application;
use Eatfit\Site\Core\View;

?>
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title><?= $this->title ?></title>

    <!-- Fonts -->
    <link
href="https://fonts.googleapis.com/css?family=Montserrat:400,7
00" rel="stylesheet">
    <link
href="https://fonts.googleapis.com/css?family=Roboto:400,700"
rel="stylesheet">

    <!-- Styles -->
    <link rel="stylesheet" type="text/css"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/b
ootstrap.min.css">
    <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/5.15.3/css/all.min.css"/>
    <link rel="stylesheet" type="text/css"
href="/css/style.css">
</head>
```

```

<body>
<header>
    <nav class="navbar navbar-expand-md navbar-dark">
        <a class="navbar-brand font-weight-bold"
href="/">Accueil</a>
        <button class="navbar-toggler" type="button" data-
toggle="collapse" data-target="#navbarNav"
            aria-controls="navbarNav" aria-
expanded="false" aria-label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="navbarNav">
            <ul class="navbar-nav mr-auto">
                <?php
                if (!Application::isGuest()) {
                    ?>
                    <li class="nav-item">
                        <a class="nav-link font-weight-bold"
href="/recipe">Ajouter une nouvelle recette</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link font-weight-bold"
href="/history">Historique des recettes consommées</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link font-weight-bold"
href="/food_type">Ajouter nouveau type de nourriture</a>
                    </li>
                <?php
                }
                ?>
            </ul>
            <ul class="navbar-nav ml-auto">
                <?php
                if (Application::isGuest()) {
                    ?>
                    <li class="nav-item">
                        <a class="nav-link font-weight-bold"
href="/login">Login</a>
                    </li>
                    <li class="nav-item">
                        <a class="nav-link font-weight-bold"
href="/register">Inscription</a>
                    </li>

                    <?php
                } else {
                    ?>
                    <li class="nav-item">
                        <a class="nav-link font-weight-bold"
href="/profile">Profil</a>

```

```

        </li>
        <li class="nav-item">
            <a class="nav-link font-weight-bold"
href="/logout">Se déconnecter</a>
        </li>
    <?php
    }
    ?>

</ul>
</div>
</nav>
</header>

<main>
    <div class="container">
        <?php

        if (Application::$app->session->getFlash('success')):
?>
            <div class="alert alert-success">
                <p><?php echo Application::$app->session-
>getFlash('success') ?></p>
            </div>
            <?php endif; ?>
            <?php

            if (Application::$app->session->getFlash('error')): ?>
                <div class="alert alert-danger">
                    <p><?php echo Application::$app->session-
>getFlash('error') ?></p>
                </div>
                <?php endif; ?>
                {{content}}
            </div>

</main>

<!-- Scripts -->
<script src="https://code.jquery.com/jquery-
3.3.1.slim.min.js"></script>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/u
md/popper.min.js"></script>
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/boo
tstrap.min.js"></script>
<script src="/js/home.js"></script>
<!--<script s></script>-->
</body>
</html>

```

_error site

```

<?php
/** @var $exception Exception */

use Eatfit\Site\Core\Application;

Application::$app->response->statusCode($exception-
>getCode());
$this->title = $exception->getCode();
?>
<main class="error-container">
    <div class="recipe-detail">
        <div class="recipe-title"><?php echo $exception-
>getMessage() ?></div>
    </div>
    <div class="consume-recipe-button-container">
        <a href="/" class="consume-recipe-button">Retour à la
page d'accueil</a>
    </div>
</main>

```

add_foodtype Site

```

<?php use Eatfit\Site\Core\Form\Form;
use Eatfit\Site\Core\View;
use Eatfit\Site\Models\FoodType;

/** @var $model FoodType */
/** @var $this View */
$this->title = 'Ajouter un nouveau type'; ?>

<h1>Ajouter nouveau type de nourriture</h1>

<?php $form = Form::begin("", "post") ?>
<div class="form-group">
    <!-- <label for="name">Food Type Name</label>-->
    <?php echo $form->field($model, 'name') ?>
    <div class="invalid-feedback">
        <?= $model->getFirstError('name') ?>
    </div>
</div>
<input type="submit" class="consume-recipe-button"
value="Ajouter">
<?php Form::end() ?>

```

add_recipe Site

```

<?php
/** @var $model Recipe */

```

```

use Eatfit\Site\Core\Application;
use Eatfit\Site\Core\Form\Form;
use Eatfit\Site\Models\FoodType;
use Eatfit\Site\Models\Recipe;

$this->title = 'Ajouter une recette';
?>

<main>
    <div class="container py-5">
        <h2 class="text-center mb-4">Ajouter une nouvelle
recette</h2>
        <div class="row justify-content-center">
            <div class="col-lg-8">
                <?php $form = Form::begin("", "post",
["enctype" => "multipart/form-data"]) ?>
                <div class="form-group">
                    <?php echo $form->field($model, 'title')
?>
                </div>
                <div class="form-group">
                    <?php echo $form->field($model,
'preparation_time')->numberField() ?>
                </div>
                <div class="form-group">
                    <?php echo $form->field($model,
'calories')->numberField() ?>
                </div>
                <div class="form-group">
                    <?php echo $form->field($model,
'difficulty')->selectField([
                        'facile' => 'Facile',
                        'moyen' => 'Moyen',
                        'difficile' => 'Difficile'
                    ]) ?>
                </div>
                <div class="form-group">
                    <?php echo $form->field($model,
'instructions')->textarea() ?>
                </div>
                <div class="form-group">
                    <?php
                        $categories = array();
                        foreach ($model->getCategories()->value as
$category) $categories[$category->name] = $category->name;
                        echo $form->field($model, 'categories')-
>selectField($categories) ?>
                </div>
            <div id="foodtype-container"></div>
        </div>
    </div>

```

```

        <div class="form-group">
            <label for="file-input">Image: </label>
            <br>
            <?php
                $acceptTypes = array();
                foreach
(Application::$ALLOWED_IMAGE_EXTENSIONS as $extension)
$acceptTypes[] = 'image/' . $extension;
                $acceptValue = implode(',', $acceptTypes);
                ?>
            <input type="file" multiple name="file-
input[]" accept="<?= $acceptValue; ?>">
        </div>

        <button type="button"
onclick="addFoodTypeField()">Ajouter un type de
nourriture</button>
        <button type="button"
onclick="removeFoodTypeField()">Supprimer le dernier type de
nourriture</button>

        <button type="submit" class="btn btn-primary
btn-block">Ajouter la recette</button>

        <?php Form::end(); ?>
        <!-- Ajouté après la balise de script du
graphique -->

        </div>
    </div>
</div>
</main>

<script>
    function addFoodTypeField() {
        var container = document.getElementById('foodtype-
container');
        var div = document.createElement('div');
        div.innerHTML = `
            <div class="form-group">
                <label for="foodtype">Type de nourriture
: </label>
                <select name="foodtype[]" class="form-
control">
                    <?php
                        foreach (FoodType::getFoodTypes()->value as $foodType)
{
                            echo "<option value='{ $foodType-
>name }'>{ $foodType->name}</option>";

```

```

    }
    ?>
        </select>
    </div>
    <div class="form-group">
        <label for="percentage">Pourcentage :</label>
        <input type="number" name="percentage[]"
class="form-control">
    </div>
    `;
    container.appendChild(div);
}

function removeFoodTypeField() {
    var container = document.getElementById('foodtype-
container');
    if (container.children.length > 0) {
        container.removeChild(container.lastChild);
    }
}

</script>

```

edit_recipe Site

```

<?php
/** @var $model Recipe */

use Eatfit\Site\Core\Application;
use Eatfit\Site\Core\Form\Form;
use Eatfit\Site\Models\FoodType;
use Eatfit\Site\Models\Recipe;

$this->title = 'Modifier une recette';
?>

<main>
    <div class="container py-5">
        <h2 class="text-center mb-4">Modifier la recette</h2>
        <div class="row justify-content-center">
            <div class="col-lg-8">
                <?php $form = Form::begin("", "post",
["enctype" => "multipart/form-data"]) ?>
                <div class="form-group">
                    <?php echo $form->field($model, 'title')
                ?>
            </div>
            <div class="form-group">
                <?php echo $form->field($model,
'preparation_time')->numberField() ?>
            </div>

```

```

        <div class="form-group">
            <?php echo $form->field($model,
'calories')->numberField() ?>
        </div>
        <div class="form-group">
            <?php echo $form->field($model,
'difficulty')->selectField([
                'facile' => 'Facile',
                'moyen' => 'Moyen',
                'difficile' => 'Difficile'
            ]) ?>
        </div>
        <div class="form-group">
            <?php echo $form->field($model,
'instructions')->textarea() ?>
        </div>
        <div class="form-group">
            <?php
                $categories = array();
                foreach ($model->getCategories()->value as
$category) $categories[$category->name] = $category->name;
                echo $form->field($model, 'categories')-
>selectField($categories) ?>
        </div>
        <div id="foodtype-container"></div>
        <table>
            <tr>
                <th>Type de nourriture</th>
                <th>Pourcentage</th>
            </tr>
            <?php foreach (FoodType::getFoodTypes()-
>value as $foodType):
                ?>
                <tr>
                    <td><?= $foodType->name ?></td>
                    <td>
                        <input type="number"
name="percentage[<?= $foodType->name ?>]" class="form-control"
                        value="<?= $model-
>foodType[trim($foodType->name)] ?? ' ' ?>">
                    </td>
                </tr>
            <?php endforeach; ?>
        </table>
        <div class="form-group">
            <br>
            <div class="image-container">
                <?php foreach ($model->image as
$image):
                    if (str_contains($image,
'default')) continue;

```



```

        ?>
        <div class="image-wrapper">
            " alt="<?=
$image ?>">
        </div>
    <?php endforeach; ?>
</div>
<br>
<label class="label-dark" for="file-
input">Ajouter une image: </label>
<br>
<?php
$acceptTypes = array();
foreach
(Application::<ALLOWED_IMAGE_EXTENSIONS as $extension)
$acceptTypes[] = 'image/' . $extension;
$acceptValue = implode(',', $acceptTypes);
?>
<label for="default">Image par défaut
    <input type="checkbox" name="default">
</label>
<br>
<input type="file" multiple name="file-
input[]" accept="<?= $acceptValue; ?>">
</div>
<button type="submit" class="btn btn-primary
btn-block">Mettre à jour la recette</button>
<?php Form::end(); ?>
</div>
</div>
</main>

```

edit_rating Site

```

<?php
/** @var $model Rating */

use Eatfit\Site\Core\Form\Form;
use Eatfit\Site\Models\Rating;

//var_dump($model);

?>
<div class="container mt-5">
    <div class="row">
        <div class="col-md-8 offset-md-2">
            <h2 class="form-label label-dark">Modifier
l'évaluation de la recette</h2>
            <?php $form = Form::begin("", "post") ?>

```

```

        <div class="mb-3">
            <?= $form->field($model, "score") -
>numberField() ?>
        </div>
        <div class="mb-3">
            <?= $form->field($model, "comment") -
>textarea() ?>
        </div>
        <div class="mb-3">
            <button type="submit" class="btn btn-primary-
custom btn-lg btn-block">Mettre à jour l'évaluation
            </button>
        </div>
        <?php Form::end() ?>
    </div>
</div>

```

history Site

```

<?php
/** @var $user array */

/** @var $model History */

use Eatfit\Site\Models\History;

$this->title = 'Historique';
?>

<h1>Historique des recettes consommées</h1>
<div class="table-container">
    <table>
        <thead>
            <tr>
                <th>Identifiant de la recette</th>
                <th>Titre de la recette</th>
                <th>Date de la réalisation</th>
                <th>Moment de la consommation</th>
                <th>Actions</th>
            </tr>
        </thead>
        <tbody>
            <?php
            $history = $model->getHistory()->value;
            if ($history == null) {
                echo "<tr><td colspan='5'>Aucune recette
consommée</td></tr>";
            } else
                foreach ($history as $item): ?>

```

```

        <tr>
            <td data-column="Identifiant de la
recette"><?php echo $item->idRecipe; ?></td>
            <td data-column="Titre de la
recette"><?php echo $item->title; ?></td>
            <td data-column="Date de la
réalisation"><?php echo $item->created_at; ?></td>
            <td data-column="Moment de la
consommation"><?php echo $item->consumption_date; ?></td>
            <td data-column="Actions">
                <a href="/history/delete/<?php echo
$item->idConsumedRecipe; ?>">Supprimer cette recette</a>
            </td>
        </tr>
    <?php endforeach; ?>
</tbody>
</table>
</div>
<a href="/history/delete">Supprimer tout l'historique</a>

```

home Site

```

<?php
/** @var $model Recipe */

use Eatfit\Site\Core\Application;
use Eatfit\Site\Core\Form\Form;
use Eatfit\Site\Models\FoodType;
use Eatfit\Site\Models\Recipe;

$this->title = 'Accueil';
// Cette fonction génère un carrousel HTML avec des recettes
filtrées selon plusieurs critères.
function renderCarouselWithSearch($category,
$recipesByCategory, $carouselId, $searchQuery,
$categoryFilter, $foodtypeFilter, $usernameFilter): void
{
    // On filtre les recettes en fonction des critères de
recherche et des filtres spécifiés.
    $recipesToDisplay = array_filter($recipesByCategory,
function ($recipe) use ($searchQuery, $categoryFilter,
$foodtypeFilter, $usernameFilter) {
        // On vérifie si le titre, la catégorie, ou la date de
la recette contiennent la requête de recherche.
        $titleContainsSearch = empty($searchQuery) ||
stripos($recipe->recipe_title, $searchQuery) !== false;
        $categoryContainsSearch = empty($searchQuery) ||
stripos($recipe->categories, $searchQuery) !== false;
        $dateContainsSearch = empty($searchQuery) ||
stripos($recipe->created_at, $searchQuery) !== false;
        // On vérifie si la catégorie, le type de nourriture,

```

```

ou le nom d'utilisateur de la recette correspondent aux
filtres spécifiés.
    $categoryMatchesFilter = empty($categoryFilter) ||
$recipe->categories == $categoryFilter;
    $foodtypeMatchesFilter = empty($foodtypeFilter) ||
stripos($recipe->foodtypes_with_percentages, $foodtypeFilter)
!= false;
    $usernameMatchesFilter = empty($usernameFilter) ||
$recipe->creator_username == $usernameFilter;
    // On retourne vrai si la recette satisfait tous les
critères.
    return ($titleContainsSearch ||
$categoryContainsSearch || $dateContainsSearch) &&
$categoryMatchesFilter && $foodtypeMatchesFilter &&
$usernameMatchesFilter;
});
// Si aucune recette ne satisfait les critères, on ne
génère pas le carrousel.
if (count($recipesToDisplay) === 0) return;
?>
<div class="carousel-container">
    <h1><?= $category->name ?></h1>
    <div class="inner-carousel">
        <div class="track" id="track<?= $carouselId ?>">
            <?php
                // Pour chaque recette à afficher, on génère
une carte avec les détails de la recette.
                foreach ($recipesToDisplay as $recipe) {
                    // On récupère les chemins des images de
la recette.
                    $recipe->image_paths = !empty($recipe-
>image_paths) && str_contains($recipe->image_paths, ',') ?
array_map('trim', explode(',', $recipe->image_paths)) :
array($recipe->image_paths);
                    if (str_contains($recipe->image_paths[0],
'default')) $recipe->image_paths = [explode('_', $recipe-
>image_paths[0])[1]];
                    ?>
                    <div class="card-container">
                        <div class="container">
                            <div class="row">
                                <div class="menu-item">
                                    <!-- On génère un lien
vers la page de détails de la recette, avec une image de la
recette. -->
                                    <a
href="/recipe/detail/<?= $recipe->recipe_id ?>">image_paths[0]
?>"
alt="<?= $recipe-

```

```

>recipe_id ?>" class="rounded"></a>
                                <div>
                                    <!-- On affiche le
titre et les instructions de la recette. -->
                                <h3 style="text-align:
left;"><?= $recipe->recipe_title ?></h3>
                                <p style="text-align:
left;"><?= $recipe->recipe_instructions ?></p>
                                </div>
                            </div>
                        </div>
                    </div>
                </div>
            <?php
        }
    ?>
</div>
<!-- On génère les boutons de navigation du
carrousel (précédent et suivant). -->
    <div class="nav" <?php if
(count($recipesToDisplay) < 5) echo 'style="display: none;"'
?>>
        <button class="prev" data-index="" id="prev<?=
$carouselId ?>" onclick="Prev(<?= $carouselId ?>)"><i
            class="fas fa-arrow-left fa-
2x"></i></button>
        <button class="next" data-index="0"
id="next<?= $carouselId ?>" onclick="Next(<?= $carouselId
?>)"><i
            class="fas fa-arrow-right fa-
2x"></i></button>
        </div>
    </div>
</div>
<?php
}

$form = Form::begin('', "post");
?>
    <div class="search-bar">
        <input type="text" id="search-input" name="search"
placeholder="Rechercher des recettes...">
        <button id="search-button">Rechercher</button>
    </div>
    <div class="filter-container">
        <select id="category-filter" name="category-filter">
            <option value="">Toutes les catégories</option>
            <?php foreach ($model->getCategories()->value as
$category) { ?>
                <option value="<?= $category->name ?>"><?=

```

```

$category->name ?></option>
    <?php } ?>
</select>

    <select id="foodtype-filter" name="foodtype-filter">
        <option value="">Tous les types
d'aliments</option>
        <?php foreach (FoodType::getFoodTypes()->value as
$foodtype) { ?>
            <option value="<?= $foodtype->name ?>"><?=
$foodtype->name ?></option>
            <?php } ?>
        </select>
        <input type="text" id="username-filter"
name="username-filter" placeholder="Filtrer par
utilisateur..."/>
        <button id="filter-button">Filtrer</button>
    </div>

<?php
Form::end();
$j = 0;
$searchQuery = $_POST['search'] ?? '';
$categoryFilter = $_POST['category-filter'] ?? '';
$foodtypeFilter = $_POST['foodtype-filter'] ?? '';
$usernameFilter = $_POST['username-filter'] ?? '';
foreach ($model->getCategories()->value as $category) {
    $recipesByCategory = $model->getRecipeByFilter("category",
$category->name)->value;
    if ($recipesByCategory === null) continue;
    renderCarouselWithSearch($category, $recipesByCategory,
$j, $searchQuery, $categoryFilter, $foodtypeFilter,
$usernameFilter);
    $j++;
}

```

login Site

```

<?php

/** @var $model LoginForm */

/** @var $this View */

use Eatfit\Site\Core\Form\Form;
use Eatfit\Site\Core\View;
use Eatfit\Site\Models\LoginForm;

$this->title = 'Login';
?>

```

```

<div class="row justify-content-center">
    <div class="col-md-9">
        <div class="login-form p-4 mt-5">
            <h2 class="text-center mb-4">Login</h2>
            <?php $form = Form::begin('', 'post') ?>
            <div class="mb-3">
                <?php echo $form->field($model, 'email')-
            >setPlaceholder("Entrez votre email") ?>
                <small class="form-text text-danger help-
block"></small>
            </div>
            <div class="mb-3">
                <?php echo $form->field($model, 'password')-
            >passwordField()->setPlaceholder("Entrez votre mot de passe")
            ?>
                <small class="form-text text-danger help-
block"></small>
            </div>
            <div class="d-grid gap-2">
                <button type="submit" class="btn btn-
primary">Se connecter</button>
            </div>
            <br>
            <div class="mb-3">
                <a href="/register"> Vous inscrire ?</a>
            </div>
            <?php if ($model->getFirstError("error") != null)
echo '    <small class="form-text text-danger help-block">' .
$model->getFirstError("error") . '</small>' ?>
                <?php Form::end() ?>
            </div>
        </div>
    </div>
</div>

```

profile Site

```

<?php
/** @var $user array */

/** @var $model ProfileModel */

/** @var $this View */

use Eatfit\Site\Core\Form\Form;
use Eatfit\Site\Core\View;
use Eatfit\Site\Models\ProfileModel;

$this->title = 'Profile';
?>
<h2 class="text-uppercase text-center text-secondary mb-0">
    >Votre profil</h2>

```

```

<hr class="star-dark mb-5 border-dark">
<div class="ms-0 me-0 row">
  <div class="col-lg-8 mx-auto">
    <?php $form = Form::begin('/profile/update', 'post',
    ['id' => 'updateProfileForm']) ?>
    <div class="control-group mb-3 row">
      <label class="form-label col label-dark"
style="margin: auto;">Email : <?= $user->email ?></label>
      <button class="consume-recipe-button"
type="button" onclick="edit('email')">Modifier</button>
    </div>
    <div class="control-group row" id="email"></div>
    <div class="control-group mb-3 row">
      <label class="form-label col label-dark"
style="margin: auto;">Username
      : <?= $user->username ?></label>
      <button class="consume-recipe-button"
type="button" onclick="edit('username')">Modifier</button>
    </div>
    <div class="control-group row" id="username"></div>
    <div class="control-group mb-3 row">
      <label class="form-label col label-dark"
style="margin: auto;">Mot de
      passe : </label>
      <button class="consume-recipe-button"
type="button" onclick="edit('password')">Modifier</button>
    </div>
    <div class="control-group row" id="password"></div>
    <div class="control-group mt-3 row">
      <button class="consume-recipe-button"
id="sendMessageButton" type="submit">Valider</button>
    </div>
    <?php Form::end() ?>

    <?php $form = Form::begin('/profile/delete', 'post')
?>

    <div class="control-group mt-3 row">
      <button class="btn btn-danger btn-xl"
id="deleteAccountButton" type="submit">Supprimer le
compte</button>
    </div>
    <?php Form::end() ?>
  </div>
</div>
<script>
  function edit(id) {
    let div = document.getElementById(id);
    if (id === "password") {
      if (div.innerHTML === "") {
        div.innerHTML += '<input class="form-control
mt-2" type="password" name="password" placeholder="Nouveau mot

```



```

de passe">';
        div.innerHTML += '<input class="form-control
mt-2" type="password" name="confirm_password"
placeholder="Confirmer le mot de passe">';
        div.innerHTML += '<div class="invalid-
feedback"></div>';
    } else div.innerHTML = "";
} else if (id === "username") {
    let placeholder = "Nouveau pseudo";
    if (div.innerHTML === "") div.innerHTML = '<input
class="form-control mb-3" name="' + id + '" placeholder="' +
placeholder + '" type="text">';
    else div.innerHTML = "";
} else if (id === "email") {
    let placeholder = "Nouveau email";
    if (div.innerHTML === "") div.innerHTML = '<input
class="form-control mb-3" name="' + id + '" placeholder="' +
placeholder + '" type="email">';
    else div.innerHTML = "";
}
}
</script>

```

recipe_details Site

```

<?php
/** @var $recipe */
/** @var $ratings Rating */

/** @var $this View */

use Eatfit\Site\Core\Application;
use Eatfit\Site\Core\Form\Form;
use Eatfit\Site\Core\View;
use Eatfit\Site\Models\Rating;

$this->title = "Détails de la recette";

$comments = $ratings->getRatingByIdRecipe()->value;
?>

<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/boot
strap.min.css">
<link rel="stylesheet" href="/css/style.css">
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.4/jquery
.min.js"></script>
<script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootst
rap.min.js"></script>

```

```

<div class="recipe-detail">
  <div class="container">
    <div id="myCarousel" class="carousel slide" data-
ride="carousel">
      <!-- Indicators -->
      <ol class="carousel-indicators">
        <?php foreach ($recipe->image_paths as $index
=> $image): ?>
          <li data-target="#myCarousel"
              data-slide-to="<?= $index ?>" <?=
$index === 0 ? 'class="active"' : '' ?>></li>
          <?php endforeach; ?>
        </ol>

      <!-- Wrapper for slides -->
      <div class="carousel-inner">
        <?php foreach ($recipe->image_paths as $index
=> $image): ?>
          <?php if (str_contains($image, 'default'))
$image = explode('_', $image)[1]; ?>
          <div class="item <?= $index === 0 ?
'active' : '' ?>">
            " alt="Image de la recette"
                style="width:600px;
height:auto;">
          </div>
          <?php endforeach; ?>
        </div>

      <!-- Left and right controls -->
      <a class="left carousel-control"
href="#myCarousel" data-slide="prev">
        <span class="glyphicon glyphicon-chevron-
left"></span>
        <span class="sr-only">Previous</span>
      </a>
      <a class="right carousel-control"
href="#myCarousel" data-slide="next">
        <span class="glyphicon glyphicon-chevron-
right"></span>
        <span class="sr-only">Next</span>
      </a>
    </div>
  </div>
  <!-- -->
  <h2 class="recipe-title" id="recipe-title"><?= $recipe-
>recipe_title ?></h2>

```

```

    <p id="recipe-author">Recette numéro : <?= $recipe-
>recipe_id ?></p>
    <p id="recipe-author">Créé par : <?= $recipe-
>creator_username ?></p>
    <p> Catégories : <?= implode(",", $recipe->categories)
?></p>
    <p id="recipe-difficulty">Difficulté : <?= $recipe-
>difficulty ?></p>
    <p id="recipe-calories">Calories : <?= $recipe->calories
?></p>
    <p id="recipe-calories">Note moyenne : <?= round($recipe-
>average_rating, 1) ?></p>
    <div class="row">
        <div class="col-md-12">
            <h3 class="instructions-title">Instructions</h3>
            <p> <?= $recipe->recipe_instructions ?> </p>
        </div>
    </div>
    <div class="row mt-4">
        <div class="col-md-6">
            <h3 class="prep-time-title">Temps de
préparation</h3>
            <p class="prep-time" id="prep-time"><?= $recipe-
>preparation_time ?> minutes</p>
        </div>
        <div class="col-md-6">
            <h3 class="creation-date-title">Date de
création</h3>
            <p class="creation-date" id="creation-date"><?=
date('d-m-Y', strtotime($recipe->created_at)) ?></p>
        </div>
    </div>

    <center>
        <canvas id="myChart" width="400"
height="400"></canvas>
    </center>
    <div class="consume-recipe-button-container">
        <a href="/addToHistory/<?= $recipe->recipe_id ?>">
            <button name="history" class="consume-recipe-
button">Consommer cette recette</button>
        </a>
        <?php
        if (Application::$app->user->idUser == $recipe-
>creator_id) {
            ?>
            <a href="/recipe/edit/<?= $recipe->recipe_id ?>">
                <button class="consume-recipe-
button">Modifier</button>
            </a>

```

```

        <a href="/recipe/delete/<?= $recipe->recipe_id
?>">
            <button class="consume-recipe-
button">Supprimer</button>
        </a>
        <?php
    }
    ?>
</div>
<div class="mt-5">
    <h3 class="mb-4 ">Commentaires</h3>
    <?php if (!$comments) : ?>
        <div class="alert background-color-dark font-
color-light" role="alert">
            Aucun commentaire pour cette recette pour le
moment. Soyez le premier à commenter !
        </div>
    <?php else: ?>
        <?php $form = Form::begin("", "post") ?>
        <?php foreach ($comments as $comment): ?>
            <div class="card mb-3 border-dark">
                <div class="card-header background-color-
dark">
                    <div class="row align-items-center">
                        <div class="col">
                            <strong class="font-color-
highlight"><?= $comment->username ?></strong>
                        </div>
                        <div class="col text-end">
                            <span class="badge bg-primary-
custom"><?= $comment->score ?> / 5</span>
                        </div>
                    </div>
                </div>
                <div class="card-body background-color-
light">
                    <p class="card-text font-color-main">
                        <?php
                            if ($comment->comment == null)
echo "Aucun commentaire";
                        ?>

                        <?php
                            if (Application::$app->user-
>idUser == $comment->idUser) {
                                ?>
                                <div class="d-flex justify-content-
end">
                                    <a href="/rating/update/<?=
$comment->idRating ?>"
                                        class="btn btn-primary-custom

```

```

btn-sm mr-2">
    Modifier
    </a>
    <a href="/rating/delete/<?=$comment->idRating ?>"
        class="btn btn-danger-custom
btn-sm">
    Supprimer
    </a>
</div>
<?php
}
?>
</div>
</div>
<?php endforeach; ?>
<?php Form::end() ?>
<?php endif; ?>
</div>

<div class="mt-4">
    <h3>Ajouter un commentaire</h3>
    <?php $form = Form::begin("", "post") ?>
    <div class="form-group">
        <?=$form->field($ratings, 'score')-
>numberField(); ?>
    </div>
    <div class="form-group">
        <?=$form->field($ratings, 'comment')->textarea();
    ?>
    </div>
    <button type="submit" class="btn btn-primary mt-
2">Envoyer</button>
    <?php Form::end() ?>
</div>
<script
src="https://cdn.jsdelivr.net/npm/chart.js"></script>
<script>
    function generateRandomColorAround(rgb = [146, 76,
22]) {
        var variation = 60;
        var r = Math.max(Math.min(Math.round(rgb[0] +
(Math.random() - 0.5) * 2 * variation), 255), 0);
        var g = Math.max(Math.min(Math.round(rgb[1] +
(Math.random() - 0.5) * 2 * variation), 255), 0);
        var b = Math.max(Math.min(Math.round(rgb[2] +
(Math.random() - 0.5) * 2 * variation), 255), 0);
        return 'rgb(' + r + ', ' + g + ', ' + b + ')';
    }

```

```

<?php
$noms = [];
$pourcentages = [];

foreach ($recipe->foodtypes_with_percentages as
$foodtype_with_percentage) {
    preg_match('/^(.*?)\s+\((\d+)\%\)$/',
$foodtype_with_percentage, $matches);
    $noms[] = $matches[1]; // Ajoute le nom du type
d'aliment au tableau des noms
    $pourcentages[] = (int)$matches[2]; // Ajoute le
pourcentage au tableau des pourcentages
}
?>

var foodTypes = <?php echo json_encode($recipe-
>foodtypes_with_percentages); ?>;
var backgroundColors = [];
var borderColors = [];

foodTypes.forEach(function (foodType) {
    var color = generateRandomColorAround();
    backgroundColors.push(color);
    borderColors.push(color);
});

var ctx =
document.getElementById('myChart').getContext('2d');
var myChart = new Chart(ctx, {
    type: 'pie',
    data: {
        labels: foodTypes.map(function (foodType) {
            return foodType;
        }),
        datasets: [{
            data: [<?php echo implode(", ",
$pourcentages); ?>],
            backgroundColor: backgroundColors,
            borderColor: borderColors,
            borderWidth: 1
        }]
    },
    options: {
        responsive: false,
    }
});
</script>

</div>

```

recipe_details_guest Site

```

<?php
/** @var $recipe */

/** @var $ratings Rating */

use Eatfit\Site\Core\Application;
use Eatfit\Site\Core\Form\Form;
use Eatfit\Site\Models\Rating;

$this->title = "Détails de la recette";
$comments = $ratings->getRatingByIdRecipe(false)->value;
?>
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/boot
strap.min.css">
<link rel="stylesheet" href="/css/style.css">
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.4/jquery
.min.js"></script>
<script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootst
rap.min.js"></script>

<div class="recipe-detail">
    <div class="container">
        <div id="myCarousel" class="carousel slide" data-
ride="carousel">
            <!-- Indicators -->
            <ol class="carousel-indicators">
                <?php foreach ($recipe->image_paths as $index
=> $image): ?>
                    <li data-target="#myCarousel"
                        data-slide-to="<?= $index ?>" <?=
$index === 0 ? 'class="active"' : '' ?>></li>
                <?php endforeach; ?>
            </ol>

            <!-- Wrapper for slides -->
            <div class="carousel-inner">
                <?php foreach ($recipe->image_paths as $index
=> $image): ?>
                    <?php if (str_contains($image, 'default'))
$index = explode('_', $image)[1]; ?>
                    <div class="item <?= $index === 0 ?
'active' : '' ?>">
                        " alt="Image de la recette"
                            style="width:100%;

```

```

height:450px;">
        </div>
        <?php endforeach; ?>
    </div>

    <!-- Left and right controls -->
    <a class="left carousel-control"
href="#myCarousel" data-slide="prev">
        <span class="glyphicon glyphicon-chevron-
left"></span>
        <span class="sr-only">Previous</span>
    </a>
    <a class="right carousel-control"
href="#myCarousel" data-slide="next">
        <span class="glyphicon glyphicon-chevron-
right"></span>
        <span class="sr-only">Next</span>
    </a>
</div>
</div>
<h2 class="recipe-title" id="recipe-title"><?= $recipe-
>recipe_title ?></h2>
<p id="recipe-author">Cr    par : <?= $recipe-
>creator_username ?></p>
<p> Cat   gories : <?= implode(",", $recipe->categories)
?></p>
<p id="recipe-difficulty">Difficult    : <?= $recipe-
>difficulty ?></p>
<p id="recipe-calories">Note moyenne : <?= round($recipe-
>average_rating, 1) ?></p>
<p id="recipe-calories">Temps de pr   paration : <?=
$recipe->preparation_time ?> minutes</p>

<div class="mt-5">
    <h3 class="mb-4 ">Commentaires</h3>
    <?php if (!$comments) : ?>
        <div class="alert background-color-dark font-
color-light" role="alert">
            Aucun commentaire pour cette recette pour le
moment. Soyez le premier     commenter ! - <a
href="/register"> Vous inscrire ?</a> / <a
href="/login"> Vous connecter ?</a>
        </div>
    <?php else: ?>
        <?php $form = Form::begin("", "post") ?>
        <?php foreach ($comments as $comment): ?>
            <div class="card mb-3 border-dark">
                <div class="card-header background-color-
dark">
                    <div class="row align-items-center">

```



```

        <div class="col">
            <strong class="font-color-
highlight"><?= $comment->username ?></strong>
        </div>
        <div class="col text-end">
            <span class="badge bg-primary-
custom"><?= $comment->score ?> / 5</span>
        </div>
    </div>
</div>
<div class="card-body background-color-
light">
    <p class="card-text font-color-main">
<?php
        if ($comment->comment == null)
echo "Aucun commentaire";
    ?></p>
</div>
</div>
<?php endforeach; ?>
<?php Form::end() ?>
<?php
endif; ?>
</div>
</div>

```

register Site

```

<?php
/** @var $model User */

use Eatfit\Site\Core\Form\Form;
use Eatfit\Site\Models\User;

$this->title = 'Register';
$form = new Form();
?>

<div class="row justify-content-center">
    <div class="col-md-9">
        <div class="login-form mt-5 p-4">
            <h2 class="text-center mb-4">Inscription</h2>
            <?php $form = Form::begin('', 'post') ?>
            <div class="mb-3">
                <?php echo $form->field($model, 'username')-
>setPlaceholder("Entrez votre nom d'utilisateur") ?>
                <small class="form-text text-danger help-
block"></small>
            </div>
            <div class="mb-3">
                <?php echo $form->field($model, 'email')-

```

```

>setPlaceholder("Entrez votre email") ?>
    <small class="form-text text-danger help-
block"></small>
    </div>
    <div class="mb-3">
        <?php echo $form->field($model, 'password')-
>passwordField()->setPlaceholder("Entrez votre mot de passe")
?>
        <small class="form-text text-danger help-
block"></small>
    </div>
    <div class="mb-3">
        <?php echo $form->field($model,
'password_confirm')->passwordField()-
>setPlaceholder("Confirmez votre mot de passe") ?>
        <small class="form-text text-danger help-
block"></small>
    </div>
    <div class="d-grid gap-2">
        <button type="submit" class="btn btn-
primary">S'inscrire</button>
    </div>
    <div class="mt-3 text-center">
        <a href="/login" class="text-muted">Déjà un
compte ? Connectez-vous ici</a>
    </div>
    <?php Form::end() ?>
</div>
</div>

```

Composer.json

```

{
    "name": "eatfit/site",
    "autoload": {
        "psr-4": {
            "Eatfit\\Site\\": "."
        }
    },
    "authors": [
        {
            "name": "LAAC172003",
            "email": "61194486+LAAC172003@users.noreply.github.com"
        }
    ],
    "require": {
        "ext-curl": "*"
    }
}

```