François-Xavier Bellan

Concepteur Développeur d'Applications

Activité 1 – Développer une application sécurisée

PROJET FIL ROUGE

Jalon 1 – Développer des sites web

"Saveurs Inédites"

2025-04-29

Remerciements

Je tiens à exprimer ma profonde gratitude à toutes les personnes qui ont contribué à la réalisation de ce projet. Leur expertise, leur soutien et leurs encouragements ont été essentiels tout au long de cette expérience enrichissante.

Merci à l'équipe pédagogique de 2ISA et à l'établissement AMIO pour la qualité de leurs enseignements, leur disponibilité et leur accompagnement. Leur implication m'a permis d'acquérir de nouvelles compétences et de gagner en confiance.

Un grand merci à la MDPH et au centre d'Osséja pour m'avoir orienté vers cette formation, une découverte qui s'est révélée passionnante. Enfin, merci à toutes les personnes, de près ou de loin, qui ont soutenu ce projet. Cette aventure m'a permis de renforcer mes compétences techniques, mon autonomie, mon organisation et mon esprit d'équipe.

Saveurs Inédites

Table des matières

Présentation personnelle

Je m'appelle François-Xavier Bellan et je suis en reconversion professionnelle dans le domaine du développement d'applications. Actuellement en formation CDA chez 2Isa, je mets à profit ma curiosité et mon envie constante d'apprendre dans un nouveau domaine qui allie rigueur et créativité.

Avant cela, j'ai eu un parcours diversifié, notamment dans le sport, la filière bois et le service à la personne. Ces expériences m'ont appris à m'investir pleinement, quelle que soit la mission. Le développement web représente aujourd'hui un véritable tournant et ce projet de fin d'activité est pour moi une opportunité concrète de mettre en pratique mes acquis.

Présentation de la formation CDA

La formation de Concepteur Développeur d'Applications dispensée par l'ESPR 2Isa (niveau Bac+3/4, reconnue par le Ministère du Travail) se déroule sur 18 mois. Elle est articulée autour de trois grandes activités, complétées par une période d'application en entreprise.

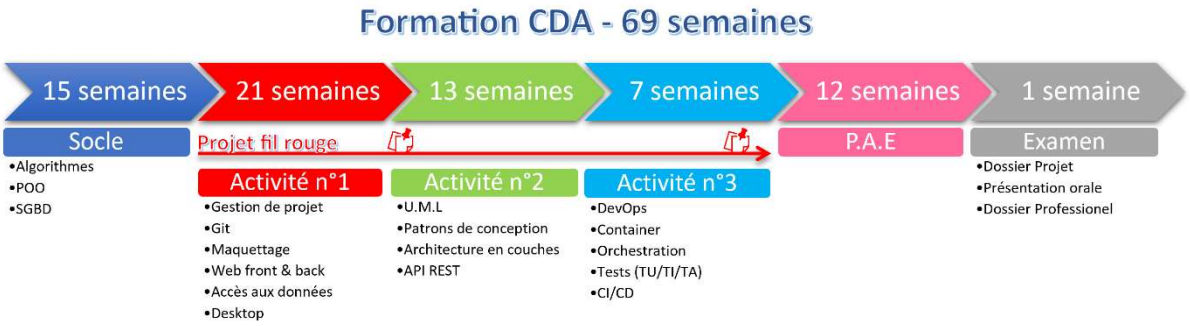
L'activité 1 : est dédiée au développement d'applications sécurisées. Les apprenants y développent les compétences professionnelles suivantes :

Installer et configurer son environnement de travail en fonction du projet

1. Développer des interfaces utilisateur
2. Développer des composants métier
3. Contribuer à la gestion d'un projet informatique

Les technologies couvertes incluent Visual Studio, Git, HTML/CSS/JavaScript, C#, .NET, Winform, SQL, SQL Server, PHP/Laravel ou ASP.Net, ainsi que les méthodologies de gestion de projet (cycle en V, agilité, Scrum, Kanban).

Cette première activité représente 21 semaines sur les 69 que compte la formation CDA.



L'activité 2 : Concevoir et développer une application sécurisée organisée en couches

1. Analyser les besoins et maquetter une application
2. Définir l'architecture logicielle d'une application
3. Concevoir et mettre en place une base de données relationnelle
4. Développer des composants d'accès aux données SQL et NoSQL

L'activité 3 : Préparer le déploiement d'une application sécurisée

1. Préparer et exécuter les plans de tests d'une application
2. Préparer et documenter le déploiement d'une application
3. Contribuer à la mise en production dans une démarche DevOps

Le projet Fil rouge s'inscrit dans le cadre de la formation CDA. Conçu par l'équipe pédagogique, il repose sur une mise en situation professionnelle visant à permettre aux étudiants d'appliquer les compétences acquises tout au long de leur apprentissage.

Généralités du projet Fil Rouge – "Saveurs Inédites"

1. Contexte et objectifs

Ce projet s'inscrit dans une démarche de formation et d'apprentissage du développement web. Il est réalisé dans un environnement simulé où les participants jouent le rôle de développeurs travaillant pour une entreprise fictive, "*Saveurs Inédites*", spécialisée dans la création et le partage de recettes culinaires innovantes.

L'objectif est de développer une application web dynamique et interactive complète en mode agile, avec des échanges réguliers et une approche itérative, permettant aux utilisateurs de rechercher, consulter, publier et commenter des recettes dans un cadre communautaire.

Le site doit ainsi offrir la possibilité de rechercher des recettes, de partager ses propres créations culinaires, et d'échanger avec les autres membres de la communauté. Une attention particulière est portée à la fluidité de l'expérience utilisateur, à l'intégration de technologies modernes, ainsi qu'au respect des bonnes pratiques en matière de sécurité et de performance.

2. Objectifs pédagogiques et techniques

Ce projet vise à offrir une expérience complète et formatrice du développement web. Les participants sont amenés à maîtriser l'ensemble du cycle de développement, aussi bien côté backend que frontend. Ils mettent en pratique les bonnes pratiques du domaine, notamment en matière de sécurité, de performance et d'accessibilité.

Le projet inclut également l'implémentation d'un système de rôles pour la gestion des utilisateurs, ainsi que le développement de fonctionnalités de recherche avancée. L'interface de l'application se veut moderne, ergonomique et entièrement responsive, afin de garantir une utilisation fluide sur tous les types de supports.

Liste des compétences du référentiel mobilisées

Le projet "Fil Rouge" mobilise plusieurs compétences issues du référentiel du développement web, en suivant un ordre cohérent aligné sur les différentes étapes du cycle de développement.

Dans un premier temps, les participants sont amenés à installer et configurer leur environnement de travail en fonction des besoins du projet. Cela inclut l'installation des outils nécessaires tels que Visual Studio pour le développement en ASP.NET Core MVC, PostgreSQL pour la gestion de la base de données, ainsi que Git et GitHub pour assurer le versioning du code et faciliter la collaboration en équipe.

La configuration du projet repose ensuite sur la mise en place d'une structure conforme à l'architecture MVC. Les participants définissent également le contexte de base de données à l'aide d'Entity Framework Core, posant ainsi les fondations techniques nécessaires au bon déroulement du développement.

La deuxième étape du projet consiste à développer des interfaces utilisateur à la fois intuitives, modernes et adaptées à tous les types d'écrans.

La conception des pages web s'appuie sur les technologies HTML5 et CSS3 afin de garantir une mise en page responsive et ergonomique. Des pages essentielles comme l'accueil, la recherche ou encore les détails d'une recette sont intégrés avec soin. Les recettes sont présentées de manière dynamique, sous forme de cartes interactives qui facilitent la navigation.

Les vues Razor sont utilisées pour générer du contenu dynamique en fonction des données stockées dans la base. Elles permettent notamment d'afficher les recettes disponibles, mais aussi d'intégrer des formulaires dédiés à l'inscription des utilisateurs ou à la publication de nouvelles recettes.

Enfin, l'interaction avec l'utilisateur est enrichie grâce à JavaScript, qui permet d'afficher les résultats de recherche en temps réel et de gérer divers événements tels que la soumission de formulaires ou la validation des champs saisis.

La troisième phase du projet est consacrée au développement des composants métier, essentiels au bon fonctionnement de l'application.

Le backend est développé en ASP.NET Core MVC. Les participants mettent en place les contrôleurs nécessaires à la gestion des différentes entités du projet, telles que les recettes, les utilisateurs et les commentaires. Ils conçoivent également les modèles correspondants (Recette, Utilisateur, Avis) et définissent les relations entre ces entités à l'aide d'Entity Framework Core.

Un système de gestion des recettes a été mis en place afin d'en permettre une utilisation souple et évolutive. Il repose sur des mécanismes de communication sécurisés, offrant la possibilité d'ajouter, de modifier et de supprimer des contenus de manière encadrée et fiable. La gestion des utilisateurs est renforcée par un système d'authentification et de rôles, garantissant un contrôle d'accès adapté aux différentes fonctionnalités de l'application.

La sécurisation des données occupe une place importante dans cette étape. Des requêtes paramétrées sont utilisées afin de prévenir les injections SQL, et des protections contre les attaques CSRF sont mises en place. L'authentification des utilisateurs est assurée de manière sécurisée grâce au chiffrement des mots de passe avec l'algorithme BCrypt.

La quatrième étape du projet vise à développer les compétences liées à la gestion d'un projet informatique, en s'appuyant sur une méthodologie Agile.

Le projet est planifié selon une approche itérative, avec un découpage en jalons et en tâches clairement définies. Le suivi de l'avancement se fait à l'aide de l'outil Trello, permettant une visualisation claire des étapes en cours, à venir ou terminées, et favorisant une organisation collaborative.

Par ailleurs, la gestion du versioning et la collaboration entre les membres de l'équipe s'appuient sur Git et GitHub. L'utilisation des branches Git, couplée à un

système de pull requests, permet de suivre précisément les modifications apportées au code tout en garantissant sa stabilité et sa cohérence tout au long du développement.

Expression du besoin

1. Objectifs

Le projet a pour objectif de créer une plateforme en ligne dédiée aux passionnés de cuisine, leur permettant de consulter, partager et interagir autour de recettes variées. Une attention particulière est portée à l'interaction entre les utilisateurs, à l'intuitivité de la navigation ainsi qu'à l'accessibilité de l'interface.

2. Utilisateurs ciblés

La plateforme s'adresse à deux types d'utilisateurs. D'une part, les utilisateurs authentifiés, qui auront la possibilité d'ajouter et de modifier des recettes, de laisser des notes ou des commentaires, et de gérer une liste de recettes favorites.

D'autre part, les administrateurs, chargés de la modération, de la gestion des contenus publiés et de la supervision des comptes utilisateurs.

3. Fonctionnalités principales

Le site proposera une page d'accueil mettant en avant des recettes sélectionnées.

Une fonctionnalité de recherche avancée permettra de filtrer les résultats selon plusieurs critères, tels que les mots-clés, les ingrédients ou le niveau de difficulté.

Chaque recette sera présentée de manière détaillée, avec la possibilité pour les utilisateurs de les créer ou de les modifier via leur espace personnel.

Les administrateurs disposeront d'outils dédiés à la modération et à la gestion du site.

4. Contraintes et attentes

Le respect des normes de sécurité est essentiel, notamment à travers une authentification fiable, la prévention des injections SQL et des failles CSRF, ainsi que la conformité au RGPD.

L'accessibilité sera assurée grâce à un design responsive, compatible avec tous les types d'écrans.

Une attention particulière sera portée à la performance de l'application, en garantissant une navigation fluide et un affichage rapide.

Enfin, l'interface se devra d'être claire, moderne et ergonomique, afin d'offrir une expérience utilisateur agréable et cohérente.

Gestion du projet

Le développement du projet s'appuie sur une méthodologie agile, favorisant la flexibilité, la collaboration et un suivi régulier de la progression à travers un découpage par jalons.

1. Méthodologie

Le projet adopte une approche itérative et incrémentale, avec des livraisons progressives permettant d'intégrer les retours au fil du développement. Cette méthode offre une grande adaptabilité, autorisant l'ajustement des fonctionnalités selon les besoins ou les remarques formulées. La collaboration entre les membres de l'équipe est encouragée par des échanges fréquents, visant à améliorer continuellement le produit.

2. Planification

La planification s'organise autour de deux jalons principaux.

Le premier jalon, prévu du 20 janvier au 2 mai 2025, vise la livraison d'une première version fonctionnelle de la plateforme.

Au cours de cette phase, plusieurs travaux ont été réalisés :

– La mise en place de l'architecture MVC avec ASP.NET Core.

– La création des pages essentielles, telles que la page d'accueil, la recherche, les détails d'une recette et l'ajout de nouvelles recettes.

– Le développement du moteur de recherche avancé.

– L'implémentation du système d'authentification pour les utilisateurs.

3. Outils utilisés

Pour assurer le bon déroulement du projet, plusieurs outils ont été mobilisés. Trello a été utilisé pour la gestion et le suivi des tâches. GitHub a permis de gérer le versioning du code et de faciliter la collaboration entre les développeurs. Enfin, la communication au sein de l'équipe a été assurée via Microsoft Teams, favorisant les échanges en temps réel et la coordination.

Présentation de Trello

Trello est un outil de gestion de projet visuel fondé sur la méthode Kanban. Il permet de représenter l'avancement d'un projet à l'aide de cartes organisées en colonnes, qui correspondent aux différentes étapes du processus (par exemple : "À faire", "En cours", "Terminé"). Intuitif et collaboratif, Trello facilite la planification, le suivi et la répartition des tâches, que ce soit en équipe ou en travail individuel.

Chaque carte Trello peut inclure divers éléments permettant de détailler et de structurer le travail : une description de la tâche, une checklist pour en suivre les sous-étapes, une date d'échéance, des fichiers joints, des commentaires, ainsi que des étiquettes colorées pour classer ou prioriser les actions.

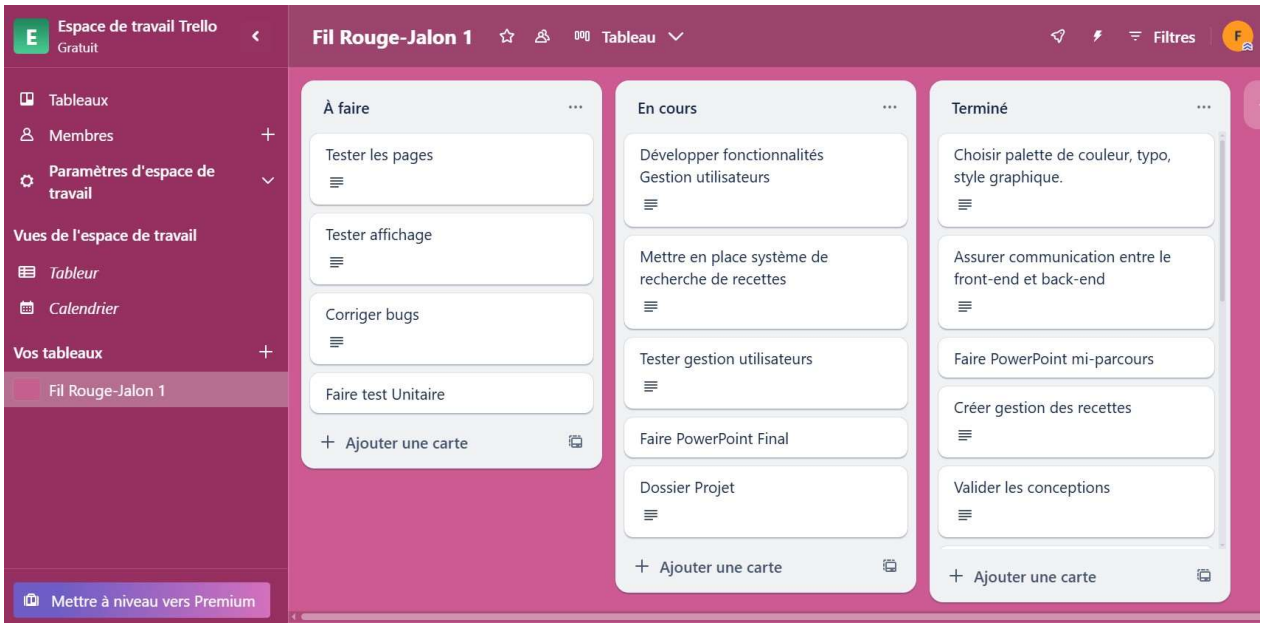
Utilisation de Trello dans le projet *Saveurs Inédites*

Dans le cadre du projet *Saveurs Inédites*, Trello a été utilisé comme principal outil d'organisation et de suivi. Il a permis de structurer les différentes étapes du projet, de l'analyse initiale à la conception, en passant par le développement, les tests.

Les fonctionnalités à développer (comme l'authentification, la gestion des recettes, le moteur de recherche ou encore l'espace administrateur) ont été intégrées sous forme de cartes, chacune représentant une fonctionnalité ou un sous-objectif spécifique.

Les colonnes "To Do", "En cours" et "Fait" ont permis de suivre l'état d'avancement des tâches, tout en assurant une priorisation claire. Des checklists ont été utilisées au sein de chaque carte pour découper les tâches techniques en étapes plus précises et faciliter leur suivi.

Trello s'est révélé particulièrement utile pour garder une vue d'ensemble sur le projet, coordonner efficacement le travail, et s'adapter aux ajustements requis par la méthode agile. Cet outil a donc joué un rôle central dans la gestion du temps, des livrables et dans la simulation d'un contexte professionnel de développement collaboratif.



4. Gestion des risques

Au cours du projet, plusieurs risques potentiels ont été identifiés, pouvant impacter la qualité ou le bon déroulement du développement.

Parmi les principaux risques figurent les problèmes de compatibilité, notamment entre différents navigateurs ou supports mobiles, les failles de sécurité telles que les injections SQL ou les attaques CSRF, ainsi que d'éventuels retards dans l'avancement du développement.

Afin de limiter ces risques, plusieurs mesures ont été mises en place. Des vérifications systématiques ont été réalisées à chaque jalon pour s'assurer de la conformité et du bon fonctionnement des fonctionnalités développées. Des tests manuels réguliers ont permis de détecter les anomalies ou incohérences en amont. Enfin, un effort constant a été fourni pour appliquer les bonnes pratiques de sécurité du web, en veillant à la robustesse du code, à la sécurisation des échanges de données et à la protection des utilisateurs.

Analyse du besoin et spécifications fonctionnelles

1. Analyse du besoin

De nombreuses plateformes culinaires en ligne souffrent d'un manque d'intuitivité, d'interactivité ou de personnalisation. Ce projet vise donc à proposer une alternative ergonomique, interactive et responsive, afin de faciliter la recherche, le partage et la notation de recettes.

Pour l'utilisateur, les objectifs sont de pouvoir rechercher des recettes en utilisant des filtres avancés, d'ajouter, modifier ou supprimer ses propres recettes, mais aussi de noter, commenter et gérer ses recettes favorites.

Du côté de l'administrateur, les objectifs consistent à gérer les recettes et les utilisateurs, ainsi qu'à modérer les contenus jugés inappropriés.

2. Spécifications fonctionnelles

Le site se compose de plusieurs pages principales et intègre des fonctionnalités clés.

La page d'accueil met en avant les recettes populaires et les nouveautés. Elle offre également un accès rapide aux différentes catégories de recettes ainsi qu'une barre de recherche directe.

La fonctionnalité de recherche permet d'explorer les recettes grâce à des filtres avancés tels que le mot-clé, les ingrédients, le temps de préparation, le niveau de difficulté ou encore les régimes alimentaires. Les résultats s'affichent sous forme de cartes incluant une image, une note, le temps de préparation et d'autres informations pertinentes.

La page de détail d'une recette fournit toutes les informations nécessaires, comme les ingrédients, les instructions, le temps de préparation et la difficulté. L'utilisateur peut interagir avec cette page en laissant une note, un commentaire ou en ajoutant la recette à ses favoris.

Lorsqu'un utilisateur est connecté, il peut ajouter ou modifier une recette via un formulaire comportant une validation des champs tels que le titre, l'image, les ingrédients et les étapes. Il peut également modifier ou supprimer ses recettes déjà publiées.

L'espace utilisateur permet de s'inscrire et de se connecter de manière sécurisée. Une fois connecté, l'utilisateur peut gérer ses recettes publiées ainsi que ses recettes favorites.

Un espace d'administration est également prévu, avec un tableau de bord permettant à l'administrateur de modérer les recettes et de gérer les comptes utilisateurs.

3. Contraintes

Sur le plan de la sécurité, l'authentification est assurée grâce à ASP.NET Identity. Des protections sont mises en place contre les injections SQL et les failles CSRF à l'aide de requêtes paramétrées et de l'encodage des données. Le site respecte

également le RGPD, notamment pour la gestion des données personnelles et des cookies.

En ce qui concerne l'accessibilité et les performances, le site adopte un design responsif pour s'adapter aux différents supports (ordinateur, mobile, tablette) et veille à un chargement rapide des contenus.

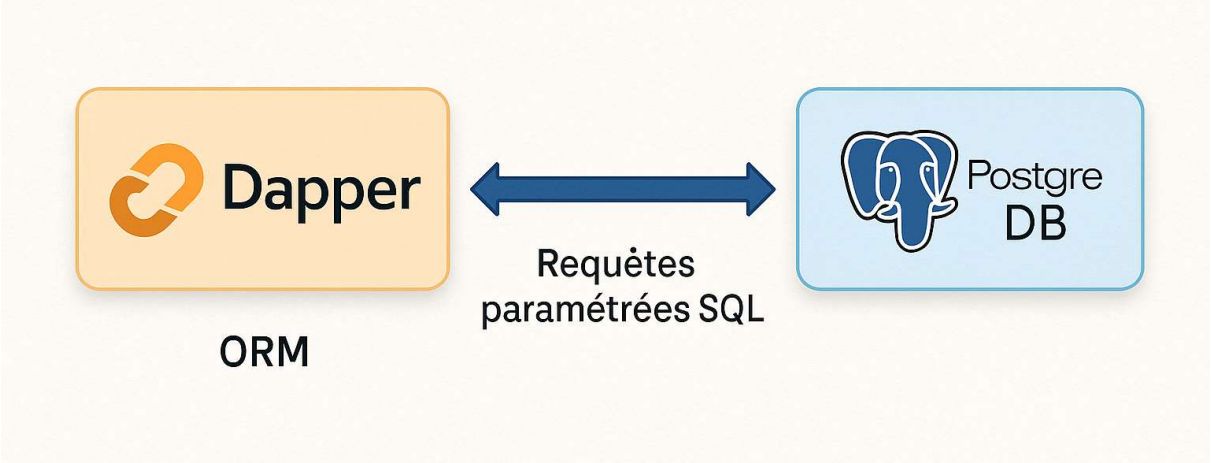
Spécifications techniques

Le projet repose sur une architecture claire, modulaire et évolutive. Il intègre des technologies modernes, sélectionnées pour garantir la performance, la sécurité et la facilité de maintenance de l'application.

La partie backend est développée en C# en s'appuyant sur le Framework ASP.NET Core MVC. Le frontend est conçu à l'aide des langages HTML, CSS et JavaScript, afin d'assurer une interface réactive et accessible depuis tous les types de supports.

Pour interagir avec la base de données, le projet s'appuie sur Dapper, un ORM léger offrant un excellent compromis entre performance et maîtrise des requêtes SQL. Cette solution permet d'exécuter des requêtes de manière efficace tout en conservant un contrôle fin sur les opérations réalisées.

Grâce à Dapper, les entités de la base sont représentées sous forme d'objets C#, ce qui simplifie grandement leur manipulation au sein de l'application et facilite l'écriture de requêtes structurées et maintenables.



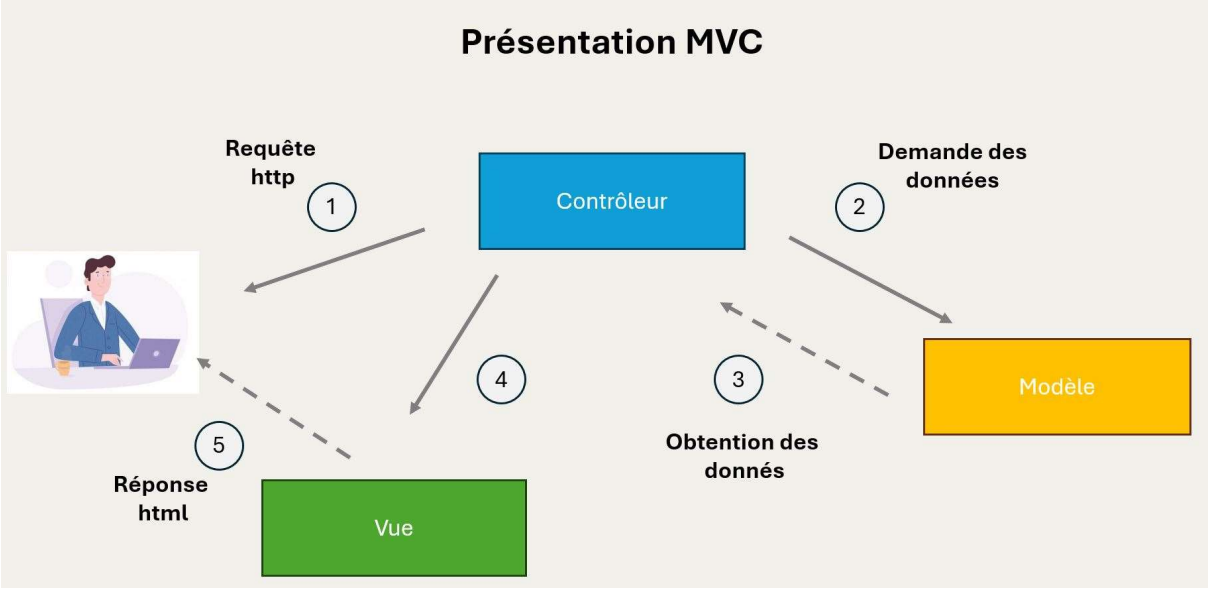
Les données sont stockées dans une base PostgreSQL, reconnue pour sa robustesse, sa compatibilité multiplateforme et ses performances.

L'authentification des utilisateurs est gérée à l'aide du système ASP.NET Identity, renforcé par le chiffrement des mots de passe via l'algorithme BCrypt, ce qui garantit un haut niveau de sécurité pour les données personnelles.

Enfin, le versioning du projet est assuré à l'aide de Git, avec un hébergement du code source sur GitHub, facilitant ainsi la collaboration, le suivi des évolutions et la gestion du cycle de vie du développement.

1. Structure du backend

Le projet *Saveurs Inédites* s'organise selon une architecture MVC bien définie.



Le dossier Controllers regroupe les différents contrôleurs chargés de gérer la logique des recettes, des utilisateurs, des avis, etc. Le dossier Models contient les entités représentant les objets du domaine, tels que les recettes, les utilisateurs ou les commentaires. Les interfaces utilisateur sont regroupées dans le dossier Views, qui contient les vues Razor au format .cshtml.

La couche d'accès aux données repose sur le dossier Data, dans lequel se trouve le fichier ApplicationDbContext.cs, jouant le rôle de passerelle entre l'application

et la base de données. La logique métier est centralisée dans le dossier Services, permettant de séparer clairement les traitements applicatifs.

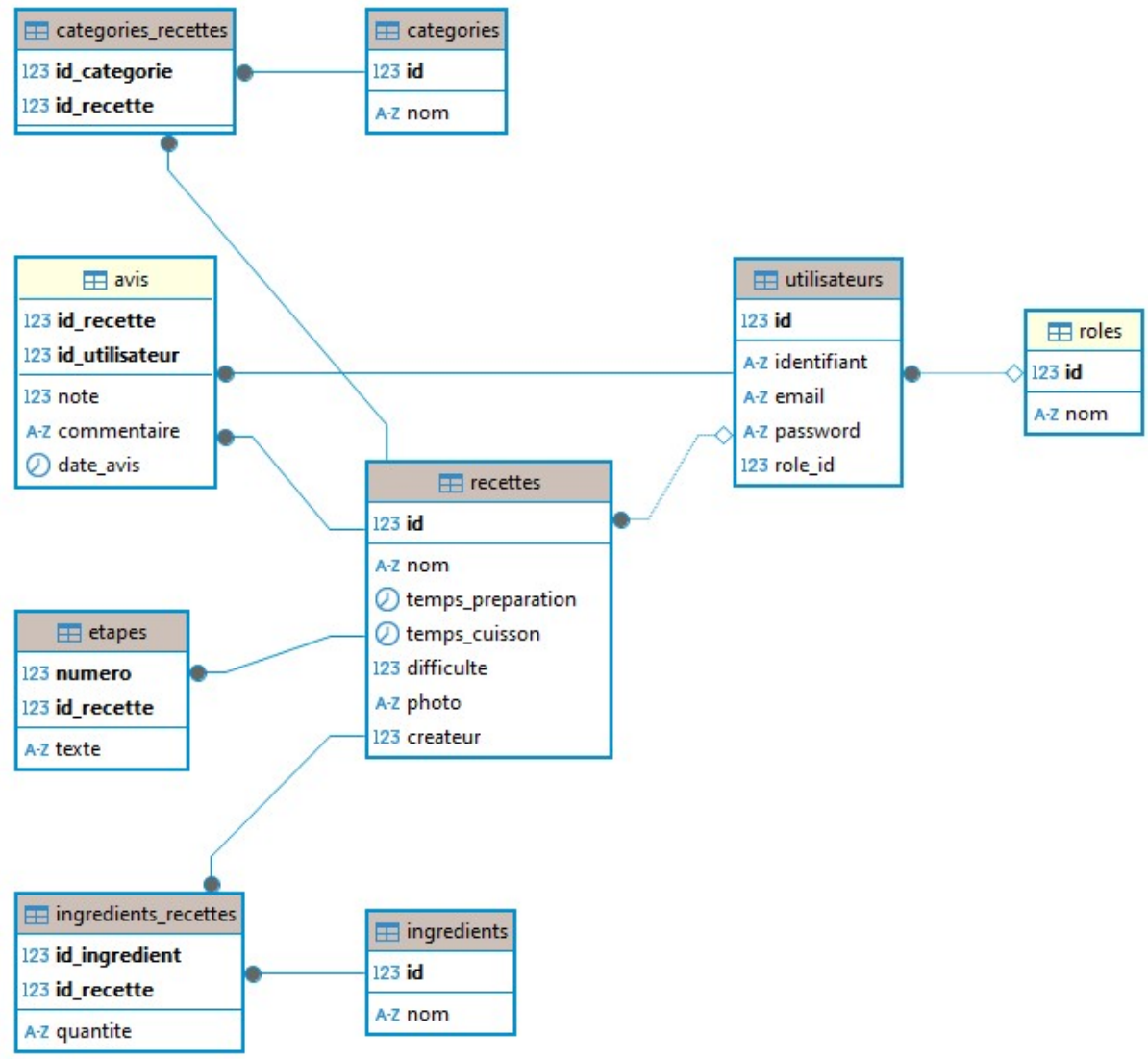
Les ressources statiques telles que les feuilles de style, les fichiers JavaScript et les images sont stockées dans le dossier wwwroot. Enfin, les fichiers Startup.cs et Program.cs assurent respectivement la configuration générale de l'application et définissent son point d'entrée.

2. Mise en place de la base de données et du modèle

Avant de pouvoir intégrer les modèles dans le cadre de l'architecture MVC, il est essentiel de disposer d'une base de données sur laquelle s'appuyer. Pour le projet *Saveurs Inédites*, nous utiliserons une base de données destinée à gérer les recettes.

Cette base de données permettra de modéliser plusieurs entités comme les recettes, les utilisateurs, les catégories et les ingrédients... Ces éléments constitueront la base du Model (le « M » de MVC), qui jouera un rôle central dans la structuration des données et leur interaction avec les autres composants de l'application.

Un schéma représentant la structure de cette base de données sera présenté ci-dessous afin d'illustrer les relations entre les différentes entités.



3. Modèle de données et entités

Les scripts SQL fournis en annexe permettent de générer l'ensemble des tables nécessaires ainsi que des données d'exemple pour initialiser la base. Le cœur du modèle repose sur plusieurs entités principales : les utilisateurs, les recettes, les rôles et les avis.

La table des utilisateurs contient les champs essentiels tels que l'identifiant, l'adresse e-mail, le mot de passe et le rôle de l'utilisateur. Les rôles sont définis dans une table distincte, permettant une gestion souple des droits d'accès à l'application. Les recettes, au centre de l'application, possèdent un identifiant unique, un nom, des informations de préparation et de cuisson (sous forme de durées), un niveau de difficulté, une photo et une référence vers le créateur.

Les avis sont associés à une recette et à un utilisateur. Chaque avis contient un commentaire, une date de publication, et permet d'enrichir l'interaction entre les membres de la plateforme.

Voici un extrait simplifié du modèle Recette en C# :

```
1   // Représente le modèle de données pour une recette
2   public class Recette
3   {
4       // Identifiant unique de la recette (clé primaire)
5       public int id { get; set; }
6
7       // Nom de la recette (obligatoire, initialisé à une chaîne vide)
8       public string nom { get; set; } = string.Empty;
9
10      // Durée de préparation de la recette (exclut le temps de cuisson)
11      public TimeSpan temps_preparation { get; set; }
12
13      // Durée de cuisson de la recette, par défaut à zéro
14      public TimeSpan temps_cuisson { get; set; } = TimeSpan.Zero;
15
16      // Niveau de difficulté (ex. : 1 = facile, 2 = moyen, 3 = difficile)
17      public int difficulte { get; set; }
18
19      // URL ou chemin de la photo associée à la recette (facultatif)
20      public string? photo { get; set; }
21  }
```

4. Routes principales

Les routes définies dans l'application suivent une structure RESTful claire et cohérente. La route GET/recettes permet d'obtenir la liste complète des recettes disponibles sur le site. Pour consulter les détails d'une recette spécifique, la route GET /recettes/detail/{id} est utilisée.

L'ajout d'une nouvelle recette, réservé aux utilisateurs authentifiés, se fait via la route POST /recettes/nouveau. Cette structure garantit une navigation simple et prévisible pour l'ensemble des utilisateurs.

5. Sécurité et authentification

Plusieurs mécanismes de sécurité ont été mis en place pour protéger les données et les interactions des utilisateurs. L'authentification repose sur ASP.NET Identity, avec un système de cookies pour maintenir les sessions. Chaque formulaire intégré dans l'application inclut un token unique, permettant de se prémunir contre les attaques de type CSRF.

Toutes les requêtes SQL sont construites de manière paramétrée, afin de prévenir les tentatives d'injection. Un système de rôles différencie les utilisateurs classiques des administrateurs, limitant ainsi l'accès à certaines fonctionnalités sensibles.

Enfin, les mots de passe sont stockés de manière sécurisée grâce à l'algorithme BCrypt, assurant une protection renforcée des informations personnelles des utilisateurs, notamment lors de l'authentification.

Fonctionnalité représentative : la recherche par nom

L'un des éléments centraux de l'application est son moteur de recherche, conçu pour offrir à l'utilisateur une expérience fluide, rapide et adaptée à ses préférences culinaires. Cette fonctionnalité joue un rôle essentiel dans la navigation et l'exploration du contenu proposé.

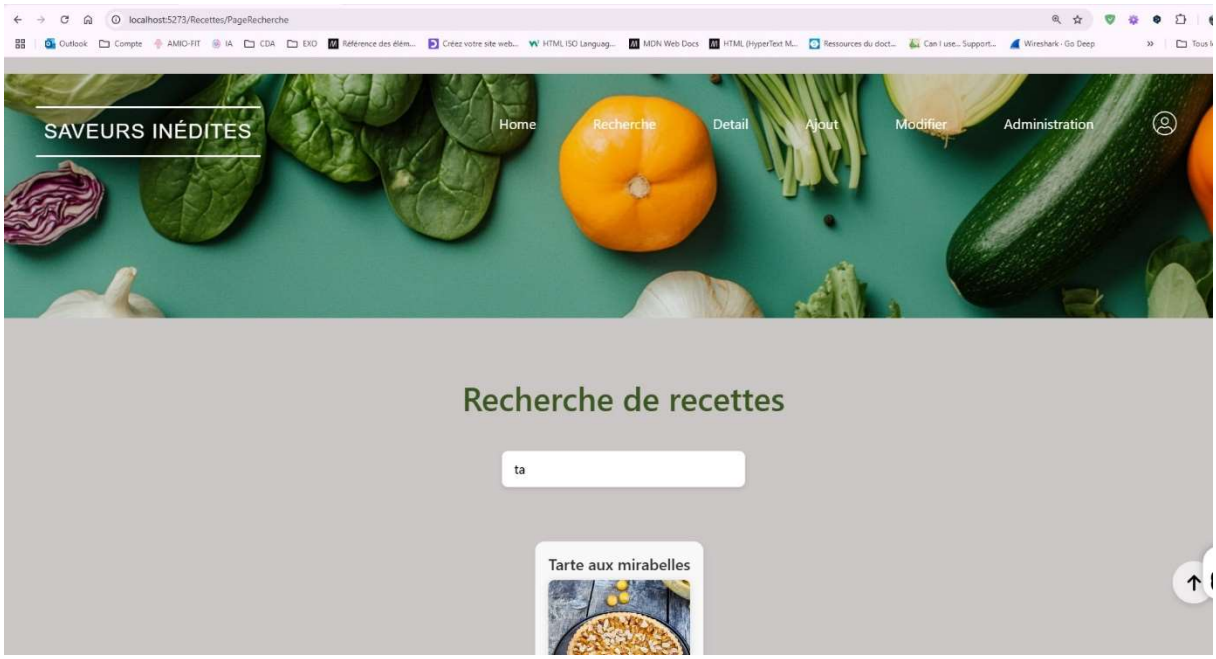
L'objectif principal est de permettre à l'utilisateur de trouver facilement des recettes correspondant à ses envies ou à des critères spécifiques. Qu'il s'agisse d'un ingrédient particulier, d'un temps de préparation limité, d'un niveau de difficulté ou encore d'un régime alimentaire, le moteur de recherche propose des filtres avancés pour affiner les résultats.

Les résultats sont présentés sous forme de cartes visuelles, mettant en avant l'image de la recette ainsi que son titre. Ce format permet une consultation rapide et agréable, tout en facilitant la comparaison entre plusieurs propositions.

1. L’utilisateur clique sur “Recherche” dans le menu

Ce que fait l’utilisateur :

Depuis la page d’accueil (Home), l’utilisateur connecté clique sur “Recherche” dans le menu de navigation, redirige vers la route /Recettes/PageRecherche.



Ce que fait l'application :

Cela déclenche un appel vers l'action suivante du contrôleur RecettesController :

```
614     // Attribut qui limite l'accès à l'action aux utilisateurs authentifiés
615     [Authorize]
616
617     // Action du contrôleur qui retourne la vue de la page de recherche
        0 références | François Xavier Bellan, il y a 18 jours | 1 auteur, 2 modifications
618     public IActionResult PageRecherche()
619     {
620         // Retourne la vue nommée "Recherche" (fichier Recherche.cshtml)
621         return View("Recherche");
622     }
```

Résultat :

La vue Recherche.cshtml est rendue dans le navigateur. Elle contient un champ de saisie et une zone vide où s'afficheront les recettes.

2. Affichage de la page "Recherche" (vue)

Voici ce que contient cette page côté client (HTML + JavaScript) :

```
9   <!-- Conteneur de la zone de recherche -->
10  <div class="zone-recherche">
11      <!-- Champ de saisie pour entrer le nom d'une recette. Il a un identifiant "recherche"
12      pour pouvoir être facilement ciblé en JavaScript -->
13      <input type="text" id="recherche" placeholder="Tapez un nom de recette..." autocomplete="off">
14      <!-- Le placeholder donne un indice à l'utilisateur sur ce qu'il doit saisir.
15      "autocomplete" désactive les suggestions automatiques du navigateur -->
16  </div>
17
18  <!-- Conteneur pour afficher les recettes recherchées. Les recettes seront insérées ici par JavaScript -->
19  <div id="recettes" class="grille-recettes"></div>
```

Ce champ permet à l'utilisateur de taper un nom de recette. En parallèle, le fichier JS contient une fonction recherche() qui va être appelée.

3. L'utilisateur tape un mot-clé (ex : "tarte")

Quand l'utilisateur tape quelque chose (ex. "tarte"), la fonction recherche() s'exécute :

```
// Déclaration de la fonction appelée lors d'une recherche
2 references
function recherche() {

    // Récupère la valeur saisie dans le champ de recherche ayant l'ID "recherche"
    let chaine = document.getElementById("recherche").value;

    // Envoie une requête HTTP GET vers l'URL "/recettes/recherche" avec la chaîne encodée en paramètre
    fetch("/recettes/recherche?recherche=" + encodeURIComponent(chaine))

        // Traite la réponse en la convertissant en JSON
        .then((reponse) => reponse.json())

        // Utilise les données JSON reçues pour mettre à jour l'affichage
        .then((json) => {

            // Sélectionne l'élément HTML qui contient les recettes (ayant l'ID "recettes")
            const conteneur = document.getElementById("recettes");

            // Vide le contenu actuel du conteneur pour afficher de nouveaux résultats
            conteneur.innerHTML = "";

            // Parcourt chaque recette dans le tableau JSON et appelle la fonction afficherRecette pour chaque élément
            json.forEach(afficherRecette);
        })

        // Gère les erreurs survenues lors de la requête ou du traitement des données
        .catch((erreur) => {
            console.error("Erreur lors de la récupération des recettes :", erreur);
        });
}
```

Ce que fait ce code :

Le code mis en place permet d'implémenter une recherche dynamique côté client. Lorsqu'un utilisateur saisit une valeur dans le champ de recherche, cette chaîne de caractères est d'abord récupérée par le script. Une requête HTTP de type GET est ensuite envoyée de manière asynchrone à l'URL /recettes/recherche, en ajoutant la valeur saisie sous forme de paramètre de requête (?recherche=tarte, par exemple).

Une fois la requête traitée par le serveur, une réponse au format JSON est renvoyée, contenant une liste d'objets représentant les recettes correspondantes. Le conteneur HTML prévu pour l'affichage des résultats est alors vidé afin d'éviter toute redondance. Enfin, pour chaque recette reçue, la fonction afficherRecette() est appelée. Celle-ci se charge d'insérer dynamiquement la recette dans la page, généralement sous forme de carte visuelle.

4. Passage côté serveur (Contrôleur)

L'appel Fetch arrive sur cette action du contrôleur :

```
620     // Attribut qui restreint l'accès à l'action aux utilisateurs authentifiés
621     [Authorize]
622
623     // Action du contrôleur qui répond à une requête de recherche
        0 références | François Xavier Bellan, il y a 18 jours | 1 auteur, 3 modifications
624     public IActionResult Recherche(string recherche)
625     {
626         // Si la chaîne de recherche est vide ou ne contient que des espaces, retourne une liste vide au format JSON
627         if (string.IsNullOrWhiteSpace(recherche))
628             return Json(new List<Recette>());
629
630         // Requête SQL : sélectionne toutes les recettes dont le nom correspond (partiellement, sans tenir compte de la casse) à la chaîne de
              recherche
631         string query = "SELECT * FROM Recettes WHERE lower(nom) LIKE @recherche";
632
633         // Déclare une liste qui contiendra les résultats des recettes trouvées
634         List<Recette> recettes;
635
636         // Ouvre une connexion à la base de données PostgreSQL
637         using (var connexion = new NpgsqlConnection(_connexionString))
638         {
639             // Exécute la requête SQL en injectant la valeur recherchée (en minuscules et entourée de %) pour faire une recherche partielle
640             recettes = connexion.Query<Recette>(query, new { recherche = $"%{recherche.ToLower()}%" }).ToList();
641         }
642
643         // Retourne la liste des recettes trouvées sous forme de JSON
644         return Json(recettes);
645     }
```

Étapes techniques :

Du point de vue technique, plusieurs étapes sont nécessaires pour traiter la requête de recherche côté serveur. Tout d'abord, le système vérifie que le champ de recherche n'est pas vide afin d'éviter des requêtes inutiles. Si une valeur est bien présente, une requête SQL est préparée dynamiquement en utilisant l'opérateur LIKE avec des jokers (%...%) pour rechercher les recettes dont le nom contient la chaîne saisie, par exemple %tarte%.

L'exécution de cette requête est assurée par l'ORM léger Dapper, qui permet de communiquer efficacement avec la base de données PostgreSQL tout en maintenant des performances optimales. Une fois les résultats obtenus, ceux-ci sont transformés en objets C# puis automatiquement sérialisés au format JSON. Cette réponse JSON est ensuite renvoyée au client pour affichage.

5. Le modèle Recette

Le JSON retourné correspond à la classe suivante :

```
// Déclare l'espace de noms (namespace) du projet pour organiser les classes
namespace SaveursInedites.Models
{
    // Déclare la classe Recette, qui représente une entité de recette
    18 références | 0 modification | 0 auteur, 0 modification
    public class Recette
    {
        // Identifiant unique de la recette (clé primaire)
        8 références | 0 modification | 0 auteur, 0 modification
        public int id { get; set; }

        // Nom de la recette (ne peut pas être nul, initialisé avec une chaîne vide par défaut)
        7 références | 0 modification | 0 auteur, 0 modification
        public string nom { get; set; } = string.Empty;

        // Chemin ou URL de la photo associée à la recette (peut être null)
        13 références | 0 modification | 0 auteur, 0 modification
        public string? photo { get; set; }
    }
}
```

6. Affichage dynamique des résultats (JavaScript)

Une fois les données reçues, chaque recette est affichée dynamiquement avec :

```
function afficherRecette(recette) {

    // Crée un élément <a> (lien) qui redirige vers la page de détail de la recette
    let lien = document.createElement("a");
    lien.href = "/Recettes/Detail/" + recette.id; // Lien vers la page de détail avec l'ID de la recette
    lien.style.textDecoration = "none";           // Supprime le soulignement du lien
    lien.style.color = "inherit";                 // Hérite de la couleur du texte environnant

    // Crée un conteneur <div> pour afficher la carte de la recette
    let div = document.createElement("div");
    div.classList.add("carte-recette");           // Ajoute une classe CSS pour le style

    // Crée un titre <h2> contenant le nom de la recette
    let nom = document.createElement("h2");
    nom.textContent = recette.nom;                // Définit le texte à partir du nom de la recette

    // Crée une image <img> représentant la recette
    let img = document.createElement("img");
    img.src = recette.photo;                      // Définit la source de l'image
    img.alt = recette.nom;                        // Texte alternatif pour l'accessibilité
    img.classList.add("image-recette");           // Classe CSS pour styliser l'image

    // Crée un paragraphe <p> contenant la description de la recette
    let description = document.createElement("p");
    description.textContent = recette.description; // Affiche la description

    // Ajoute le titre, l'image et la description à la carte (div)
    div.append(nom, img, description);

    // Insère la carte dans le lien cliquable
    lien.appendChild(div);

    // Ajoute le lien complet dans le conteneur principal des recettes (élément avec l'ID "recettes")
    document.getElementById("recettes").appendChild(lien);
}
```

Résultat :

Le résultat de cette recherche se manifeste par la génération dynamique d'une carte HTML pour chaque recette trouvée. Chaque carte met en avant les éléments visuels essentiels : une image représentative de la recette, son nom, ainsi qu'un court descriptif permettant à l'utilisateur d'en saisir rapidement l'essence.

Ces cartes ne sont pas uniquement informatives ; elles sont également interactives. Chacune d'elles est cliquable et redirige l'utilisateur vers la page de détails correspondante, accessible via l'URL /Recettes/Detail/{id}, où {id} représente l'identifiant unique de la recette sélectionnée.

7. Résumé des étapes

Le processus de recherche s'enclenche dès que l'utilisateur clique sur le bouton "Recherche". Cette action déclenche l'exécution de la méthode PageRecherche() dans le contrôleur, laquelle affiche la vue correspondante Recherche.cshtml. Cette vue contient un champ de saisie pour le texte ainsi qu'un script JavaScript chargé de gérer la recherche côté client.

Lorsque l'utilisateur commence à taper une requête, la fonction JavaScript recherche() est appelée. Cette fonction utilise fetch() pour envoyer une requête asynchrone vers l'URL /recettes/recherche, en y ajoutant la chaîne recherchée en paramètre.

Côté serveur, l'action Recherche() du contrôleur ASP.NET reçoit la requête, prépare une instruction SQL avec un filtre LIKE appliqué sur la base de données PostgreSQL, et renvoie en réponse une liste de recettes au format JSON.

Enfin, de retour sur le client, le script JavaScript récupère cette liste et génère dynamiquement des cartes HTML pour chaque recette trouvée. Ces cartes sont ensuite injectées dans la page, offrant à l'utilisateur un affichage immédiat et interactif des résultats.

Présentation de deux types d'attaques web

1. Injection SQL

A. Définition

Une injection SQL est une attaque où un utilisateur malveillant injecte du code SQL dans une requête, généralement via un champ de formulaire ou une URL. Cela permet de contourner l'authentification, consulter ou modifier des données, voire supprimer des tables.

B. Exemple de code vulnérable (C# / ASP.NET)

```
// Action du contrôleur qui traite la connexion d'un utilisateur
public IActionResult Connexion(string username, string password)
{
    // Construction de la requête SQL pour vérifier si un utilisateur avec le login et mot de passe
    // spécifiés existe dans la base de données
    string query = $"SELECT * FROM Utilisateurs WHERE login = '{username}' AND mot_de_passe = '{password}'";

    // Ouverture d'une connexion à la base de données PostgreSQL
    using (var connexion = new NpgsqlConnection(_connexionString))
    {
        // Exécution de la requête et récupération du premier utilisateur correspondant, ou null si aucun résultat
        var utilisateur = connexion.QueryFirstOrDefault<Utilisateur>(query);

        // Si un utilisateur a été trouvé, redirige vers la page d'accueil (Index de Home)
        if (utilisateur != null)
            return RedirectToAction("Index", "Home");

        // Si aucun utilisateur n'a été trouvé, retourne la vue "ErreurConnexion" pour afficher un message d'erreur
        return View("ErreurConnexion");
    }
}
```

C. Exploitation

Un pirate entre cette valeur dans le champ mot de passe :

```
// Tentative d'injection SQL où la condition '1'='1' est toujours vraie,
//permettant à l'attaquant de contourner l'authentification.
' OR '1'='1
```

La requête SQL devient :

```
//Cette requête tente une injection SQL où la condition '1'='1' est toujours vraie,
//permettant à l'attaquant de contourner l'authentification
//et de récupérer tous les utilisateurs de la base de données, sans vérifier le mot de passe réel.
SELECT * FROM Utilisateurs WHERE login = 'admin' AND mot_de_passe = '' OR '1'='1';
```

Résultat : la condition OR '1'='1' est toujours vraie → l'utilisateur est connecté sans connaître le mot de passe.

D. Solution sécurisée (parade) : Requête paramétrée

```
// Action du contrôleur qui gère la tentative de connexion d'un utilisateur
public IActionResult Connexion(string username, string password)
{
    // Requête SQL paramétrée pour éviter les injections SQL
    string query = "SELECT * FROM Utilisateurs WHERE login = @username AND mot_de_passe = @password";

    // Ouverture d'une connexion à la base PostgreSQL à l'aide de la chaîne de connexion
    using (var connexion = new NpgsqlConnection(_connexionString))
    {
        // Exécution de la requête avec les paramètres sécurisés (pas d'interpolation directe)
        var utilisateur = connexion.QueryFirstOrDefault<Utilisateur>(
            query,                          // Requête SQL
            new { username, password }      // Paramètres fournis par l'utilisateur
        );

        // Si un utilisateur correspondant a été trouvé, redirige vers la page d'accueil
        if (utilisateur != null)
            return RedirectToAction("Index", "Home");

        // Sinon, retourne une vue d'erreur de connexion
        return View("ErreurConnexion");
    }
}
```

E. Pourquoi c'est sécurisé ?

- Les paramètres sont transmis séparément de la requête SQL.
- Cela empêche tout code injecté d'être interprété comme une commande SQL.
- Les bibliothèques comme Dapper, ADO.NET, Entity Framework ou PDO (en PHP) permettent ces requêtes paramétrées.

2. CSRF (Cross-Site Request Forgery)

A. Qu'est-ce qu'une attaque CSRF ?

Une attaque CSRF consiste à exploiter la session authentifiée d'un utilisateur sur un site web pour exécuter des actions malveillantes à son insu. L'attaquant incite l'utilisateur à faire une requête (ex : transfert d'argent, modification d'email, etc.) en profitant du fait que l'utilisateur est connecté au site cible.

Exemple : Vous êtes connecté à votre banque. Vous cliquez sur un lien dans un email piégé et, sans le savoir, vous transférez de l'argent vers le compte de l'attaquant.

B. Exemple de code vulnérable (ASP.NET MVC)

Prenons une action qui permet de changer l'adresse email d'un utilisateur connecté :

a. Contrôleur (vulnérable)

```
// Attribut indiquant que cette action répond uniquement aux requêtes HTTP POST
[HttpPost]

// Action du contrôleur permettant à l'utilisateur de mettre à jour son adresse email
public ActionResult UpdateEmail(string newEmail)
{
    // Récupère l'identifiant de l'utilisateur actuellement connecté depuis la session
    var userId = Session["UserId"];

    // Recherche l'utilisateur correspondant dans la base de données et met à jour son adresse email
    db.Users.Find(userId).Email = newEmail;

    // Enregistre les modifications dans la base de données
    db.SaveChanges();

    // Redirige l'utilisateur vers sa page de profil après la mise à jour
    return RedirectToAction("Profile");
}
```

b. Vue correspondante

```
// Formulaire HTML qui envoie une requête POST à l'action UpdateEmail du contrôleur Account
<form method="post" action="/Account/UpdateEmail">

    //Champ de saisie pour la nouvelle adresse email, avec validation HTML5 intégrée (type="email")
    <input type="email" name="newEmail" />

    //Bouton pour soumettre le formulaire
    <button type="submit">Mettre à jour</button>
</form>
```

C. Exploitation CSRF

L'attaquant peut créer une page externe contenant ce code HTML piégé :

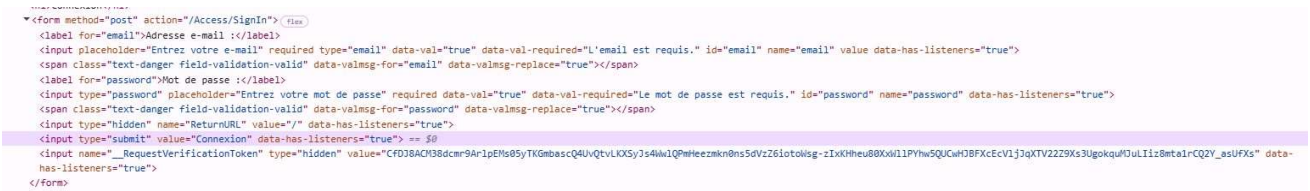
```html
//Formulaire HTML malveillant qui envoie une requête POST à l'action UpdateEmail de ton site
<form method="post" action="https://votresite.com/Account/UpdateEmail">

    //Champ caché contenant une fausse adresse email, injectée sans que l'utilisateur ne le voie
    <input type="hidden" name="newEmail" value="attacker@example.com" />

    //Bouton trompeur qui incite l'utilisateur à cliquer, sous un faux prétexte
    <input type="submit" value="Clique ici pour gagner un iPhone !" />
</form>

<script>
    // Script JavaScript qui soumet automatiquement le formulaire sans interaction de l'utilisateur
    document.forms[0].submit();
</script>
```

Si la victime est connectée à votre site, cette requête POST sera acceptée car les cookies d'authentification sont automatiquement envoyés.

D. Parade CSRF : AntiForgeryToken

a. Ajout d'un token CSRF dans la vue

Dans la vue Razor :



b. Vérification du token dans le contrôleur

Dans le contrôleur :

```
// Attribut indiquant que cette action accepte uniquement les requêtes HTTP POST
[HttpPost]

// Attribut qui active la validation du jeton anti-CSRF envoyé depuis la vue
[ValidateAntiForgeryToken]

// Action du contrôleur qui permet de mettre à jour l'adresse email de l'utilisateur connecté
public ActionResult UpdateEmail(string newEmail)
{
    // Récupère l'identifiant de l'utilisateur en session (assumé ici comme étant déjà authentifié)
    var userId = Session["UserId"];

    // Recherche de l'utilisateur dans la base de données et mise à jour de son adresse email
    db.Users.Find(userId).Email = newEmail;

    // Enregistrement des modifications dans la base
    db.SaveChanges();

    // Redirection vers la page de profil après mise à jour
    return RedirectToAction("Profile");
}
```

ASP.NET générera un token unique dans le formulaire et vérifiera que la requête contient le même token. Une attaque CSRF externe échouera car l'attaquant ne peut pas deviner ou générer ce token valide.

E. Résumé des bonnes pratiques

Afin de garantir la sécurité des échanges et de prévenir les attaques de type CSRF, plusieurs bonnes pratiques doivent être rigoureusement appliquées. Lors de l'implémentation d'actions sensibles en POST, il est essentiel d'utiliser l'attribut [ValidateAntiForgeryToken] côté serveur. Ce dernier doit être accompagné, dans la vue, de l'ajout de la directive @Html.AntiForgeryToken() à l'intérieur des formulaires, afin de générer automatiquement un jeton de sécurité.

Il est également recommandé de désactiver les actions sensibles accessibles via la méthode GET, en réservant ces traitements aux requêtes POST uniquement. Dans les contextes où les formulaires HTML ne sont pas utilisés, le mécanisme CSRF peut être mis en œuvre via l'envoi d'en-têtes personnalisés, comme X-CSRF-Token, qui seront ensuite vérifiés explicitement côté serveur pour valider l'origine de la requête.

Bilan du projet

1. Évaluation

Le projet a respecté les principaux objectifs fixés, et le cahier des charges a été scrupuleusement suivi. Les fonctionnalités essentielles, telles que la recherche de recettes, l'ajout de recettes, et la gestion des utilisateurs, sont pleinement opérationnelles. La mise en œuvre des fonctionnalités a permis de garantir que l'application répondait aux attentes initiales tout en respectant les contraintes de performance et de sécurité.

2. Apports personnels

Sur le plan technique, ce projet m'a permis de renforcer ma maîtrise du stack ASP.NET et de PostgreSQL, en approfondissant mes connaissances dans le développement d'applications web full-stack. Méthodologiquement, j'ai acquis une meilleure structuration du travail, ainsi qu'une approche plus agile, ce qui a permis une gestion de projet plus fluide et une adaptation rapide aux besoins évolutifs.

En matière de communication, j'ai amélioré ma gestion de projet et mes échanges avec les parties prenantes, ce qui a facilité la prise de décisions et le suivi des tâches. Enfin, au niveau professionnel, ce projet m'a permis de développer une posture plus autonome et rigoureuse, qualités essentielles dans le milieu du développement logiciel.

3. Compétences développées

Au terme de ce projet, j'ai enrichi mes compétences en développement full-stack .NET, particulièrement dans la mise en place d'applications web sécurisées. Ces compétences techniques se sont accompagnées de connaissances approfondies sur la sécurisation des applications, en particulier la gestion de l'authentification et de l'autorisation via ASP.NET Identity.

4. Défis rencontrés

Parmi les défis techniques, l'implémentation d'un système d'authentification robuste s'est avérée complexe, nécessitant une attention particulière pour garantir la sécurité des données utilisateur. L'intégration de Razor et la prise en main d'Entity Framework ont également représenté des défis, car ces technologies demandaient un temps d'adaptation pour une utilisation optimale et la gestion efficace de la base de données. Enfin, la compréhension du code en anglais technique a constitué une difficulté supplémentaire, rendant l'assimilation de certains concepts plus complexe.

Perspectives

Pour l'avenir, plusieurs améliorations sont envisageables. Tout d'abord, l'ajout de suggestions personnalisées basées sur l'historique de recherche et les recettes précédemment consultées pourrait enrichir l'expérience utilisateur en offrant des recommandations pertinentes et adaptées aux préférences individuelles.

Par ailleurs, le développement d'une application mobile en complément du site web pourrait élargir l'accessibilité et la praticité de l'application. Une version mobile permettrait aux utilisateurs de consulter, ajouter ou partager des recettes depuis leurs appareils mobiles, augmentant ainsi l'engagement et l'utilisation de la plateforme.

Annexes

A1. Script SQL de la base de données

1. Script de création des tables (CREATE TABLE)

```sql
-- Suppression des tables dans le bon ordre pour éviter les erreurs de contraintes
DROP TABLE IF EXISTS categories_recettes;
DROP TABLE IF EXISTS ingredients_recettes;
DROP TABLE IF EXISTS etapes;
DROP TABLE IF EXISTS avis;
DROP TABLE IF EXISTS recettes;
DROP TABLE IF EXISTS categories;
DROP TABLE IF EXISTS ingredients;
DROP TABLE IF EXISTS utilisateurs;

-- Création des tables

CREATE TABLE utilisateurs (
id SERIAL PRIMARY KEY,
identifiant VARCHAR(20) UNIQUE NOT NULL,
email VARCHAR(50) UNIQUE NOT NULL,
password VARCHAR(255) NOT NULL
);

CREATE TABLE recettes (
id SERIAL PRIMARY KEY,
nom VARCHAR(100) NOT NULL,
temps_preparation INTERVAL NOT NULL,
temps_cuisson INTERVAL NOT NULL DEFAULT '00:00:00',
difficulte INT CHECK (difficulte BETWEEN 1 AND 5) NOT NULL,
photo VARCHAR(100),
createur INT REFERENCES utilisateurs(id) ON DELETE CASCADE
);
```

```
29
30  CREATE TABLE avis (
31  id_recette INT NOT NULL REFERENCES recettes(id) ON DELETE CASCADE,
32  id_utilisateur INT NOT NULL REFERENCES utilisateurs(id) ON DELETE CASCADE,
33  note INT CHECK (note BETWEEN 1 AND 5) NOT NULL,
34  commentaire VARCHAR(500),
35  PRIMARY KEY (id_recette, id_utilisateur)
36  );
37
38  CREATE TABLE etapes (
39  numero INT NOT NULL,
40  id_recette INT NOT NULL REFERENCES recettes(id) ON DELETE CASCADE,
41  texte VARCHAR(500) NOT NULL,
42  PRIMARY KEY (numero, id_recette)
43  );
44
45  CREATE TABLE ingredients (
46  id SERIAL PRIMARY KEY,
47  nom VARCHAR(50) UNIQUE NOT NULL
48  );
49
50  CREATE TABLE ingredients_recettes (
51  id_ingredient INT NOT NULL REFERENCES ingredients(id) ON DELETE CASCADE,
52  id_recette INT NOT NULL REFERENCES recettes(id) ON DELETE CASCADE,
53  quantite VARCHAR(40) NOT NULL,
54  PRIMARY KEY (id_ingredient, id_recette)
55  );
56
57   CREATE TABLE categories (
58   id SERIAL PRIMARY KEY,
59   nom VARCHAR(50) UNIQUE NOT NULL
60   );
61
62   CREATE TABLE categories_recettes (
63   id_categorie INT NOT NULL REFERENCES categories(id) ON DELETE CASCADE,
64   id_recette INT NOT NULL REFERENCES recettes(id) ON DELETE CASCADE,
65   PRIMARY KEY (id_categorie, id_recette)
66   );
```

```sql
DROP TABLE IF EXISTS utilisateurs;
DROP TABLE IF EXISTS roles ;

-- Création des tables

CREATE TABLE utilisateurs (
id SERIAL PRIMARY KEY,
identifiant VARCHAR(20) UNIQUE NOT NULL,
email VARCHAR(50) UNIQUE NOT NULL,
password VARCHAR(255) NOT NULL
role_id INT REFERENCES Roles(id)
);

-- Utilisateurs
INSERT INTO utilisateurs (identifiant, email, password) VALUES
('admin', 'admin@example.com', 'motdepassehash');

-- Table des rôles
CREATE TABLE Roles (
id SERIAL PRIMARY KEY,
nom VARCHAR(50) UNIQUE NOT NULL
);

CREATE TABLE Roles (
id SERIAL PRIMARY KEY,
nom VARCHAR(50) UNIQUE NOT NULL
);


ALTER TABLE utilisateurs
ADD COLUMN role_id INT REFERENCES Roles(id);
```

2. Script d’insertion de données test (INSERT INTO)

```
-- Insertion des données

-- Utilisateurs
INSERT INTO utilisateurs (identifiant, email, password) VALUES
('admin', 'admin@example.com', 'motdepassehash');

-- Recettes
INSERT INTO recettes (nom, temps_preparation, temps_cuisson, difficulte, photo, createur) VALUES
('Feuilletés apéritif au chaource', '00:20:00', '00:15:00', 2, NULL, 1),
('Soupe chou kale et chorizo', '00:30:00', '00:25:00', 3, NULL, 1),
('Salade de céleri-branche, rhubarbe et maquereau fumé', '00:15:00', DEFAULT, 2, NULL, 1), -- Utilise la valeur par
défaut
('Pot-au-feu', '00:30:00', '02:30:00', 5, NULL, 1),
('Salade d''épeautre aux haricots verts et bleu', '00:20:00', DEFAULT, 2, NULL, 1),
('Tarte aux mirabelles', '00:30:00', '00:40:00', 3, NULL, 1);

-- Insertion des ingrédients
INSERT INTO ingredients (nom) VALUES
('Pâte feuilletée'), ('Chaource'), ('Chou kale'), ('Chorizo'),
('Céleri-branche'), ('Rhubarbe'), ('Maquereau fumé'), ('Viande de bœuf'),
('Légumes'), ('Épeautre'), ('Haricots verts'), ('Fromage bleu'),
('Mirabelles'), ('Pâte brisée');

-- Association ingrédients-recettes
INSERT INTO ingredients_recettes (id_ingredient, id_recette, quantite) VALUES
(1, 1, '1 rouleau'), (2, 1, '100g'),
(3, 2, '200g'), (4, 2, '100g'),
(5, 3, '200g'), (6, 3, '100g'), (7, 3, '1 filet'),
(8, 4, '1kg'), (9, 4, '500g'),
(10, 5, '200g'), (11, 5, '150g'), (12, 5, '100g'),
(13, 6, '500g'), (14, 6, '1 rouleau');


  -- Insertion des étapes
  INSERT INTO etapes (numero, id_recette, texte) VALUES
  (1, 1, 'Préchauffer le four à 180°C.'),
  (2, 1, 'Découper la pâte, ajouter le chaource.'),
  (3, 1, 'Cuire 15 min.'),
  (1, 2, 'Faire revenir le chorizo.'),
  (2, 2, 'Ajouter le chou kale et cuire 25 min.'),
  (1, 3, 'Couper les ingrédients, mélanger et servir.'),
  (1, 4, 'Cuire la viande 3h.'),
  (2, 4, 'Ajouter les légumes en fin de cuisson.'),
  (1, 5, 'Cuire l''épeautre, ajouter les autres ingrédients et mélanger.'),
  (1, 6, 'Disposer les mirabelles sur la pâte et cuire 40 min à 180°C.');

  -- Insertion des catégories
  INSERT INTO categories (nom) VALUES
  ('Apéritif'), ('Soupe'), ('Entrées'), ('Plats'), ('Plats végétariens'), ('Desserts');

  -- Association catégories-recettes
  INSERT INTO categories_recettes (id_categorie, id_recette) VALUES
  (1, 1), (2, 2), (3, 3), (4, 4), (5, 5), (6, 6);
```

A2. Fichiers de configuration

1. Program.cs

SaveursInedites

```
using Microsoft.AspNetCore.Authentication.Cookies;
using Microsoft.Extensions.Options;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddRazorPages();
builder.Services.Configure<CookiePolicyOptions>(options =>
{
    // This lambda determines whether user consent for non-essential
    // cookies is needed for a given request.
    options.CheckConsentNeeded = context => true;

    options.MinimumSameSitePolicy = SameSiteMode.Lax;
});



// Add services to the container.
builder.Services.AddControllersWithViews();

builder.Services.AddAuthentication(CookieAuthenticationDefaults.AuthenticationScheme).AddCookie(
            option =>
            {
                option.LoginPath = "/Access/SignIn";
                option.ExpireTimeSpan = TimeSpan.FromMinutes(20);
                option.AccessDeniedPath = "/Home/HandleError/403";
            }
        );

var app = builder.Build();

if (!app.Environment.IsDevelopment())
{
    // Redirige vers la méthode Error si il y'a une exception
    app.UseExceptionHandler("/Home/Error");
    app.UseHsts();
}
app.UseStatusCodePagesWithReExecute("/Home/HandleError/{0}");



app.UseHttpsRedirection();

app.UseStaticFiles();
app.UseCookiePolicy();

app.UseRouting();

app.UseAuthentication();
app.UseAuthorization();

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Recettes}/{action=Index}/{id?}");


app.Run();
```