

WIMF - What's in my fridge



Trouve des idées recettes avec les ingrédients de ton frigo

Projet réalisé dans le cadre de la présentation au
**Titre Professionnel Développeur Web et Web
Mobile**

présenté par

Antoine PETIT

La Plateforme - Promotion 2025/2026

Sommaire

Sommaire	3
Introduction	5
Résumé du projet	6
Cahier des charges	7
I. Besoins et objectifs	7
A.	7
B. Objectifs du projet	7
C. Besoins fonctionnels et techniques	7
II. Personae et User Stories	11
A. Personae	11
B. User Stories	12
III. MVP	13
A. Objectif principal du MVP	13
B. Fonctionnalités retenues (Must Have)	13
C. Fonctionnalités exclues du MVP	13
D. Parcours utilisateur MVP	13
IV. Contraintes techniques et organisationnelles	14
A. Techniques	14
B. Organisationnelles	15
V. Wireframes	15
VI. Charte graphique	16
A. Ton général	16
B. Logo	17
C. Palette de couleurs et typographie	17
D. Exemple de maquettes	18
Gestion de projet	22
I. Méthodologie	22
II. Planning et outils utilisés	22
Analyse et conception	25
I. Base de données	25
II. Back-end	26
III. Front-end	28
IV. Frameworks utilisés	29
Développement	31
Scanner d'ingrédients	31
A. Back-end	31
B. Front-end	35
Génération de recette	38
A. Back-end	38
B. Front-end	39
Authentification utilisateur	40
A. Back-end	40

B. Front-end	44
Tests et validation	50
Tests unitaires	50
Test de performance et accessibilité	51
Tests de compatibilité	52
Problèmes rencontrés et résolutions	53
Bilan et conclusion	54
Annexe	55

Introduction

Depuis mon plus jeune âge, j'ai toujours été passionné d'informatique, que ce soit au travers du jeu vidéo, du montage vidéo ou de la programmation. Pour ces raisons, j'ai décidé, dès que je l'ai pu au lycée, de découvrir les métiers du numérique. J'ai ainsi découvert Python, HTML et CSS. Suite au lycée, mon parcours universitaire s'est compliqué : je me suis éloigné de la programmation et j'ai fini par abandonner mes études.

Passionné par le jeu vidéo, j'ai eu une courte carrière professionnelle de joueur sur le jeu *Rainbow Six Siege*. Lorsque cette aventure s'est terminée, j'ai voulu me former en programmation, et j'ai donc été logiquement attiré par la formation de développeur à la carte en premier lieu, puis **Développeur Web & Web Mobile** par la suite.

Au cours de cette formation, j'ai développé beaucoup de compétences et j'ai pu prendre plaisir à travailler dans le domaine du développement web. Ayant un projet à réaliser pour valider mon titre, j'ai pu traiter d'une problématique qui peut concerner tout le monde.

En effet, j'ai observé que les gens ont régulièrement des difficultés à trouver des idées de recettes à cuisiner à partir des ingrédients qu'ils ont en leur possession. L'idée de WIMF est de fournir une solution à ces utilisateurs avec une application web proposant une fonctionnalité d'analyse d'ingrédients à partir d'une photo d'un réfrigérateur, permettant de proposer des recettes ou d'en générer grâce à l'intelligence artificielle à partir des ingrédients disponibles.

Résumé du projet

J'ai remarqué, en discutant avec mon entourage, que les gens ont régulièrement du mal à trouver quoi cuisiner avec les ingrédients dans leur réfrigérateur. Il m'a également été dit que beaucoup de sites de cuisine n'étaient pas adaptés aux personnes souffrant de troubles de l'alimentation. Ces deux problèmes ont été soulevés par une observation personnelle et par une analyse auprès des utilisateurs grâce à un questionnaire en ligne posté sur X. Comme j'aime beaucoup cuisiner, j'ai décidé de développer un site de cuisine qui permettrait de résoudre ces deux problèmes.

J'ai donc décidé de créer l'application WIMF (pour What's In My Fridge) pour aider ces personnes à cuisiner à partir des ingrédients à leur disposition, afin de leur retirer cette charge mentale, notamment à la fin d'une journée de travail. Le projet permettra également aux personnes atteintes de troubles de naviguer librement sur l'application sans avoir à s'inquiéter de voir des informations pouvant déclencher leur trouble.

L'application aura pour cible plusieurs groupes de personnes. La cible principale sera les utilisateurs de moins de 30 ans souhaitant simplifier la manière de trouver des recettes, confortables avec l'utilisation d'un appareil mobile, pour une utilisation régulière à quotidienne. La cible secondaire sera les personnes atteintes de troubles de l'alimentation, comme évoqué précédemment, sur mobile et ordinateur. Enfin, la version ordinateur ciblera tous les autres utilisateurs cherchant un site de cuisine simple.

Ce projet me tient à cœur car j'aime beaucoup cuisiner, et j'aimerais résoudre des problématiques auxquelles mon entourage et moi avons déjà eu à faire face. Je pense qu'il est important de développer un outil qui m'intéresse et qui sera utile au plus grand nombre autant qu'à moi.

Cahier des charges

I. Besoins et objectifs

A. Objectifs du projet

Je définis d'abord plusieurs objectifs pour le projet en suivant la méthode SMART :

- Réaliser un outil d'analyse de frigo et de génération de recettes à l'aide d'une IA (Gemini), fonctionnel avant janvier 2025.
- Réaliser un site de cuisine accessible aux personnes atteintes de troubles de l'alimentation avec une personnalisation de l'expérience utilisateur pour tous.
- Optimiser le site en version mobile afin d'en faire une application téléchargeable pour de meilleures performances.

B. Besoins fonctionnels et techniques

Pour bien définir les besoins du projet, je commence par identifier les acteurs internes et externes impliqués dans le projet, ainsi que leurs rôles :

→ Antoine Petit :

- ◆ Porte la vision du projet
- ◆ Planifie et organise le projet
- ◆ Conçoit l'interface et l'expérience utilisateur
- ◆ Réalise les fonctionnalités techniques

→ Utilisateurs :

- ◆ Destinataires de la solution
- ◆ Effectuent des retours/tests
- ◆ Participe à des sondages durant l'élaboration du projet ainsi qu'après sa publication

Les utilisateurs sont divisés en 2 catégories : les visiteurs et les membres. Je réalise une grille d'analyse des besoins afin de définir quels besoins sont fonctionnels ou non fonctionnels, ainsi que les parties prenantes :

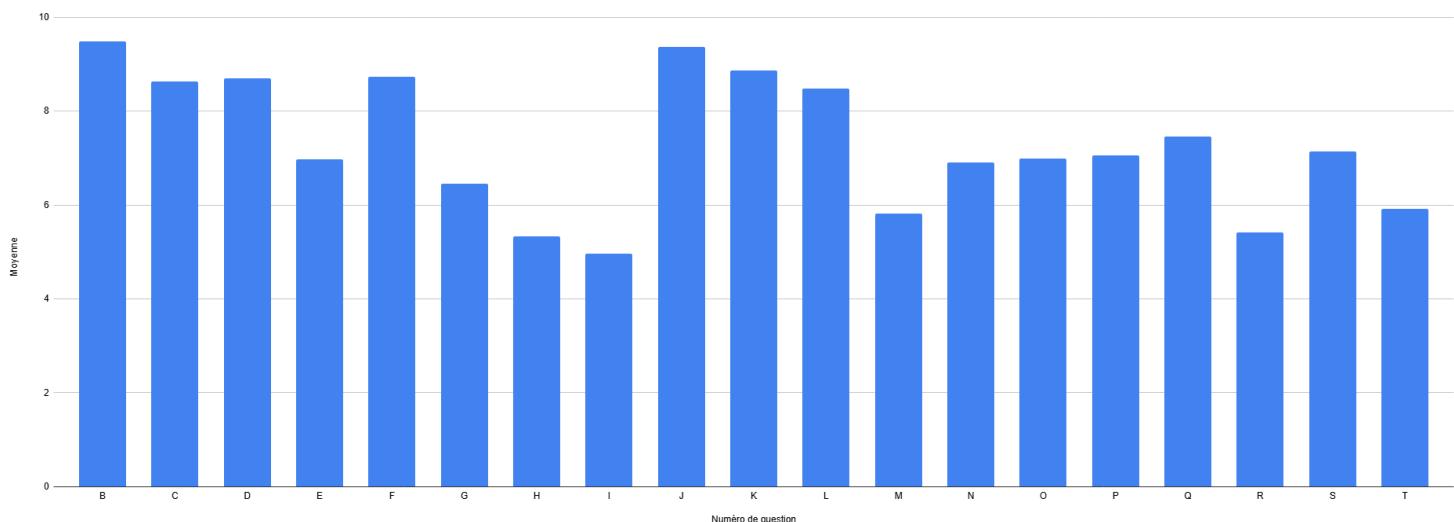
Partie prenante	Besoin exprimé	Fonctionnel	Non fonctionnel
Visiteurs	Créer un compte	X	
Visiteurs, membres	Rechercher une recette	X	
Visiteurs, membres	Utiliser l'outil d'analyse de frigo	X	
Membres	Gérer le compte	X	
Membres	Personnaliser l'expérience sur le site	X	X
Membres	Commenter les recettes	X	
Membres	Ajouter aux favoris/Like une recette	X	
Visiteurs, membres	Utiliser l'outil sur mobile, tablette et desktop		X
Membres	Gérer les préférences de régime alimentaire	X	X
Visiteurs, membres	Avoir des informations sur les recettes (prix moyen, temps de préparation, difficulté...)	X	
Visiteurs, membres	Modifier le nombre de personne pour avoir les bonnes quantités	X	
Membres	Avoir des suggestions en fonction des recettes en favori, choix du chef etc...	X	X
Visiteurs, membres	Présence d'un générateur de liste de courses à partir de recettes sélectionnées	X	
Visiteurs, membres	Sécurité sur le site		X
Visiteurs, membres	Fluidité dans la navigation et le chargement des différents médias/outils		X

L'utilisateur étant le destinataire de l'application, je réalise un formulaire d'analyse des besoins ainsi qu'un formulaire concernant l'inclusivité que je poste sur X afin de récolter des retours directs. Dans mon formulaire d'analyse des besoins, je liste plusieurs fonctionnalités possibles que les utilisateurs peuvent noter de 1 à 10, ainsi qu'un champ de suggestions. Les fonctionnalités présentes dans le formulaire sont :

- B : Possibilité de chercher une recette par son nom.
- C : Possibilité de rechercher des recettes à partir d'un ingrédient.

- D : Possibilité de chercher des recettes à partir d'une liste d'ingrédients.
- E : Laisser des commentaires aux recettes.
- F : Mettre des recettes en favoris.
- G : Poster ses propres recettes.
- H : Rechercher un utilisateur pour voir ses recettes personnelles.
- I : Voir les recettes favorites d'un autre utilisateur.
- J : Modifier le nombre de personnes pour avoir les justes proportions sur la page de la recette.
- K : Régler son régime alimentaire (végétarien, végan, sans gluten...).
- L : Caractéristiques de la recette (prix des ingrédients, temps de préparation, difficulté de la préparation...).
- M : Informations sur le taux calorique.
- N : Suggestion selon la popularité de la recette.
- O : Suggestion selon les recettes déjà en favori (recettes contenant des ingrédients similaires).
- P : Suggestion selon une mise en avant par le site (choix du chef, choix de la communauté par vote...).
- Q : Présence d'un générateur de liste de courses selon une liste de recettes pré-sélectionnées.
- R : Certification d'utilisateurs pour mettre en avant leurs recettes.
- S : Vidéos sur les recettes vérifiées pour montrer les différentes étapes de préparation.
- T : Présence d'un espace "Réduction" montrant les meilleures offres du moment dans les magasins proches de l'utilisateur.

À partir de ces questions et des 57 réponses récoltées par le questionnaire, j'ai pu établir un graphique montrant les priorités générales des utilisateurs :



Grâce à ce formulaire, je peux définir des fonctionnalités clés aux yeux des utilisateurs et réaliser une matrice de priorisation des fonctionnalités avec la méthode MoSCoW :

Fonctionnalité	Must	Should	Could	Won't
Rechercher une recette (nom de recette, ingrédient(s))	X			
Modifier le nombre de personnes pour avoir les bonnes quantités sur une recette	X			
Créer un compte utilisateur	X			
Gérer les préférences du compte utilisateur	X			
Fonctionnalité d'analyse de frigo	X			
Gérer le régime alimentaire dans les recherches et sur le compte utilisateur		X		
Fonctionnalité d'affichage/masquage des calories sur les recettes/mention de calories dans la majorité du site		X		
Caractéristiques de la recette (prix moyen, calories (si actif), temps de préparation, difficulté...)		X		
Laisser des commentaires sous les recettes			X	
Ajouter aux favoris/Like une recette		X		
Poster ses propres recettes				X
Rechercher un utilisateur pour voir ses recettes				X
Voir les recettes favorites d'un autre utilisateur				X
Suggestions de recettes (Selon popularité, recettes en favoris, mise en avant par le site)			X	
Présence d'un générateur de liste de course			X	
Certification d'utilisateurs pour mettre en avant leurs recettes				X
Vidéo sur les recettes certifiées pour présenter les étapes de la recette			X	
Présence d'un espace « Réduction » pour les magasins autour de l'utilisateur				X

II. Personae et User Stories

A. Personae

Je crée des personae simulant les utilisateurs cibles de l'application :

Mael 26 ans, Ingénieur



- Cuisine régulièrement mais peu à l'aise
- Recherche des solutions mobiles pour trouver des recettes spontanément selon les ingrédients à sa disposition
- Veut alléger sa charge mentale le soir au moment du repas
- Végétarien

Nathalie 32 ans, Professeure des écoles



- Aime cuisiner
- Est atteinte de troubles de l'alimentation
- A des difficultés à trouver des sites de cuisines adaptés à ses troubles (affichage des calories, etc...)

Valérie 51 ans, Femme au foyer



- Cuisine quotidiennement
- Recherche spontanément des recettes de cuisine sur son ordinateur
- Veut rechercher des recettes correspondant à des ingrédients précis

B. User Stories

À partir des informations collectées jusqu'à maintenant, je peux définir des user stories générales :

En tant que	Je souhaite	Afin de
Visiteur	avoir accès à une fonction de recherche	trouver des recettes correspondants à mes envies
Utilisateur mobile	pouvoir scanner mon réfrigérateur	rechercher des recherches correspondants aux ingrédients à ma disposition pour alléger ma charge mentale
Utilisateur sur ordinateur	pouvoir créer une liste d'ingrédient manuellement	rechercher des recherches correspondants à ces ingrédients
Visiteur	pouvoir générer une recette à partir d'une liste d'ingrédient	trouver une recette correspondant exactement à mes ingrédients
Utilisateur atteint de troubles de l'alimentation	personnaliser mon expérience afin de	cacher ou d'afficher les informations nutritives des recettes
Visiteur	ajuster le nombre de personnes sur la recette	obtenir les proportions correctes peu importe le nombre de personnes à table
Visiteur	avoir des informations claires sur les quantités et les temps de préparation des recettes	mieux m'organiser le soir et alléger ma charge mentale
Visiteur régulier	créer un compte	sauvegarder mes préférences d'affichage quelle que soit la plateforme sur laquelle je me trouve
Végétarien	enregistrer mon régime alimentaire	avoir uniquement des recettes adaptées à mes besoins
Utilisateur connecté	ajouter des recettes à mes favoris	pouvoir accéder facilement à des recettes que j'ai déjà essayé et validé
Utilisateur connecté	noter et commenter les recettes	faire un retour sur la recette et conseiller les autres utilisateurs de l'application
Utilisateur connecté	recevoir des recommandations de recettes adaptées	découvrir de nouveaux plats correspondants à mes goûts
Visiteur	visionner des vidéos des étapes de la recette	mieux comprendre les différentes étapes et réaliser correctement la recette

III. MVP

A. Objectif principal du MVP

Permettre à un utilisateur de s'inscrire, personnaliser ses préférences, rechercher des recettes, modifier le nombre de personnes pour la recette et utiliser l'outil de scan d'ingrédients.

B. Fonctionnalités retenues (Must Have)

Fonctionnalité	Justification
Recherche de recette	Fonction indispensable du site
Inscription d'utilisateur	Fonction majeure pour assurer la personnalisation de l'expérience utilisateur
Scan d'ingrédients/frigo	Fonction majeure et vitrine du site
Modification des préférences utilisateurs	Complète la fonction de création d'utilisateur
Modification du nombre de personne pour une recette	Fonction basique d'un site de cuisine pour simplifier l'expérience utilisateur

C. Fonctionnalités exclues du MVP

- Recherche par ingrédient
- Ajout de favori
- Informations supplémentaires sur les recettes (prix, difficulté)
- Générateur de liste de courses
- Suggestions de recettes
- Vidéo des différentes étapes des recettes

D. Parcours utilisateur MVP

- L'utilisateur crée un compte
- L'utilisateur gère ses préférences (affichage)
- L'utilisateur recherche une recette avec l'outil de recherche/avec l'outil de scan d'ingrédients
- L'utilisateur règle le nombre de personnes pour la recette recherchée.

IV. Contraintes techniques et organisationnelles

A. Techniques

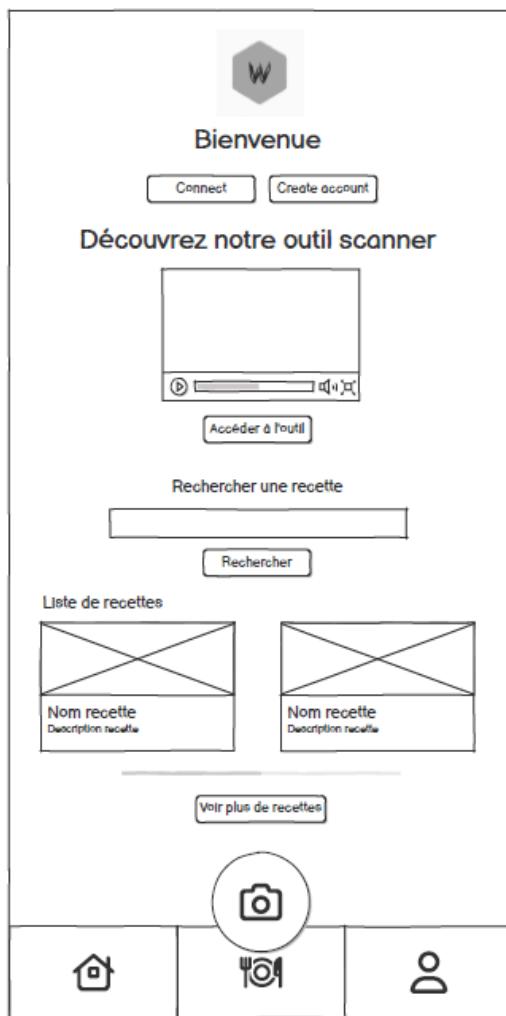
Type de contrainte	Détail	Impact sur le projet
Front-end	React + SASS	Développement rapide, composants UI réutilisables, mobile-first
Back-end	Node.js + express	Création d'API REST, bonne performance pour applications temps réel
Base de données	Psql (relationnelle)	Schéma de données fixe donc plus pratique à manipuler pour ce projet
Intelligence Artificielle	Gemini	Gestion de l'analyse d'image pour l'outil d'analyse de frigo + génération de recette sur demande
Authentification	Login interne au site	Gestion des infos utilisateurs directement dans la base de données
RGPD	Politique de confidentialité	Consentement explicite requis à la création du compte
Accessibilité	Conformité RGAA : contrastes, navigation clavier, alt text	Test régulier, amélioration UX pour les personnes en situation de handicap
Sécurité	HTTPS, validation des entrées, protection XSS/SQLi, rôles utilisateurs	Sécurisation côté client et serveur, logique de gestion des accès
Navigateurs supportés	Chrome, Firefox, Safari, Edge	Nécessite des tests multi-navigateurs pour éviter les incompatibilités
Performance	Chargement optimisé des images, lazy loading	Réduction du temps de chargement, meilleure expérience utilisateur

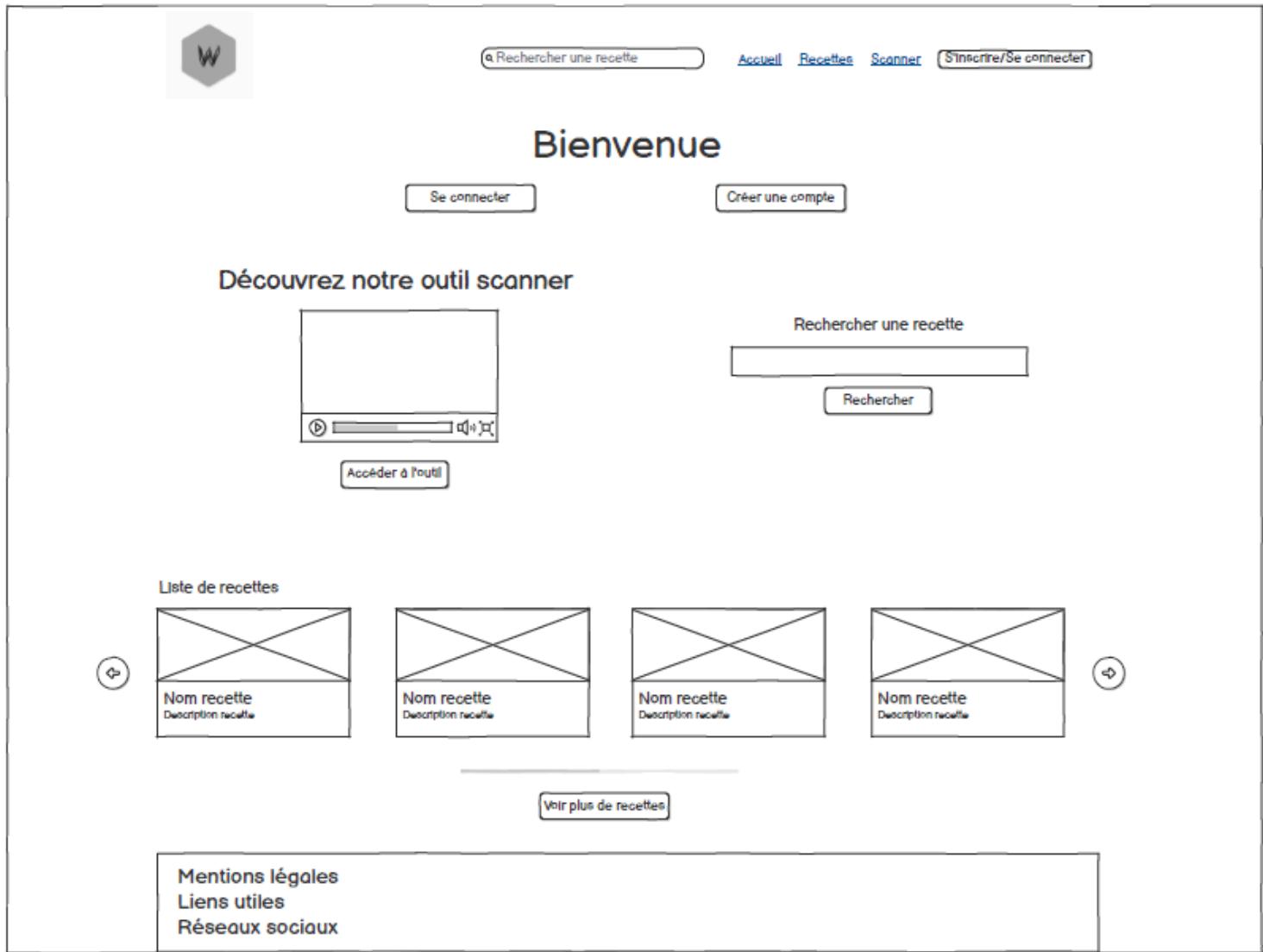
B. Organisationnelles

Pour ce qui est des contraintes organisationnelles, j'avais à terminer le MVP du projet pour au plus tard la date d'examen. Au vu des dates de fin de la formation en présentiel, je me devais de réaliser le plus de tâche possible avant le 23 novembre. J'étais également tenu de réaliser ce dossier projet et un dossier professionnel pour au plus tard le 9 février 2026.

V. Wireframes

Voici un exemple de wireframes pensé pour la page d'accueil, en version mobile et en version ordinateur :





VI. Charte graphique

A. Ton général

Pour la rédaction des textes, instructions de recettes et mails (réinitialisation de mot de passe), j'ai décidé d'utiliser un langage familier avec tutoiement de l'utilisateur. Le ton général du site est assez léger et accueillant, visant à mettre à l'aise tout utilisateur. Cette décision vient du fait que le public cible principal du projet est jeune (20 à 30 ans), sachant que les personnes jeunes sont habitués à ce genre de langage utilisé dans la majorité des contenus qu'ils consomment quotidiennement. Je cherche à renvoyer des valeurs inclusives au travers du projet, notamment en simplifiant l'expérience aux utilisateurs atteints de handicap ou de troubles, ainsi que de la communication autour du projet (que je réalisera moi-même sur les réseaux sociaux). Pour continuer sur la sensation de confort, j'ai souhaité intégrer des formes arrondies pour les différents éléments, car moins agressifs à l'heure que des lignes et angles droits.

B. Logo

Le logo du site a été réalisé par un artiste du nom de Benjamin Valquevis (@benverever sur Instagram), qui a accepté de m'aider dans mon travail. Il y a eu plusieurs versions du logo, voici la première version imaginée :



Le logo final représente un oeuf au plat dont le blanc prend la forme d'un coeur, avec le jaune d'oeuf reprenant les initiales de l'application (WIMF) :



Ce logo ainsi que les couleurs utilisées représentent le ton accueillant et réconfortant que je souhaitais exprimer au travers de l'application.

C. Palette de couleurs et typographie

- ❖ Couleur principale : Orange #FCAE26
- ❖ Couleur de fond : Blanc floral #fff7ed
- ❖ Couleurs des textes :
 - Graphite (pour les textes plus sombres) #303030
 - Charbon de bois (pour les textes plus clairs) #565656

Ces couleurs ont été choisies afin de respecter les normes d'accessibilité. Le orange renvoie un sentiment chaleureux et enthousiaste à l'utilisateur, en accord avec le ton de l'application.

- ❖ Typographies :
 - Titre : Sora
 - Texte : Carlito

Le choix de la typographie Sora pour les titres correspond à l'envie d'avoir des formes arrondies et confortables à l'œil, tandis que la typographie Carlito est également confortable à l'œil et reste parfaitement lisible pour tout utilisateur.

D. Exemple de maquettes

L'entièreté des maquettes a été réalisée sur Figma, outil gratuit et ergonomique pour la création de maquettes graphiques. D'abord, un exemple de maquette moyenne fidélité (contenant l'ancien logo) pour la page d'accueil, en version mobile et ordinateur :

Bienvenue sur WIMF

Trouve des idées recettes avec les ingrédients de ton frigo

Se connecter Crée un compte
Régler les paramètres d'accessibilité

Découvrez notre outil scanner

Accéder à l'outil scanner

Rechercher une recette

Rechercher une recette, un ingrédient, un tag...

Rechercher

Recettes recommandées

Nom recette
Description recette
Tag 1 Tag 2 Tag 3 Tag 4

Nom recette
Description recette
Tag 1 Tag 2 Tag 3

Voir plus de recettes

Accéder à l'outil scanner

Rechercher une recette

Rechercher une recette, un ingrédient, un tag...

Rechercher

Bienvenue sur WIMF

Trouve des idées recettes avec les ingrédients de ton frigo

se connecter Crée un compte
Régler les paramètres d'accessibilité

Découvre notre outil scanner

Il te reste des ingrédients au frigo mais tu ne sais pas quoi cuisiner ? Prends simplement une photo de l'intérieur avec notre scanner : il reconnaît automatiquement les aliments disponibles et te propose une sélection de recettes adaptées.

Aucune ne te convient ? Pas de souci : notre Intelligence artificielle peut t'en créer une sur mesure. Et si la recette te plait, partage-la avec nous pour qu'elle ait une chance de rejoindre notre catalogue !

Accéder à l'outil scanner

Rechercher une recette

Rechercher une recette, un ingrédient, un tag...

Rechercher

Recettes recommandées

Nom recette
Description recette
Tag 1 Tag 2 Tag 3 Tag 4

Nom recette
Description recette
Tag 1 Tag 2 Tag 3 Tag 4

Nom recette
Description recette
Tag 1 Tag 2 Tag 3 Tag 4

Nom recette
Description recette
Tag 1 Tag 2

Voir plus de recettes

Mentions légales Politique de confidentialité Préférences d'accessibilité
2025 WIMF What's In My Fridge - Tous droits réservés

Voici plusieurs exemples de maquettes haute fidélité :

Bienvenue sur WIMF

Trouve des idées recettes avec les ingrédients de ton frigo

Découvre notre outil scanner

Scanner mon frigo

Rechercher une recette

Ex: Poke Bowl

Rechercher

Recettes recommandées

Poke Bowl au Saumon Risotto aux champignons

Voir plus de recettes

Accueil Recettes Scanner Profil

Bienvenue sur WIMF

Trouve des idées recettes avec les ingrédients de ton frigo

Découvre notre outil scanner

Il te reste des ingrédients au frigo mais tu ne sais pas quoi cuisiner ?
Prends simplement une photo de l'intérieur avec notre scanner : il reconnaît automatiquement les aliments disponibles et te propose une sélection de recettes adaptées.

Aucune ne te convient ? Pas de souci : notre intelligence artificielle peut t'en créer une sur mesure. Et si la recette te plait, partage-la avec nous pour qu'elle ait une chance de rejoindre notre catalogue !

Exclusivement disponible sur mobile !

Rechercher une recette

Ex: Poke Bowl

Rechercher

Recettes recommandées

Poke Bowl au Saumon Risotto aux champignons Poke Bowl au Saumon Risotto aux Champignons

Voir plus de recettes

Politique de confidentialité Conditions d'utilisation Mentions légales

© 2025 WIMF - Tous droits réservés



Poke Bowl au saumon



Préparation
10 min



Cuisson
20 min



Repos
10 min

Ingrédients

✓ Pour 5 personnes

Saumon	650 g
Avocat	1
Gingembre en poudre	0.5 càc
Sauce soja	80 g
Huile de sésame	20 g
Sucre en poudre	5 càs
Poudre de wasabi (facultatif)	1 pincée
Graine de sésame grillée	1 poignée
Riz à sushi	500 g
Vinaigre de riz	5 càs
Sel	1 càc

Préparation

- Fais cuire le riz à sushi selon les indications du paquet. Pendant ce temps, prépare l'assaisonnement du riz en mélangeant le vinaigre de riz, trois cuillères à soupe de sucre en poudre et une pincée de sel.
- Lorsque le riz est cuit, laisse-le reposer 10 minutes, puis verse l'assaisonnement par-dessus. Mélange délicatement, puis laisse tiédir ou refroidir.
- Coupe le saumon et l'avocat en petits cubes de 0,5 à 1 cm et dépose-les dans un saladier. Prépare la sauce en mélangeant le gingembre, la sauce soja, l'huile de sésame, deux cuillères à soupe de sucre et le wasabi.
- Verse cette sauce sur le poisson et mélange bien. Réserve au frais jusqu'au moment de servir.
- Remplis un bol de riz, ajoute le mélange de saumon et d'avocat par-dessus, puis parsème de graines de sésame avant de servir.

Valeurs nutritionnelles



Valeurs nutritionnelles



Calories	657 kcal
Matière grasses totales	35.3 g
Acides gras saturés	3.8 g
Cholestérol	0 mg
Sodium	1190 mg
Glucides totaux	47.9 g
Fibres alimentaires	4.2 g
Sucre totaux	10.6 g
Protéines	35.0 g
Calcium	0 mg
Fer	0 mg
Potassium	0 mg

Poke Bowl au saumon



Temps de préparation



Préparation
10 min



Cuisson
20 min



Repos
10 min

Ingrédients

✓ Pour 5 personnes

Saumon	650 g	Avocat	1
Gingembre en poudre	0.5 càc	Sauce soja	80 g
Huile de sésame	20 g	Sucre en poudre	5 càs
Poudre de wasabi (facultatif)	1 pincée	Graine de sésame grillée	1 poignée
Riz à sushi	500 g	Vinaigre de riz	5 càs
Sel	1 càc		

Préparation

- Fais cuire le riz à sushi selon les indications du paquet. Pendant ce temps, prépare l'assaisonnement du riz en mélangeant le vinaigre de riz, trois cuillères à soupe de sucre en poudre et une pincée de sel.
- Lorsque le riz est cuit, laisse-le reposer 10 minutes, puis verse l'assaisonnement par-dessus. Mélange délicatement, puis laisse tiédir ou refroidir.
- Coupe le saumon et l'avocat en petits cubes de 0,5 à 1 cm et dépose-les dans un saladier. Prépare la sauce en mélangeant le gingembre, la sauce soja, l'huile de sésame, deux cuillères à soupe de sucre et le wasabi.
- Verse cette sauce sur le poisson et mélange bien. Réserve au frais jusqu'au moment de servir.
- Remplis un bol de riz, ajoute le mélange de saumon et d'avocat par-dessus, puis parsème de graines de sésame avant de servir.

Valeurs nutritionnelles (1 portion)

Calories	657 kcal
Matière grasses totales	35.3 g
Acides gras saturés	3.8 g
Cholestérol	0 mg
Sodium	1190 mg
Glucides totaux	47.9 g
Fibres alimentaires	4.2 g
Sucre totaux	10.6 g
Protéines	35.0 g
Calcium	0 mg
Fer	0 mg
Potassium	0 mg

Politique de confidentialité

Conditions d'utilisation

Mentions légales

© 2025 WIMF - Tous droits réservés



Accueil



Recettes



Scanner



Profil

Mes informations

TestUser
testuser@mail.com

Nom d'utilisateur
TestUser

Adresse email
testuser@mail.com

Mot de passe

Confirmer le mot de passe
NouveauMotDePasse

Mettre à jour mes informations

Déconnexion

Mes informations **Préférences** **Informations légales**

Accueil Recettes Scanner Profil

Mes informations

Rechercher une recette

Accueil Recettes

Nom d'utilisateur
TestUser

Adresse email
testuser@mail.com

Mot de passe

Confirmer le mot de passe
NouveauMotDePasse

Mettre à jour mes informations

Déconnexion

Politique de confidentialité Conditions d'utilisation Mentions légales

© 2025 WIMF - Tout droits réservés

De légères modifications sont à attendre lors du développement afin d'obtenir un résultat optimal et cohérent avec les différents appareils et les réglementations, ou simplement des changements de décisions au cours du projet.

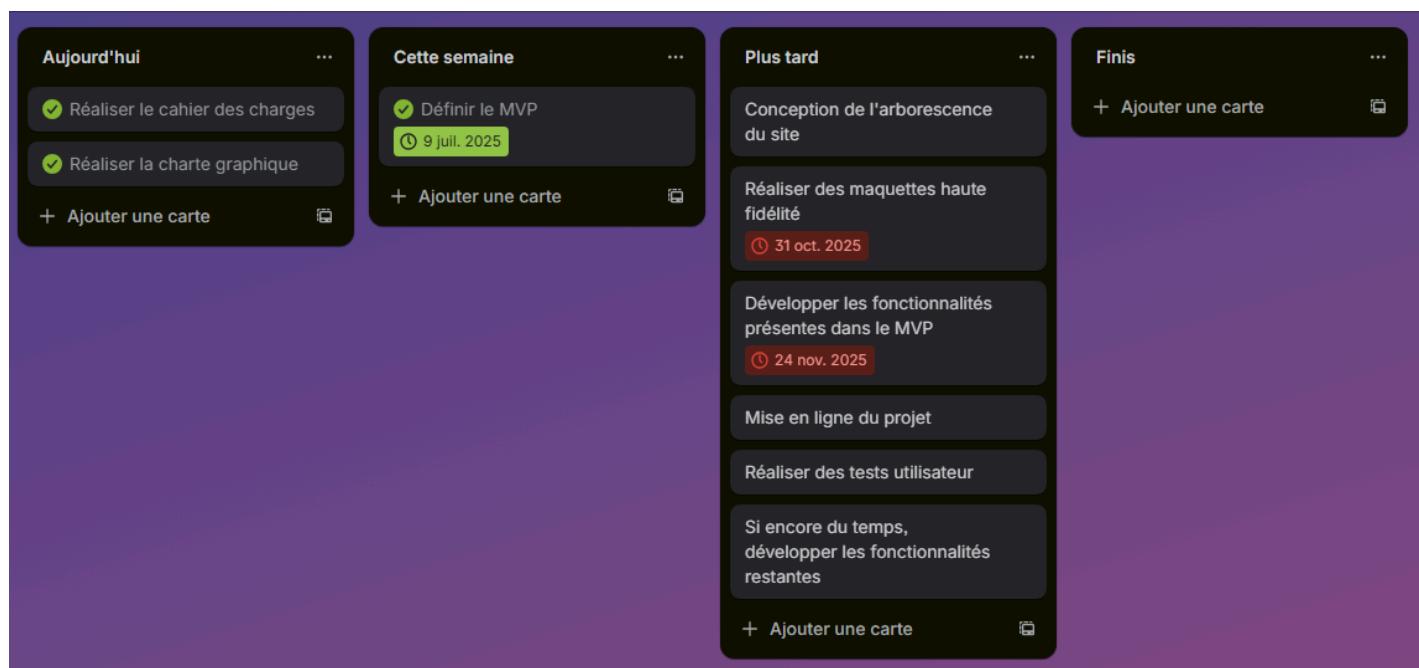
Gestion de projet

I. Méthodologie

Pour réaliser mon projet, j'ai décidé de suivre la méthode Agile. Chaque semaine, je fixais plusieurs objectifs (par exemple : réaliser les maquettes de la page d'accueil et de la page de recette, développer la page d'accueil), puis chaque jour je visais à réaliser différentes parties de ces objectifs, avec chaque soir un compte rendu de ma journée pour pouvoir mieux suivre si j'étais en avance, en adéquation ou en retard avec mes objectifs de la semaine. Grâce à cette méthode, j'ai pu avancer sans me désorganiser entre plusieurs tâches, ce qui aurait ainsi pu retarder l'évolution des différentes fonctionnalités et du projet dans sa globalité. J'ai également pu compter sur mes formateurs qui me dirigeaient lorsque je prenais du retard ou bien quand je partais dans une mauvaise direction. Étant mon premier projet, leur soutien et leur aide m'ont été très précieux pour mener à bien ce projet.

II. Planning et outils utilisés

Pour m'organiser au mieux, j'ai, au début du projet, mis en place un planning global sur Trello classant les différentes phases générales du projet, avec des dates butoirs sur les plus importantes. J'ai créé plusieurs colonnes, me permettant de visualiser clairement les objectifs à court et long terme. Voici un exemple d'à quoi pouvait ressembler le Trello tôt dans la conception du projet :



Lors de la phase de développement du projet, j'ai utilisé un second tableau Trello suivant la méthode Kanban. J'ai ainsi créé trois colonnes : To-Do, Doing, Done. J'ai ajouté toutes mes tâches, MVP et hors MVP, accompagnées de différents tags (front, back, conception, base de données, documentation, tests) me permettant ainsi de me retrouver au mieux quotidiennement. Pour définir mes objectifs quotidiens, j'ouvrais mon tableau, je regardais quelles tâches étaient présentes dans la colonne Doing, et ainsi je savais précisément où j'en étais dans mon travail. La colonne Doing était limitée à trois tâches simultanément au maximum, afin de contrôler correctement le flux de travail et de ne pas me perdre entre trop de tâches différentes. Voici un exemple du tableau lors de la phase de tests liée au front (page suivante) :

WIMF 2		
To-DO	Doing	Done
<p>Front</p> <p>Intégration de la page de préférences diètes/ingrédients bannis</p> <p>Back Base de données</p> <p>Population de la base de données</p> <p>4/6</p>	<p>Front Tests</p> <p>Tests de performance</p> <p>Front Tests</p> <p>Tests unitaires</p> <p>Front Tests</p> <p>Test de compatibilité</p> <p>+ Ajouter une carte</p>	<p>Front</p> <p>✓ Optimisation des performances du site</p> <p>Conception</p> <p>✓ Révision de la charte graphique</p> <p>Front</p> <p>✓ Optimisation de l'accessibilité du site</p> <p>Front</p> <p>✓ Intégration de la page de résultat de recherche</p> <p>Front</p> <p>✓ Développement des composants UI</p> <p>Front</p> <p>✓ Intégration du design responsive desktop</p> <p>Front</p> <p>✓ Intégration de la page de mise à jour d'informations utilisateurs</p> <p>Back</p> <p>Transfert de données</p> <p>+ Ajouter une carte</p>

J'utilisais un autre outil lors de mon travail quotidien, Clockify, me permettant de chronométrier chacune de mes tâches dans la journée. Grâce à cet outil, je pouvais avoir une trace écrite de mon travail, et je pouvais également me rendre compte de la charge de travail que représente chacune des tâches. À chaque fin de journée, je pouvais

générer un rapport de ma journée, m'indiquant mes heures de travail ainsi que le temps passé sur chaque tâche. Par exemple, voici le résumé de ma journée de travail du 4 novembre 2025 :

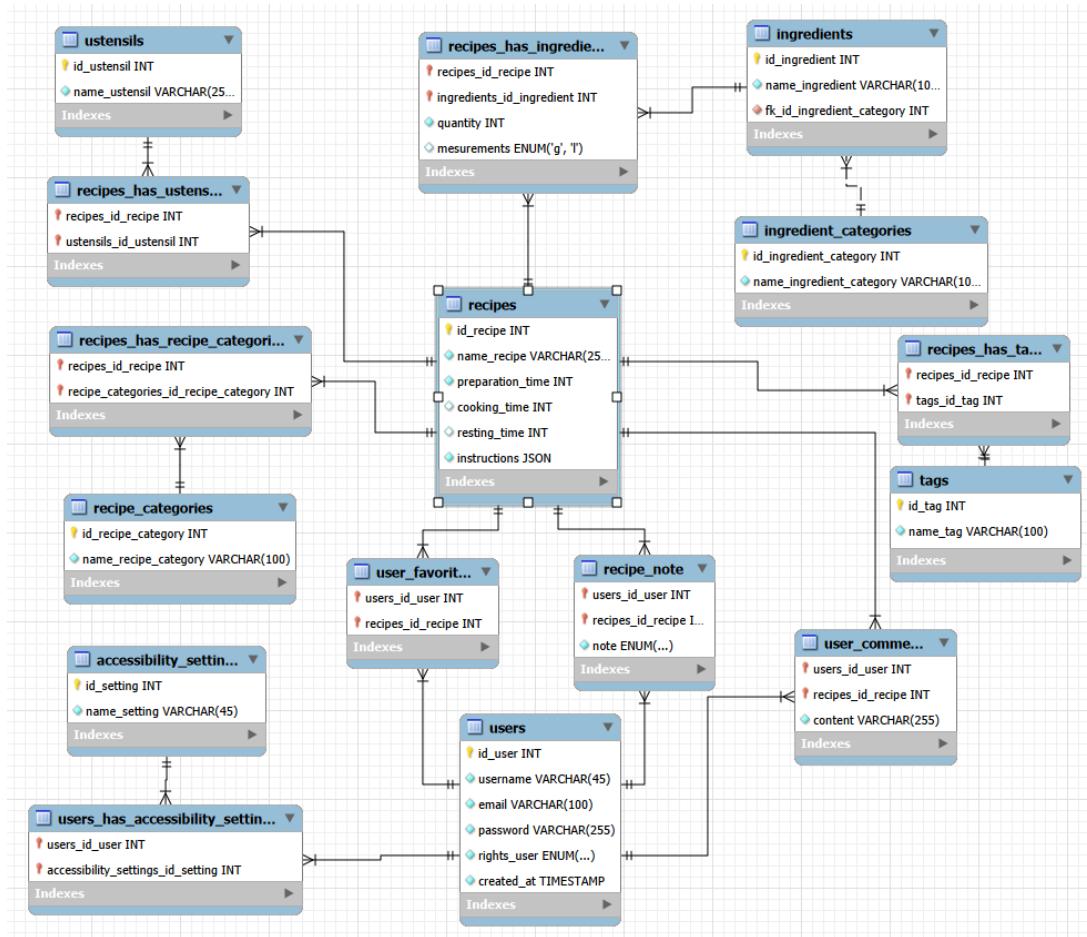


Analyse et conception

I. Base de données

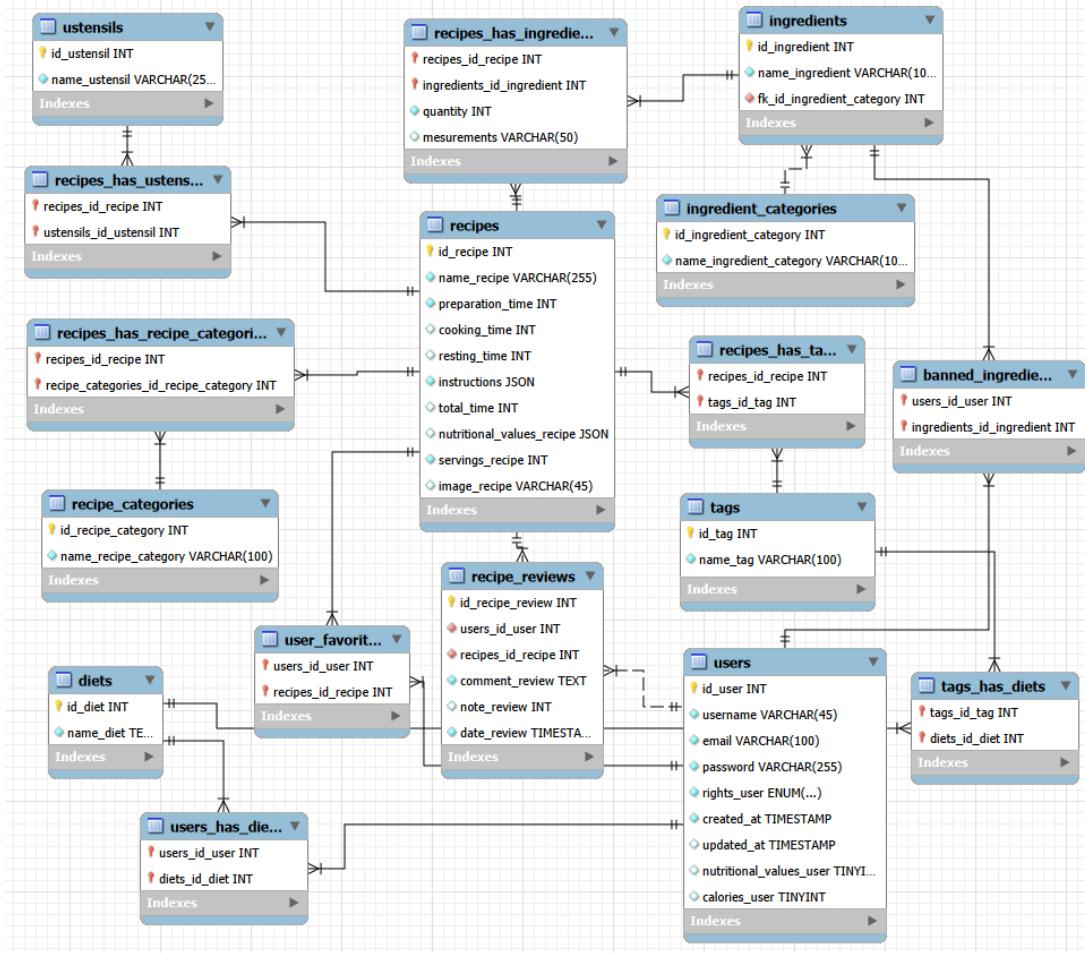
Pour ma base de données, j'ai décidé de travailler en PostgreSQL. Le PSQL me permet de facilement gérer les relations entre les différentes tables grâce aux clés étrangères et aux tables de jointure. Pour un projet comme le mien, cela me paraissait un choix cohérent, surtout au vu de la complexité de la base de données. J'aurais pu la réaliser en NoSQL, mais j'aurais probablement beaucoup alourdi mon back avec des requêtes multiples pour arriver au même résultat qu'une seule requête SQL.

Lors de la conception de ma base de données, j'ai pris la décision de modéliser directement les tables hors MVP. Ainsi, je m'assure de limiter au maximum les difficultés et modifications lors de l'évolution du projet, afin d'éviter les problèmes de transfert de données en cas de modification majeure de la base de données. J'ai d'abord réalisé une version initiale de la base de données sur MySQL Workbench :



Au cours de l'avancée du projet, je suis venu à modifier à plusieurs reprises ma base de données afin de mieux gérer les requêtes et le traitement des données. J'ai supprimé la table concernant l'accessibilité pour que ces informations soient directement liées à l'utilisateur. J'ai également rassemblé les tables de notes et de commentaires de recettes en une seule table "review". J'ai mis à jour la table de recette pour y ajouter les images

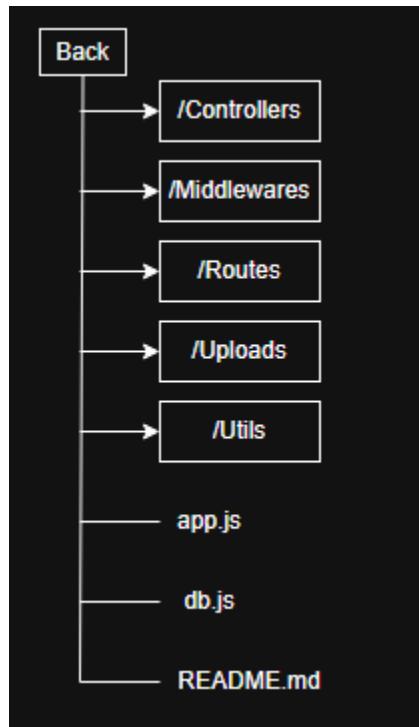
des recettes et également les valeurs nutritionnelles ainsi que la colonne "servings" qui me permettra de calculer dynamiquement les quantités selon le nombre de services prévus par l'utilisateur. Enfin j'ai ajouté les tables concernant les préférences alimentaires afin de pousser la personnalisation de l'expérience utilisateur (hors MVP). Voici donc la version finale de la base de données :



Pour ce qui est de l'hébergement de ma base de données, j'ai décidé d'utiliser Neon. Neon permet d'héberger des bases de données en PSQL gratuitement et de façon performante, tout en ayant une interface me permettant de bien suivre l'état de chacune de mes tables, et donc de réaliser des maintenances manuelles en cas de besoin.

II. Back-end

Je décide de réaliser mon back-end avec Node.js et Express. Express me permet de gérer efficacement chacune des requêtes envoyées à ma base de données grâce aux routes et aux controllers, ainsi que la création de middlewares pour la gestion d'autorisations. Voici l'arborescence de mon back-end :



Présentation rapide de chaque dossier/fichier :

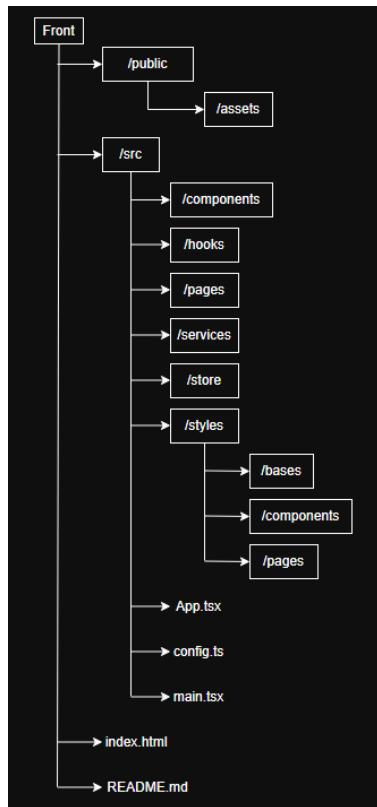
- Controllers : Contient toutes les fonctions qui seront utilisées pour établir des interactions entre le front-end et le back-end
- Middlewares : Contient les fonctions de vérification (ex : vérification du niveau de droit utilisateur, vérification des tokens de connexion ou de réinitialisation de mails)
- Routes : Contient les fichiers de définition de routes pour la connexion au front-end et à quelle fonction correspond chaque route
- Uploads : Contient les différentes images (ex : photo de recette)
- Utils : Contient les fonctions plus “complexes” à gérer à part (ex : fonction de mailing pour le mot de passe oublié)
- app.js : Contient la configuration du back-end (ex : routes globales, cors, sécurités)
- db.js : Contient la connexion à la base de données
- README.md : Contient la documentation générale du back-end

Grâce à cette arborescence, je peux efficacement maintenir, mettre à jour ou ajouter des fonctionnalités à mon back-end. La séparation de chaque élément permet de mieux m'organiser dans mon travail et de plus facilement gérer la connexion au front-end.

Pour l'hébergement de mon back-end, je décide d'utiliser Render. Render est gratuit et adapté à l'hébergement de back-end de façon performante. Il permet également de paramétrier facilement les variables d'environnement, ce qui sera utile lors du déploiement du back-end. Un défaut de Render est qu'un projet non utilisé régulièrement sera mis “en veille” et donc mettra du temps à répondre à une nouvelle requête du front. Pour pallier ce problème, j'utiliserai UptimeRobot qui enverra des pings réguliers à mon back-end pour le maintenir actif.

III. Front-end

Le front-end est réalisé à l'aide de React + TS et de SASS pour les styles. React est parfaitement adapté à une application mobile-first, et permet de gagner grandement en performance. En effet, en utilisant React, lors d'un changement de page sur l'application, les éléments récurrents ne seront pas rechargés, ce qui permet de gagner grandement en performance. React permet de travailler avec des composants réutilisables, ce qui permet d'économiser du code inutile ainsi qu'une meilleure maintenabilité du site. Je choisis également de travailler avec TypeScript étant donné qu'il est nécessaire de typer chaque variable. C'est plus sûr que JavaScript car cela évite les erreurs lors de la manipulation des données, je sais exactement avec quoi je travaille. Pour le style, l'utilisation de SASS me permettra de créer des variables globales au projet, que ce soit pour les couleurs ou bien les polices d'écritures utilisées, des mixins pour les styles récurrents ou des animations. Ainsi, j'aurai une bien meilleure maintenabilité en cas de mise à jour de la charte graphique plus tard. Voici l'arborescence du front-end du projet :



Une nouvelle fois, présentation rapide des différents dossiers et fichiers :

- public : contient les images nécessaires à l'affichage du site (ex : logo, illustration générale)
 - assets : contient les images utilisées pour les styles (ex : SVG pour le fond du site)
- src : contient le corps du site
 - components : contient les composants réutilisables sur chaque page (ex : header, footer, navbar) et tous les composants des différentes pages

- hooks : contient les “hooks” personnalisés, réutilisables sur chaque page afin d’éviter de recopier du code utilisé régulièrement (ex : isMobile pour vérifier la taille d’écran)
 - pages : contient les pages du site (ex : home, recipe)
 - services : contient toutes les fonctions de connexion au back-end, regroupé en un seul fichier pour une meilleure maintenabilité
 - store : comprend la logique métier derrière la connexion utilisateur
 - styles : contient les styles du site
 - bases : contient les styles réutilisables du site (ex : variables, animations, global)
 - components : contient les styles de chaque composant
 - pages : contient les styles de chaque page
 - App.tsx : gère le routing du projet et la navigation entre les pages du site
 - config.ts : permet d’appeler les différentes variables d’environnement dans le projet
 - main.tsx : contient la configuration du routing du projet
- index.html : permet l’affichage du site
 - README.md : contient les informations et documentation du projet

Cette arborescence me permet une bonne maintenabilité dans mon projet et me permet de facilement ajouter ou mettre à jour du contenu.

Pour l’hébergement du front-end, j’utilise Vercel car excessivement simple d’utilisation, il me suffit d’y lier mon repository Github et d’y insérer les variables d’environnement, en plus de fournir un retour clair sur les différentes erreurs du projet empêchant le déploiement du site.

IV. Frameworks utilisés

Je vais lister les différents frameworks utilisés ainsi que pourquoi j’ai décidé de les utiliser. Pour le front-end :

- Zustand : Zustand me permet de gérer la connexion utilisateur en accédant aux stockages en variables de session ou en localStorage. Grâce à Zustand, je peux stocker un token de connexion afin de vérifier constamment si l’utilisateur est connecté et donc fournir une expérience différente selon un visiteur ou un utilisateur connecté
- Lucide-react : Lucide fournit une importante bibliothèque d’icône gratuitement, je me sers de la version React car elle est plus adaptée à mon projet.
- Swiper : Swiper permet de créer facilement des carrousels performants. Je l’utilise pour mon espace “découverte de recettes” sur ma page d’accueil. Swiper est performant et très personnalisable, ce qui correspondait à mes besoins.
- React-webcam : React webcam est un framework permettant d’accéder à la caméra de l’utilisateur, sur mobile comme sur ordinateur. J’avais nécessairement besoin d’accéder à la caméra de l’utilisateur pour l’outil de scanner, j’ai donc trouvé ce framework que j’ai configuré et intégré à mon projet.

Pour le back-end :

- Bcrypt : Bcrypt est primordial pour le cryptage et le stockage du mot de passe utilisateur dans la base de données. Il permet d'encrypter le mot de passe utilisateur et le rendre impossible à utiliser en cas de fuite de données.
- Cors : Nécessaire pour la sécurité du back-end, il me permet de filtrer les accès et donc de n'autoriser que les requêtes provenant de mon front-end et non d'un autre site externe.
- Dotenv : Permet de créer des variables d'environnement réutilisables dans mon projet.
- Helmet : Utile pour sécuriser les requêtes Express et ajoute donc une nouvelle couche de sécurité à mon back-end.
- Jsonwebtoken : Permet la création de tokens à partir d'une clé (stockée en variable d'environnement). Il est très utile pour l'authentification utilisateur ou bien pour la réinitialisation du mot de passe par exemple.
- Nodemailer : Utilisé pour gérer l'envoi de mails directement depuis le back, dans mon cas pour le mail de réinitialisation de mot de passe.

Développement

Scanner d'ingrédients

Dans ce projet, la fonctionnalité vitrine est le scanner d'ingrédients. J'ai, pour des raisons logiques, pris la décision de le rendre accessible uniquement sur mobile, car aucun utilisateur ne va utiliser la webcam de son ordinateur pour utiliser l'outil. J'avais donc besoin d'accéder à la caméra de l'utilisateur ainsi qu'à ses fonctionnalités pour le bon fonctionnement de l'outil. Comme mentionné plus tôt, j'ai décidé d'utiliser le framework react-webcam car il est simple à configurer et performant pour l'utilisation que j'en ai.

A. Back-end

Je commence d'abord par réaliser les API nécessaires au bon fonctionnement de la fonctionnalité. J'ai donc besoin de deux paires controller/route : une pour les ingrédients, et une pour l'intelligence artificielle qui analysera l'image. Je commence par réaliser la paire controller/route pour l'IA.

Pour le controller, je commence par initialiser Google Gemini qui me servira ici à l'analyse de l'image envoyée par l'utilisateur. Je définis deux sécurités initiales : une taille de fichier limite pour l'image et une liste d'"ingrédients" interdits pour éviter les faux retours. Je limite la taille des fichiers à 4 Mo pour optimiser les performances, et j'ajoute à ma liste d'ingrédients interdits des mots tels que "emballages", "pack", "pot" ou "bocal" afin d'avoir une liste plus simple à traiter après analyse.

Je crée une fonction "getIngredientsFromPicture" que j'exporte directement, afin de pouvoir l'appeler sur ma route. J'englobe l'intégralité du contenu de ma fonction dans un "try/catch" pour gérer les erreurs serveur. J'initialise ensuite mon modèle IA grâce à la clé API Gemini générée sur le site de Gemini directement, et je récupère la photo envoyée par l'utilisateur dans le body de la requête.

```
1 const { GoogleGenAI } = require(id: "@google/genai");
2
3 const MAX_IMAGE_SIZE: number = 4 * 1024 * 1024; // 4 Mo
4
5 const FORBIDDEN_INGREDIENTS: string[] = [
6   "récipient",
7   "emballage",
8   "bouteille",
9   "boite",
10  "boîte",
11  "pot",
12  "bocal",
13  "étiquette",
14  "étagère",
15  "sachet",
16  "film",
17  "carton",
18  "pack",
19  "canette",
20  "couvercle",
21  "plateau",
22  "tiroir",
23  "tupperware",
24  "main",
25  "doigt",
26  "doigts",
27  "main humaine",
28  "ombre",
29  "fond",
30  "table",
31];
```

Je réalise ensuite plusieurs vérifications et manipulations concernant l'image pour éviter des appels inutiles à l'API Gemini, ayant un nombre limité de token sur une période déterminée. Dans l'ordre, je commence par valider la présence d'une image au format texte, car notre image doit être traitée au format Base64, pour de meilleures performances lors de l'envoi de l'image par l'utilisateur. Je "nettoie" ensuite la chaîne de caractères contenant l'image en Base64 pour ne garder que l'image. Je vérifie ensuite que la chaîne de caractère reçue correspond bien à une image au bon format, et enfin je vérifie que l'image correspond bien aux limites de taille de fichier fixées plus tôt.

```
38 // Validation de base
39 if (!picture || typeof picture !== "string") {
40   return res.status(400).json({ error: "Aucune image valide fournie." });
41 }
42
43 const cleanedBase64: string = picture.replace(searchValue: /data:image\/\w+;base64,/ , replaceValue: "");
44
45 // Vérification base64
46 const isBase64: boolean = /^[A-Za-z0-9+/]{0,2}$.test(cleanedBase64);
47 if (!isBase64) {
48   return res.status(400).json({ error: "Format Base64 invalide." });
49 }
50
51 // Taille limite
52 const imageBuffer: Buffer<ArrayBuffer> = Buffer.from(cleanedBase64, encoding: "base64");
53 if (imageBuffer.length > MAX_IMAGE_SIZE) {
54   return res.status(413).json({ error: "Image trop lourde (4Mo max)." });
55 }
```

Je prépare ensuite le contenu de ma requête à Gemini, en incluant l'image fournie par l'utilisateur, ainsi qu'un prompt indiquant l'IA quoi faire. Voici le prompt :

"Tu es un assistant qui liste uniquement les ingrédients visibles ou mentionnés, en français.

- Ne fais jamais de traduction vers l'anglais.
- Ne crée pas de mots inventés, incomplets ou des variantes inutiles.
- Répond uniquement avec un JSON strictement valide.
- La clé doit être "ingredients" et la valeur un tableau de chaînes de caractères.
- Ne mets que les ingrédients réels, exactement comme ils apparaissent, sans commentaires, sans clés supplémentaires.

Exemple de sortie correcte :

```
{
  "ingredients": ["crème fraîche", "carottes", "lentilles", "bière"]
}
```

J'envoie ensuite la requête à Gemini (j'utilise ici Gemini 2.5 flash lite) en lui demandant une réponse précise à une température de 0.1. Cela m'assure de limiter au maximum les erreurs. Je récupère ensuite la réponse de Gemini, et vérifie qu'elle existe bien, et si elle est au bon format (soit format text).

Une fois la réponse récupérée, je commence à la traiter et la re-vérifier avant de la renvoyer à l'utilisateur. D'abord, comme j'ai demandé à Gemini de renvoyer un json, je nettoie les potentiels éléments de mise en forme ajoutés par Gemini. Je vérifie ensuite que la réponse est bien un JSON valide, dans le cas contraire je renvoie une erreur et la réponse IA en brut. Je vérifie une seconde fois que le format du json correspond à la demande (une clé "ingredient" et une valeur en forme de tableau de texte), dans le cas contraire je renvoie une erreur avec le JSON brut. Ensuite, je normalise le résultat en mettant tous les ingrédients en minuscule, au singulier et sans caractères spéciaux et je génère un nouveau tableau avec les ingrédients normalisés. Je filtre ensuite les ingrédients interdits pour éliminer tout retour inutile, puis vérifie simplement les ingrédients dupliqués. Enfin je renvoie la liste "propre" au front.

```

95 // Parsing JSON
96 const cleanJsonText: string = resultText
97 .replace(searchValue: /```json/i, replaceValue: "")
98 .replace(searchValue: /```/g, replaceValue: "")
99 .trim();
100
101 let ingredientsJson;
102 try {
103   ingredientsJson = JSON.parse(cleanJsonText);
104 } catch (err) {
105   return res.status(500).json({
106     error: "La réponse IA n'est pas un JSON valide.",
107     raw: resultText,
108   });
109 }
110
111 if (
112   !ingredientsJson.ingredients ||
113   !Array.isArray(ingredientsJson.ingredients)
114 ) {
115   return res.status(500).json({
116     error: "Format JSON incorrect.",
117     raw: ingredientsJson,
118   });
119 }
120
121 // Normalisation
122 const normalize: (str: any) => any = (str: any) : any =>
123   str.toLowerCase().trim().replace(/[()]/g, "").replace(/\s+/g, " ");
124
125 let ingredients: any = ingredientsJson.ingredients.map(normalize);
126
127 // Filtrage des ingrédients interdits
128 ingredients = ingredients.filter(
129   (ing: any) : any => ing && !FORBIDDEN_INGREDIENTS.includes(ing) && ing.length > 1
130 );
131
132 // Remove duplicates
133 ingredients = [...new Set(ingredients)];

```

Une fois mon controller réalisé, j'ajoute la route pour pouvoir appeler cette fonction depuis le front. Je crée un fichier aiRoutes.js qui contiendra toutes les routes faisant appel à l'IA dans le projet. J'ajoute une route en POST, car l'utilisateur envoie une information au back (ici une photo), et j'y attribue la route "/" car c'est la route principale des appels IA. J'y attribue ma fonction de mon controller et j'exporte les routes. Enfin, dans mon fichier app.js, j'ajoute une route "/api/v1/ai" et j'y attribue toutes mes routes IA. Ma route est désormais fonctionnelle.

```

1  const express = require(id: 'express')
2  const router = express.Router()
3  const aiControllers = require(id: '../controllers/aiControllers')
4
5  router.post(path: '/', aiControllers.getIngredientsFromPicture)
6
7  module.exports = router

```



```

45  const aiRoutes : Router = require(id: "./routes/aiRoutes");

```



```

67  // Analyse IA d'image
68  app.use(path: "/api/v1/ai", aiRoutes);

```

Je crée ensuite un controller pour récupérer les informations d'une liste d'ingrédients grâce à leur nom. Cette fonction permettra notamment de récupérer les IDs des ingrédients pour plus tard. Pour cela, j'établis une route

en GET pour permettre de rechercher un ou plusieurs ingrédients. Ensuite, le fonctionnement du controller sera simple : récupérer une liste de noms d'ingrédients séparés par un “+”, transformer cette liste en un tableau en filtrant les éléments vides (par exemple deux “+” à la suite), créer une chaîne de caractères correspondant aux conditions utilisées dans la requête SQL, et enfin envoyer la requête à la base de données pour récupérer tous les ingrédients ayant au moins partiellement le nom de l'ingrédient (ex : rechercher “moutarde” renverra “moutarde”, “moutarde de Dijon” et “moutard à l'ancienne”). Cette logique permet de faire des recherches générales et ensuite de filtrer manuellement.

```
35 exports.searchIngredients = async (req, res) => {
36   try {
37     const { search } = req.params;
38
39     // Coupe à chaque "+" dans les paramètres, puis assure qu'il n'y a aucun espace autour de chaque mot, et ne compte pas les mots "vides" (par exemple
40     // si la recherche est "moutarde+dijon", supprime le contenu entre les deux '+')
41     const splitSearch: any = search
42       .split("+")
43       .map((word: any): any => word.trim())
44       .filter(Boolean);
45
46     const conditions: any = splitSearch
47       .map((_ : any, idx: any): string => `i.name_ingredient ILIKE ${idx + 1}`)
48       .join(" OR ");
49
50     const values: any = splitSearch.map((word: any): string => `%${word}%`);
51
52     const searchResult: any[] = await db.any(
53       query: `SELECT i.id_ingredient, i.name_ingredient, c.name_ingredient_category
54       FROM ingredients AS i
55       LEFT JOIN ingredient_categories AS c
56       ON i.fk_id_ingredient_category = c.id_ingredient_category
57       WHERE ${conditions}`,
58       values
59     );
60
61     res.status(200).json(searchResult);
62   } catch (err) {
63     return res.status(500).json({ message: err.message });
64   }
65 };
66
67 router.get(path: "/search/:search", ingredientControllers.searchIngredients); // Rechercher un(des ingrédient(s) par le nom
```

37 const ingredientRoutes: Router = require(id: "./routes/ingredientRoutes");

51 // Ingrédients
52 app.use(path: "/api/v1/ingredients", ingredientRoutes);

Enfin, le dernier controller qui servira dans la fonctionnalité de scanner est le controller pour rechercher les recettes. J'utilise la même logique de récupération d'information que dans le controller d'ingrédient, à l'exception de l'utilisation d'ID cette fois plutôt que de noms d'ingrédients. À partir de cette liste d'ID, je réalise un tableau contenant les IDs, en vérifiant bien qu'il s'agit bien de nombres. Je réalise ensuite une recherche dans ma base de données grâce à la table de jointure en regroupant les éléments par l'ID de recette et en classant par nombre d'ingrédients présents dans la recette. Ainsi, je classe d'avance mes données correctement pour n'avoir plus qu'à les afficher à l'utilisateur.

```

314  exports.searchRecipesByIngredients = async (req, res) => {
315    try {
316      const { ids } = req.params;
317
318      const ingredientIds : any = ids
319        .split("+")
320        .map((id : any) : number => parseInt(id.trim()))
321        .filter((id : any) : boolean => !isNaN(id));
322
323      if (ingredientIds.length === 0) {
324        return res.status(400).json({ message: "Aucun ID d'ingrédient valide fourni" });
325      }
326
327      const recipes : any[] = await db.any(
328        query: `SELECT r.*, COUNT(rhi.fk_id_ingredient) as match_count
329        FROM recipes r
330        INNER JOIN recipes_has_ingredients rhi ON r.id_recipe = rhi.fk_id_recipe
331        WHERE rhi.fk_id_ingredient = ANY($1)
332        GROUP BY r.id_recipe
333        ORDER BY match_count DESC`,
334        [ingredientIds]
335      );
336
337      res.status(200).json(recipes);
338    } catch (err) {
339      res.status(500).json({ message: err.message });
340    }
341  };

```

```

12  router.get(path: "/search/ingredients/:ids", recipeControllers.searchRecipesByIngredients); // Rechercher des recettes par IDs d'ingrédients
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47  // Recettes
48  app.use(path: "/api/v1/recipes", recipeRoutes);

```

B. Front-end

La partie back-end réalisée, je peux maintenant réaliser les fonctionnalités front-end du scanner. Je crée donc un fichier Scanner.tsx dans mon dossier src/pages qui me permettra de gérer si l'utilisateur souhaite scanner avec son appareil photo, ou bien simplement ajouter manuellement les ingrédients à sa liste. Afin que tout soit géré sur une seule et même page (sans changement d'URL), je crée plusieurs variables :

- showCamera : Permet d'afficher ou non la caméra de l'utilisateur pour prendre une photo
- showIngredients : Permet d'afficher la gestion de liste d'ingrédients pour que l'utilisateur puisse la manipuler
- isScanned : Permet de savoir si la liste d'ingrédients provient d'un scan ou bien d'une entrée purement manuelle
- ingredients : Contient la liste des ingrédients scannés ou ajoutés par l'utilisateur
- scanError : Permet de renvoyer une erreur à l'utilisateur en cas d'échec du scan (erreur provenant du back-end)
- search : Permet d'effectuer la recherche de recettes à partir de la liste d'ingrédients
- isMobile : Permet de gérer l'affichage mobile ou ordinateur

Toutes ces variables, à l'exception de isMobile, sont gérées dynamiquement par un useState, qui est un hook React permettant de lier une fonction à une variable pour la manipuler dynamiquement et permettant ainsi de manipuler le DOM à partir de ces variables.

Maintenant que ces éléments sont définis, je peux créer mon composant CameraScanner.tsx, pour gérer la capture de photo ainsi que l'envoi de la photo au back-end pour être traitée par le controller créé précédemment.

Avant de commencer à développer, j'importe tous les éléments dont je vais avoir besoin, soit : une icône "X" pour fermer la caméra, les hooks useCallback, useRef et useState de React, l'élément "Webcam" du framework react-webcam, la méthode "scanImage" pour l'envoi de l'image au back-end, ainsi que la feuille de style du composant. Je définis également les contraintes pour la caméra utilisateur (uniquement la caméra arrière du téléphone, et uniquement en 720p). Enfin, je définis une interface pour les propriétés de l'élément CameraScanner, c'est-à-dire les différentes fonctions créées plus tôt pour gérer l'affichage de la page.

```

1 import { X } from "lucide-react";
2 import { useCallback, useRef, useState } from "react";
3 import Webcam from "react-webcam";
4 import { scanImage } from "../services/api";
5 import "./styles/components/CameraScanner.scss";
6
7 const videoConstraints = {
8   facingMode: { exact: "environment" },
9   width: { ideal: 1280 },
10  height: { ideal: 720 },
11 };
12
13 You, 22 hours ago | 1 author (You)
14 interface CameraScannerProps {
15   setShowCamera: (value: boolean) => void;
16   setIngredients: (ingredients: string[]) => void;
17   setIsScanned: (value: boolean) => void;
18   setShowIngredients: (value: boolean) => void;
19   setScanError: (value: boolean) => void;
20 }
21 const CameraScanner: ({ setShowCamera, setIngredients, setShow... = ({|
22   setShowCamera,
23   setIngredients,
24   setShowIngredients,
25   setIsScanned,
26   setScanError,
27 }: CameraScannerProps) => {|

```

Maintenant que tout est prêt, je commence par définir les variables. webcamRef initialise un useRef pour permettre l'appel à la méthode getScreenshot de Webcam, et capturedImage est initialisé avec un useState pour stocker l'image capturée à l'aide de la fonction associée. Je crée ensuite 3 fonctions :

- takePhoto : Utilise la méthode getScreenshot pour capturer l'écran utilisateur sans créer de re-rendu grâce au useRef, puis stocke l'image capturée dans la variable capturedImage. J'utilise useCallback pour éviter que la fonction soit recréée à chaque rendu du composant

```

34   const takePhoto: () => void = useCallback(() => {
35     if (webcamRef.current) {
36       const imageSrc: any = webcamRef.current.getScreenshot();
37       if (imageSrc) setCapturedImage(imageSrc);
38     }
39   }, []);

```

- retryPhoto : Supprime simplement la photo enregistrée pour permettre de reprendre une nouvelle photo
- validatePhoto : Permet d'envoyer l'image au back-end et de récupérer la liste d'ingrédients générée, tout en gérant les potentielles erreurs

```

43  const validatePhoto : () => Promise<void> = async () => {
44    if (!capturedImage) return;
45
46    try {
47      const data : any = await scanImage(capturedImage);
48      setIngredients(data.ingredients);
49    } catch (err) {
50      console.error("Erreur upload:", err);
51      setScanError(true);
52    }
53    setIsScanned(true);
54    setShowIngredients(true);
55    setShowCamera(false);
56  };

```

Dans la fonction validatePhoto, j'utilise la méthode "scanImage" avec pour argument l'image capturée. Cette méthode gère la liaison avec le back et est stockée dans le fichier api.ts. Je regroupe tous les appels au back-end dans un seul et même fichier pour une meilleure maintenabilité. La fonction consiste en un fetch vers la route créée précédemment, en méthode POST, avec l'image dans le body de la requête, puis elle renvoie le json reçu en retour.

```

107 // AI
108 export const scanImage : (picture: string) => Promise<any> = async (picture: string) => {
109   const res : Response = await fetch(`[${apiUrl}]/api/v1/ai`, {
110     method: "POST",
111     headers: { "Content-Type": "application/json" },
112     body: JSON.stringify({ picture }),
113   });
114   return res.json();
115 };

```

Une fois l'analyse IA réalisée, je ferme l'affichage de la caméra utilisateur, et j'affiche le résultat du scan avec le composant ScannerIngredient.tsx. Pour ce composant, le fonctionnement est plus basique. J'utilise des useState pour la gestion du chargement et la gestion d'erreur, et à partir de la liste d'ingrédients générée (si et seulement si elle a été générée), je réalise une recherche dans la base de données à l'aide d'une autre route et d'un autre controller de recherche d'ingrédients, pour récupérer les ingrédients et leur ID dans la base de données.

```

97  export const searchIngredients : (query: string) => Promise<any> = async (query: string) => {
98    const res : Response = await fetch(`[${apiUrl}]/api/v1/ingredients/search/${query}`);
99    return res.json();
100 };

```

En récupérant ainsi les ingrédients, je pourrais ensuite faire une recherche pour les recettes contenant ces ingrédients, en les classant de la recette ayant le plus d'ingrédients correspondant à la liste, à celle en ayant le moins.

```

84  export const searchRecipesByIngredients : (ingredientIds: string) => Promise<any> = async (ingredientIds: string) => {
85    const res : Response = await fetch(
86      `${apiUrl}/api/v1/recipes/search/ingredients/${ingredientIds}`
87    );
88    return res.json();
89  };

```

Je peux ensuite afficher les recettes récupérées dans la base de données grâce à mon back-end.

Génération de recette

Une fois que l'utilisateur a réalisé sa recherche à partir de la liste d'ingrédients, il pourra choisir d'afficher plus de recettes (pour éviter de surcharger la page), et lorsque plus aucune recette supplémentaire ne pourra être affichée, il pourra décider de générer une recette n'utilisant que les ingrédients présents dans la liste utilisée pour la recherche.

A. Back-end

Pour réaliser cette génération de recette, je crée une nouvelle route ainsi qu'un nouveau controller dans les mêmes fichiers que ceux utilisés pour l'analyse d'image. Dans mon controller, je commence par vérifier la présence de la liste d'ingrédients pour éviter une génération vide. Je prépare ensuite mon prompt dans lequel j'indique à l'IA tous les éléments à renvoyer ainsi que leur type pour assurer ensuite un bon traitement. Le prompt étant très long, il est disponible en annexe. Ensuite, pour le traitement de la réponse de l'IA, je réalise les mêmes manipulations que pour l'analyse pour nettoyer et vérifier la réponse. La différence principale est que cette fois-ci, je vérifie bien la présence des clés "recipe" et "ingredients" dans la réponse, sans quoi je renvoie une erreur.

```

253  // Vérification basique du format attendu
254  const requiredKeys : string[] = ["ingredients", "recipe"];
255
256  for (const key of requiredKeys) {
257    if (!recipeJson[key]) {
258      return res.status(500).json({
259        error: `Clé manquante dans le JSON généré : ${key}`,
260        raw: recipeJson,
261      });
262    }
263  }

```

B. Front-end

Pour gérer la recette générée, je crée une fonction dans le composant ScannerRecipes.tsx, servant à afficher les résultats de recherche par ingrédients, permettant la génération et l'affichage de la recette. Je commence par modifier le bouton de génération de recette grâce à une variable "isGenerating" créée avec un useState, pour donner un retour à l'utilisateur sur son action. Ensuite, je fais appel à la méthode "generateRecipe" avec pour argument les ingrédients utilisés dans la recherche. Dans cette méthode, je formate les ID d'ingrédients reçus pour préparer correctement ma recherche. J'utilise ensuite la méthode "getIngredientsByIds" avec les IDs formatés, ce qui va me renvoyer les informations des ingrédients. À partir de ces informations, je crée un tableau contenant uniquement les noms des ingrédients. J'envoie ensuite cette liste de nom à mon back-end pour la génération grâce à un fetch. Je renvoie la réponse du back-end dans le composant ensuite.

```
117  export const generateRecipe : (ingredientIds: string) => Promise<any> = async (ingredientIds: string) => [
118    const formattedIds : string = ingredientIds.replace(/ /g, '+');
119    const ingredientsData : any = await getIngredientsByIds(formattedIds);
120    const ingredients : any = ingredientsData.map((ing: { name_ingredient: string }) => ing.name_ingredient);
121
122    const res : Response = await fetch(`$apiUrl/api/v1/ai/generate-recipe`, {
123      method: "POST",
124      headers: { "Content-Type": "application/json" },
125      body: JSON.stringify({ ingredients }),
126    });
127    return await res.json();
128  ];
```

Une fois la recette générée, je redirige l'utilisateur sur la page recette, avec une URL personnalisée. Au lieu de contenir l'ID de la recette comme pour une recette normale, elle contient "generated", ce qui me permettra de rendre la page différemment. J'envoie également les informations récupérées lors de la génération avec un "state" pour pouvoir les manipuler.

```
62  const handleGenerateRecipe : () => Promise<void> = async () => {
63    setIsGenerating(true);
64    try {
65      const data : any = await generateRecipe(search);
66      navigate("/recipes/recipe/generated", {
67        state: {
68          recipe: data.recipe,
69          ingredients: data.ingredients,
70        },
71      });
72    } catch (error) {
73      console.error(error);
74      alert("Erreur lors de la génération de la recette. Veuillez réessayer.");
75    } finally {
76      setIsGenerating(false);
77    }
78  };
```

Sur ma page recette, j'utilise une fonction "getRecipe" qui vérifie si la recette est générée ou non. Dans le cas où elle est générée, elle récupère les informations dans le "state", sinon elle récupère grâce à l'ID comme en temps normal.

```

32  const getRecipe : () => Promise<void> = async () => {
33    if (isGenerated) {
34      if (location.state?.recipe) {
35        setRecipe({
36          ...location.state.recipe,
37          ingredients: location.state.ingredients,
38        });
39      } else {
40        setReqError("Une erreur est survenue lors de la génération de la recette, réessaie plus tard !");
41      }
42      setIsLoading(false);
43      return;
44    }

```

Elle rend ensuite la page comme en temps normal, sans image de la recette mais avec un cadre précisant que la recette est générée par IA et peut donc contenir des erreurs.

```

151   {isGenerated && (
152     <div id="is-ia">
153       <strong>⚠ Recette générée par IA</strong>
154       <p>
155         Cette recette a été créée automatiquement. Vérifiez toujours les
156         ingrédients, les quantités et les temps de cuisson. En cas
157         d'allergies ou de régime spécifique, consultez un professionnel de
158         santé.
159       </p>
160     </div>
161   )}

```

Authentification utilisateur

Pour gérer l'affichage personnalisé, j'ai décidé de passer par l'authentification utilisateur. Ainsi, quelle que soit la plateforme sur laquelle l'utilisateur est connecté, il gardera les mêmes paramètres d'affichage.

A. Back-end

Pour la partie back-end de l'authentification utilisateur, je crée 4 routes :

- “/register” - POST : Permet à un utilisateur de s’inscrire
- “/login” - POST : Permet à l’utilisateur de se connecter
- “/forgot-pass” - POST : Permet d’envoyer un mail à l’utilisateur avec un lien de réinitialisation de mot de passe
- “/reset-pass” - PUT : Permet de réinitialiser le mot de passe de l’utilisateur grâce à un token fourni dans le lien envoyé par mail

Pour l'inscription utilisateur, je commence par vérifier l'existence d'un utilisateur dans la base de données utilisant soit l'adresse mail, soit le nom d'utilisateur. Dans le cas où l'un des deux est pris, je renvoie une erreur à l'utilisateur. Je ne fais pas la distinction entre adresse mail et nom d'utilisateur dans l'erreur pour des raisons de sécurité. Ensuite, je crée une variable de date à la valeur de l'instant T, pour dater la date de création du compte de l'utilisateur. Je crypte ensuite le mot de passe utilisateur avec bcrypt. Maintenant que tous les éléments sont prêts, j'insère le nouvel utilisateur dans la base de données en tant que "Member", ce qui servira plus tard lors de l'évolution du projet pour gérer les autorisations d'utilisateur. Une fois l'utilisateur enregistré, je crée un token de connexion JWT pour que l'utilisateur soit directement connecté sur la page de modification de profil et je le renvoie à l'utilisateur.

```
20 exports.register = async (req, res) => {
21   const { username, email, password } = req.body;
22   You, 5 months ago * feat: add user authentication ...
23   try {
24     const user : any = await db.oneOrNone(
25       query: "SELECT * FROM users WHERE email_user = $1 OR username_user = $2",
26       [email, username]
27     );
28
29     if (user)
30       res.status(409).json({
31         message: "Tu ne peux pas utiliser ce nom d'utilisateur ou cet email.",
32       });
33
34     const date = new Date();
35
36     const salt : string = await bcrypt.genSalt(parseInt(string: 12));
37     const passwordHash : string = await bcrypt.hash(password, salt);
38
39     const newUser : any = await db.one(
40       query: `INSERT INTO users (username_user, email_user, password_user, created_at, updated_at, "rights_user") VALUES ($1, $2, $3, $4, $5, 'Member')
41       RETURNING id_user, username_user, email_user`,
42       [username, email, passwordHash, date, date]
43     );
44
45     const token : never = jwt.sign(
46       {
47         id: newUser.id_user,
48         username: newUser.username_user,
49       },
50       process.env.JWT,
51       { expiresIn: "7d" }
52     );
53
54     res.status(201).json({
55       message: "Utilisateur connecté avec succès",
56       user: {
57         id: newUser.id_user,
58         username: newUser.username_user,
59         email: newUser.email_user,
60       },
61       token,
62     });
63   } catch (err) {
64     res.status(500).json({ message: err.message });
65   }
};
```

Pour la connexion de l'utilisateur, je sélectionne l'utilisateur et ses informations dans la base de données grâce à son adresse mail, et renvoie une erreur 404 en cas d'utilisateur non trouvé. Si je trouve bien l'utilisateur, je vérifie que le mot de passe est bon grâce à bcrypt, et renvoie une erreur 401 si c'est incorrect. Si le mot de passe est correct, je crée un token de connexion et le renvoie à l'utilisateur.

```

81 exports.login = async (req, res) => {
82   const { email, password } = req.body;
83
84   try {
85     const user : any = await db.oneOrNone(
86       query: "SELECT id_user, email_user, password_user, username_user FROM users WHERE email_user ILIKE $1",
87       email
88     );
89
90     if (!user)
91       return res
92         .status(404)
93         .json({ message: "Cet email n'est lié à aucun compte" });
94
95     const isMatch: boolean = await bcrypt.compare(password, user.password_user);
96     if (!isMatch)
97       return res.status(401).json({ message: "Le mot de passe est incorrect" });
98
99     const token: never = jwt.sign(
100       { id: user.id_user, username: user.username_user },
101       process.env.JWT,
102       { expiresIn: "7d" }
103     );
104
105     res.json(token);
106   } catch (err) {
107     res.status(500).json({ message: err.message });
108   }
109 }

```

Ensuite, pour le mot de passe oublié, je commence par vérifier si l'email est bien lié à un utilisateur existant. Ensuite, je crée un token JWT valide pendant 15 minutes, et j'appelle la méthode "emailForgotPass" pour envoyer le mail à l'utilisateur. La méthode est créée dans un fichier mailerForgotPass.js, elle est réalisée grâce à nodemailer. Je configure d'abord le transporteur, avec l'adresse mail et un mot de passe d'application généré par Gmail. Ensuite je prépare le mail avec le destinataire, l'objet et le contenu. Enfin, j'envoie avec la méthode "sendMail" de nodemailer.

```

1  const nodemailer = require(id: "nodemailer");
2
3  async function emailForgotPass(email : any, username : any, token : any): Promise<SentMessageInfo> {
4    const transporter = nodemailer.createTransport({
5      service: "gmail",
6      auth: {
7        user: process.env.GMAIL_USER,
8        pass: process.env.GMAIL_PASS,
9      },
10    });
11
12    const mailOptions = {
13      from: `WIMF ${process.env.GMAIL_USER}`,
14      to: email,
15      subject: "Réinitialisation du mot de passe - mot de passe oublié",
16      text: `Bonjour ${username},
17
18 Nous avons reçu une demande réinitialisation de mot de passe pour ton compte WIMF. Si tu n'es pas à l'origine de cette demande, contente toi d'ignorer ce
mail.
19
20 Si tu es bien à l'origine de cette demande, voici ton lien de réinitialisation de mot de passe : ${process.env.URL_WEBSITE}reset-pass/${token} ! You
21
22 Ce lien sera fonctionnel pendant les 15 prochaines minutes !`,
23
24
25  try {
26    const info : SentMessageInfo = await transporter.sendMail(mailOptions); // optionnel : tu peux retourner l'info si ton contrôleur en a besoin
27    return info;
28  } catch (error) {
29    // Important : on relance l'erreur pour que le contrôleur puisse la gérer
30    throw new Error(message: "Échec de l'envoi de l'email");
31  }
32}

```

Enfin, pour la réinitialisation du mot de passe, je commence par vérifier le token utilisateur grâce à un middleware dans lequel je décrypte le token fourni.

```
11  const resetPassMiddleware : (req: Object, res: Object, next: Function...) = (req, res, next) => {
12    const resetPassHeader : any = req.headers.authorization;
13
14    if (!resetPassHeader || !resetPassHeader.startsWith("Bearer ")) {
15      return res.status(401).json({ message: "Token manquant !" });
16    }
17
18    const token : any = resetPassHeader.split(" ")[1];
19
20    try {
21      const decoded : Jwt & JwtPayload & void = jwt.verify(token, process.env.JWT);
22      req.user = decoded;
23      next();
24    } catch (err) {
25      res.status(401).json({ message: "Token invalide ou expiré" });
26    }
27  };
28
29  module.exports = resetPassMiddleware;
```

Je vérifie si l'utilisateur existe bien, s'il essaie bien réinitialiser sa propre adresse mail, et enfin, comme pour l'inscription de l'utilisateur, je crypte le mot de passe et met à jour la date de mise à jour utilisateur dans la base de données.

```
167  exports.resetPassword = async (req, res) => {
168    const { email, password } = req.body;
169
170    try {
171      const userToReset : any = await db.oneOrNone(
172        query: "SELECT email_user FROM users WHERE id_user = $1",
173        req.user.id
174      );
175
176      if (!userToReset)
177        return res.status(404).json({ message: "Utilisateur introuvable" });
178
179      if (email != userToReset.email_user)
180        return res
181          .status(401)
182          .json({ message: "Tu n'es pas autorisé à réaliser cette action" });
183
184      if (password != null) {
185        const salt : string = await bcrypt.genSalt(parseInt(string: 12));
186        const passwordHash : string = await bcrypt.hash(password, salt);
187
188        const updateDate = new Date();
189
190        const userReset : any = await db.one(
191          query: `UPDATE users
192            SET "password_user" = $1,
193            "updated_at" = $2
194            WHERE id_user = $3
195            RETURNING id_user, username_user, email_user`,
196            [passwordHash, updateDate, req.user.id]
197        );
198
199        res.json(userReset);
200      }
201    } catch (err) {
202      res.status(500).json({ message: err.message });
203    }
204  };
```

Je prépare ensuite les routes pour l'authentification.

```
3 // Intégrer le middleware d'authentification
4 const resetPassMiddleware : (req: Object, res: Object, next: Function) = require('../middlewares/resetPassMiddleware')
5 // Intégrer le controller des utilisateurs
6 const authControllers = require(id: "../controllers/authControllers");
7
8 // Authentification
9 router.post(path: "/register", authControllers.register); // inscription
10 router.post(path: "/login", authControllers.login); // connexion
11 router.post(path: "/forgot-pass", authControllers.forgotPass); // Mot de passe oublié
12 router.put(path: "/reset-pass", resetPassMiddleware, authControllers.resetPassword); // Réinitialisation
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45 // Authentification
46 app.use(path: "/api/v1/auth", authRoutes);
```

B. Front-end

Pour la partie front-end, je gère l'inscription et la connexion sur une seule et même page, avec la possibilité pour l'utilisateur de choisir entre les deux actions grâce à deux boutons. La page Profile.tsx contient plusieurs composants affichés selon le choix utilisateur entre le formulaire d'inscription et de connexion, et le formulaire de mot de passe oublié dans le second composant. L'affichage dynamique est géré grâce à plusieurs useState pour éviter un rechargeement complet de la page. La page vérifie également la présence ou non d'un token de connexion, dans le cas où il est présent, l'utilisateur est redirigé vers la page de modification de profil.

```
13 const Profile : () => Element = () => {
14   const token : string | null = useAuthStore((state : AuthState) => state.token);
15   const [showSignIn, setShowSignIn] = useState<boolean>(false);
16   const [showLogIn, setShowLogIn] = useState<boolean>(false);
17   const [showButtons, setShowButtons] = useState<boolean>(false);
18   const navigate : NavigateFunction = useNavigate();
19   const isMobile : boolean = useIsMobile();
20
21   document.title = "Mon compte";
22
23   useEffect(() => {
24     if (token) {
25       navigate("/profile infos");
26       return;
27     }
28     setShowButtons(true);
29     setShowLogIn(false);
30     setShowSignIn(false);
31   }, [token, navigate]);
```

Pour la création de compte, je crée un formulaire HTML simple, demandant à l'utilisateur son nom d'utilisateur, adresse mail, mot de passe, confirmation du mot de passe et enfin la validation RGPD. Je prépare également des messages d'erreur sous chaque champ de saisie, permettant d'indiquer à l'utilisateur quel champ empêche son inscription. L'affichage d'erreur est géré encore une fois grâce à des useState et des affichages conditionnels. Par exemple pour le mot de passe :

```

<div>
  <label htmlFor="password">Mot de passe</label>
  <input type="password" id="password" name="password" required aria-required="true" aria-invalid={errorPassword} aria-describedby={errorPassword ? "password-error" : undefined} />
  {errorPassword && (
    <div id="password-error" role="alert">
      <p>Le mot de passe doit :</p>
      <ul>
        <li>Faire au moins 8 caractères</li>
        <li>Contenir au moins une lettre majuscule</li>
        <li>Contenir au moins une lettre minuscule</li>
        <li>Contenir au moins un chiffre</li>
        <li>Contenir au moins un caractère spécial (@$!%*?&)</li>
      </ul>
    </div>
  )}
</div>

```

Ensuite, je crée une fonction pour gérer la vérification de chaque champ puis l'envoi du formulaire. Je crée donc des regex pour vérifier le contenu des champs. Je vérifie également la confirmation du mot de passe ainsi que le case RGPD. En cas d'erreur sur un quelconque élément, une variable isError booléenne reçoit la valeur “true”, ce qui annule l'envoi du formulaire. Si tous les éléments sont corrects, j'appelle la méthode “register” pour inscrire l'utilisateur. J'utilise la méthode handleResponse qui gère automatiquement la réception d'erreur HTTP.

```

39   // Regex
40   const usernameRegex = /^[a-zA-Z0-9_-]{3,20}$/;
41   const emailRegex = /^[^\s@]+@[^\s@]+\.\[^ \s@]+\$/;
42   const passwordRegex =
43     /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[$!%*?&])[A-Za-z\d@$!%*?&]{8,})$/;

```

```

25   export const register : (username: string, email: string, password: string) => Promise<any> = async (
26     username: string,
27     email: string,
28     password: string
29   ) => {
30     const res : Response = await fetch(`${
31       apiUrl
32     }/api/v1/auth/register`, {
33       method: "POST",
34       headers: { "Content-Type": "application/json" },
35       body: JSON.stringify({ username, email, password })
36     });
37     return handleResponse(res);
38   };

```

La méthode register renvoie un token comme préparé plus tôt dans le back-end. Si la réponse contient bien un token (inscription réussie), j'utilise la méthode “setToken” pour maintenir l'utilisateur connecté. J'expliquerai son fonctionnement en même temps que la connexion utilisateur.

```

73   try {
74     const res : any = await register(username, email, password);
75     if (res.token) {
76       setToken(res.token, true);
77       setIsRegistered(true);
78     } else {
79       setIsRegistered(false);
80     }

```

Pour la connexion utilisateur, le fonctionnement est plus simple. Le formulaire de connexion contient un champ pour l'adresse mail, un champ pour le mot de passe, un bouton pour le mot de passe oublié et une checkbox

permettant à l'utilisateur de choisir de rester connecté ou non. Lors de l'envoi du formulaire, je vérifie d'abord que les deux champs sont bien remplis pour éviter un appel back-end inutile, puis si les champs sont bien remplis, j'appelle la méthode "login".

```
15 // Auth
16 export const login : (email: string, password: string) => Promise<Response> = async (email: string, password: string) => {
17   const res : Response = await fetch(`.${apiUrl}/api/v1/auth/login`, {
18     method: "POST",
19     headers: { "Content-Type": "application/json" },
20     body: JSON.stringify({ email, password }),
21   });
22   return handleResponse(res);
23 };
```

Si la connexion utilisateur réussie, je stocke le token grâce à Zustand et redirige l'utilisateur vers la page de gestion de profil, sinon j'affiche une notification d'erreur pendant 3 secondes.

```
48 if (!newErrors.email && !newErrors.password) {
49   try {
50     const res : any = await login(email, password);
51     if (res.token || res) {
52       setToken(res, remainConnected);
53       setShowLogIn(false);
54       navigate("/profile infos");
55     }
56   } catch (error) {
57     // Affichage de l'erreur retournée par l'API
58     const errorMessage : string = error instanceof Error ? error.message : "Erreur de connexion";
59     setNotification(errorMessage);
60     setTimeout(() => setNotification(null), 3000);
61   }
62 }
```

Je vais maintenant expliquer le stockage du token grâce à Zustand. Pour commencer, je type les différents éléments présents dans mon élément useAuthStore, et je crée une fonction de vérification de l'expiration du token.

```
1 import { create } from "zustand";
2
3 You, 2 months ago | 1 author (You)
4 interface AuthState {
5   token: string | null;
6   setToken: (token: string, rememberMe: boolean) => void;
7   clearToken: () => void;
8   checkTokenExpiration: () => void;
9   loadToken: () => void;
10 }
11 const isTokenExpired : (token: string) => boolean = (token: string): boolean => {
12   try {
13     const payload : any = JSON.parse(atob(token.split(".")[1]));
14     return payload.exp * 1000 < Date.now();
15   } catch {
16     return true;
17   }
18 };
```

Une fois cette étape passée, je crée mon élément useAuthStore dans lequel je vais définir une variable "token" ainsi que plusieurs méthodes pour sa gestion. D'abord, la méthode setToken, que j'ai utilisée plus tôt, utilise la variable du token utilisateur, puis grâce à une variable "rememberMe" (qui correspond à la case cochée lors de la

connexion utilisateur), stocke le token en variable de session si l'utilisateur n'a pas coché la case, et en variable dans le localStorage si l'utilisateur l'a cochée.

```
22 |   setToken: (token: string, rememberMe: boolean) => {
23 |     set({ token });
24 |     if (rememberMe) {
25 |       localStorage.setItem("auth-token", token);
26 |       sessionStorage.removeItem("auth-token");
27 |     } else {
28 |       sessionStorage.setItem("auth-token", token);
29 |       localStorage.removeItem("auth-token");
30 |     }
31 |   },
```

La méthode "clearToken" sert à déconnecter l'utilisateur. Elle supprime tout élément "token" des variables de session ou du localStorage lorsqu'elle est appelée.

```
32 |   clearToken: () => {
33 |     set({ token: null });
34 |     localStorage.removeItem("auth-token");
35 |     sessionStorage.removeItem("auth-token");
36 |   },
```

Ensute, la méthode "checkTokenExpiration" vérifie que le token n'est pas expiré grâce à la fonction créée plus tôt dans le document. Dans le cas où il expire, la méthode "clearToken" est appelée.

```
37 |   checkTokenExpiration: () => {
38 |     const token: string | null = get().token;
39 |     if (token && isTokenExpired(token)) {
40 |       get().clearToken();
41 |     }
42 |   },
```

La méthode "loadToken" charge le token pour afficher correctement les informations si l'utilisateur est connecté tout en vérifiant encore une fois la validité du token.

```
43 |   loadToken: () => {
44 |     const token: string | null = localStorage.getItem("auth-token") || sessionStorage.getItem("auth-token");
45 |     if (token && !isTokenExpired(token)) {
46 |       set({ token });
47 |     } else if (token) {
48 |       get().clearToken();
49 |     }
50 |   },
```

Enfin pour terminer, j'appelle la méthode "loadToken" au démarrage de l'application pour pouvoir vérifier le token avec Zustand. J'ajoute également une vérification toutes les minutes du token pour m'assurer de sa validité et déconnecter l'utilisateur si le token n'est plus valide pendant son utilisation de l'application.

```

53  // Chargement du token au démarrage
54  useAuthStore.getState().loadToken();
55
56  // Vérification périodique toutes les minutes
57  setInterval(() => {
58    useAuthStore.getState().checkTokenExpiration();
59  }, 60000);

```

Voilà pour l'utilisation de Zustand dans le projet. Je vais maintenant pouvoir expliquer le fonctionnement de la réinitialisation du mot de passe utilisateur en cas d'oubli.

Je crée un composant contenant un formulaire uniquement composé d'un champ pour l'adresse mail et un bouton d'envoi. Lors de l'envoi du formulaire par l'utilisateur, je modifie l'affichage du bouton pour qu'il sache que son action a bien été prise en compte, et j'appelle la méthode "forgotPassword". Cette méthode lance un appel au back-end sur la route créée plus tôt.

```

38  export const forgotPassword : (email: string) => Promise<Response> = async (email: string) => {
39    const res : Response = await fetch(`.${apiUrl}/api/v1/auth/forgot-pass`, {
40      method: "POST",
41      headers: { "Content-Type": "application/json" },
42      body: JSON.stringify({ email }),
43    });
44    return res;
45  };

```

Selon la réponse du serveur, l'utilisateur reçoit comme information :

- Une erreur si le mail est incorrect
- Une erreur serveur est survenue (erreur serveur ou erreur nodemailer)
- Un message de confirmation de l'envoi du mail

```

19  try {
20    const res : Response = await forgotPassword(email);
21    if (res.status === 404) {
22      setErrorEmail(true);
23    } else if (!res.ok) {
24      setErrorServer(true);
25    } else {
26      setSuccess(true);
27    }
28  } catch {
29    setErrorServer(true);
30  } finally {
31    setLoading(false);
32  }
33};

```

Le mail envoyé à l'utilisateur contient un lien de réinitialisation contenant le token généré par le back. L'utilisateur arrive sur une page avec un formulaire lui demandant son email, ainsi que deux champs de mot de passe (modification et confirmation). Lors de l'envoi du formulaire, la page vérifie la présence du token ainsi que le mot de passe (conforme aux regex vus plus tôt) et la confirmation du mot de passe. Une fois ces vérifications faites, le bouton d'envoi du formulaire change pour faire savoir à l'utilisateur que la modification est en cours. J'appelle

alors la méthode “resetPassword” qui envoie le mail ainsi que le mot de passe au back, en utilisant le token comme autorisation.

```
47  export const resetPassword : (token: string, email: string, password: ... = async (
48    token: string,
49    email: string,
50    password: string
51  ) => {
52    const res : Response = await fetch(` ${apiUrl}/api/v1/auth/reset-pass`, {
53      method: "PUT",
54      headers: {
55        "Content-Type": "application/json",
56        Authorization: `Bearer ${token}`,
57      },
58      body: JSON.stringify({ email, password }),
59    });
60    return res;
61  );
```

En cas d'erreur 401, j'affiche l'erreur à l'utilisateur. En cas de 404, je précise à l'utilisateur que son adresse mail n'est liée à aucun compte. Si le back-end ne renvoie pas un statut “ok”, j'affiche à l'utilisateur une erreur serveur. Enfin, si tout est bon, j'affiche un message de confirmation avec un lien renvoyant au profil pour que l'utilisateur puisse se connecter.

```
51  setLoading(true);
52  try {
53    const res : Response = await resetPassword(token, email, password);
54    if (res.status === 401) {
55      const data : any = await res.json();
56      setErrorMessage(data.message);          You, 2 months ago • feat(auth)
57    } else if (res.status === 404) {
58      setErrorMessage("Utilisateur introuvable");
59    } else if (!res.ok) {
60      setErrorMessage("Erreur serveur, réessayez plus tard");
61    } else {
62      setSuccess(true);
63    }
64  } catch {
65    setErrorMessage("Erreur serveur, réessayez plus tard");
66  } finally {
67    setLoading(false);
68  }
```

Tests et validation

Tests unitaires

Pour réaliser les tests dans mon projet, j'ai décidé d'utiliser Jest. Jest est un framework de test très accessible, permettant de réaliser différents types de tests et d'avoir des retours précis sur ses erreurs ainsi que la couverture globale des tests réalisés. Pour cette démonstration, je vais présenter un test réalisé pour mon controller d'authentification utilisateur. Je commence par créer un fichier authController.test.js, qui contiendra tous les tests concernant le controller authController.js. J'importe toutes les fonctions à tester, ainsi que les dépendances utilisées dans ces fonctions. Une fois tout importé, je réalise un mock de chaque dépendance, ce qui me permet de simuler leur comportement sans réellement les utiliser. Cela permet par exemple de ne pas manipuler la base de données mais simplement de simuler le comportement du controller vis-à-vis de la base de données.

```
8 // Import des dépendances
9 const db: IDatabase<{}, IClient> = require(id: "../db");
10 const jwt = require(id: "jsonwebtoken");
11 const bcrypt = require(id: "bcrypt");
12 const emailForgotPass: (email: any, username: any, token: any) => ... = require("../utils/mailForgotPass");
13
14 jest.mock("jsonwebtoken");
15 jest.mock("bcrypt");
16 jest.mock("../utils/mailForgotPass.js");
17 jest.mock("../db.js");
```

Le format des tests unitaires Jest est le suivant :

- J'utilise "describe" pour classer les tests en catégories (ex : "Auth Controllers" puis "Register" pour tester la fonction "register"). Cela permet de s'y retrouver dans les résultats des tests.

```
19 describe("Auth Controllers", () : void => {
20   const mockDate = new Date(value: "2025-10-09T13:33:02.815Z");
21
22   describe("Register", () : void => {
```

- J'utilise "beforeEach" et "afterEach" pour définir un comportement avant et après chaque test. (ex : préparer le faux "body" et les fausses fonctions puis les réinitialiser à la fin du test). C'est ce qui nous permet de simuler les envois et réponses lors des tests.

```
25 beforeEach(): void => {
26   req = {
27     body: {
28       username: "testuser",
29       email: "test@mail.com",
30       password: "password123",
31     },
32   };
33   res = {
34     status: jest.fn().mockReturnThis(),
35     json: jest.fn(),
36   };
37   process.env.JWT = "test-secret";
38   jest.useFakeTimers();
39   jest.setSystemTime(mockDate);
40 } // /beforeEach
41
42 afterEach(): void => {
43   jest.clearAllMocks();
44   jest.useRealTimers();
45 } // /afterEach
```

- J'utilise "it" pour définir quel test je vais réaliser (ex : it "devrait enregistrer et connecter l'utilisateur"). Encore une fois, c'est pour bien voir quel test correspond à quoi lors d'erreurs.

```
47 | | | it("should register and connect new user", async () : Promise<void> => {
```

- Je rédige le test en mockant chaque fonction et chaque comportement attendu grâce aux méthodes Jest.

```
54 | | | db.oneOrNone.mockResolvedValue(null);
```

- J'appelle ensuite la fonction à tester

```
63 | | | await register(req, res);
```

- Enfin, je teste les résultats grâce à "expect" suivi de l'action attendue.

```
65 | | | expect(db.oneOrNone).toHaveBeenCalledWith(
66 | | |     "SELECT * FROM users WHERE email_user = $1 OR username_user = $2",
67 | | |     ["test@mail.com", "testuser"]
68 | | );
```

Cet exemple de test est fourni en entier dans l'annexe.

Pour lancer les tests, j'utilise "npm test" après avoir configuré Jest comme outil de test par défaut. Je peux également utiliser "npm test -- --coverage" pour tester la couverture globale des tests. Cela permet de vérifier que chaque élément des controllers est bien couvert. Il faut toutefois faire attention, une couverture de 100% peut donner un faux sentiment de sécurité et faire l'impasse sur certaines failles du controller.

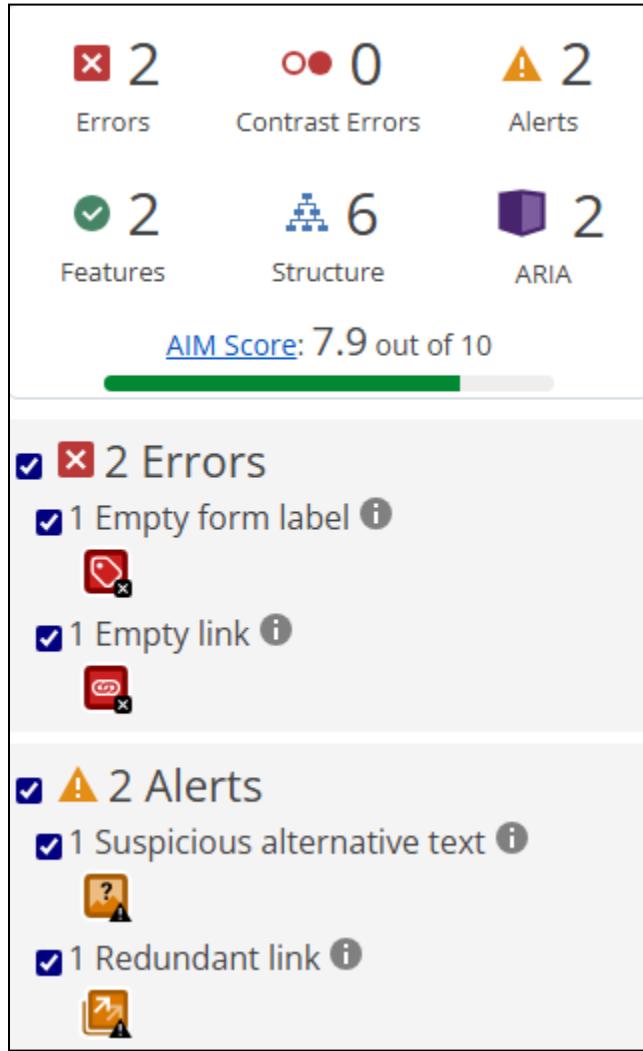
Test de performance et accessibilité

Pour réaliser les tests de performance, j'utilise un plug-in sur mon navigateur appelé "Lighthouse". Ce plug-in analyse la page choisie et offre un compte rendu des performances, de l'accessibilité, des bonnes pratiques ainsi que du SEO de la page. Par exemple, voici les résultats de mes derniers tests Lighthouse sur ma page d'accueil :



Grâce à ce test, je sais par exemple que je dois maintenant me focaliser sur le SEO de mon projet.

Pour tester l'accessibilité, je peux également utiliser WAVE pour par exemple voir les textes alternatifs manquants sur les images ou bien les problèmes de contraste. Voilà par exemple le résultat pour ma page d'erreur 404 :



Les résultats doivent par contre être interprétés correctement. Par exemple, les deux erreurs ici viennent d'un lien et d'un label contenant uniquement une icône, qui sont considérés comme vides. Il faut donc bien regarder les erreurs et ne pas corriger aveuglément pour obtenir le meilleur résultat possible.

Tests de compatibilité

Pour réaliser les tests de compatibilité, j'ai téléchargé les divers navigateurs que je voulais tester, et j'ai navigué sur mon site sur chacun des navigateurs pour y voir des erreurs, notamment de style. Par exemple, en testant Firefox, j'ai pu voir une erreur sur ma barre de recherche dans le header en version desktop, que j'ai ainsi pu corriger pour améliorer l'expérience utilisateur.

Problèmes rencontrés et résolutions

Ce projet était mon premier projet complet et d'ampleur assez importante, j'ai donc pu rencontrer plusieurs difficultés. Par exemple, lors de la phase de développement du back, à cause de mon manque d'expérience, j'ai commencé à développer plus de controllers que nécessaires, oubliant mon MVP et prenant donc ainsi du retard sur sa réalisation. Mon formateur m'a permis de lister les tâches qu'il me restait à faire et ainsi de me rendre compte de mon écart vis-à-vis du MVP. J'ai pu apprendre à mieux m'organiser et à suivre plus en détail mon outil de gestion de projet, afin d'optimiser mon temps de travail.

J'ai pu également rencontrer des difficultés lors de la création graphique de mon projet, n'ayant que peu voire pas de notions de design. Je n'étais pas satisfait de mon travail lors de la création des maquettes basses fidélités, j'ai alors décidé d'utiliser des outils en ligne de génération de design pour m'inspirer, notamment dans la disposition des éléments, et ainsi pouvoir réaliser des designs plus modernes et accessibles. J'ai également été aidé par mon formateur pour mieux comprendre la gestion des espaces vides pour ne pas étouffer l'utilisateur sous trop de contenu d'un coup par exemple.

Lors de la création de mes outils d'authentification, j'ai voulu intégrer Nodemailer pour l'envoi du mail de réinitialisation de mot de passe. J'ai passé de nombreuses heures à difficilement configurer, sans succès, l'outil. Malgré de nombreux tests, je n'arrivais pas à faire fonctionner. Un de mes camarades de formation, ayant déjà mis en place son outil, m'a fourni ses paramètres et m'a expliqué comment trouver ma configuration (par exemple le mot de passe d'application Gmail) ce qui m'a permis de gagner beaucoup de temps sur cet outil.

J'ai également pu rencontrer des difficultés lors de la recherche d'images d'illustration pour les recettes. Je ne pouvais légalement pas voler les images d'autres sites de cuisine. J'ai donc essayé de contacter plusieurs sites importants (Marmiton, Jow) pour demander un droit temporaire à l'utilisation de leurs illustrations. Le résultat est assez décevant, n'ayant soit pas eu de réponse, soit un refus, j'ai dû chercher une solution. J'ai donc décidé de, au fur et à mesure du temps, réaliser moi-même les recettes ajoutées, et donc réaliser moi-même les photos d'illustration. Ainsi, je n'aurais aucun problème de droit d'auteur. Ma solution temporaire est d'ajouter une photo d'une assiette vide comme image par défaut pour toutes les recettes n'ayant pas encore leur propre illustration.

Bilan et conclusion

Ce projet m'a énormément apporté sur tous les aspects de sa réalisation. J'ai déjà appris à mieux gérer un projet, et à travailler plus rigoureusement selon un planning et selon des tâches précises à réaliser. Étant mon premier projet personnel en autonomie de A à Z, j'ai dû trouver moi-même des solutions aux problématiques posées sur beaucoup d'éléments que je n'avais que très peu vus auparavant. J'ai beaucoup aimé travailler sur la conception de la base de données et sur l'arborescence de mon back-end pour pouvoir mieux la manipuler. Les points où j'ai pu avoir plus de difficultés sont la conception graphique ainsi que l'arborescence du site. J'ai pu facilement me perdre en mélangeant mes idées ou bien à cause de mon manque de notions graphiques. Malgré tout, je suis satisfait de mes réalisations et du travail déployé sur l'entièreté du projet.

Ce projet m'a fait réaliser certains points sur lesquels je dois maintenant travailler pour m'améliorer. Par exemple, je dois continuer à mieux suivre les outils de gestion de projet pour continuer d'améliorer mon organisation de travail. Mes notions graphiques vont devoir être retravaillées pour pouvoir réaliser plus rapidement et efficacement des maquettes dans mes prochains projets. Je souhaite également m'améliorer sur la conception du back-end, car, même en étant satisfait, je sais qu'il y a beaucoup d'axes d'amélioration dans ma façon de travailler sur ce point, notamment dans la rédaction des controllers et des routes.

Pour ce qui est du projet, j'aimerais le faire évoluer en quelque chose de plus ergonomique. J'ai déjà commencé à développer les controllers et les routes pour l'ajout de catégories de recettes (Entrée, Plat, Dessert, Boisson) permettant aux utilisateurs de classer les recettes selon leur recherche. Je travaille également sur des éléments d'inclusivité, notamment l'ajout d'ingrédients bannis, pour les allergies par exemple, la possibilité de choisir un régime précis (végan, végétarien, sans gluten, etc...) et donc de bloquer des recettes grâce à des tags liés à ces régimes. Je compte ajouter également des informations concernant les ustensiles utilisés dans les différentes recettes, pour faciliter l'organisation de l'utilisateur. Enfin, je vais également travailler sur l'ajout d'un espace de commentaires et de notations, afin de rendre l'application un peu plus communautaire.

L'objectif est de réaliser mon site de cuisine "de rêve" pour pouvoir l'utiliser moi-même au quotidien, en espérant également toucher des gens à qui il pourrait servir. Si le site aboutit comme je le souhaite, j'aimerais également en faire une version application téléchargeable, car je considère cette solution comme plus pratique et plus optimisée que de passer entièrement par navigateur (pour la version mobile). J'ai pris beaucoup de plaisir à solutionner mes différents problèmes et difficultés lors de ce projet. Le fait de pouvoir réaliser ce projet me tenait à cœur, et je suis fier de l'avoir rendu fonctionnel, ergonomique et moderne.

Annexe

Prompt utilisé pour la génération de recette (page 38) :

Tu es un assistant culinaire expert.

Tu dois générer UNE recette qui utilise UNIQUEMENT les ingrédients suivants :

\${JSON.stringify(ingredients, null, 2)}

Répond STRICTEMENT avec un JSON valide et rien d'autre.

N'ajoute aucun texte avant ou après.

FORMAT EXACT ATTENDU :

```
{  
  "ingredients": [  
    {  
      "name_ingredient": "string",  
      "quantity": number,  
      "measurements": "string"  
    }  
  ],  
  "recipe": {  
    "name_recipe": "string",  
    "preparation_time": number,  
    "cooking_time": number,  
    "resting_time": number,  
    "instructions": {  
      "steps": ["string", "string", "string"]  
    },  
    "servings_recipe": number,  
    "nutritional_values_recipe": {  
      "totalFat": {"name": "Matières grasses totales", "quantity": number, "unit": "g"},  
      "saturatedFat": {"name": "Acides gras saturés", "quantity": number, "unit": "g"},  
      "cholesterol": {"name": "Cholestérol", "quantity": number, "unit": "mg"},  
      "sodium": {"name": "Sodium", "quantity": number, "unit": "mg"},  
      "totalCarbohydrate": {"name": "Glucides totaux", "quantity": number, "unit": "g"},  
      "dietaryFiber": {"name": "Fibres alimentaires", "quantity": number, "unit": "g"},  
      "totalSugars": {"name": "Sucres totaux", "quantity": number, "unit": "g"},  
      "protein": {"name": "Protéines", "quantity": number, "unit": "g"},  
      "calcium": {"name": "Calcium", "quantity": number, "unit": "mg"},  
      "iron": {"name": "Fer", "quantity": number, "unit": "mg"},  
    }  
  }  
}
```

```

    "potassium": {"name": "Potassium", "quantity": number, "unit": "mg"},  

    "calories": {"name": "Calories", "quantity": number, "unit": "kcal"}  

}  

}  

}

```

CONTRAINTES IMPORTANTES :

- Le JSON doit être parfaitement valide.
- "ingredients" doit être un tableau d'objets, un pour chaque ingrédient fourni.
- "name_ingredient" doit correspondre EXACTEMENT aux ingrédients fournis (en minuscules si nécessaire).
- "quantity" : mettre 0 si la quantité n'est pas explicitable.
- "measurements" : indiquer une unité réaliste ("g", "ml", "pièce", etc.). Si inconnu → "" (chaîne vide).
- Dans "recipe", n'utilise QUE ces ingrédients.
- Pour les valeurs nutritionnelles, si l'IA n'est pas certaine → mettre "quantity": 0.
- AUCUNE clé supplémentaire.
- AUCUN texte autour, strict JSON.

Règles pour les ingrédients :

- Tu dois extraire tous les ingrédients mentionnés dans la recette.
- S'il manque la quantité ou l'unité dans le texte, tu dois FAIRE UNE ESTIMATION COHÉRENTE pour une préparation de 2 personnes.
- Il est strictement interdit d'utiliser quantity = 0 ou measurements = "" sauf si l'ingrédient ne peut vraiment pas être quantifié (ex : "sel" → mettre une estimation comme "1 pincée").
- Toujours donner une unité logique ("g", "ml", "c.à.s", "c.à.c", "pincée", "unité", etc.).
- Les quantités doivent être réalistes (pas 1 g ou 2000 g sans raison).

À la 3ème ligne, on s'assure que l'IA utilise bien tous les ingrédients fournis en paramètre dans la recette générée.

Test unitaire (page 50) :

```
47 it("should register and connect new user", async () : Promise<void> => {
48   const mockNewUser = {
49     id_user: 1,
50     username_user: "testuser",
51     email_user: "test@mail.com",
52   };
53
54   db.oneOrNone.mockResolvedValue(null);
55
56   bcrypt.genSalt.mockResolvedValue("mock-salt");
57   bcrypt.hash.mockResolvedValue("mock-hash");
58
59   db.one.mockResolvedValue(mockNewUser);
60
61   jwt.sign.mockReturnValue("mock-token");
62
63   await register(req, res);
64
65   expect(db.oneOrNone).toHaveBeenCalledWith(
66     "SELECT * FROM users WHERE email_user = $1 OR username_user = $2",
67     ["test@mail.com", "testuser"]
68   );
69   expect(bcrypt.genSalt).toHaveBeenCalled();
70   expect(bcrypt.hash).toHaveBeenCalled("password123", "mock-salt");
71   expect(db.one).toHaveBeenCalled(
72     "INSERT INTO users (username_user, email_user, password_user, created_at, updated_at, rights_user) VALUES ($1, $2, $3, $4, $5, 'Member') RETURNING id_user, username_user, email_user",
73     ["testuser", "test@mail.com", "mock-hash", mockDate, mockDate]
74   );
75   expect(jwt.sign).toHaveBeenCalled(
76     {
77       id: 1,
78       username: "testuser",
79     },
80     process.env.JWT,
81     { expiresIn: "7d" }
82   );
83   expect(res.status).toHaveBeenCalledWith(201);
84   expect(res.json).toHaveBeenCalled({
85     message: "Utilisateur connecté avec succès",
86     user: [
87       {
88         id: 1,
89         username: "testuser",
90         email: "test@mail.com",
91       },
92       token: "mock-token",
93     ];
94   });
95}); // /it
```

Liens utiles :

- Lien de l'application déployée : <https://wimf-ten.vercel.app/>
- Lien des repository :
 - Front : <https://github.com/AntoinePetit/front-wimf>
 - Back : <https://github.com/AntoinePetit/back-wimf>
- Lien du figma du projet : [Figma](#)