RAPPORT AP2
MEDMANAGER

SOMMAIRE

I. Contexte GSB

Galaxy Swiss Bourdin (GSB) a été créé en 2009 par la fusion de Galaxy, qui se spécialise dans les maladies virales (SIDA, hépatites), et Swiss Bourdin, qui se concentre sur des médicaments plus traditionnels. Ce regroupement a créé une entreprise majeure dans le secteur pharmaceutique, avec un siège administratif à Paris et un siège social à Philadelphie, aux États-Unis. Le domaine pharmaceutique, caractérisé par de nombreuses fusions, génère de grandes rentrées de fonds, mais est critiqué pour ses pratiques de visite médicale considérées comme désinformées. À la suite de la fusion, GSB a mis en place une restructuration et des économies d'échelle pour optimiser ses activités, avec 480 visiteurs médicaux en France et 60 dans les DOM-TOM, répartis dans 6 secteurs géographiques.

La centralisation de l'infrastructure informatique de GSB à Paris inclut des services administratifs et des salles de serveurs sécurisées. Les informations, perçues comme stratégiques, sont répliquées chaque jour aux États-Unis. La direction des systèmes d'information occupe une position stratégique en intégrant des outils de recherche, de production et de communication avancés, tandis que la partie commerciale a été moins automatisée. Le réseau est divisé en VLAN afin de garantir la sécurité et la fluidité du trafic, avec des règles d'accès rigoureuses pour chaque VLAN.

GSB désire consolider sa capacité commerciale afin d'obtenir une meilleure compréhension de l'activité sur le terrain et de redonner confiance aux équipes après les fusions. Les patients, qui jouent un rôle essentiel dans l'information des prescripteurs sur les produits du laboratoire, ont un impact indirect sur les prescriptions médicales. Cette structure partagée par les délégués médicaux, qui découle de la politique de Galaxy, a pour objectif de standardiser et de moderniser les méthodes de visite médicale.

II. Fonctionnalité présente

Voici un résumé des fonctionnalités présentes dans le projet :

1. Authentification sécurisée :

- Vérification des identifiants et redirection en fonction du rôle de l'utilisateur (visiteur, comptable, administrateur).

2. Fonctionnalités Principales

2.1 Gestion des Utilisateurs

- **Authentification sécurisée**
  - Compte unique par médecin
  - Système d'authentification robuste
  - Accès exclusif aux professionnels habilités

2.2 Gestion des Profils Patients

- Création et gestion de profils détaillés comprenant :
  - Informations personnelles (nom, âge, sexe)
  - Antécédents médicaux
  - Allergies connues
  - Historique médical

2.3 Gestion des Ordonnances

- **Fonctionnalités complètes de prescription**
  - Création de nouvelles ordonnances
  - Modification d'ordonnances existantes
  - Annulation d'ordonnances
  - Export automatique au format PDF
  - Traçabilité complète des prescriptions

2.4 Base de Données Médicaments

- **Base de données exhaustive**
  - Référentiel complet des médicaments
  - Informations détaillées sur chaque médicament
  - Système d'alerte intelligent
  -

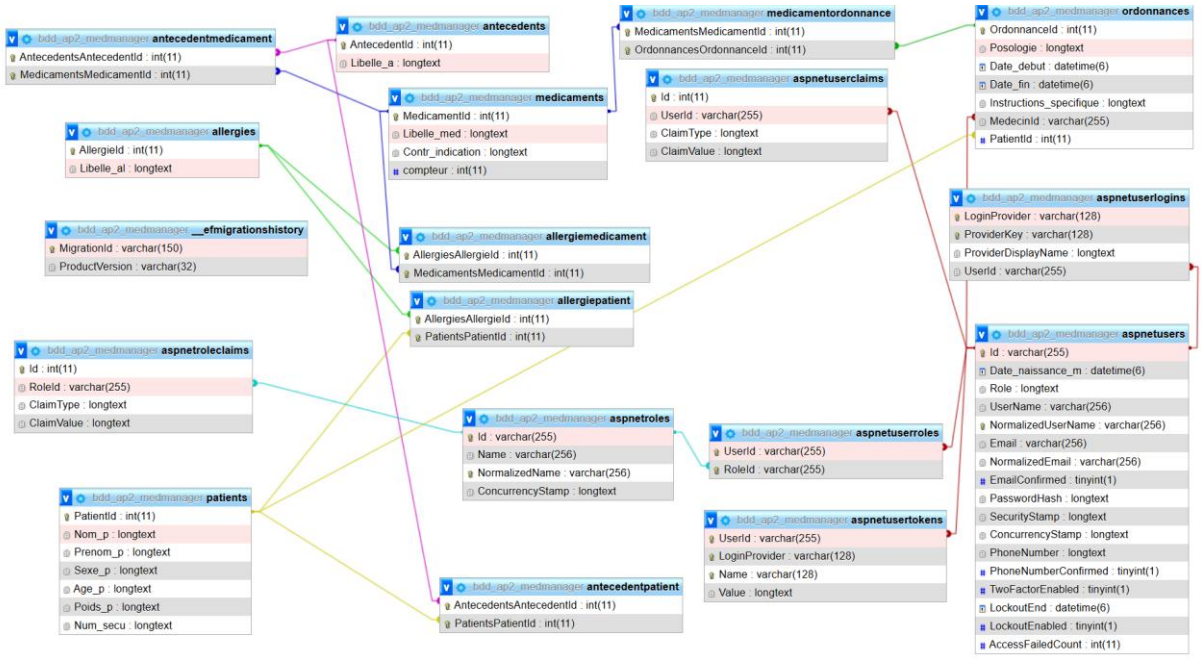
3. Sécurité renforcée :

- Utilisation de paramètres SQL pour prévenir les injections SQL et gestion des erreurs pour assurer la fiabilité des interactions avec la base de données.

Ces fonctionnalités permettent une gestion efficace et sécurisée des frais pour les utilisateurs de l'application.

III. BDD

Voici mon MPD :



Le diagramme représente le modèle relationnel de la base de données de votre application .NET MVC, qui gère les informations médicales des patients, les médicaments, les allergies, les ordonnances, et la gestion des utilisateurs.

Structure globale :

Gestion des patients : La table principale patients est liée aux informations médicales comme les antécédents (antecedents via antecedentpatient), les allergies (allergies via allergiepatient), et les ordonnances (ordonnances).

Médicaments et ordonnances : Les médicaments sont centralisés dans la table medicaments et associés aux ordonnances via la table medicamentordonnance. Cela permet de lier des médicaments spécifiques à des prescriptions.

Gestion des allergies : Les patients peuvent être liés à des allergies spécifiques via des relations dans les tables allergies et allergiemedicament, pour identifier les médicaments contre-indiqués.

Authentification et gestion des rôles : La partie utilisateur repose sur les tables aspnetusers, aspnetroles, et leurs relations associées (aspnetuserroles, aspnetuserclaims, etc.). Cela gère les comptes, les rôles (administrateur, médecin, etc.), et les permissions dans l'application.

Logs et historique : La table __efmigrationshistory trace les versions et migrations de la base de données pour le suivi des évolutions.

Relations globales :

Chaque table est liée par des clés primaires/secondaires pour :

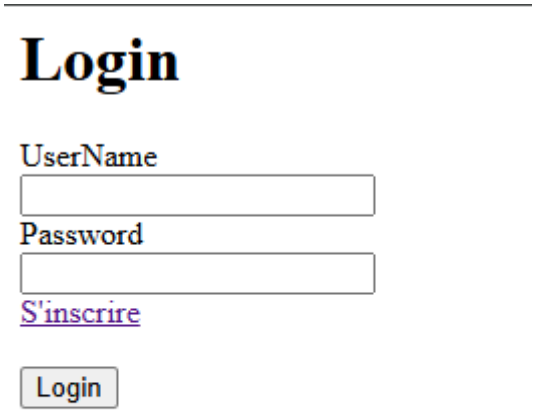
- Relier les données des patients à leur historique médical.
- Associer des ordonnances à des médicaments et des médecins.
- Gérer les rôles et les droits des utilisateurs.

En résumé, cette base de données suit un schéma relationnel bien structuré, centralisant les données médicales et utilisateurs pour un fonctionnement fluide de votre application.

IV. Application

Cette application, développée avec le framework **ASP.NET MVC**, permet de gérer les informations des patients d'un établissement médical. Elle offre une interface utilisateur intuitive pour effectuer les opérations courantes de gestion des patients, telles que l'ajout, la modification, la suppression et la consultation des informations. L'objectif est de simplifier et d'automatiser la gestion des données des patients.

A. Page de connexion et Page d'inscription



Ceci est mon forms de ma page de connexion.

```
<h1>Login</h1>
<form asp-action="Login" asp-controller="Account" method="post">
  <div asp-validation-summary="ModelOnly" class="text-danger"></div>
  <div class="form-group row">
    <label asp-for="UserName" class="col-sm-2 col-form-label"></label>
    <div class="col-sm-10">
      <input asp-for="UserName" class="form-control" />
      <span asp-validation-for="UserName" class="text-danger"></span>
    </div>
  </div>
  <div class="form-group row">
    <label asp-for="Password" class="col-sm-2 col-form-label"></label>
    <div class="col-sm-10">
      <input asp-for="Password" class="form-control" />
      <span asp-validation-for="Password" class="text-danger"></span>
    </div>
  </div>
  <div class="form-group row">
    <div class="col-sm-10 offset-sm-2">
      <div class="form-check <input asp-for="RememberMe" class="form-check </div>
      <span  asp-validation-for="RememberMe" class="text-danger"></span>
      <a asp-action="Register" class="text-danger">S'inscrire</a>
    </div>
  </div>
  <br />


  <input type="submit" value="Login" class="btn btn-primary" />
</form>
```

```
0 references
public class LoginViewModel
{
    [Required(ErrorMessage = "Nom d'utilisateur requis.")]
    0 references
    public required string UserName { get; set; }

    [Required(ErrorMessage = "Mot de passe requis.")]
    [DataType(DataType.Password)]
    0 references
    public required string Password { get; set; }

    [Display(Name = "Remember me?")]
    0 references
    public bool RememberMe { get; set; }
}
```

L'application intègre un système de gestion des utilisateurs, notamment à travers un formulaire de connexion. Ce formulaire utilise le modèle MVC (Model-View-Controller) et repose sur une approche orientée données grâce à l'utilisation d'un ViewModel.

1. Présentation du formulaire de connexion

Le formulaire de connexion, défini dans une vue Razor, permet à l'utilisateur de s'authentifier en saisissant un nom d'utilisateur et un mot de passe. Ce formulaire s'appuie sur les fonctionnalités ASP.NET Razor pour simplifier la génération de champs et la gestion des erreurs.

- **Structure générale :**
  - Le formulaire utilise la directive asp-action="Login" et asp-controller="Account", ce qui indique que les données saisies seront envoyées à l'action **Login** du contrôleur **Account** via une requête HTTP POST.
  - Une validation côté client et côté serveur est mise en œuvre pour garantir l'intégrité des données saisies.
- **Champs et validation :**
  - Le champ pour le **nom d'utilisateur** est généré à l'aide de la directive asp-for="UserName", qui se lie dynamiquement à la propriété correspondante dans le ViewModel (LoginViewModel).
  - De manière similaire, le champ pour le **mot de passe** utilise asp-for="Password".
  - Une validation des données est intégrée grâce à asp-validation-for. Si une erreur est détectée (par exemple, si un champ est laissé vide), un message d'erreur s'affiche dynamiquement en fonction des annotations définies dans le modèle.
- **Autres fonctionnalités :**
  - Un champ de type case à cocher, lié à la propriété RememberMe, permet à l'utilisateur de demander à ce que sa session reste active.

- Un lien d'inscription (asp-action="Register") est également inclus pour rediriger les nouveaux utilisateurs vers une page de création de compte.

2. Définition du ViewModel : LoginViewModel

Le **LoginViewModel** est une classe C# qui sert de modèle de données pour le formulaire. Elle structure les informations nécessaires à la connexion tout en implémentant des règles de validation via des annotations de données.

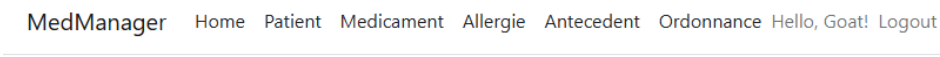
- **Propriétés principales :**
  - UserName : Représente le nom d'utilisateur saisi par l'utilisateur.
  - Password : Représente le mot de passe. L'attribut [DataType(DataType.Password)] garantit que ce champ sera traité comme un mot de passe (masquage des caractères dans la vue).
  - RememberMe : Représente une option booléenne pour maintenir la session active.
- **Validation des données :**
  - Les propriétés UserName et Password sont annotées avec [Required] pour s'assurer qu'elles ne peuvent pas être laissées vides.
  - Les messages d'erreur sont personnalisés grâce à l'attribut ErrorMessage, permettant d'afficher des indications claires en cas de saisie incorrecte.

3. Intégration entre Vue et Modèle

Le formulaire Razor et le ViewModel sont étroitement liés :

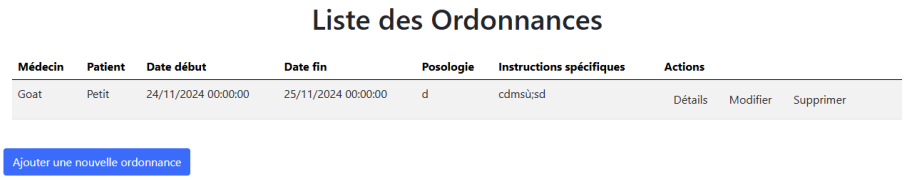
- Les directives comme asp-for et asp-validation-for permettent de lier automatiquement les champs du formulaire aux propriétés du modèle.
- Cela simplifie la gestion des données et centralise les règles de validation dans le ViewModel, améliorant ainsi la maintenabilité et la cohérence de l'application.

```
1 reference
public class AccountController : Controller
{
    2 references
    private readonly UserManager<Medecin> _userManager;

    4 references
    private readonly SignInManager<Medecin> _signInManager;
    0 references
    public AccountController(SignInManager<Medecin> signInManager, UserManager<Medecin> userManager)
    {
        _signInManager = signInManager;
        _userManager = userManager;
    }

    [HttpGet]
    0 references
    public IActionResult Index()
    {
        ViewBag.HideNavBar = true;
        return View();
    }

    [HttpGet]
    0 references
    public IActionResult Login()
    {


        return View();
    }
```



La barre de navigation de l'application constitue un élément central du layout, facilitant l'accès aux différentes fonctionnalités. Elle inclut le nom de l'application, "MedManager", affiché en haut à gauche pour identifier clairement l'outil. Les liens de navigation permettent de se déplacer entre les sections principales : "Home" pour la page d'accueil ou le tableau de bord, "Patient" pour gérer les informations des patients, "Médicament" pour administrer la liste des médicaments, "Allergie" et "Antécédent" pour documenter les allergies et antécédents médicaux, et "Ordonnance" pour gérer les prescriptions. Sur la droite, un message personnalisé, "Hello, Goat!", s'adresse à l'utilisateur connecté, accompagné d'un lien "Logout" pour se déconnecter. Ce design garantit une navigation intuitive et une expérience utilisateur fluide.

"Ajouter un Patient" offre un formulaire structuré permettant de saisir les données essentielles telles que le nom, le prénom, le sexe, l'âge, le poids, et le numéro de sécurité sociale, avec un bouton dédié pour valider et enregistrer les informations. La vue "Liste des Patients" présente un tableau clair où chaque patient est identifié par un ID, un nom et un prénom, avec des options pour consulter les détails, modifier ou supprimer les données.



Elle présente une interface organisée sous forme de tableau affichant des informations clés telles que le nom du médecin, celui du patient, les dates de début et de fin de validité, la posologie et les instructions spécifiques associées à chaque ordonnance. Des actions interactives permettent d'accéder aux détails, de modifier ou de supprimer une ordonnance. De plus, un formulaire dédié offre la possibilité de créer une nouvelle ordonnance en sélectionnant un patient, en définissant les dates de validité, en précisant la posologie, en ajoutant des instructions spécifiques et en choisissant les médicaments à prescrire à partir d'une liste préétablie. Cette solution optimise la gestion des prescriptions de manière structurée et intuitive.

V. Conclusion

Cette application se distingue par sa simplicité et son efficacité, rendant la gestion des patients fluide et accessible. Elle illustre les principes fondamentaux d'une architecture MVC bien structurée, où les vues offrent une expérience utilisateur intuitive et les contrôleurs assurent le traitement des données en arrière-plan.