

METZ NUMERIC SCHOOL

Dossier projet

Trame du dossier projet

Kraimbah Said / CDA / METZ NUMERICSCHOOL
2024 - 2025

Table des matières

1. La liste des compétences mises en œuvre dans le cadre du projet.....	0
2. Le cahier des charges ou l'expression des besoins du projet.....	1
3. la présentation de l'entreprise et du service.....	3
4. La gestion de projet.....	4
5. Les spécifications fonctionnelles du projet.....	6
5.1. Les contraintes du projet et livrables attendus.....	8
5.2. L'architecture logicielle du projet.....	9
5.3. Les maquettes et enchaînement des maquettes.....	11
5.4. Le modèle entités-associations et modèle physique de la base de données.....	14
5.5. Création et modification de la base de données.....	20
5.6. Le diagramme du comportement des fonctionnalités de type cas d'utilisations..	23
5.7. Le diagramme du détail des cas d'utilisations les plus significatifs de type diagramme de séquence.....	27
6. Les spécifications techniques du projet.....	28
7. Les réalisations du candidat comportant les extraits de code les plus significatifs, les arguments qui ont conduit à ses choix, y compris pour la sécurité.....	31
7.1. Les captures d'écran d'interfaces utilisateur et le code correspondant.....	31
7.2. Des extraits de code de composants métier.....	34
7.3. Des extraits de code de composants d'accès aux données.....	38
7.4. Des extraits de code d'autres composants (contrôleurs, utilitaires...)	41
8. La présentation d'éléments de sécurité de l'application.....	42
9. Déploiement.....	46
10. plan de tests et jeu d'essai élaboré par le candidat.....	49
11. la description de la veille, effectuée par le candidat.....	52

1. La liste des compétences mises en œuvre dans le cadre du projet

- Installer et configurer son environnement de travail en fonction du projet
- Développer des interfaces utilisateur
- Développer des composants métier
- Contribuer à la gestion d'un projet informatique
- Analyser les besoins et maquetter une application
- Définir l'architecture logicielle d'une application
- Concevoir et mettre en place une base de données relationnelle
- Développer des composants d'accès aux données SQL et NoSQL
- Préparer et exécuter les plans de tests d'une application
- Préparer et documenter le déploiement d'une application
- Contribuer à la mise en production dans une démarche DevOps

2. Le cahier des charges ou l'expression des besoins du projet

Introduction

L'application de gestion pour un club canin vise à optimiser l'organisation des cours, en facilitant l'inscription des propriétaires, la gestion des plannings et le suivi des chiens. Elle doit offrir une solution intuitive et efficace pour organiser les séances en fonction de l'âge et des besoins des chiens, tout en garantissant une expérience utilisateur fluide.

Contexte et Enjeux

L'application doit permettre une gestion simplifiée et structurée des cours en fonction de l'âge et des besoins des chiens. Elle doit assurer un processus d'inscription fluide pour les propriétaires, une organisation optimisée des cours et une réservation contrôlée en fonction des places disponibles. De plus, l'éligibilité des chiens aux cours devra être vérifiée automatiquement en fonction de leur âge. Enfin, un système de suivi et de communication sera mis en place afin de garantir une interaction efficace entre le club et ses membres.

Objectifs et Acteurs

L'application vise à offrir un outil performant pour la gestion des cours et des inscriptions, tout en améliorant la communication entre les propriétaires et le club. Elle doit permettre un suivi simplifié des séances et proposer une interface intuitive pour tous les utilisateurs.

Les principaux acteurs de cette application sont le responsable du club, qui supervise les cours et gère les inscriptions, les propriétaires qui inscrivent leurs chiens et consultent les disponibilités, ainsi que les coachs et entraîneurs chargés d'encadrer les séances et de suivre la présence des participants.

Fonctionnalités Principales

L'application intègre une gestion complète des utilisateurs, incluant la création de compte, une authentification sécurisée et la possibilité de modifier les profils des adhérents. Une section dédiée aux chiens permet d'ajouter et de mettre à jour leurs informations, notamment la race, le sexe et l'âge, avec une catégorisation selon ces critères.

La gestion des cours repose sur une organisation structurée des séances, avec des catégories définies en fonction de l'âge du chien. Les cours sont ainsi répartis en plusieurs niveaux : l'école du chiot pour les chiens de moins de cinq mois, ainsi que des sessions d'éducation adaptées aux tranches d'âge suivantes : 6 à 12 mois, 1 à 2 ans et plus de 2 ans. Un calendrier interactif affichera les séances disponibles, et un système d'inscription en ligne permettra aux propriétaires de réserver des places dans la limite des disponibilités. Une validation automatique de l'éligibilité en fonction de l'âge du chien sera également mise en place.

Un tableau de bord permettra de suivre les événements et les disponibilités des cours. Des notifications, notamment par email, rappelleront aux propriétaires les séances à venir. Un historique des cours permettra de suivre la progression des chiens et d'optimiser leur apprentissage.

Contraintes Techniques et Architecture Applicative

L'application devra être accessible sur tous types d'appareils grâce à un design responsive. Elle devra être compatible avec les navigateurs modernes tels que Chrome, Firefox, Safari et Edge. La sécurité des données sera une priorité, avec une mise en conformité au RGPD, ainsi que le cryptage et l'anonymisation des informations sensibles. Une approche éco-conçue sera privilégiée pour optimiser l'utilisation des ressources.

Côté architecture, le backend sera développé en C# avec .NET Core et reposera sur une base de données SQL Server. Les frameworks Identity et Entity Framework seront utilisés pour la gestion des utilisateurs et des données. Le frontend sera conçu avec Blazor et adoptera une interface moderne via Bootstrap.

User Stories

Les principales fonctionnalités seront développées à travers plusieurs scénarios utilisateurs. Les propriétaires pourront créer et modifier leur compte, ajouter et mettre à jour les informations de leurs chiens, consulter et réserver des cours adaptés, et recevoir des rappels avant chaque séance. De leur côté, les coachs auront accès à la liste des participants pour chaque cours, tandis que le responsable du club disposera d'un tableau de bord récapitulatif des inscriptions et des activités en cours.

Organisation du Développement

Le développement de l'application s'appuiera sur des outils de gestion de projet tels que Trello et des diagrammes de Gantt pour structurer les étapes. Le contrôle de version sera assuré via Git afin de garantir un suivi des évolutions du projet efficace .

3. la présentation de l'entreprise et du service

Présentation de l'Entreprise - Educanin

Educanin est une entreprise spécialisée dans l'éducation canine et l'accompagnement des propriétaires de chiens. Fondée par des passionnés du monde canin, elle a pour mission de proposer des solutions adaptées à tous les types de chiens, du chiot au chien adulte, en prenant en compte leurs spécificités et leurs besoins. Grâce à une équipe d'experts en comportement animal et en dressage, Educanin offre un cadre bienveillant et structuré, propice à un apprentissage progressif et positif, tout en renforçant le lien entre le chien et son maître.

Le Service Proposé

Educanin propose une plateforme permettant de gérer les inscriptions aux cours, d'organiser les séances en fonction de l'âge et du niveau du chien, et d'assurer un suivi personnalisé pour chaque animal. Cette approche moderne, basée sur des outils technologiques, facilite la gestion des activités du club canin et améliore l'interaction entre les propriétaires, les éducateurs et les responsables du centre. Parmi les services offerts, on retrouve des cours d'éducation canine adaptés à chaque tranche d'âge et à chaque niveau, une interface intuitive pour l'inscription et la planification des séances, un suivi individualisé avec un historique des progrès et des recommandations, ainsi qu'un système de notifications pour rappeler les rendez-vous et annoncer les événements du club.

4. La gestion de projet

Dans ce projet, je vais gérer l'ensemble du processus de développement en solo. Afin d'assurer une organisation optimale et un suivi structuré, j'ai sélectionné plusieurs outils et méthodologies qui me permettront de suivre l'avancement du projet, de m'assurer de la bonne gestion des tâches et de centraliser les informations importantes. En tant que développeur seul sur ce projet, l'utilisation de ces outils est essentielle pour respecter les délais et assurer la qualité de l'application.

Trello

Trello est un outil de gestion de projet basé sur la méthode Kanban. Il est particulièrement efficace pour diviser le travail en projets et tâches. Chaque projet est représenté par un tableau, et chaque tâche est une carte qui peut être personnalisée avec différents statuts et autres options. J'ai configuré mon tableau Trello en suivant les principes de la méthode SCRUM afin de garantir un suivi efficace de l'avancement du projet, en respectant les sprints et les priorités.

Notion

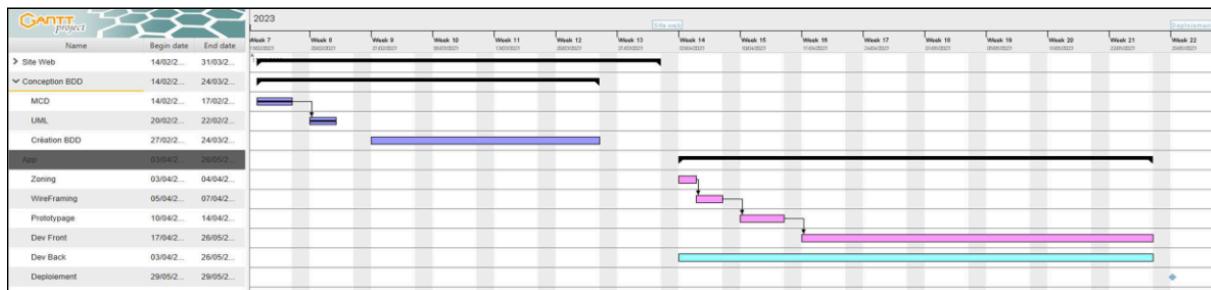
Notion est un outil de productivité tout-en-un qui offre des fonctionnalités pour la prise de notes, la gestion des tâches, les calendriers et la création de wikis. Ce logiciel est très flexible et personnalisable, ce qui permet de structurer l'espace de travail en fonction des besoins spécifiques du projet. Notion a été un excellent allié pour centraliser la documentation technique et les informations relatives aux technologies utilisées dans le projet.

The screenshot shows the Notion application interface. On the left, there is a sidebar with various sections like 'Pages privées', 'Rechercher', 'Accueil', 'Boîte de réception', 'Paramètres et membres', 'Anglais', 'Système de prise de notes...', 'Calculateur de notes', 'Réseau CMS', 'Culture informatique', 'POO / Algo', 'TRE', 'Ressources', 'Calendrier', 'Créer un espace d'équipe', 'Modèles', 'Corbeille', 'Aide et service client'. The main content area has a search bar at the top. Below it, there's a diagram titled 'Architecture MVC' with three main components: 'Contrôleur', 'Vue', and 'Modèle'. The 'Contrôleur' box receives a 'Requête http' from the outside. It interacts with the 'Route' (labeled '1') and the 'Modèle' (labeled '3'). The 'Route' leads to the 'Vue' (labeled '4'). The 'Modèle' provides 'Demande des données' to the 'Vue' (labeled '2') and also provides 'Obtention des données' back to the 'Contrôleur' (labeled '5'). The 'Vue' then displays the final output. At the bottom of the main content area, there's a note: 'Aucune page à l'intérieur'.

Screenshot de Notion

GanttProject

GanttProject est un logiciel open source qui permet de créer des diagrammes de Gantt pour planifier et suivre l'avancement des tâches d'un projet. Cet outil m'a permis de visualiser la progression du projet sur une chronologie unique, de planifier les différentes phases de développement, et d'ajuster les délais en fonction des contraintes rencontrées. Grâce à GanttProject, j'ai pu estimer avec précision la durée de chaque tâche et la durée totale du développement.



Screenshot de GanttProject

GitHub

GitHub est une plateforme de gestion de versions et de développement collaboratif qui repose sur Git. Elle est essentielle pour suivre et gérer les modifications du code source tout au long du projet. Même en travaillant seul, GitHub me permet de gérer les versions de mon code, de centraliser la documentation et de sauvegarder mon avancement dans le cloud pour le sécuriser en cas de perte de mon ordinateur. Cet outil est indispensable pour maintenir un historique des évolutions du projet et garantir la qualité du code.

File	Message	Time
.github/workflows	Deleting sonarQube	2 days ago
EduCanin	Ended main feature	2 days ago
Educanin.Tests	Dog form to register at courses added	4 days ago
.gitignore	adding sonar to CI	2 weeks ago
EduCanin.sln	adding sonar to CI	2 weeks ago
EduCanin.sln.DotSettings.user	First Version	2 weeks ago
README.md	Initial commit	2 weeks ago

Screenshot du dépôt GitHub d'EduCanin

5. Les spécifications fonctionnelles du projet

Spécifications fonctionnelle

En s'appuyant sur les besoins exprimés dans le cahier des charges, cette section précise les actions que l'application devra permettre, tout en garantissant une utilisation simple, fluide et adaptée à tous les profils d'utilisateurs.

L'accès à l'application sera sécurisé grâce à un système d'authentification par identifiants. Chaque utilisateur pourra créer un compte, se connecter à son espace personnel, et accéder aux fonctionnalités qui lui sont réservées. L'inscription au club sera une condition préalable indispensable pour accéder aux cours. Le responsable du club jouera un rôle central dans l'administration de la plateforme : il pourra gérer les comptes utilisateurs, superviser les inscriptions aux cours, et assurer la cohérence de l'organisation générale.

Les propriétaires de chiens disposeront d'un espace dédié dans lequel ils pourront renseigner et modifier les informations relatives à leurs chiens. Ces données incluront notamment la race, le sexe et la date de naissance de l'animal, qui seront utilisées pour déterminer l'éligibilité à certains cours. Le système devra donc être capable de valider automatiquement les inscriptions en fonction de critères précis tels que l'âge ou la catégorie de cours.

La gestion des cours constitue le cœur de l'application. Chaque type de cours sera défini par une tranche d'âge cible et un nombre maximal de participants. Cela permettra de garantir une répartition cohérente des chiens selon leur stade de développement et d'éviter la surcharge des séances. Par exemple :

- L'école du chiot sera réservée aux chiens de moins de cinq mois.
- Les cours d'éducation seront segmentés en plusieurs tranches : de 6 à 12 mois, de 1 à 2 ans, et plus de 2 ans.

Ces règles devront être strictement respectées lors de l'inscription. Le système devra également gérer les cas particuliers, comme l'inéligibilité à un cours ou le dépassement du nombre de participants autorisés.

L'application permettra la création et la gestion des séances de cours, avec la possibilité de planifier les horaires, d'attribuer un type de cours, et de consulter l'ensemble des sessions dans un calendrier interactif. Ce calendrier sera visible à la fois pour les responsables du club et pour les adhérents.

Lorsqu'un propriétaire souhaite inscrire son chien à une séance, il devra sélectionner un cours depuis le calendrier. Le système vérifiera alors automatiquement l'âge de l'animal, la disponibilité du créneau, et la compatibilité avec les règles du cours.

Deux interfaces principales seront mises en place : une pour l'administration et une autre pour les membres du club. La page d'accueil du responsable affichera un tableau de bord synthétique avec les indicateurs clés comme le nombre de séances planifiées, le taux de remplissage des cours, et les tâches à traiter. L'interface de l'adhérent mettra en avant les prochaines séances de son chien, les inscriptions

passées, et un accès simplifié aux outils d'inscription et de gestion du profil.

L'application devra être entièrement responsive afin d'offrir une navigation optimale sur mobile, tablette et ordinateur. L'ergonomie sera pensée pour convenir à un public large, quel que soit l'âge ou la familiarité avec les outils numériques. Le respect des standards du web (HTML/CSS à jour, compatibilité navigateurs récents) est requis, tout comme la conformité au RGPD. Les données sensibles, telles que les identifiants et les informations personnelles, devront être anonymisées et chiffrées.

Trois profils d'utilisateurs interagiront avec le système : les responsables du club, les propriétaires de chiens, et éventuellement les coachs ou entraîneurs. Ces derniers pourront consulter les plannings, suivre les présences et préparer leurs séances. La répartition des rôles garantira une organisation claire et efficace, tout en sécurisant l'accès aux données et aux fonctions critiques de l'application.

5.1. Les contraintes du projet et livrables attendus

Ce projet est une initiative étudiante proposée par MNS dans le cadre de notre formation. L'objectif principal était de nous offrir une expérience pratique pour appliquer les concepts théoriques vus en cours et développer des compétences concrètes en développement logiciel. Conçu pour simuler un environnement professionnel, le projet nous a permis de travailler sur des tâches et des défis proches de ceux rencontrés dans le monde du travail.

Nous avons bénéficié de peu de contraintes quant aux technologies à utiliser. Cependant, il nous a été demandé de conceptualiser notre base de données selon la méthode Merise et de développer le backend de l'application en utilisant un langage de programmation orienté objet (OOP). Ce qui a été particulièrement enrichissant dans ce projet, c'est la liberté qui nous a été laissée pour choisir les technologies et frameworks que nous jugions les plus adaptés à nos besoins.

En ce qui concerne les livrables, des dates de rendu des livrables sont imposées. Le projet a débuté en février 2025, et l'objectif était de livrer une application web opérationnelle pour la fin août, afin de pouvoir la présenter lors de l'examen.

5.2. L'architecture logicielle du projet

Architecture Logicielle MVC pour Application Web ASP.NET: Guide Complet

Lors de la réalisation d'une application, il est impératif de maintenir une cohérence tout au long du projet. Cela est essentiel non seulement pour des raisons d'organisation, mais aussi pour assurer une séparation claire des responsabilités dans le code. Les architectures logicielles jouent un rôle crucial en fournissant un cadre général pour structurer l'application.

Architecture Model-View-Controller (MVC)

L'architecture MVC est largement utilisée et reconnue pour sa capacité à séparer les préoccupations, facilitant ainsi la maintenance et l'évolution du projet. Elle divise l'application en trois composants principaux. Le Modèle représente les données de l'application et la logique métier, constituant ainsi le cœur fonctionnel du système. La Vue gère l'interface utilisateur et présente les données sous forme de pages interactives, offrant une expérience visuelle adaptée aux besoins des utilisateurs. Le Contrôleur, quant à lui, gère les interactions de l'utilisateur, traite les requêtes entrantes et coordonne les mises à jour entre le modèle et la vue, servant d'intermédiaire essentiel dans cette architecture tripartite.

Pourquoi choisir l'architecture MVC?

Après avoir évalué plusieurs options, l'architecture MVC s'avère la plus adaptée pour plusieurs raisons significatives. Tout d'abord explorer efficacement le paradigme MVC et ainsi enrichir son expertise technique. La popularité et l'adoption généralisée du pattern MVC en font un standard incontournable que tout développeur professionnel doit maîtriser. De plus, le support intégré offert par Visual Studio à travers des templates de projets ASP.Net Core MVC permet d'implémenter cette architecture de manière fluide, rapide et structurée, accélérant considérablement le processus de développement initial.

Avantages du MVC dans une WebApp ASP.NET

L'adoption de l'architecture MVC dans une WebApp en ASP.Net offre de nombreux bénéfices tangibles pour le développement moderne. La séparation des préoccupations permet à chaque section d'évoluer indépendamment, rendant le code plus propre et plus facile à maintenir sur le long terme. Cette isolation des composants facilite grandement les tests unitaires qui deviennent plus ciblés et plus efficaces, chaque partie pouvant être validée indépendamment des autres. Le MVC favorise également la réutilisabilité du code, particulièrement au niveau des modèles qui peuvent servir dans différentes parties de l'application ou même être transposés vers d'autres projets connexes. La flexibilité et l'évolutivité inhérentes à cette architecture permettent d'intégrer de nouvelles fonctionnalités sans perturber les composants existants, aspect crucial pour le développement itératif moderne. Enfin, le support natif fourni par le framework ASP.Net Core MVC, notamment à travers le système de routage des contrôleurs, simplifie considérablement le développement quotidien.

Architecture enrichie: MVC + Services + Repositories

Pour renforcer l'architecture standard, le projet intègre judicieusement deux couches supplémentaires qui enrichissent le modèle MVC traditionnel. La couche de Services encapsule la logique métier spécifique à l'application, agissant comme un intermédiaire essentiel entre les contrôleurs qui gèrent les interactions utilisateur et les accès aux données persistantes. Cette couche permet de centraliser les règles métier et d'assurer leur cohérence à travers l'ensemble de l'application. Parallèlement, la couche de Repositories prend en charge toutes les opérations CRUD sur la base de données via le modèle et Entity Framework, isolant ainsi la logique d'accès aux données et facilitant la maintenance ou l'évolution du modèle de persistance. Cette organisation architecturale avancée respecte davantage les principes SOLID et augmente significativement la maintenabilité du code sur le long terme, tout en facilitant les évolutions futures du système.

Technologies complémentaires

Le projet s'appuie sur un écosystème riche de technologies complémentaires qui renforcent sa robustesse et sa flexibilité. Entity Framework simplifie considérablement l'accès aux données grâce à son système de mapping objet-relationnel, permettant aux développeurs de manipuler des objets plutôt que des requêtes SQL directes. La base de données SQL Server offre une solution éprouvée et performante pour la persistance des données, avec des capacités avancées de sécurité et d'indexation. Pour la gestion des utilisateurs, ASP.NET Identity fournit une infrastructure complète d'authentification et d'autorisation, intégrant nativement les meilleures pratiques de sécurité. L'expérience utilisateur s'appuie sur Bootstrap, HTML et CSS pour créer des interfaces responsive et esthétiques, tandis que JavaScript apporte la dimension dynamique nécessaire à une application web moderne, fluidifiant les interactions.

Flux de traitement d'une requête

Le traitement d'une requête utilisateur dans cette architecture suit un parcours méthodique et bien défini qui garantit l'intégrité du processus. Lorsqu'un utilisateur initie une interaction avec l'interface web, que ce soit via la soumission d'un formulaire ou un simple clic, sa demande est immédiatement acheminée vers le Contrôleur approprié qui prend en charge son traitement initial. Ce contrôleur, après avoir validé les données entrantes, délègue ensuite les opérations métier nécessaires à un Service spécialisé qui applique les règles et transformations requises. Si l'opération implique une persistance ou une consultation de données, le Service fait appel au Repository correspondant, qui interagit avec la base de données via le modèle et Entity Framework en respectant les patterns d'accès optimisés. Une fois le traitement métier terminé, les résultats remontent progressivement la chaîne jusqu'au Contrôleur qui sélectionne alors la Vue appropriée pour présenter le résultat final à l'utilisateur dans un format adapté à ses besoins. Cette organisation structurée du flux d'information garantit une séparation claire des responsabilités et une maintenance simplifiée tout au long du cycle de vie de l'application.

5.3. Les maquettes et enchaînement des maquettes

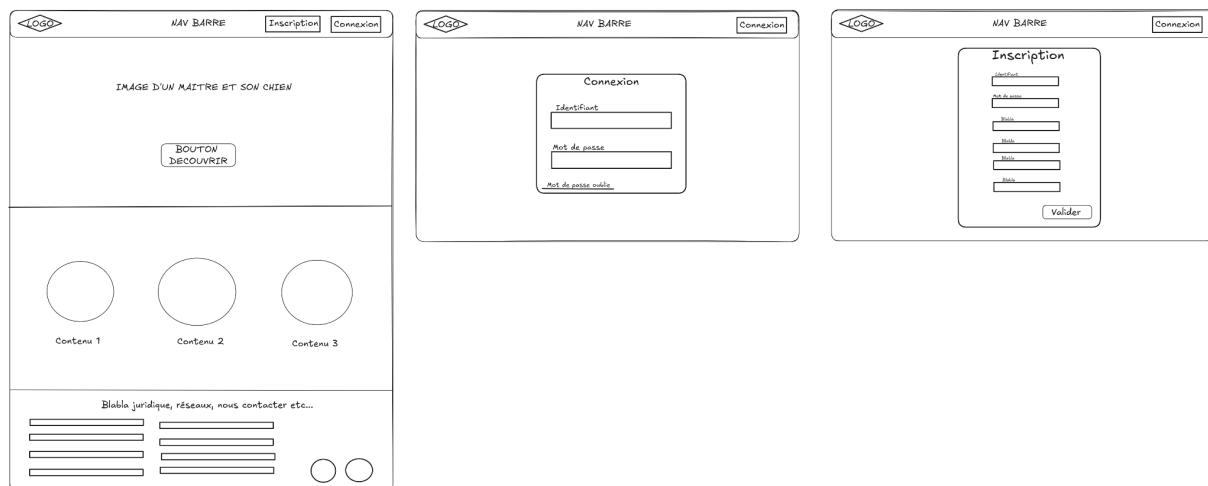
La réalisation d'une maquette représente une étape essentielle dans la conception d'un projet. Pour ce travail, j'ai opté pour Figma Desktop, un outil reconnu à la fois pour sa convivialité et la richesse de ses fonctionnalités. La maquette permet d'ajuster rapidement les éléments en cas de retours ou de modifications demandées par le client. Elle offre également une visualisation claire et interactive du projet, bien avant le début du développement. Afin de structurer efficacement ce processus, j'ai choisi de le répartir en quatre étapes distinctes.

Croquis

Le croquis consiste en une réalisation rapide et approximative des premiers visuels de l'application. Ces esquisses préliminaires permettent de m'assurer que je suis sur la même longueur d'onde que le client concernant les éléments de base du projet. À ce stade, il est essentiel d'établir une compréhension commune des objectifs principaux et des attentes, ce qui facilitera les étapes suivantes. Les croquis peuvent être réalisés à la main ou à l'aide d'outils numériques simples, offrant ainsi une première vision tangible des idées abstraites du client.

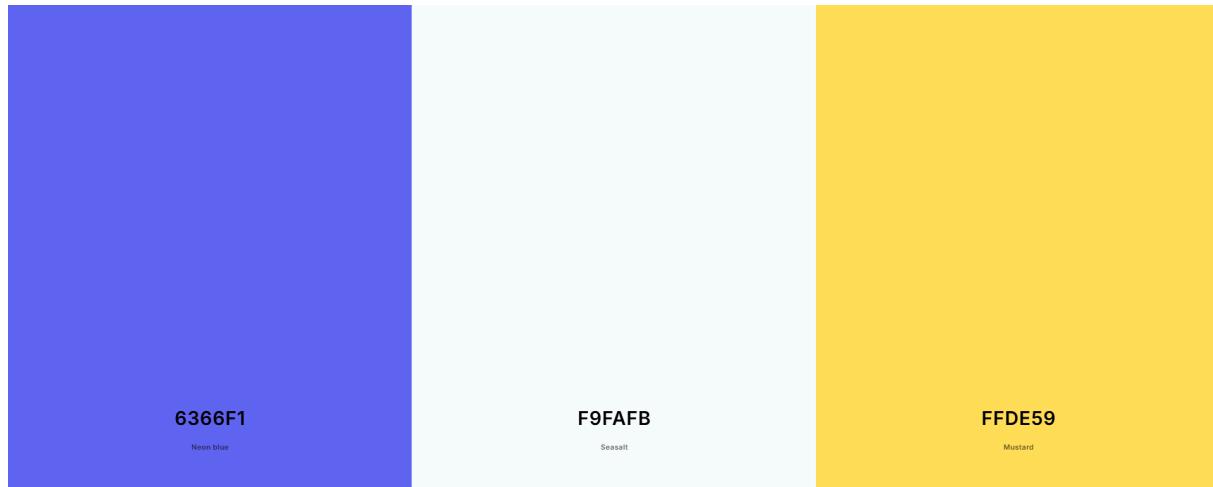
Wireframe basse fidélité

Un wireframe basse fidélité est une représentation simple et schématique d'une interface, utilisée principalement pour explorer rapidement des idées de conception. Il permet de poser les grandes lignes d'une interface sans s'attarder sur les détails graphiques. Généralement en noir et blanc ou en niveaux de gris, ce type de wireframe n'inclut ni polices réelles, ni icônes détaillées, et le contenu y est souvent remplacé par des blocs ou du texte fictif comme du Lorem ipsum. Il peut être dessiné à la main ou créé avec des outils simples comme exalidraw. Sa rapidité de création en fait un excellent support pour le brainstorming et les discussions initiales avec les parties prenantes, car il est aussi très facile à modifier.



Charte graphique

La charte graphique représente l'ensemble des éléments visuels qui définissent l'identité visuelle de l'application. Elle regroupe les choix typographiques, les couleurs, les logos, ainsi que les règles de mise en page et de hiérarchie des informations. L'objectif de la charte graphique est d'assurer une cohérence visuelle tout au long du projet, en harmonisant les éléments pour que l'application soit esthétique et professionnelle.

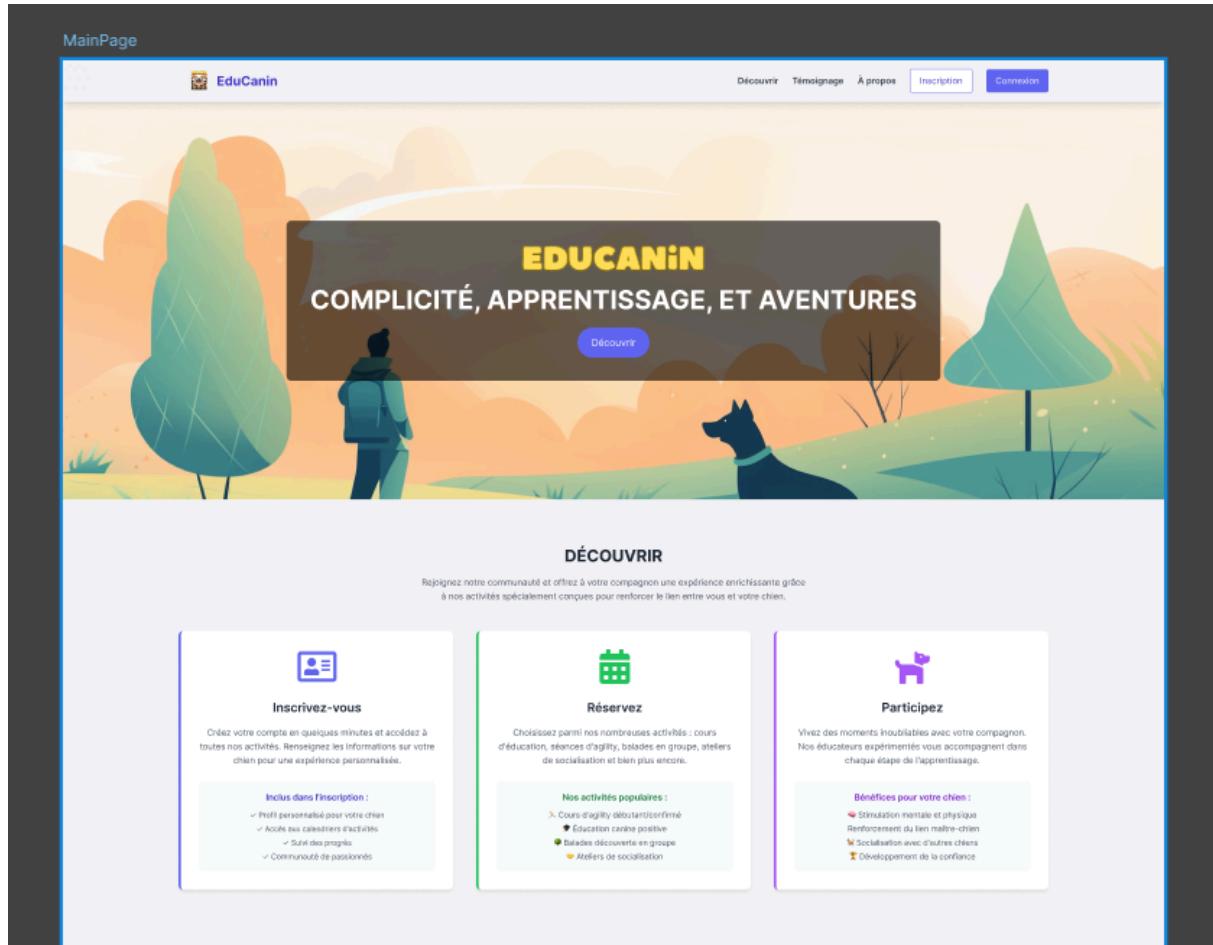


Wireframe haute fidélité

Le wireframe haute fidélité est une version plus détaillée et proche du design final. À ce stade, j'ai intégré des éléments graphiques plus précis et des informations supplémentaires sur les interactions. Ces wireframes incluent des détails tels que les typographies, les couleurs et les images, offrant ainsi une vision plus réaliste de l'apparence finale de l'application. J'ai utilisé Figma pour créer ces wireframes, ce qui m'a permis de visualiser de manière plus concrète le produit fini.

Prototype

Le prototype représente la dernière étape du maquettage. Il s'agit d'une version interactive du wireframe haute fidélité qui permet de simuler la navigation et les interactions de l'application, comme si elle était déjà fonctionnelle. Les prototypes sont particulièrement utiles pour obtenir des retours concrets de la part des utilisateurs et du client avant de démarrer le développement. Ils permettent de tester et valider les interactions et transitions, ce qui facilite l'identification des ajustements nécessaires. En outre, un prototype bien conçu sert de guide pour les développeurs, réduisant les ambiguïtés et assurant une meilleure compréhension des spécifications du projet.



Wireframe haute fidélité de EduCanin

5.4. Le modèle entités-associations et modèle physique de la base de données

Méthodologie Merise

Pour ce projet, j'ai adopté la méthodologie Merise. J'ai suivi scrupuleusement les étapes du Modèle Conceptuel des Données (MCD), du Modèle Logique des Données (MLD) et du Modèle Physique des Données (MPD). Cette approche m'a permis de me conformer aux règles spécifiques de chaque étape, réduisant ainsi les risques d'erreurs de conception. J'ai constaté qu'il est vraiment avantageux de consacrer une part significative de temps au début du projet à cette méthode, car elle permet d'éviter de nombreuses erreurs potentielles, ce qui nous fait gagner un temps précieux sur le long terme.

Collecte des informations

La phase de collecte des informations constitue une étape fondamentale dans la méthode Merise. Dans mon cas, les données nécessaires à l'analyse ont été extraites du cahier des charges fourni par le client. Ce document a servi de point de départ pour identifier les règles de gestion, les besoins fonctionnels ainsi que les éléments clés du futur système d'information. Bien que le cahier des charges contienne une partie des exigences, il a été nécessaire d'interpréter certains passages pour en déduire les règles implicites. Cette démarche s'inscrit dans la logique du travail collaboratif entre le client et le concepteur, visant à clarifier et valider les attentes avant toute modélisation. Une bonne compréhension dès cette phase permet de limiter les erreurs et les malentendus dans les étapes suivantes du projet.

Dictionnaire des données

Le dictionnaire des données est un document centralisé qui décrit les données nécessaires pour un système informatique. Il utilise des formats génériques tels que le type alphabétique (caractères uniquement), alphanumérique (caractères et chiffres), numérique (nombres), date et logique (0-1, Vrai-Faux, Oui-Non). Ce dictionnaire recense et classe toutes les informations et règles de gestion (comme le calcul d'un taux de remise). Il garantit une compréhension commune et une utilisation cohérente des données, assurant ainsi leur qualité et leur précision dans le projet.

Nom	ID	Type
Email	Email	VARCHAR
FirstName	FirstName	VARCHAR
LastName	LastName	VARCHAR
Password	Password	VARCHAR
RegisterDate	RegisterDate	DATETIME
Role	Role	ENUM

Extrait de dictionnaire des données d'Educanin

Élaboration des dépendances fonctionnelles

L'élaboration des dépendances fonctionnelles, en lien étroit avec le dictionnaire des données, permet de représenter les relations entre les différentes données du système à l'aide d'une matrice. Cette matrice, sous forme de tableau, utilise les en-têtes de lignes pour les données sources et les en-têtes de colonnes pour les données cibles. Un "1" dans une cellule indique une dépendance fonctionnelle. Cette étape est cruciale pour assurer une structure de base de données optimisée et cohérente, évitant ainsi les erreurs de conception futures.

Modèle Conceptuel des Données - MCD

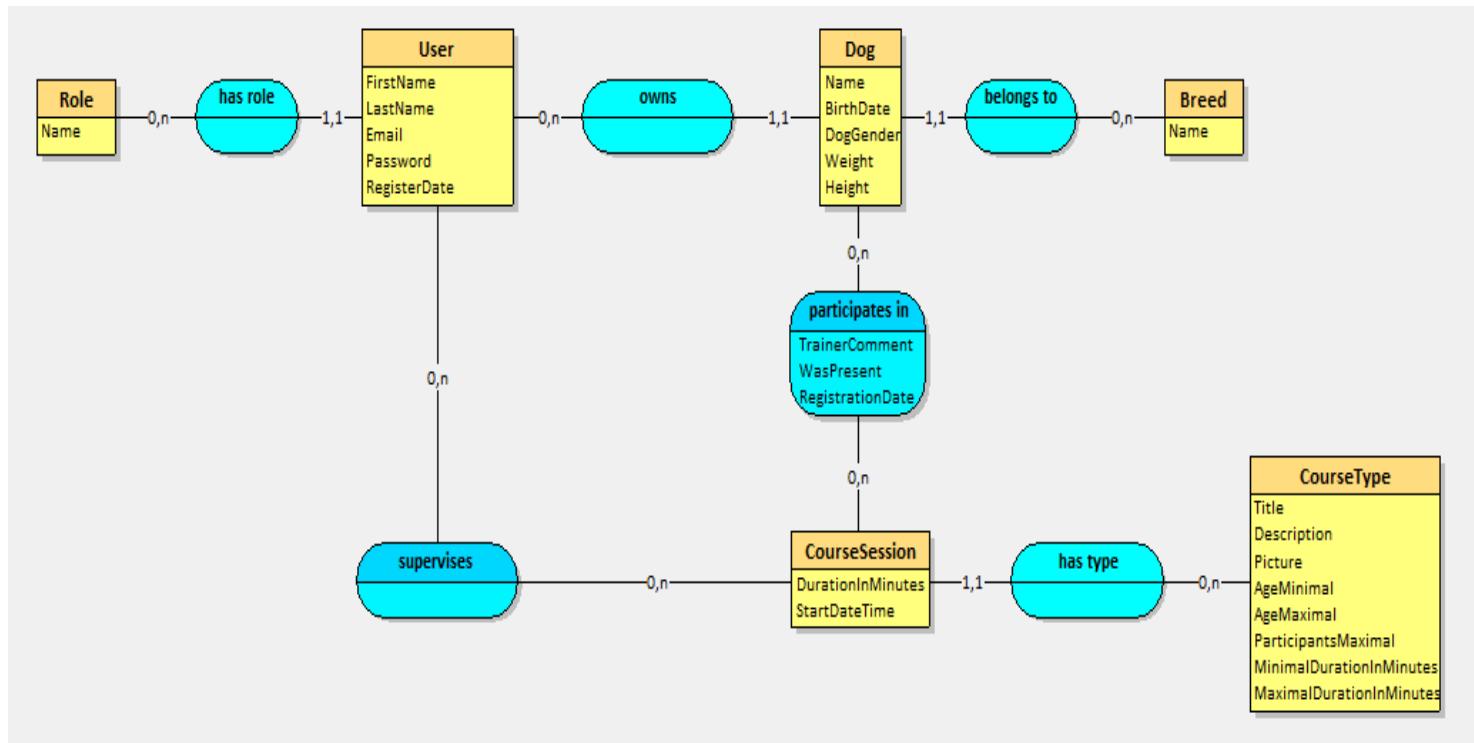
Le Modèle Conceptuel des Données (MCD) est un outil clé utilisé pour structurer les entités, les attributs, et les relations entre elles afin de refléter fidèlement les besoins métier. Le MCD est utile car il permet une compréhension commune et une communication claire entre les différentes parties prenantes du projet, facilitant ainsi la conception d'une base de données cohérente et adaptée aux exigences fonctionnelles avant de passer à des phases de modélisation plus techniques.

Pour la réalisation de mon MCD je me suis tourné vers le logiciel Looping qui est un outil de modélisation de bases de données basé sur la méthode Merise. Il permet de créer des Modèles Conceptuels de Données (MCD) et des Modèles Logiques de Données (MLD), facilitant ainsi la conception, la visualisation et la gestion des bases de données relationnelles.

En me basant sur la matrice j'ai commencé par la création des entités en leur donnant un nom et en introduisant les propriétés correspondantes. J'ai également déterminé les clés primaires.

Une fois les entités créées j'ai procédé à l'établissement des relations entre chacune. J'ai tracé un lien entre les tables possédant une relation commune et je l'ai définie avec un verbe. Ensuite j'ai défini les cardinalités qui expriment le nombre de fois où l'occurrence d'une entité participe aux occurrences de la relation. Les cardinalités les plus courantes sont :

- 1,1 – A possède une seule fois B
- 0,1 – A possède entre zéro et une seule fois B
- 1,n – A possède entre une et une infinité de fois B
- 0,n – A possède entre zéro et une infinité de fois B



Exemple complet du Modèle Conceptuel des Données de EduCanin

Modèle Logique des Données - MLD

Le MLD est une traduction du Modèle Conceptuel des Données (MCD) vers une représentation plus proche de l'implémentation physique, tout en restant indépendante d'un SGBD particulier. Chaque entité du MCD devient une table, et les associations sont transformées en clés étrangères ou en tables associatives, selon les règles de transformation dépendant des cardinalités.

Afin de transformer les relations selon les règles du MLD il m'a suffi d'observer les cardinalités maximales entre chaque entité. Selon les valeurs on peut en déduire trois types de relations :

- One-to-many :

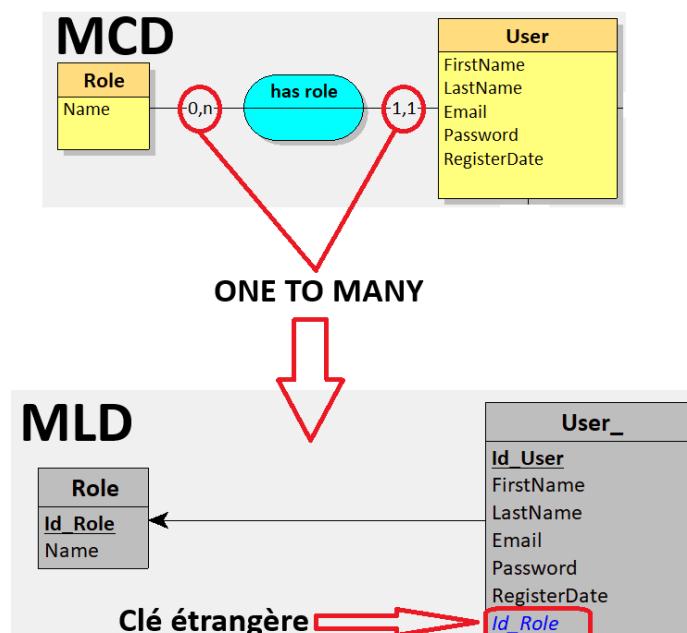
Si une association relie deux tables avec une cardinalité maximale de 1 d'un côté et n de l'autre, alors la table du côté "n" reçoit une clé étrangère vers la table du côté "1".

- Many-to-many :

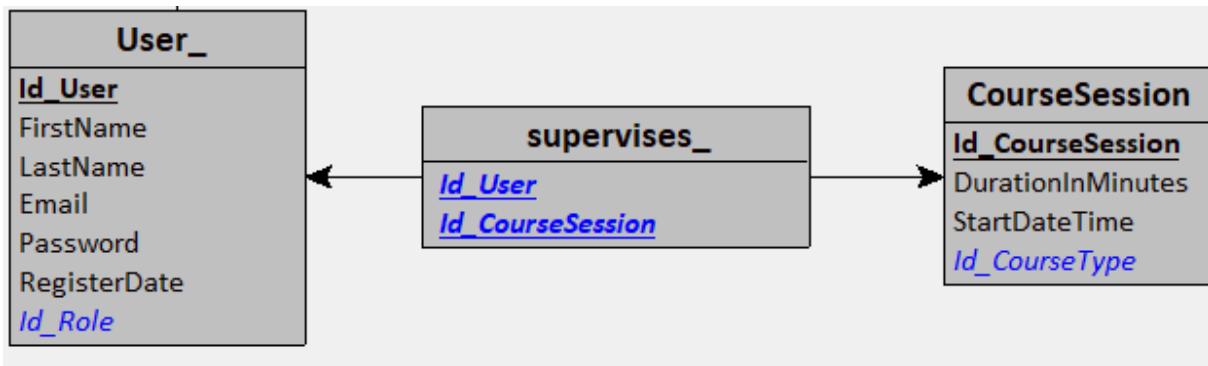
Lorsque la cardinalité maximale est supérieure à 1 des deux côtés, l'association devient une table associative, contenant les clés étrangères des deux tables concernées, et éventuellement ses propres attributs.

- One-to-one :

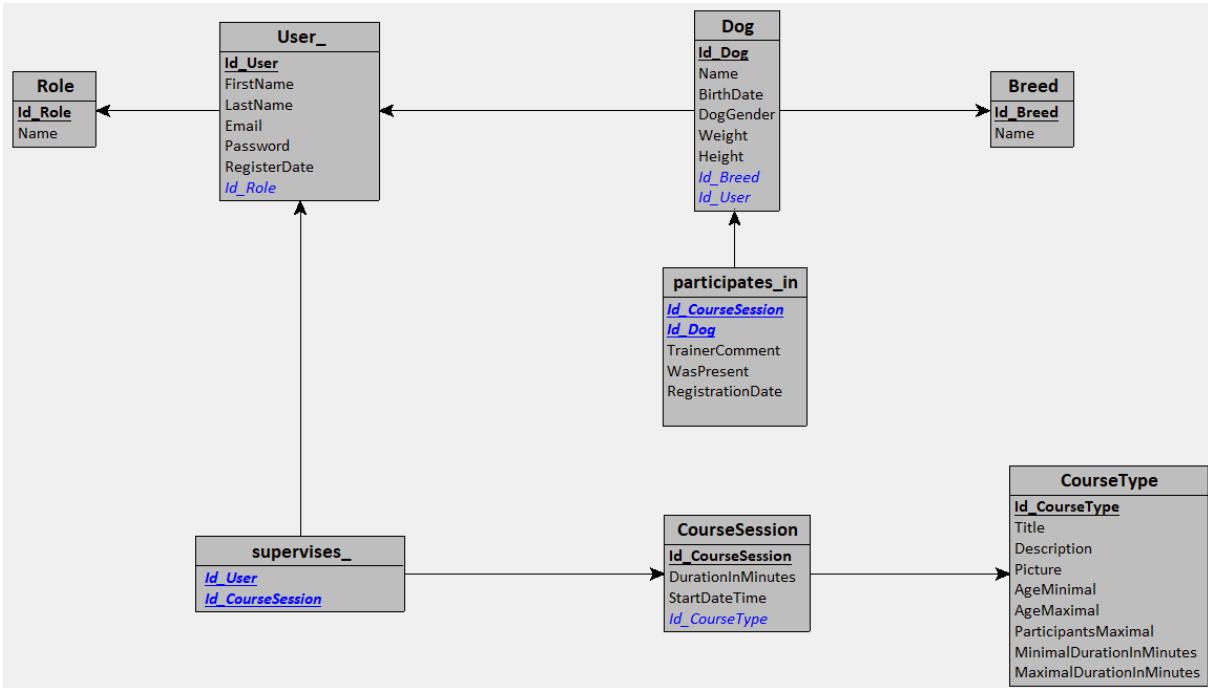
Lorsque la cardinalité maximale est de 1 des deux côtés, on peut fusionner les deux tables si elles sont fortement liées et ne justifient pas une séparation. Sinon, on conserve deux tables, et la table la plus faible (souvent facultative ou dépendante) intègre une clé étrangère vers la table la plus forte.



Exemple de transformation des relations entre le MCD et le MLD sur base des entités Statut Professionnel et Client



Exemple de création d'une entité à la suite d'une relation Many-to-Many



Exemple complet du Modèle Logique des Données d'EduCanin

Modèle Physique des Données - MPD

Le Modèle Physique des Données (MPD) est une représentation détaillée de la structure de la base de données, telle qu'elle sera effectivement implémentée dans un Système de Gestion de Base de Données (SGBD). À ce stade, je définis les **types de données** de chaque attribut, les **longueurs maximales** des champs texte, ainsi que les **contraintes techniques** (clés primaires, clés étrangères, valeurs par défaut, etc.), en accord avec le dictionnaire des données. Cette étape permet de traduire le modèle logique en instructions SQL prêtes à être exécutées dans le SGBD cible.

Dog	
<u>Id_Dog</u>	COUNTER
Name	VARCHAR(50)
BirthDate	DATE
DogGender	INT
Weight	INT
Height	INT
<u>Id_Breed</u>	INT
<u>Id_User</u>	VARCHAR(50)

Table Dog du Modèle Physique de Données (MPD), avec types de données, longueurs et clés.

Exemples de types de données utilisés dans un MPD :

- COUNTER : Représente un **entier auto-incrémenté**, souvent utilisé comme **identifiant unique** dans certains SGBD.
- VARCHAR(50) : Représente une chaîne de caractères de longueur variable, limitée ici à 50 caractères.
- TEXT : Utilisé pour stocker des chaînes de grande longueur, sans limite stricte.
- DATE : Représente une date complète (année, mois, jour).
- DATETIME : Représente une date accompagnée de l'heure (année, mois, jour, heure, minute, seconde).
- INT : Représente un nombre entier, positif ou négatif.

5.5. Création et modification de la base de données

Pour ce projet, j'ai d'abord choisi d'utiliser SQL Server comme système de gestion de base de données. Ce choix s'est naturellement imposé car SQL Server s'intègre parfaitement avec l'écosystème Microsoft, en particulier avec ASP.NET et Entity Framework, que j'utilise pour développer l'application. Cela m'a permis de bénéficier d'un environnement cohérent et bien documenté dès les premières étapes du développement. Cependant, au fil du projet, j'ai pris la décision de migrer vers PostgreSQL. Cette transition s'est justifiée principalement par les possibilités d'hébergement gratuit plus nombreuses et plus accessibles avec PostgreSQL, notamment sur des services comme Render ou Neon.

Pour connecter l'application à la base de données, j'ai utilisé une chaîne de connexion (connection string) stockée dans un fichier de configuration nommé appsettings.Development.json. Ce fichier est destiné uniquement à l'environnement de développement local, et n'est pas versionné dans Git afin de ne pas exposer d'informations sensibles publiquement. Lors du déploiement de l'application en production, j'ai opté pour une méthode plus sécurisée en stockant la chaîne de connexion directement dans une variable d'environnement du serveur, sous la clé ConnectionStrings__DefaultConnection.

Utiliser une variable d'environnement offre une bien meilleure sécurité car elle permet d'isoler les données sensibles du code source. Contrairement à un fichier de configuration, qui peut être accidentellement versionné ou consulté, une variable d'environnement est stockée côté serveur, en dehors du répertoire de l'application. Elle n'est donc pas exposée publiquement ni accessible depuis le navigateur, et ne circule pas dans les fichiers de déploiement. Cela respecte les bonnes pratiques DevOps en matière de séparation des secrets et du code.

Concernant la logique de sélection de la chaîne de connexion, Entity Framework (via le builder de DbContext) suit un ordre de priorité défini : il vérifie d'abord l'existence d'une variable d'environnement, et si celle-ci est présente, elle est utilisée comme source principale. Si aucune variable n'est trouvée, il parcourt ensuite les fichiers de configuration, comme appsettings.json ou appsettings.Development.json, en fonction de l'environnement actif. Ce mécanisme automatique permet à l'ORM de s'adapter dynamiquement au contexte d'exécution, que ce soit en local ou sur un serveur distant, sans nécessiter de modification manuelle du code. Grâce à cette organisation, j'ai pu développer dans un cadre flexible tout en garantissant un niveau de sécurité adapté aux exigences de la mise en production.

Environment Variables

Set environment-specific config and secrets (such as API keys), then read those values from your code. [Learn more](#).

Key	Value
ASPNETCORE_ENVIRONMENT	Production
ConnectionStrings__DefaultConnection	-----

Export ▾ Edit



Capture du panneau de configuration des variables d'environnement sur Render

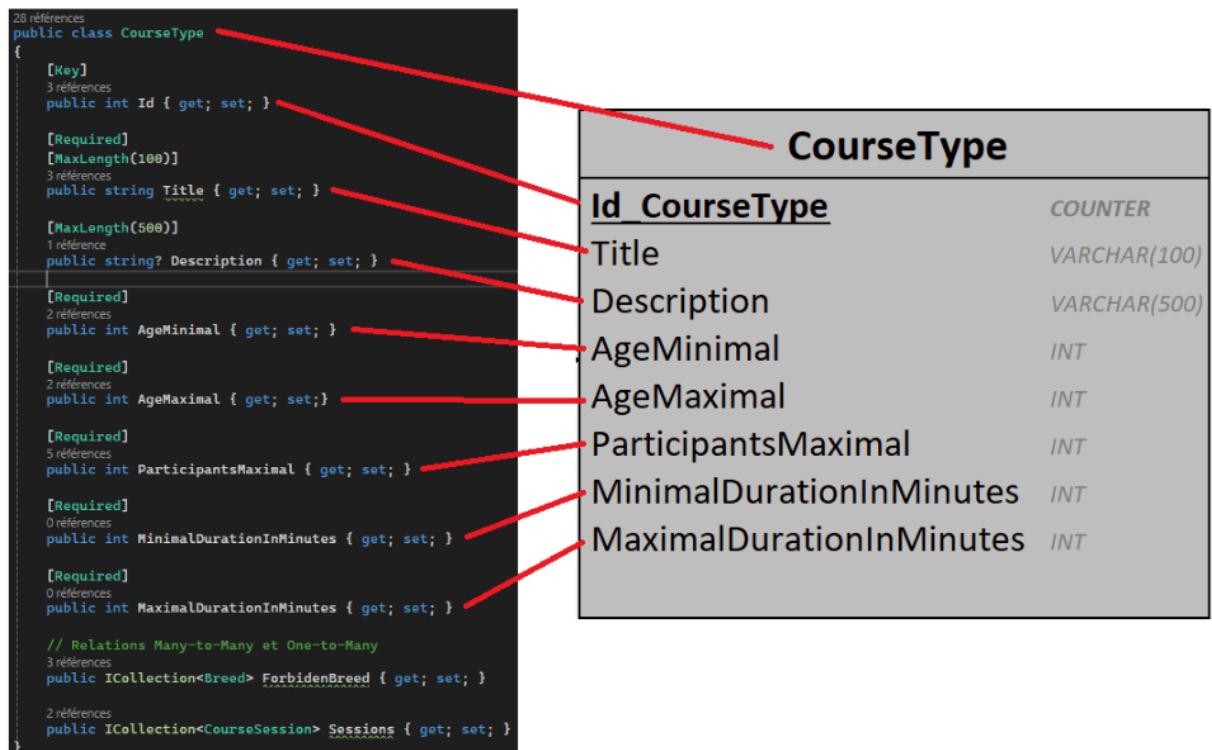
```
var connectionString = builder.Configuration.GetConnectionString("DefaultConnection");
builder.Services.AddDbContext<ApplicationContext>(options =>
    options.UseNpgsql(connectionString));
```

Extrait de code montrant la configuration de la chaîne de connexion dans le builder de services

Ensuite, j'ai créé les modèles. Il s'agit de classes C# conçues pour représenter les entités que l'on retrouve dans le MPD (Modèle Physique de Données). Ces modèles remplissent deux fonctions essentielles : d'une part, ils permettent de manipuler facilement les données en créant des objets en mémoire à l'image des enregistrements en base de données ; d'autre part, ils servent à générer la structure même de la base grâce à l'approche *Code First* d'Entity Framework.

Grâce à cet ORM, j'ai pu utiliser les *Data Annotations*, des attributs placés directement sur les propriétés des classes pour guider la création des tables. Ces annotations offrent à Entity Framework des indications précieuses, comme par exemple :

- [Key] pour définir une clé primaire ;
- [Required] pour rendre une propriété obligatoire (non nullable) ;
- [MaxLength(50)] pour limiter la taille d'une chaîne de caractères ;
- [Range(1, 100)] pour imposer une plage de valeurs à un champ numérique ;
- [ForeignKey] pour indiquer explicitement une clé étrangère.



Comparaison du modèle Document et de sa table créée avec Entity.

Migration

Après avoir soigneusement conçu mes modèles de données pour qu'ils soient conformes à mon modèle physique de données (MPD), j'ai lancé le processus de migration avec Entity Framework Core. Celui-ci permet de traduire automatiquement la structure des classes C# en instructions compréhensibles par le moteur de base de données. À partir de la configuration de mon DbContext, le framework analyse l'ensemble des entités et prépare un fichier contenant les opérations nécessaires à la création ou la mise à jour du schéma.

Pour cela, deux étapes principales sont réalisées depuis la console de l'IDE :

- **dotnet ef migrations add InitialCreate** : cette commande génère un fichier de migration contenant du code C# qui décrit les opérations à appliquer à la base de données (création de tables, colonnes, clés étrangères, etc.). Ce fichier représente une image de l'état actuel des modèles et sert de point de référence pour les futures évolutions.
- **dotnet ef database update** : cette commande établit la connexion avec le serveur de base de données et applique les instructions définies dans la migration en les traduisant en SQL. C'est à ce moment que les objets sont réellement créés ou modifiés dans la base.

Ce système de migration permet de suivre l'évolution du modèle de données tout au long du projet. À chaque nouvelle modification, une nouvelle migration peut être générée ; Entity Framework se charge alors de comparer les différences avec l'état précédent et d'appliquer uniquement les changements nécessaires.

```
migrationBuilder.CreateTable(
    name: "BreedCourseType",
    columns: table => new
    {
        CourseTypesWithRestrictionId = table.Column<int>(type: "integer", nullable: false),
        ForbidenBreedId = table.Column<int>(type: "integer", nullable: false)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_BreedCourseType", x => new { x.CourseTypesWithRestrictionId, x.ForbidenBreedId });
        table.ForeignKey(
            name: "FK_BreedCourseType_Breeds_ForbidenBreedId",
            column: x => x.ForbidenBreedId,
            principalTable: "Breeds",
            principalColumn: "Id",
            onDelete: ReferentialAction.Cascade);
        table.ForeignKey(
            name: "FK_BreedCourseType_CourseTypes_CourseTypesWithRestrictionId",
            column: x => x.CourseTypesWithRestrictionId,
            principalTable: "CourseTypes",
            principalColumn: "Id",
            onDelete: ReferentialAction.Cascade);
    });
}
```

Exemple de code généré automatiquement par Entity Framework dans un fichier de migration

5.6. Le diagramme du comportement des fonctionnalités de type cas d'utilisations

L'application du Club Canin repose sur trois types d'acteurs clairement définis. Ces rôles permettent de structurer les droits d'accès et les fonctionnalités disponibles pour chaque utilisateur selon son profil, tout en facilitant la gestion du club. Chaque rôle correspond à un besoin spécifique dans le fonctionnement de la plateforme. Voici une présentation détaillée des trois rôles principaux :

- **Propriétaire** : C'est le rôle attribué par défaut lorsqu'un utilisateur s'inscrit au club. Il peut créer et modifier son profil, ajouter les informations relatives à ses chiens (âge, race, sexe, etc.), consulter le calendrier des cours, et inscrire ses animaux aux séances disponibles. L'accès aux cours est conditionné par l'âge de l'animal et la disponibilité des places, conformément aux règles définies par le club. Le propriétaire peut également consulter ses réservations et recevoir des notifications.
- **Coach** : Le coach est un membre du personnel encadrant chargé de la gestion et de l'animation des séances. Il a accès à son propre profil, peut visualiser les sessions auxquelles il est assigné, consulter la liste des chiens inscrits à chaque cours, signaler une absence ou un comportement particulier, et valider la présence des participants. Il peut aussi accéder à certaines données des chiens pour adapter les séances à leur niveau ou à leurs besoins spécifiques.
- **Administrateur (Responsable du club)** : L'administrateur dispose de l'ensemble des droits sur l'application. Il gère les utilisateurs (validation, modification ou suppression de compte), supervise les inscriptions, définit les types de cours et les critères d'éligibilité (âge, race, durée, nombre maximal de participants), et planifie les sessions. Il peut également consulter les statistiques de participation, gérer les absences signalées, créer de nouveaux rôles si nécessaire, et contrôler le bon déroulement des activités du club via un panneau d'administration dédié.

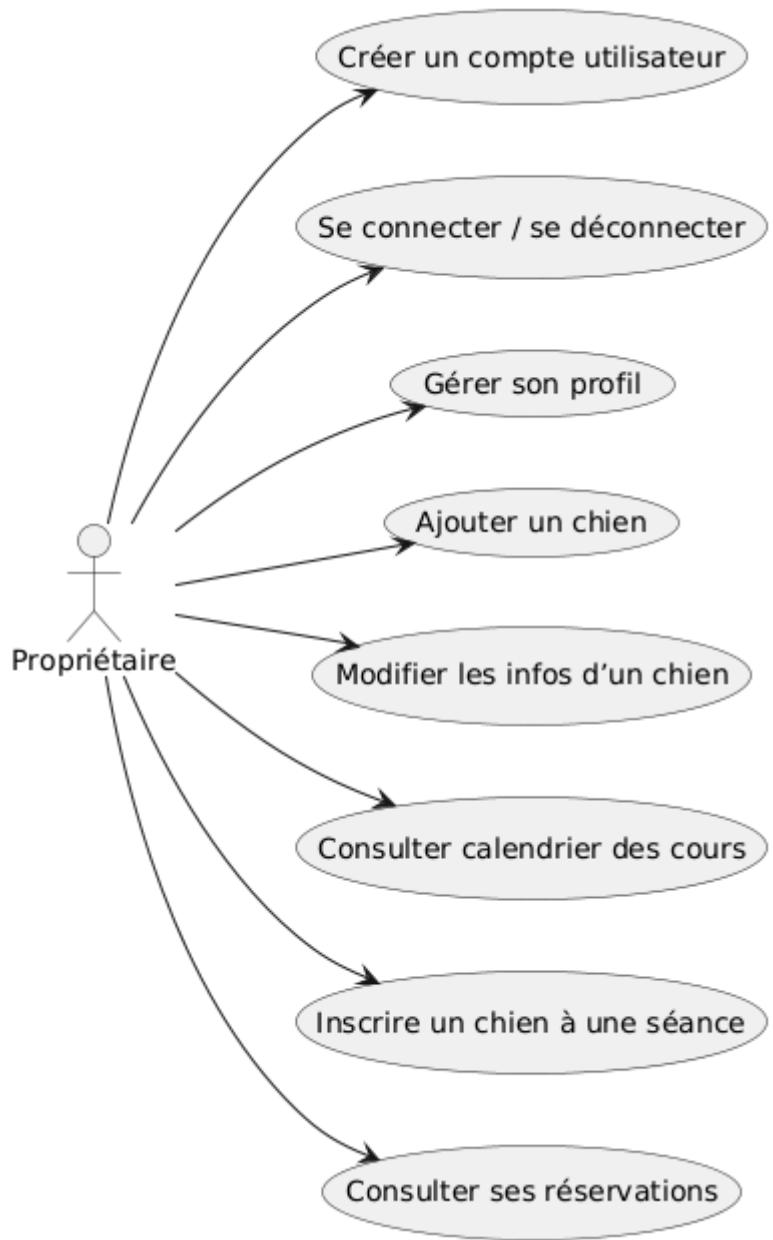


Diagramme des cas d'utilisations de l'acteur "Propriétaire"

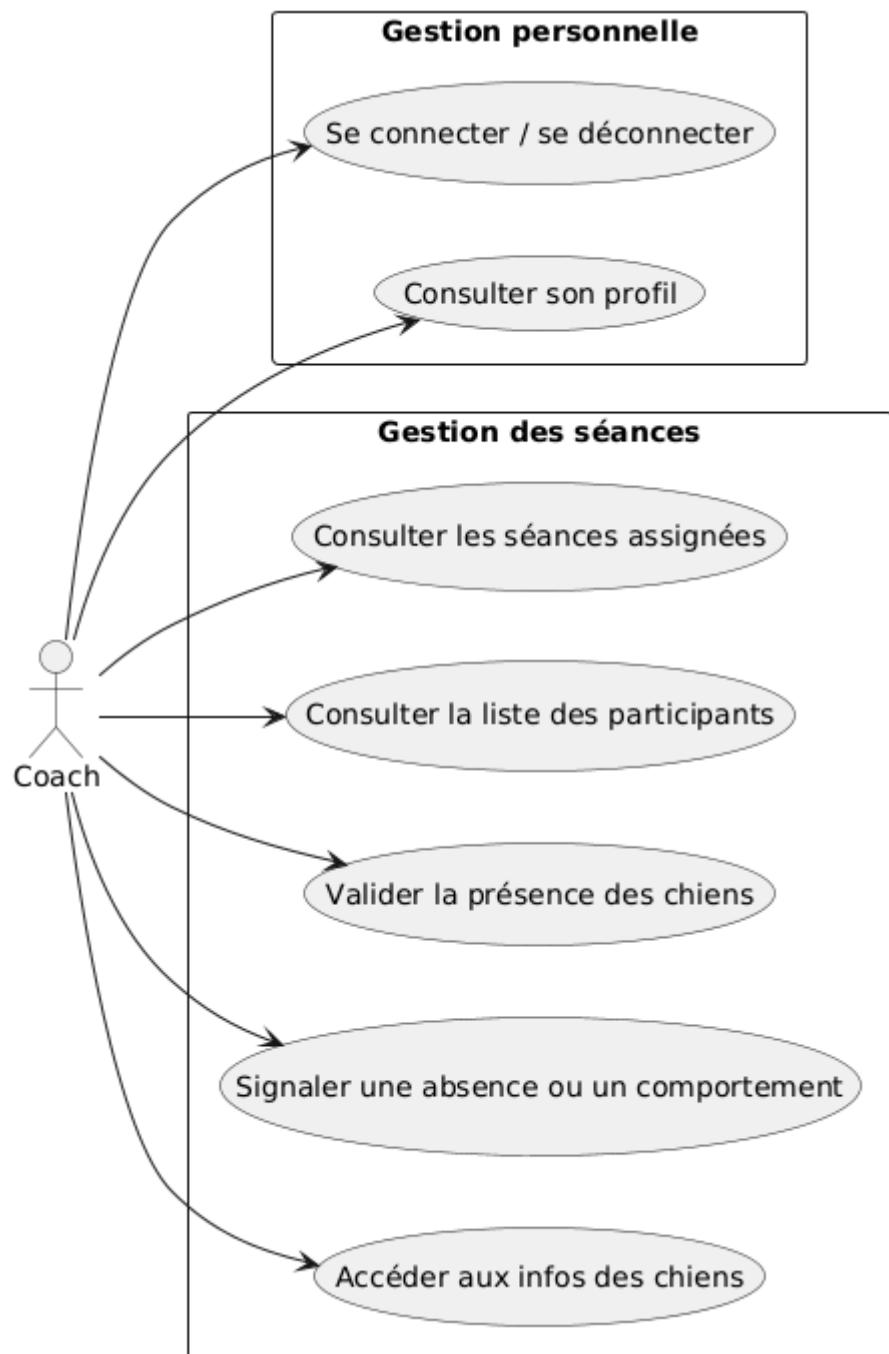


Diagramme des cas d'utilisations de l'acteur "Coach"

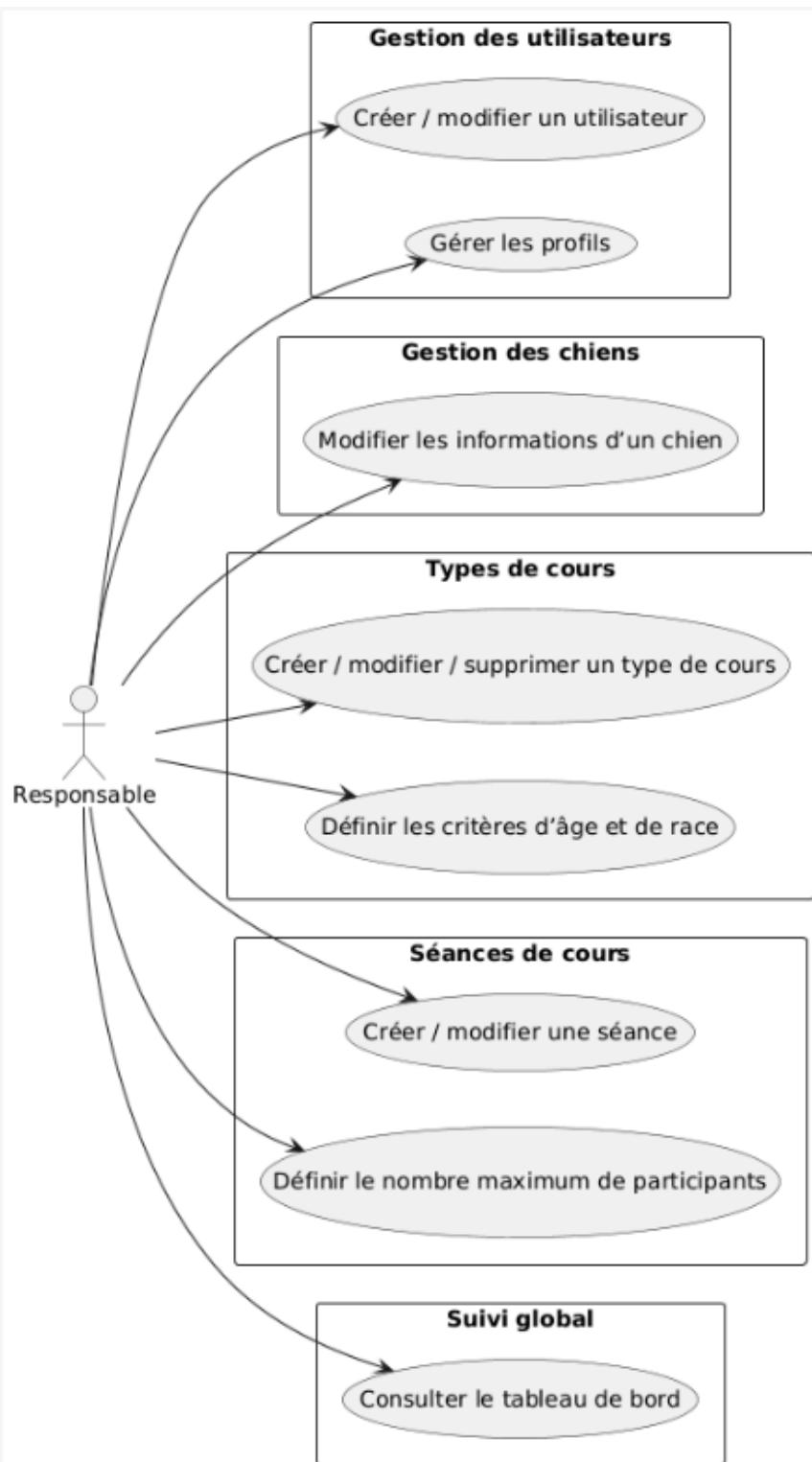


Diagramme des cas d'utilisations de l'acteur “Administrateur”

5.7. Le diagramme du détail des cas d'utilisations les plus significatifs de type diagramme de séquence.

Un diagramme des cas d'utilisation est un outil de modélisation utilisé en conception logicielle pour représenter visuellement les interactions entre les utilisateurs (appelés acteurs) et le système. Il met en évidence les différentes fonctionnalités ou services accessibles selon chaque type d'acteur. Ce type de diagramme permet de clarifier les exigences fonctionnelles et de présenter de façon simple et lisible les capacités du système ainsi que les relations entre les rôles et les actions possibles.

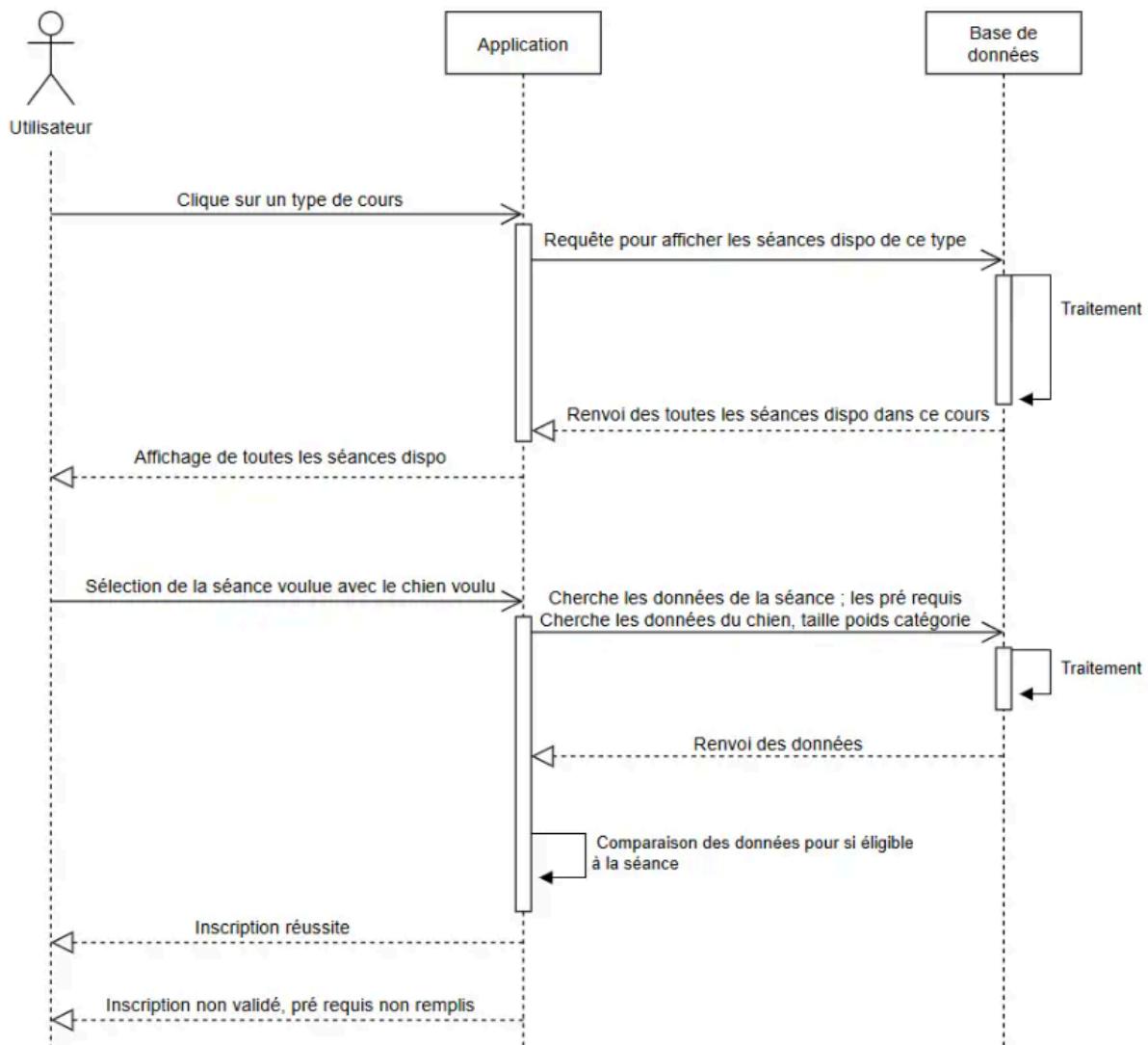


Diagramme de séquence illustrant la fonctionnalité de réservation d'une séance de cours par un propriétaire.

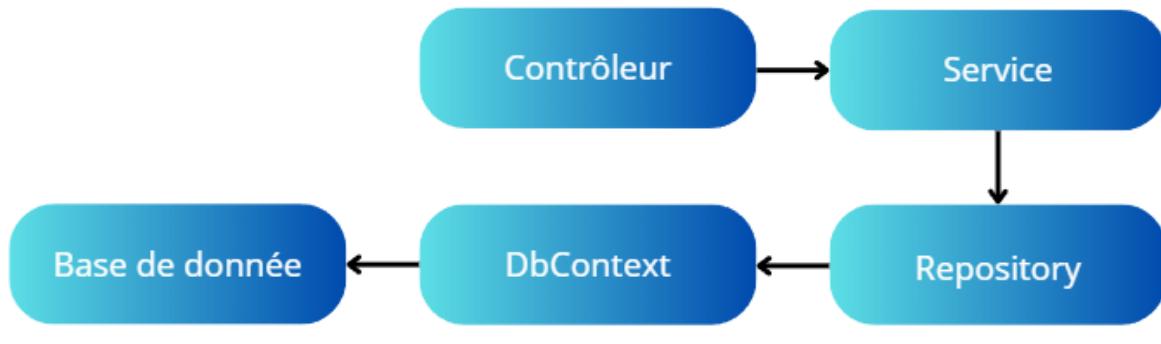
6. Les spécifications techniques du projet

Tout au long du développement, et plus particulièrement lors de la phase de conception, j'ai été amené à effectuer des choix techniques structurants. Ces décisions ont été guidées par plusieurs facteurs : les besoins fonctionnels exprimés dans le cahier des charges, les contraintes de temps et de maintenabilité, ainsi que mes préférences personnelles en termes d'outils et de technologies. Mon objectif a toujours été de trouver le bon équilibre entre efficacité, clarté du code, performance de l'application, et respect des standards professionnels. Cette section a pour but de présenter les principales orientations techniques prises dans le cadre du projet, en expliquant les raisons de chaque choix et les avantages que j'en ai tirés.

C# et ASP.NET Core MVC

Le langage C# a été au cœur du développement de cette application. Je l'ai découvert et approfondi dans le cadre de ma formation, et j'ai rapidement pris goût à sa syntaxe claire, sa puissance et sa modernité. Des fonctionnalités comme LINQ, les propriétés avec getter/setter automatiques ou encore la gestion des collections facilitent énormément l'écriture de code propre et lisible. Mais au-delà de la syntaxe, c'est l'ensemble de l'écosystème .NET qui m'a convaincu : il offre des outils robustes pour créer des applications web, concevoir des API, interagir avec des bases de données ou encore structurer un projet selon les principes du génie logiciel.

J'ai choisi d'organiser mon application selon le pattern **MVC (Model - View - Controller)**, enrichi par deux couches supplémentaires : **les services** et **les repositories**. Cette structure m'a permis de bien séparer les responsabilités au sein de l'application. Le contrôleur reçoit et gère les requêtes, le service encapsule la logique métier, et le repository s'occupe de la communication avec la base de données via Entity Framework. Cette organisation respecte les principes SOLID, améliore la lisibilité du projet, facilite les tests unitaires, et permet de faire évoluer facilement les différentes couches sans impacter l'ensemble de l'application.



Entity Framework et SQL Server / PostgreSQL

J'ai utilisé **Entity Framework** comme ORM pour faciliter l'accès à la base de données. Cette bibliothèque m'a permis de manipuler mes données sous forme d'objets, ce qui rend le code plus fluide et réduit considérablement la quantité de requêtes SQL écrites à la main. J'ai d'abord travaillé avec **SQL Server**, qui s'intègre naturellement à l'environnement .NET. Par la suite, j'ai migré vers **PostgreSQL** pour bénéficier d'options d'hébergement gratuit plus accessibles, tout en conservant la compatibilité avec Entity Framework.

ASP.NET Identity

Pour la gestion des utilisateurs, j'ai choisi d'utiliser ASP.NET Identity, une solution native et largement utilisée dans l'écosystème .NET. Bien que riche et parfois complexe, j'en ai utilisé uniquement les fonctionnalités essentielles : inscription, connexion, et sécurisation de l'accès aux pages. Ce système m'a permis d'implémenter rapidement une base d'authentification robuste et sécurisée, tout en restant ouvert à des personnalisations futures si nécessaire. Son intégration fluide dans ASP.NET Core, ainsi que sa documentation abondante, m'ont facilité la prise en main.

```
IdentityResult result = await _userManager.CreateAsync(newUser, model.Password);
if (!result.Succeeded)
    return result;

if (!await _roleManager.RoleExistsAsync("User"))
{
    await _roleManager.CreateAsync(new IdentityRole("User"));

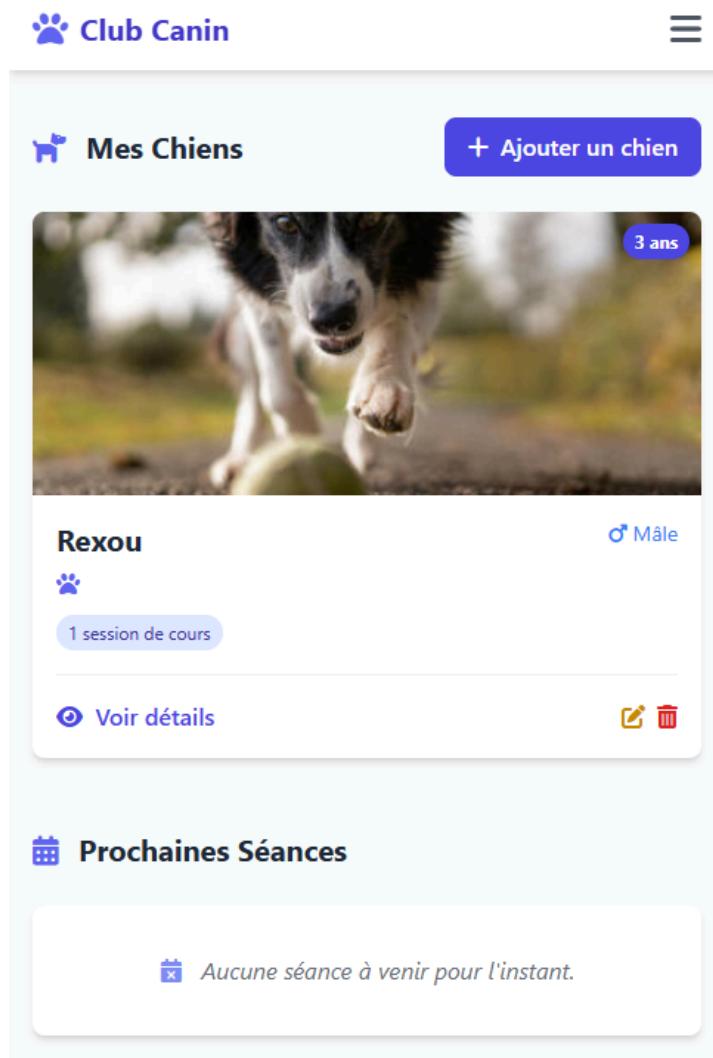
    await _userManager.AddToRoleAsync(newUser, "User");
    await _signInManager.SignInAsync(newUser, isPersistent: false);
}

return IdentityResult.Success;
```

Création d'un utilisateur avec identity

Tailwind CSS, HTML, CSS et JavaScript

Pour la partie visuelle de l'application, j'ai opté pour **Tailwind CSS**, un framework moderne et utilitaire qui permet de construire des interfaces rapidement et avec précision. Contrairement à Bootstrap que j'utilisais auparavant, Tailwind m'a offert plus de flexibilité dans le design sans imposer de composants prédéfinis. Cela m'a permis de créer des pages plus légères, mieux personnalisées et plus cohérentes avec l'identité visuelle du projet. L'interface est responsive, accessible sur tous types d'écrans, et enrichie par quelques scripts **JavaScript** pour assurer une meilleure fluidité des interactions. L'ensemble repose sur une base solide en **HTML** et **CSS**, garantissant une bonne compatibilité entre navigateurs.



Screenshot du DashBoard d'Educanin dans un format d'écran de téléphone.

7. Les réalisations du candidat comportant les extraits de code les plus significatifs, les arguments qui ont conduit à ses choix, y compris pour la sécurité

7.1. Les captures d'écran d'interfaces utilisateur et le code correspondant

Razor Views

Comme précisé plus haut, j'ai utilisé des vues Razor, qui permettent d'injecter du code C# directement dans le HTML pour dynamiser le contenu. Ce code peut faire référence à des données provenant du "code-behind", c'est-à-dire à la logique métier écrite dans le contrôleur ou le ViewModel. Ce fonctionnement rappelle celui de PHP, où l'on mélange code et affichage, mais la solution de Microsoft me semble plus lisible et mieux structurée, notamment grâce à la séparation claire entre la vue et la logique.

```
<div class="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-6 mb-8">
    @foreach (CourseViewModel courseViewModel in Model)
    {
        <div class="course-card bg-white rounded-lg shadow-md overflow-hidden transition duration-300 card-hover"
            data-category="@courseViewModel.DataCategoryFilter">
            <div class="p-6">
                <div class="flex items-center mb-4">
                    <div class="@courseViewModel.BoxIconTailwindClass">
                        <i class="@courseViewModel.IconTailwindClass"></i>
                    </div>
                    <h3 class="text-xl font-bold text-gray-800">@courseViewModel.Title</h3>
                </div>
                <p class="text-gray-600 mb-4">@courseViewModel.Description</p>
                <a asp-action="CourseSessionByCourseType"
                    asp-controller="CourseSession"
                    asp-route-courseTypeId="@courseViewModel.Id"
                    class="@courseViewModel.ButtonTailwindClass inline-flex items-center justify-center cursor-pointer focus:outline-none">
                    Voir les séances <i class="fas fa-arrow-right ml-2"></i>
                </a>
            </div>
        </div>
    }
</div>
```

 **École du chiot**

Apprentissage des bases pour les chiots de 2 à 6 mois. Socialisation, propreté et premiers ordres dans une ambiance ludique.

[Voir les séances →](#)

 **Éducation 6-12 mois**

Perfectionnement des ordres de base et apprentissage de nouveaux comportements pour les jeunes chiens en pleine croissance.

[Voir les séances →](#)

 **Éducation 1-2 ans**

Cours intensifs pour chiens adolescents. Renforcement des acquis et travail sur la concentration malgré les distractions.

[Voir les séances →](#)

 **Éducation +2 ans**

Pour les chiens adultes, perfectionnement avancé et activités spécifiques (agility, obéissance rythmée, etc.).

[Voir les séances →](#)

 **Chiots spécial famille**

Cours adapté aux familles avec enfants. Apprentissage des interactions sécuritaires entre chiots et jeunes enfants.

[Voir les séances →](#)

 **Cours avancé**

Pour les chiens ayant validé tous les niveaux précédents. Préparation aux compétitions et activités canines spécialisées.

[Voir les séances →](#)

 **Défense & protection (Chiens d'attaque)**

Cours spécialisés pour les chiens de type défense (ex : Malinois, Rottweiler, Dobermann). Apprentissage de l'obéissance en situation de stress, contrôle de la morsure, protection maîtrisée.

[Voir les séances →](#)

 **Instinct de troupeau (Chiens de berger)**

Destiné aux chiens de berger (ex : Border Collie, Berger Australien). Exercices de conduite, canalisation de l'instinct de rassemblement, travail à distance.

[Voir les séances →](#)

 **Réactivité & canalisation (Chiens primitifs)**

Cours pour chiens primitifs (ex : Husky, Shiba Inu, Akita). Apprentissage du rappel, de la gestion de la frustration, et du lien renforcé avec le maître.

[Voir les séances →](#)

Affichage du code et du rendu visuel correspondant

- **@foreach (CourseViewModel courseViewModel in Model)** est une boucle en C# car elle est précédée du symbole @. Elle va parcourir l'ensemble de la collection Model qui contient des objets CourseViewModel. À chaque tour de boucle, une variable temporaire courseViewModel prend la valeur d'un élément de la liste, permettant de générer automatiquement et dynamiquement une carte de cours pour chaque entrée.
- **<div class="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-6 mb-8">** indique que l'on utilise une grille responsive grâce à Tailwind CSS. Le contenu de l'attribut class définit plusieurs styles : grid active l'affichage en grille, grid-cols-1, md:grid-cols-2 et lg:grid-cols-3 adaptent le nombre de colonnes selon la taille de l'écran (1 sur mobile, 2 sur tablette, 3 sur grand écran). gap-6 ajoute un espace entre les cartes et mb-8 une marge inférieure pour aérer l'affichage.
- **<div class="course-card bg-white rounded-lg shadow-md overflow-hidden transition duration-300 card-hover" data-category="@courseViewModel.DataCategoryFilter">** crée le contenant de chaque carte. On retrouve ici plusieurs classes Tailwind : bg-white applique un fond blanc, rounded-lg des coins arrondis, shadow-md une ombre légère et overflow-hidden masque le contenu qui dépasserait. transition duration-300 apporte une animation douce au survol et card-hover peut contenir des effets personnalisés. L'attribut data-category stocke une information (la catégorie du cours) pour d'éventuelles fonctions JavaScript de filtrage.
- **<div class="flex items-center mb-4">** utilise le système Flexbox pour aligner horizontalement l'icône et le titre. À l'intérieur, le **@courseViewModel.BoxIconTailwindClass** applique un style

particulier (par exemple un fond coloré ou une forme) à l'icône, et l'élément `<i class="@courseViewModel.IconTailwindClass"></i>` affiche l'icône correspondant au cours. Le titre `<h3 class="text-xl font-bold text-gray-800">@courseViewModel.Title</h3>` met en avant le nom du cours avec une taille de texte plus grande (text-xl), en gras (font-bold) et en gris foncé (text-gray-800).

- `<p class="text-gray-600 mb-4">@courseViewModel.Description</p>` affiche la description du cours dans un texte gris plus doux (text-gray-600) et ajoute un petit espace en bas (mb-4).
- `<a asp-action="CourseSessionByCourseType" asp-controller="CourseSession" asp-route-courseTypeId="@courseViewModel.Id" class="@courseViewModel.ButtonTailwindClass inline-flex items-center justify-center cursor-pointer focus:outline-none">` est un lien HTML stylisé comme un bouton. Il utilise des Tag Helpers ASP.NET Core (asp-action, asp-controller, asp-route-courseTypeId) pour générer automatiquement une URL vers l'action CourseSessionByCourseType du contrôleur CourseSession, en passant l'ID du cours sélectionné. La classe `@courseViewModel.ButtonTailwindClass` adapte le style du bouton (par exemple couleur ou taille) selon le cours. Enfin, l'icône `<i class="fas fa-arrow-right ml-2"></i>` de FontAwesome ajoute une flèche après le texte "Voir les séances".

The screenshot shows the Club Canin login page. Red arrows point from specific UI elements to their corresponding code snippets in the provided HTML source.

- Email:** Points to the email input field in the UI and its corresponding code snippet in the HTML source.
- Password:** Points to the password input field in the UI and its corresponding code snippet in the HTML source.
- Se souvenir de moi:** Points to the remember me checkbox in the UI and its corresponding code snippet in the HTML source.
- Mot de passe oublié ?**: Points to the forgot password link in the UI and its corresponding code snippet in the HTML source.
- Se connecter:** Points to the "Se connecter" button in the UI and its corresponding code snippet in the HTML source.

```

<form asp-action="Login" asp-controller="Account" method="post" class="p-6 space-y-6">
    @Html.AntiForgeryToken()

    <div>
        <label asp-for="Email" class="block text-sm font-medium text-gray-700 mb-1"></label>
        <div class="relative">
            <div class="absolute inset-y-0 left-0 pl-3 flex items-center pointer-events-none">
                | <i class="fas fa-envelope input-icon"></i>
            </div>
            <input asp-for="Email" class="block w-full pl-10 pr-3 py-2 border border-gray-300 rounded-lg shadow-sm focus:outline-none focus:ring-2 focus:ring-blue-500 focus:border-blue-500" placeholder="votre@email.com" />
            <span asp-validation-for="Email" class="text-red-500 text-sm"></span>
        </div>
    </div>

    <div>
        <label asp-for="Password" class="block text-sm font-medium text-gray-700 mb-1"></label>
        <div class="relative">
            <div class="absolute inset-y-0 left-0 pl-3 flex items-center pointer-events-none">
                | <i class="fas fa-lock input-icon"></i>
            </div>
            <input asp-for="Password" class="block w-full pl-10 pr-3 py-2 border border-gray-300 rounded-lg shadow-sm focus:outline-none focus:ring-2 focus:ring-blue-500 focus:border-blue-500" placeholder="*****" />
            <span asp-validation-for="Password" class="text-red-500 text-sm"></span>
        </div>
    </div>

    <div class="flex items-center justify-between">
        <div class="flex items-center">
            <input asp-for="RememberMe" type="checkbox" class="h-4 w-4 text-blue-600 focus:ring-blue-500 border-gray-300 rounded">
            <label asp-for="RememberMe" class="ml-2 block text-sm text-gray-700"></label>
        </div>
        <div class="text-sm">
            <a href="#" class="font-medium text-blue-600 hover:text-blue-500">Mot de passe oublié ?</a>
        </div>
    </div>

    <div>
        <button type="submit" class="btn-primary w-full flex justify-center py-2 px-4 border border-transparent rounded-lg shadow-sm text-sm font-medium text-white bg-blue-600 hover:bg-blue-700 focus:outline-none focus:ring-2 focus:ring-offset-2">Se connecter</button>
    </div>
</form>

```

Code lié à la vie destiné à la connexion des utilisateurs

7.2. Des extraits de code de composants métier

Processus d'inscription à un cours

Dans les exemples ci-dessous, l'on peut voir une présentation en détail et étape par étape d'une fonctionnalité qui permet à un utilisateur d'inscrire son chien à une séance de cours. L'utilisateur commence par se rendre dans la section "cours". Il doit d'abord sélectionner un type de cours (par exemple : école du chiot, instinct de troupeau).

Une fois ce type choisi, la page affiche automatiquement toutes les séances disponibles correspondant à ce cours. L'utilisateur doit ensuite choisir une séance précise. À ce moment-là, le système effectue plusieurs vérifications liées à la date : il s'assure que la séance n'est pas déjà passée, qu'elle n'est pas complète et qu'elle est encore ouverte aux inscriptions.

Une fois la séance validée, l'utilisateur sélectionne le chien qu'il souhaite inscrire. De nouvelles vérifications sont faites sur le chien : son âge (certains cours exigent un âge minimum ou maximum) et sa race (certaines races ne sont pas acceptées dans certains types de cours).

Si toutes les conditions sont remplies, l'inscription est enregistrée en base de données. Elle apparaît immédiatement sur le tableau de bord de l'utilisateur, dans la rubrique "Mes séances à venir", ce qui lui permet de consulter facilement toutes les prochaines séances auxquelles ses chiens sont inscrits.

Nos Cours Canins

Filtrer par âge :

- Tous les cours**
- Chiots (2-6 mois)
- Jeunes (6+ mois)
- Ados / Adultes (12+ mois)
- Avancés (24+ mois)

Éducation +2 ans

Pour les chiens adultes, perfectionnement avancé et activités spécifiques (agility, obéissance rythmée, etc.).

Voir les séances →

Chiot spécial famille

Cours adapté aux familles avec enfants. Apprentissage des interactions sécuritaires entre chiots et jeunes enfants.

Voir les séances →

Filtrer les séances :

Date :

Disponibilité :

Appliquer ▾

vendredi 08 août 2025 **6 place(s)**

08:45 - 09:56

Justine

S'inscrire 🐾

vendredi 08 août 2025 **6 place(s)**

10:30 - 11:41

Jean

S'inscrire 🐾

Inscription à la séance : Éducation +2 ans

Merci de sélectionner le chien à inscrire ci-dessous

Détails de la séance

Date : Vendredi 08 août 2025

Heure : 08:45 - 09:56

Durée : 71 minutes

Places restantes : 6 / 6

Prérequis

- Âge minimum : 25 mois
- Âge maximum : 120 mois

Choix du chien

Sélectionner un chien :

Inscrire ce chien

Coach

Justine
Éducateur canin certifié
Éducation +2 ans

Mes Chiens

Rexou 3 ans

2 sessions de cours

Voir détails

Prochaines Séances

Vendredi 08 août 2025 à 08:45
 Éducation +2 ans
 Justine Lola
 Rexou

Historique des Cours

Vendredi 25 Juillet 2025 à 15:30
 Éducation +2 ans
 Justine Lola
 Rexou

Vues de la fonctionnalité qui permet de réserver un cours

Gestion du formulaire

L'extrait de code ci-dessous représente une action `HttpPost` d'un contrôleur ASP.NET MVC, permettant à un utilisateur connecté d'inscrire son chien à une session de cours. L'action commence par vérifier si le modèle reçu est valide. En cas d'erreur de validation, le contrôleur recharge les données nécessaires à l'affichage du formulaire via un service, puis retourne la vue avec un `ViewModel` mis à jour. Si le modèle est valide, une vérification métier est effectuée pour savoir si le chien peut effectivement être inscrit à la session. Si cette vérification échoue, un message d'erreur est ajouté au `ModelState`, et la vue est renvoyée avec les informations actualisées.

Dans le cas où l'inscription est autorisée, un nouvel objet représentant l'inscription (`DogCourseSession`) est créé, initialisé avec l'identifiant du chien, celui de la session, la date du jour et un indicateur de présence par défaut. Cet objet est ensuite enregistré en base de données via un service dédié. L'utilisateur est enfin redirigé vers la page d'index des cours. Ce contrôleur suit les bonnes pratiques MVC en déléguant la logique métier aux services, en utilisant un `ViewModel` pour structurer les données échangées avec la vue, et en assurant un traitement asynchrone des opérations.

```
[HttpPost]
[ValidateAntiForgeryToken]
0 références
public async Task<IActionResult> CourseSessionRegister(DogCourseSessionRegistrationViewModel dogCourseSessionRegistrationViewModel)
{
    string? userId = User.FindFirstValue(ClaimTypes.NameIdentifier);
    if (!ModelState.IsValid)
    {
        DogCourseSessionRegistrationViewModel? dogCourseSessionRegistrationViewModelRefresh = await _courseSessionService.GetInformationForRegister(dogCourseSessionRegistrationViewModel.CourseSessionId, userId);
        return View(dogCourseSessionRegistrationViewModelRefresh);
    }

    string statusMessage = await _courseSessionService.CheckIfDogCanBeRegisteredAsync(dogCourseSessionRegistrationViewModel.SelectedDogId, dogCourseSessionRegistrationViewModel.CourseSessionId);
    TempData["SuccessMessage"] = statusMessage;

    if (statusMessage != "Inscription validée !")
    {
        ModelState.AddModelError("", statusMessage);
        DogCourseSessionRegistrationViewModel? refreshedViewModel = await _courseSessionService.GetInformationForRegister(dogCourseSessionRegistrationViewModel.CourseSessionId, userId);
        return View(refreshedViewModel);
    }

    DogCourseSession registration = new DogCourseSession
    {
        DogId = dogCourseSessionRegistrationViewModel.SelectedDogId,
        CourseSessionId = dogCourseSessionRegistrationViewModel.CourseSessionId,
        RegistrationDate = DateTime.UtcNow,
        WasPresent = false
    };

    await _dogCourseSessionService.AddAsync(registration);

    return RedirectToAction("Index", "Course");
}
```

Validation métier

La méthode CheckIfDogCanBeRegisteredAsync est une méthode métier asynchrone du service, chargée de valider les conditions nécessaires à l'inscription d'un chien à une session de cours. Elle commence par récupérer, via les repositories, le chien et la session concernés. Si l'un des deux est introuvable, un message d'erreur spécifique est immédiatement retourné. Une série de vérifications est ensuite effectuée : la méthode contrôle d'abord si le chien est déjà inscrit à cette session.

Elle poursuit avec des vérifications métier supplémentaires : elle s'assure que la race du chien n'est pas interdite pour ce type de cours, que le nombre maximal de participants n'est pas déjà atteint, puis que l'âge du chien est bien compris dans l'intervalle autorisé par le cours. À chaque étape, si une condition n'est pas respectée, un message explicite est retourné pour informer l'utilisateur. Si toutes les conditions sont remplies, la méthode retourne un message de validation. Cette méthode centralise donc la logique de validation d'inscription, rendant le contrôleur plus lisible et facilitant la maintenance de la logique métier.

```
2 références
public async Task<string> CheckIfDogCanBeRegisteredAsync(int dogId, int courseSessionId)
{
    Dog? dog = await _dogRepository.GetByIdAsync(dogId);
    CourseSession? courseSession = await _courseSessionRepository.GetByIdAsyncWithAll(courseSessionId);

    if(dog == null )
    {
        return "Ce chien est introuvable.";
    }

    if (courseSession == null)
    {
        return "La séance est introuvable.";
    }

    bool isAlreadyRegistered = courseSession.DogParticipants.Any(participant => participant.DogId == dogId);
    if (isAlreadyRegistered)
    {
        return "Ce chien est déjà inscrit à cette séance.";
    }

    bool isForbiddenBreed = courseSession.CourseType.ForbiddenBreed.Any(breed => breed.Id == dog.BreedId);
    if (isForbiddenBreed)
    {
        return "La race de ce chien est interdite pour ce cours.";
    }

    int maximalParticipants = courseSession.CourseType.ParticipantsMaximal;
    bool isFull = courseSession.DogParticipants.Count >= maximalParticipants;
    if (isFull)
    {
        return "Il n'y a plus de place disponible pour cette séance.";
    }

    if ((dog.AgeInMonths) < courseSession.CourseType.AgeMinimal)
    {
        return "Votre chien est trop jeune pour ce cours.";
    }

    if ((dog.AgeInMonths) > courseSession.CourseType.AgeMaximal)
    {
        return "Votre chien est trop âgé pour ce cours.";
    }

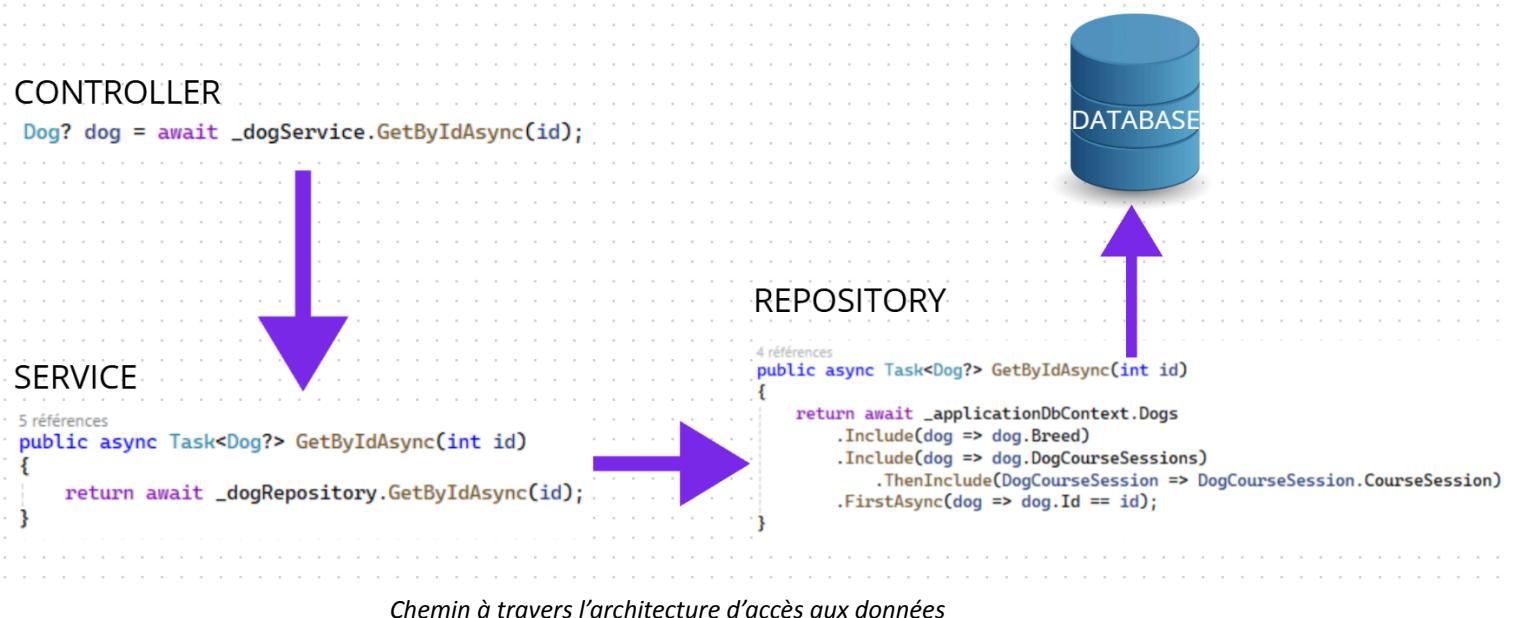
    return "Inscription validée !";
}
```

7.3. Des extraits de code de composants d'accès aux données

Organisation des couches

Dans Educanin, la majorité des pages de l'application nécessitent des données issues de la base pour être générées dynamiquement. J'ai donc accordé une attention particulière à la conception de la couche d'accès aux données, afin de garantir un code structuré, maintenable et évolutif. Pour cela, j'ai adopté le pattern service-repository (voir partie 6. Les spécifications techniques du projet), qui permet de séparer proprement les responsabilités entre les différentes couches de l'application. Ce choix architectural s'inscrit dans une démarche de respect des principes SOLID, en particulier le principe de responsabilité unique (Single Responsibility Principle) et celui d'inversion des dépendances (Dependency Inversion Principle).

Concrètement, l'architecture que j'ai mise en place fonctionne de la manière suivante : les contrôleurs injectent des services, qui contiennent la logique métier de l'application. Ces services injectent eux-mêmes des repositories, responsables des interactions directes avec la base de données via Entity Framework. Ainsi, le contrôleur ne connaît ni la base de données ni la structure interne des entités : il se contente d'appeler les services, qui s'occupent de récupérer, traiter ou enregistrer les données nécessaires. Cette approche rend le code plus lisible, facilite les tests unitaires, et permet de faire évoluer la logique métier ou la structure de la base sans impacter la couche présentation.



Cœur des données

Dans une application ASP.NET Core utilisant Entity Framework Core, la classe ApplicationDbContext joue un rôle fondamental : elle représente le pont entre l'application et la base de données. Elle hérite généralement de DbContext ou, lorsqu'on utilise ASP.NET Identity, de IdentityDbContext, afin de gérer à la fois les entités métier et les entités liées à l'authentification. Le DbContext permet de définir quelles classes de l'application doivent être mappées à des tables dans la base de données, via des propriétés de type DbSet<T>. Chaque DbSet représente une table logique dans la base, et permet d'effectuer des opérations comme l'ajout, la suppression, la mise à jour ou la lecture des données pour une entité donnée. Par exemple, un DbSet<Dog> correspondra à la table des chiens dans la base, et permettra de manipuler les objets de type Dog directement depuis le code C#. Le DbContext est donc au cœur de l'accès aux données dans une application ASP.NET Core, et constitue la pièce maîtresse pour toutes les interactions entre le code métier et la base de données relationnelle.

```
public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
{
    7 références
    public DbSet< ApplicationUser> ApplicationUsers { get; set; }
    6 références
    public DbSet< CourseSession > CourseSessions { get; set; }
    5 références
    public DbSet< CourseType > CourseTypes { get; set; }
    4 références
    public DbSet< Dog > Dogs { get; set; }
    5 références
    public DbSet< DogCourseSession > DogCourseSessions { get; set; }
    5 références
    public DbSet< Breed > Breeds { get; set; }
```

Screenshot du DbContext d'Educanin

Personnalisation EF Core

La méthode `OnModelCreating` est une étape essentielle dans la configuration du modèle de données avec Entity Framework Core. Elle est appelée automatiquement lors de la construction du modèle de la base de données et permet de personnaliser la manière dont les classes C# sont mappées aux tables et relations dans la base de données. Par défaut, EF Core crée automatiquement un modèle basé sur les conventions (noms des propriétés, types, relations implicites), mais dans de nombreux cas, il est nécessaire de préciser ou modifier certains comportements pour répondre aux besoins spécifiques de l'application.

`OnModelCreating` donne accès à un objet `ModelBuilder` qui permet de définir explicitement les clés primaires, les relations entre entités (un-à-plusieurs, plusieurs-à-plusieurs, un-à-un), les règles de suppression en cascade, les contraintes, et bien d'autres paramètres. C'est dans cette méthode qu'on peut corriger ou affiner la structure générée automatiquement, notamment pour gérer des cas plus complexes comme les clés composites, les relations many-to-many personnalisées, ou des configurations particulières de navigation. En résumé, cette méthode est le point central où l'on maîtrise précisément la modélisation des données et l'organisation des tables pour garantir que le schéma généré correspond parfaitement aux besoins fonctionnels et techniques de l'application.

```
protected override void OnModelCreating(ModelBuilder builder)
{
    base.OnModelCreating(builder);

    builder.Entity<DogCourseSession>(entity =>
    {
        entity.HasKey(DogCourseSession => new { DogCourseSession.DogId, DogCourseSession.CourseSessionId });

        entity.HasOne(DogCourseSession => DogCourseSession.Dog)
            .WithMany(Dog => Dog.DogCourseSessions)
            .HasForeignKey(DogCourseSession => DogCourseSession.DogId)
            .onDelete(DeleteBehavior.Cascade);

        entity.HasOne(DogCourseSession => DogCourseSession.CourseSession)
            .WithMany(CourseSession => CourseSession.DogParticipants)
            .HasForeignKey(DogCourseSession => DogCourseSession.CourseSessionId)
            .onDelete(DeleteBehavior.Restrict);
    });
}
```

Screenshot de la méthode `OnModelCreating`

7.4. Des extraits de code d'autres composants (contrôleurs, utilitaires...)

Injection de dépendances

Afin d'accéder aux différentes fonctionnalités de la couche service dans mes contrôleurs, j'ai mis en place l'injection de dépendances, un mécanisme fondamental en ASP.NET Core. Ce principe consiste à déléguer au framework la responsabilité de créer et de fournir les instances des objets dont une classe a besoin pour fonctionner (appelés dépendances), plutôt que de les instancier directement à l'intérieur de la classe. Dans le cadre d'Educanin, cela signifie que les services métier comme `ICourseService`, `IDogService` ou `ICourseSessionService` sont automatiquement injectés dans les contrôleurs par le biais du constructeur. Ce choix permet de renforcer le découplage entre les composants de l'application : les contrôleurs ne dépendent plus d'implémentations concrètes, mais uniquement d'interfaces, ce qui favorise l'évolutivité et les tests unitaires.

Cette approche s'inscrit pleinement dans le respect du dernier principe du SOLID, à savoir le principe d'inversion des dépendances (Dependency Inversion Principle). Ce principe repose sur l'idée que les modules de haut niveau (comme les contrôleurs) ne doivent pas dépendre des modules de bas niveau (comme les services ou les repositories concrets), mais qu'ils doivent tous deux dépendre d'abstractions (interfaces). Grâce à l'injection de dépendances, ce sont ces abstractions qui sont utilisées et injectées, ce qui permet une plus grande flexibilité du code, une meilleure réutilisabilité des composants, et une facilité de remplacement ou de simulation lors des tests. J'ai également veillé à appliquer les bonnes pratiques associées à ce pattern : seuls les services réellement nécessaires sont injectés dans chaque contrôleur, évitant ainsi des dépendances inutiles qui alourdiraient le code ou compliqueraient les tests. Les services injectés sont stockés dans des champs privés en lecture seule (`private readonly`), ce qui permet de respecter les principes d'encapsulation et d'immutabilité. Enfin, grâce au conteneur d'injection de dépendances intégré à ASP.NET Core, la configuration est centralisée dans le fichier `Program.cs`, ce qui facilite la gestion du cycle de vie des objets (`transient`, `scoped`, `singleton`) et améliore la lisibilité de l'architecture globale.

```
//Debut de mon injection de dependance
//Ici les Repository
builder.Services.AddScoped<IApplicationUserRepository, ApplicationUserRepository>();
builder.Services.AddScoped<ICourseSessionRepository, CourseSessionRepository>();
builder.Services.AddScoped<ICourseTypeRepository, CourseTypeRepository>();
builder.Services.AddScoped<IBreedRepository, BreedRepository>();
builder.Services.AddScoped<IDogCourseSessionRepository, DogCourseSessionRepository>();
builder.Services.AddScoped<IDogRepository, DogRepository>();

//Ici les Services
builder.Services.AddScoped<IAccountService, AccountService>();
builder.Services.AddScoped<IApplicationUserService, ApplicationUserService>();
builder.Services.AddScoped<IDogService, DogService>();
builder.Services.AddScoped<IBreedService, BreedService>();
builder.Services.AddScoped<ICourseTypeService, CourseTypeService>();
builder.Services.AddScoped<ICourseSessionService, CourseSessionService>();
builder.Services.AddScoped<IDogCourseSessionService, DogCourseSessionService>();
```

Injection de dépendance faite dans le Program.cs

8. La présentation d'éléments de sécurité de l'application

La sécurité informatique est devenue un enjeu crucial ces dernières années. C'est pourquoi j'ai accordé une attention particulière à cet aspect dans mon projet. Pour renforcer mes connaissances, je me suis principalement formé grâce à la documentation officielle et aux ateliers interactifs proposés par la CNIL (Commission Nationale de l'Informatique et des Libertés), ainsi que par le site de l'OWASP (Open Worldwide Application Security Project).

Rôles et autorisation

On va commencer avec La sécurisation des pages avec les rôles d'Identity en ASP.NET Core repose sur une logique simple : définir qui a accès à quoi. Chaque utilisateur se voit attribuer un ou plusieurs rôles (comme Admin, Coach ou User), et ces rôles servent de filtre pour autoriser ou bloquer l'accès à certaines pages. Le cœur du système, c'est l'attribut [Authorize]. Placé sur un contrôleur ou une action, il exige d'abord que l'utilisateur soit connecté. Il peut aussi vérifier un rôle précis : [Authorize(Roles = "Admin")] bloque l'accès à tous sauf aux Admins. En listant plusieurs rôles, par exemple [Authorize(Roles = "Admin,Coach")], on ouvre l'accès aux deux. Pour que ces vérifications fonctionnent, ASP.NET Core s'appuie sur deux middlewares. UseAuthentication() identifie l'utilisateur (via un cookie, un token...) et crée un profil (ClaimsPrincipal). UseAuthorization() prend le relais pour comparer ce profil aux rôles demandés et décider si l'accès est accordé. Ces deux étapes doivent être activées dans Program.cs (ou Startup.cs) et exécutées dans cet ordre avant le routage. Ce mécanisme apporte une structure claire : chaque rôle correspond à un périmètre d'action défini. Cela simplifie la maintenance (un nouveau rôle ou une règle se gèrent au même endroit), renforce la sécurité (les pages sensibles restent verrouillées) et permet même d'adapter l'interface : certains boutons ou menus n'apparaissent qu'aux bons utilisateurs.

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

```
[Authorize]
1 référence
public class DashBoardController : Controller
```

```
[Authorize(Roles ="Admin")]
0 références
public IActionResult DashBoardAdmin()
{
    return View();
}
```

```
[Authorize(Roles ="Admin,Coach")]
0 références
public IActionResult DashBoardCoach()
{
    return View();
}
```

L'attaque CSRF

Le CSRF (Cross-Site Request Forgery) est une attaque bien connue dans le monde du web. Elle consiste à piéger un utilisateur déjà authentifié sur un site (par exemple connecté à sa banque en ligne) pour qu'il envoie, à son insu, une requête malveillante vers ce même site. Le principe est simple : si le navigateur de la victime détient déjà un cookie de session valide, une page malveillante peut forcer une requête (un formulaire ou un script) vers le site cible, et celui-ci pensera que la requête est légitime car elle vient d'un utilisateur "connecté". Résultat : l'attaquant peut faire valider un virement, changer un mot de passe ou supprimer un compte, sans que la victime s'en rende compte.

C'est là qu'intervient le HTML Anti-Forgery Token d'ASP.NET Core. Il agit comme une "preuve d'intention" : chaque fois qu'un formulaire est affiché, le serveur génère un token unique et imprévisible lié à la session de l'utilisateur et l'injecte dans la page via `@Html.AntiForgeryToken()`. Quand l'utilisateur soumet le formulaire, ce token est renvoyé avec la requête POST. Du côté serveur, l'attribut `[ValidateAntiForgeryToken]` (placé sur l'action du contrôleur) compare le token reçu avec celui qui a été généré pour la session. Si le token est absent ou incorrect, la requête est immédiatement bloquée. Cette vérification empêche un site tiers de simuler une requête authentique : même si le navigateur envoie le cookie de session, il manquera toujours le bon token, que seul le site d'origine peut fournir.

```
<form asp-action="Login" asp-controller="Account" method="post" class="p-6 space-y-6">
    @Html.AntiForgeryToken()

    [HttpPost]
    [ValidateAntiForgeryToken]
    0 références
    public async Task<IActionResult> Login(LoginViewModel model)
```

Exemple d'utilisation des Anti-Forgery Token

Les View Models

Les View Models jouent aussi un rôle important dans la sécurité d'une application ASP.NET Core, notamment contre les attaques de surpostage (over-posting). Le surpostage se produit lorsqu'un utilisateur mal intentionné envoie plus de données que prévu via un formulaire, par exemple en ajoutant manuellement des champs supplémentaires dans la requête POST. Si l'application utilise directement son entité de base de données comme paramètre d'action (par exemple un objet User complet), ces champs supplémentaires peuvent être liés automatiquement et modifier des propriétés sensibles, comme le rôle d'un utilisateur ou son statut d'administration.

C'est là que les View Models interviennent. Au lieu de lier directement l'action à l'entité, on crée une classe dédiée (ex. LoginViewModel, RegisterViewModel...) qui ne contient que les propriétés nécessaires au scénario. Par exemple, un formulaire de connexion n'aura besoin que d'un email et d'un mot de passe. ASP.NET Core ne pourra donc lier que ces champs, même si un utilisateur tente d'en injecter d'autres. On y ajoute souvent des Data Annotations comme [Required], [EmailAddress] ou [StringLength] pour valider automatiquement les données reçues avant même qu'elles n'atteignent la logique métier.

En isolant strictement les données attendues et en les validant dès leur réception, les View Models réduisent la surface d'attaque, empêchent la modification non désirée d'informations sensibles et garantissent que seules les données prévues circulent entre la vue et le serveur. C'est une bonne pratique à la fois pour la sécurité, la clarté du code et la robustesse de l'application.

```
public class LoginViewModel
{
    [Required(ErrorMessage = "L'adresse email est requise.")]
    6 références
    public string Email { get; set; }

    [Required(ErrorMessage = "Le mot de passe est requis.")]
    [DataType(DataType.Password)]
    6 références
    public string Password { get; set; }

    [Display(Name = "Se souvenir de moi")]
    5 références
    public bool RememberMe { get; set; }
}
```

Exemple avec le ViewModel du Login

Les Injections SQL

Les failles SQL Injection sont l'une des attaques les plus répandues : elles consistent à injecter du code SQL malveillant dans un champ d'entrée (par exemple un formulaire de connexion) pour manipuler la base de données. Une requête construite naïvement en concaténant des chaînes de caractères avec les données fournies par l'utilisateur devient vulnérable : si quelqu'un saisit "" OR 1=1 --", la condition est toujours vraie et peut permettre de contourner une authentification ou de lire des informations sensibles.

Avec Entity Framework Core, ce problème est largement évité car le framework utilise des requêtes paramétrées par défaut. Quand on écrit `context.Users.Where(u => u.Email == email)`, EF ne colle pas la valeur de `email` directement dans la requête SQL : il génère un paramètre (@p0) et envoie la valeur séparément. La base de données traite donc cette valeur comme du texte brut, et non comme du code SQL exécutable. Cela empêche l'injection de commandes malveillantes et sécurise les échanges avec la base, tout en gardant le code clair et lisible.

- Attaques par injection dans LINQ to Entities :

Même s'il est possible de composer des requêtes dans LINQ to Entities, cette opération est effectuée via l'API de modèle objet. Contrairement aux requêtes Entity SQL, les requêtes LINQ to Entities ne sont pas composées par manipulation ou concaténation de chaînes, et elles ne sont pas sujettes à des attaques par injection de code SQL au sens classique du terme.

Extrait de la documentation Microsoft officiel sur le sujet

9. Déploiement

Déploiement sur Render

La partie déploiement de mon projet a été conçue pour être fluide, automatisée et professionnelle, tout en restant gratuite. J'ai choisi Render comme hébergeur, un PaaS (Platform as a Service) capable de gérer facilement les déploiements et offrant un plan gratuit adapté à ce projet. Render ne prend pas ASP.NET Core en charge de manière native, donc pour contourner cette limitation j'ai utilisé Docker. Grâce au Dockerfile intégré au projet, l'application est construite et exécutée dans un conteneur, ce qui permet à Render de la faire tourner sans se soucier de la technologie utilisée.

```
FROM mcr.microsoft.com/dotnet/sdk:9.0 AS build

WORKDIR /src

COPY . .

WORKDIR /src/EduCanin

RUN dotnet publish "EduCanin.csproj" -c Release -o /app/out

FROM mcr.microsoft.com/dotnet/aspnet:9.0

WORKDIR /app

COPY --from=build /app/out .

ENV ASPNETCORE_URLS=http://+:10000

EXPOSE 10000

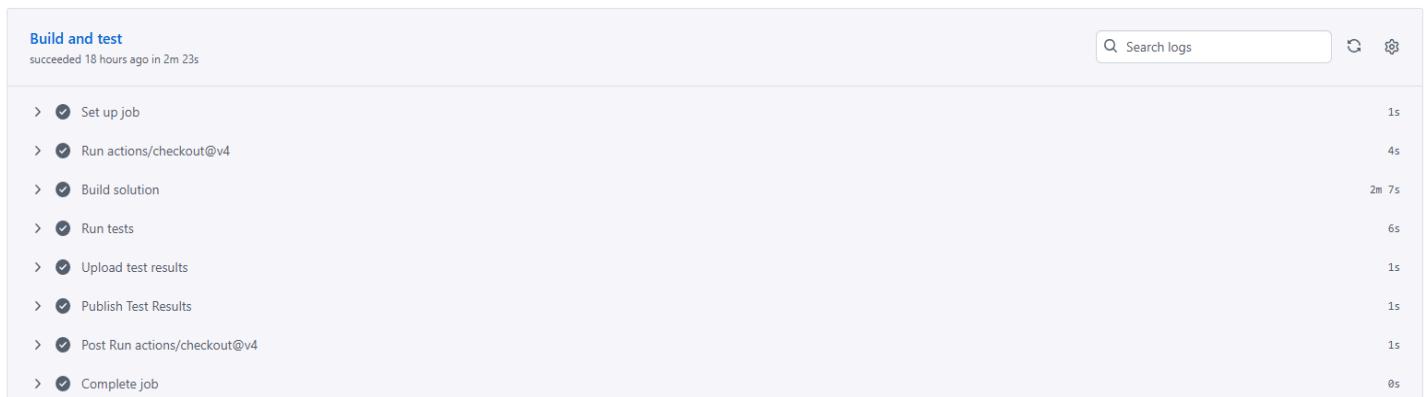
ENTRYPOINT ["dotnet", "EduCanin.dll"]
```

Dockerfile de l'application EduCanin

Migration & CI/CD

La seule vraie contrainte a été de migrer ma base de données de SQL Server vers PostgreSQL, puisque Render ne prend pas SQL Server en charge nativement. J'ai donc adapté mon code et mes migrations pour que tout fonctionne parfaitement avec PostgreSQL.

Côté automatisation, j'ai mis en place une pipeline CI/CD basée sur GitHub Actions. Concrètement, à chaque push sur mon dépôt GitHub, GitHub Actions exécute une série de jobs : restauration des packages, compilation, exécution des tests unitaires... et si tout est vert, Render déploie automatiquement la nouvelle version de l'application. C'est un gain de temps énorme : je n'ai plus à intervenir manuellement, et je suis sûr que la version en ligne est toujours stable et à jour.



The screenshot shows a GitHub Actions pipeline named "Build and test". It indicates a success status with the message "succeeded 18 hours ago in 2m 23s". A search bar labeled "Search logs" is at the top right. The log itself lists nine steps: "Set up job" (1s), "Run actions/checkout@v4" (4s), "Build solution" (2m 7s), "Run tests" (6s), "Upload test results" (1s), "Publish Test Results" (1s), "Post Run actions/checkout@v4" (1s), and "Complete job" (0s). Each step is preceded by a green checkmark icon and a right-pointing arrow.

Pipeline CI/CD GitHub Actions validée toutes les étapes (build, tests, résultats) sont passées avec succès.

SonarQube

J'avais initialement intégré SonarQube dans ma pipeline pour analyser la qualité et la sécurité du code. L'idée était bonne : chaque push lançait une analyse pour détecter des bugs ou des mauvaises pratiques. Mais la version gratuite s'est révélée trop rigide : elle scannait même les fichiers de migrations EF Core, générés automatiquement, qui ne respectent pas toujours les standards qu'elle impose. Résultat : mes pipelines échouaient inutilement, et je n'avais aucun moyen d'ajuster les règles sans passer sur un plan payant. J'ai donc choisi de retirer SonarQube pour éviter de bloquer la CI/CD avec des problèmes hors de mon contrôle.

Choix de l'hébergement des données

Pour la base de données, Render propose bien du PostgreSQL, mais sur le plan gratuit, elle est effacée après un certain temps. Ce n'était pas viable, car je voulais que mon projet reste en ligne indéfiniment pour pouvoir l'intégrer à mon portfolio sans devoir intervenir. J'ai donc opté pour Neon DB, un service PostgreSQL gratuit et persistant, parfait pour un usage long terme. Concrètement, l'application est sur Render, la base est sur Neon, et j'ai appliqué mes migrations EF Core sur cette base distante. Ce découplage rend le projet plus durable et maintenable.

Said's projects

New project Import database

Account Usage Jul 1, 2025 to now • Upgrade

Storage 0.03 / 0.5 GB	Compute 3.2 / 191.9 h	Branch Compute 0 / 5 h	Data Transfer 0.01 / 5 GB	Projects 1 / 10
---------------------------------	---------------------------------	----------------------------------	-------------------------------------	---------------------------

Metrics may be delayed up to one hour. [Learn more here.](#)

Name	Region	Created at	Storage	Postgres version	Integrations	⋮
Educanin	Azure Germany West Central (Frankfurt)	Jun 17, 2025 10:46 am	32.97 MB	16	Add	⋮

Ma base de données sur Neon

10. plan de tests et jeu d'essai élaboré par le candidat

Tests unitaires

Pour Educanin, j'ai mis en place une batterie de tests unitaires indispensables dans le développement d'applications web. Leur objectif est de vérifier que les méthodes de l'application retournent les résultats attendus et se comportent correctement dans différentes situations. Les tests unitaires sont très flexibles : ils peuvent simuler des scénarios variés afin de détecter des bugs avant qu'ils ne se produisent en conditions réelles, évitant ainsi des problèmes en production.

Pour les mettre en place, j'ai commencé par ajouter un projet de test dédié à ma solution, en y important une référence vers le projet principal afin d'accéder aux classes et services de l'application. J'ai ensuite installé les packages nécessaires, notamment MSTest et Moq, deux outils essentiels pour cette phase.

- **MSTest** : framework de tests unitaires open source pour .NET, il permet d'écrire et d'exécuter des tests afin de s'assurer que le code fonctionne comme prévu. Il est simple, fiable et parfaitement intégré à l'écosystème Visual Studio.
- **Moq** : bibliothèque de *mocking* pour .NET, elle permet de créer des **objets simulés** pour remplacer des dépendances externes (comme une base de données, un service d'email ou un gestionnaire d'identité). Grâce à Moq, j'ai pu tester la **logique métier isolément**, sans dépendre d'implémentations réelles ni de ressources externes.

```

[TestMethod]
0 références
public async Task RegisterAsync_ShouldReturnSuccess_WhenUserCreatedAndRoleAssigned()
{
    // Arrange : on prépare les données d'inscription
    var model = new RegisterViewModel
    {
        Email = "newuser@mail.com",
        Password = "P@ssword1",
        FirstName = "John",
        LastName = "Doe"
    };

    // On simule que l'utilisateur est bien créé
    _userManagerMock.Setup(x => x.CreateAsync(It.IsAny< ApplicationUser>(), model.Password))
        .ReturnsAsync(IdentityResult.Success);

    // On simule que le rôle "User" n'existe pas
    _roleManagerMock.Setup(x => x.RoleExistsAsync("User"))
        .ReturnsAsync(false);

    // On simule que le rôle est créé avec succès
    _roleManagerMock.Setup(x => x.CreateAsync(It.IsAny< IdentityRole>()))
        .ReturnsAsync(IdentityResult.Success);

    // On simule que l'ajout au rôle "User" réussit
    _userManagerMock.Setup(x => x.AddToRoleAsync(It.IsAny< ApplicationUser>(), "User"))
        .ReturnsAsync(IdentityResult.Success);

    // On simule que la connexion de l'utilisateur réussit après l'inscription
    _signInManagerMock
        .Setup(x => x.SignInAsync(It.IsAny< ApplicationUser>(), false, null))
        .Returns(Task.CompletedTask);

    // Act : on appelle RegisterAsync
    var result = await _accountService.RegisterAsync(model);

    // Assert : on s'assure que l'inscription est un succès
    Assert.IsTrue(result.Succeeded);
}

```

Ce test suit la structure classique Arrange, Act, Assert, une façon claire et lisible d'écrire des tests unitaires.

Dans la phase Arrange, on prépare tout ce dont la méthode aura besoin pour être testée. Ici, on crée un RegisterViewModel avec les informations d'un nouvel utilisateur (email, mot de passe, prénom, nom). On configure ensuite tous les mocks pour simuler le comportement attendu : le UserManager va répondre que l'utilisateur est bien créé, le RoleManager indique que le rôle “User” n'existe pas encore puis qu'il est créé sans problème, l'ajout de l'utilisateur à ce rôle réussit, et enfin le SignInManager simule une connexion automatique réussie après l'inscription. Cette étape prépare donc un scénario où tout se passe bien.

Vient ensuite la phase Act : on exécute la méthode à tester, ici _accountService.RegisterAsync(model). On agit vraiment comme si l'utilisateur remplissait le formulaire d'inscription, sauf que tout est contrôlé grâce aux mocks.

Enfin, la phase Assert sert à vérifier le résultat. On s'assure que l'inscription a bien été un succès en contrôlant la valeur result.Succeeded. Si une seule étape simulée n'avait pas fonctionné comme prévu, cette assertion aurait échoué.

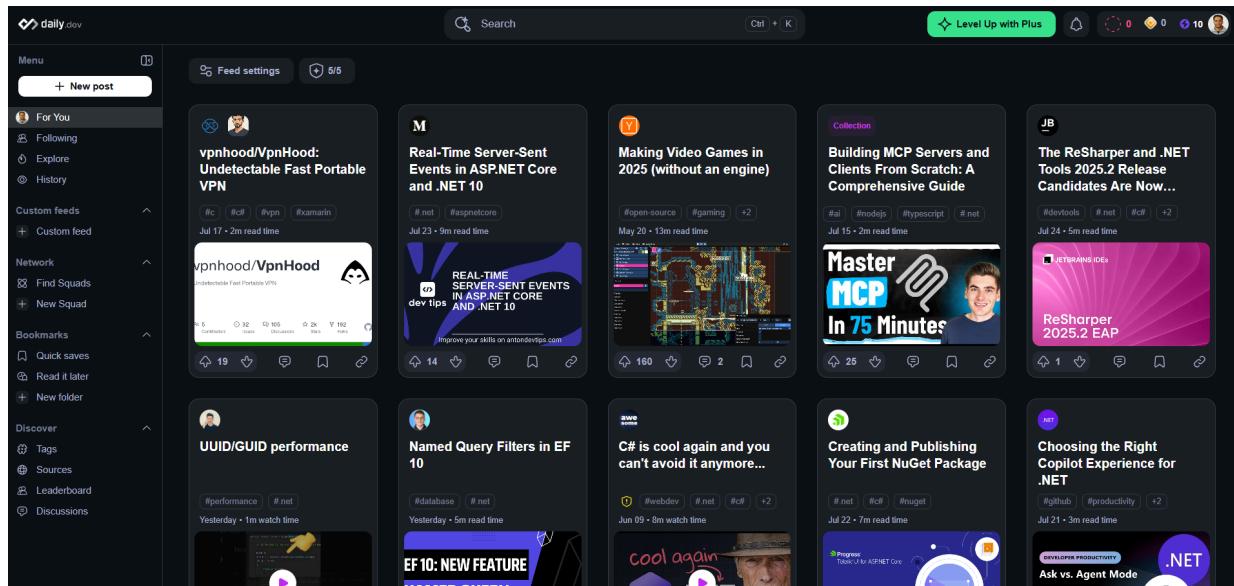
▲ ✓ Educanin.Tests (9)	2,1 s
▲ ✓ Educanin.Tests (3)	820 ms
▲ ✓ AccountServiceTests (3)	820 ms
✓ LoginAsync_ShouldReturnSuccess	273 ms
✓ LogoutAsync_ShouldCallSignOut	272 ms
✓ RegisterAsync_ShouldReturnSuccess	275 ms

Résultats d'exécution des tests unitaires du projet

11. la description de la veille, effectuée par le candidat

La veille technologique est indispensable en développement, car elle permet de rester informé des évolutions, outils et bonnes pratiques dans un secteur en constante évolution. Afin de me tenir à jour, aussi bien dans le domaine de l'informatique en général que plus spécifiquement dans le software et le développement, je consulte régulièrement de multiples sources d'information. Ces sources, variées dans leur format, me permettent d'anticiper les tendances, d'adopter des solutions plus efficaces et de maintenir un code moderne, sécurisé et conforme aux standards actuels.

Daily.dev: Daily.Dev est une extension que j'ai installée sur mon navigateur Mozilla Firefox et qui sert de véritable hub d'actualité dédié au développement. Elle agrège en continu un grand nombre d'articles, tutoriels et retours d'expérience issus de la communauté tech. Les contenus sont classés et mis en avant en fonction de leur pertinence et des retours des utilisateurs, ce qui permet d'accéder rapidement aux informations les plus utiles. L'extension offre aussi la possibilité de personnaliser les sujets suivis comme C#, ASP.NET, Microsoft, HTML ou CSS pour n'afficher que les articles liés à mes centres d'intérêt, rendant ma veille à la fois ciblée et efficace.

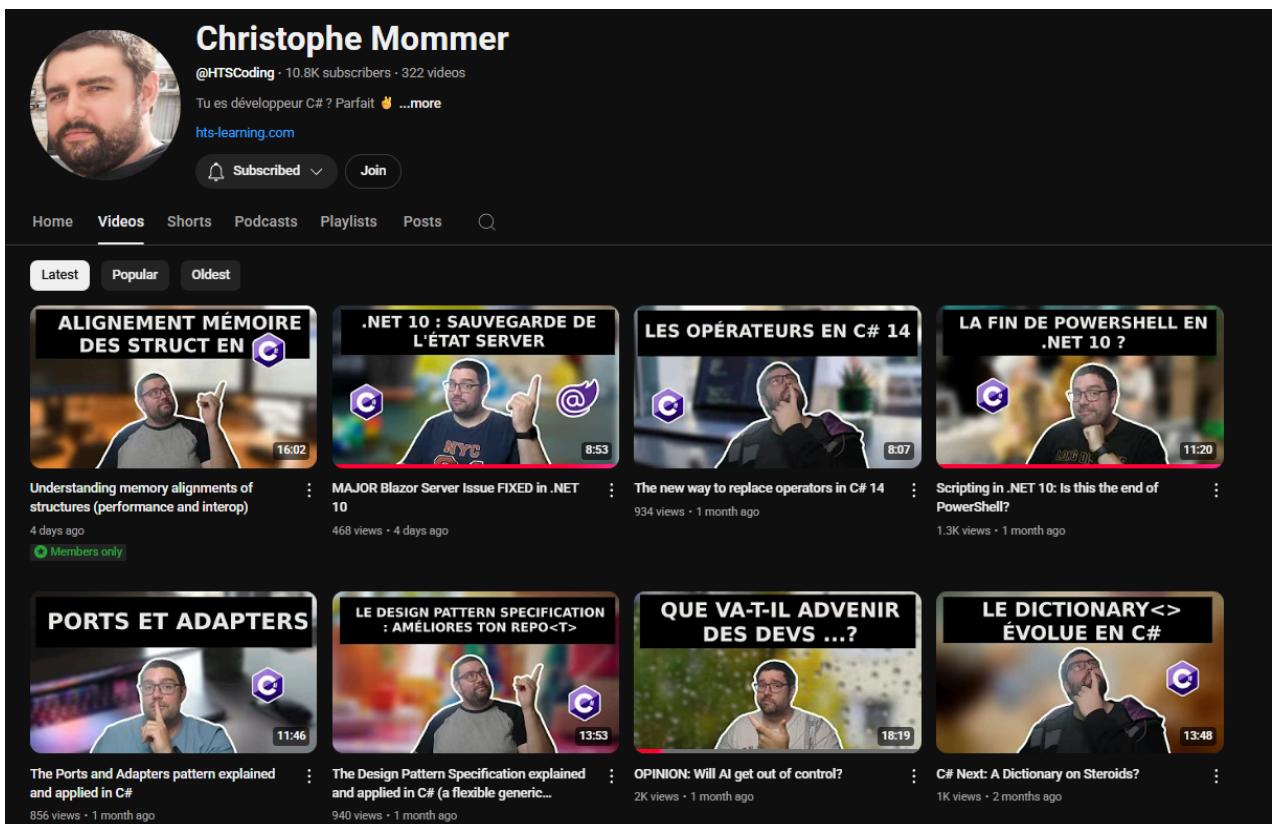


Screenshot de l'accueil de Daily.Dev

Linkedin : Linkedin est un réseau social professionnel qui me permet de suivre des pages spécialisées (par exemple sur des langages ou des frameworks) ainsi que des développeurs du monde entier qui partagent articles, retours d'expérience et bonnes pratiques.

Reddit : Reddit est une plateforme sociale américaine organisée en milliers de forums thématiques appelés subreddits. On y trouve de nombreuses communautés dédiées à l'informatique, au développement ou encore à la cybersécurité. C'est un excellent moyen de découvrir des discussions techniques, des astuces ou des tendances directement auprès d'autres passionnés et professionnels.

Youtube : YouTube, de son côté, est une ressource incontournable pour la veille et l'apprentissage continu. On y trouve des chaînes spécialisées dans le développement (tutoriels, conférences, présentations de nouvelles technologies) qui permettent de rester informé tout en découvrant des exemples concrets en vidéo.



Chaîne youtube de Christophe Mommer, Développeur C#/Net Senior