ANTOINE DINH

RAPPORT DE PROJET

Novembre 2024

AP2 GSB

Bts SIO OPtion SLAM

SOMMAIRE

01

CONTEXTE GSB

Contexte GSB

Galaxy Swiss Bourdin (GSB) a émergé d'une fusion stratégique entre deux grands noms de l'industrie pharmaceutique : le géant américain Galaxy, spécialisé dans le traitement des maladies virales comme le SIDA et les hépatites, et le conglomérat européen Swiss Bourdin, connu pour ses médicaments traditionnels. Cette collaboration en 2009 a donné naissance à un leader incontesté du secteur pharmaceutique.

Siège Social et Administration

GSB a établi son siège administratif à Paris pour son entité européenne, tandis que le siège social de la multinationale se situe à Bagneux, dans les Hauts-de-Seine. Ces choix stratégiques reflètent l'engagement de GSB envers l'excellence et l'innovation au cœur de l'Europe.

Force de Vente et Présence Médicale

Avec une équipe de vente composée de 480 visiteurs médicaux en France métropolitaine et 60 dans les départements et territoires d'outre-mer, GSB maintient une présence significative et influente dans le domaine médical, assurant une couverture complète et efficace sur tout le territoire national.

Expertise Clinique et Commerciale

GSB est reconnu comme un expert mondial en études cliniques, contribuant de manière significative à la progression de la recherche médicale. Au niveau national, GSB excelle dans la vente de médicaments en B2B avec les professionnels de santé et en C2B avec les clients particuliers, consolidant ainsi sa position d'expert en commercialisation de solutions thérapeutiques.

02

BESOIN ET OBJECTIF

Expression des besoins et objectifs

Authentification Sécurisée : Chaque médecin doit avoir un compte unique, protégé par un système d'authentification sécurisé. Cela assure la confidentialité des données médicales et garantit que seuls les professionnels autorisés peuvent accéder aux fonctionnalités de l'application.

Gestion des Profils des Patients : Les profils des patients doivent contenir des informations détaillées, telles que le nom, l'âge, le sexe, les antécédents médicaux et les allergies. Cela permet d'assurer une prise en charge personnalisée et sécurisée.

Création et Gestion des Ordonnances : Les médecins doivent avoir la possibilité de créer, modifier et annuler des ordonnances. Chaque ordonnance doit inclure le nom du médicament, la posologie, la durée du traitement et, si nécessaire, des instructions spéciales. De plus, chaque ordonnance doit pouvoir être exportée au format .pdf pour des utilisations externes.

Base de Données des Médicaments : L'application doit intégrer une base de données complète des médicaments, incluant les informations sur les contre-indications et les interactions médicamenteuses. Elle doit également alerter le médecin en cas d'interactions dangereuses potentielles ou de contre-indications basées sur le profil du patient.

Ces objectifs sont fondamentaux pour développer une application robuste et sécurisée qui répond aux exigences des professionnels de santé. Ils s'inscrivent également dans les meilleures pratiques en matière de gestion des données sensibles et de sécurité des patients.

03

ANALYSE FONCTIONNELLE ET CHOIX TECHNIQUES

Analyse fonctionnelle et choix techniques

Listage des fonctionnalités

L'application créée pour GSB présente diverses fonctionnalités clés, divisées pour objectif finale la création de l'ordonnance

Authentification et gestion des utilisateurs :

- Écran de connexion pour les médecins : Permet aux médecins de se connecter de manière sécurisée.
- Écran de gestion de profil médecin : Offre la possibilité de visualiser et de modifier les informations du profil.

Gestion des patients :

- 1.Consultation de l'historique des fiches précédentes : Permet aux visiteurs de vérifier l'état des fiches passées et d'accéder aux détails.
- 2.Visualisation de la fiche en cours : Offre la possibilité de consulter les détails de la fiche du mois en cours.
- 3.Ajout de frais forfaitaires : Les visiteurs peuvent intégrer des frais prédéfinis à leurs fiches.
- 4.Ajout de frais hors forfait : Permet comparé à la précédente, d'ajouter des frais non compris les tarifs prédéfinis

Gestion des ordonnances :

- Liste des ordonnances par patient : Affiche toutes les ordonnances associées à un patient.
- Création d'une nouvelle ordonnance : Permet de générer une nouvelle ordonnance.
- Modification d'une ordonnance existante : Pour mettre à jour une ordonnance déjà créée.
- Visualisation détaillée d'une ordonnance : Affiche les détails complets d'une ordonnance.
- Fonction d'export d'ordonnance en PDF : Pour exporter et sauvegarder une ordonnance au format PDF.

Base de données des médicaments :

- Liste des médicaments avec fonction de recherche : Pour rechercher des médicaments spécifiques.
- Fiche détaillée d'un médicament : Affiche les informations complètes d'un médicament.
- Informations sur les contre-indications et interactions : Inclut les détails sur les contre-indications et les interactions médicamenteuses.

Création d'ordonnance :

- Interface de sélection des médicaments : Pour choisir les médicaments à prescrire.
- Saisie de la posologie et de la durée du traitement : Pour définir la posologie et la durée du traitement.
- Ajout d'instructions spéciales : Pour ajouter des instructions particulières à l'ordonnance.
- Système d'alerte pour les interactions médicamenteuses et contre-indications : Avertit en cas de potentiels problèmes.

Création d'ordonnance :

- Interface de sélection des médicaments : Pour choisir les médicaments à prescrire.
- Saisie de la posologie et de la durée du traitement : Pour définir la posologie et la durée du traitement.
- Ajout d'instructions spéciales : Pour ajouter des instructions particulières à l'ordonnance.
- Système d'alerte pour les interactions médicamenteuses et contre-indications : Avertit en cas de potentiels problèmes.

Tableau de bord du médecin :

- Vue d'ensemble des patients récents : Affiche les patients récemment consultés.
- Ordonnances en cours : Liste les ordonnances actuellement actives.
- Alertes et notifications : Notifie des interactions médicamenteuses potentielles ou d'autres alertes importantes.

Gestion des allergies et antécédents :

- Interface d'ajout/modification des allergies pour un patient : Pour gérer les allergies d'un patient.
- Interface d'ajout/modification des antécédents médicaux pour un patient : Pour gérer les antécédents médicaux d'un patient.

Rapports et statistiques :

- Rapport sur les prescriptions par période : Génère des rapports sur les prescriptions sur une période donnée.
- Statistiques sur les médicaments les plus prescrits : Affiche les statistiques des médicaments les plus souvent prescrits.

Paramètres de l'application :

- Configuration des alertes : Permet de configurer les alertes de l'application.

Aide et support :

- Guide d'utilisation intégré : Fournit un guide pour l'utilisation de l'application.
- Page de FAQ : Répond aux questions fréquemment posées.
- Formulaire de contact pour le support technique : Pour contacter le support technique en cas de problème.

Langage et technologies utilisées

L'utilisation de C#, ASP.NET Core MVC et Entity Framework pour le développement d'une application de gestion de fiche de frais peut être justifiée de la manière suivante :

C# :

- Sécurité et fiabilité : Dans l'industrie pharmaceutique, la sécurité et la fiabilité des données sont primordiales. C# est un langage fortement typé qui aide à prévenir les erreurs de programmation et assure la fiabilité du code.
- Intégration avec .NET : C# est le langage de programmation principal pour le développement .NET, ce qui signifie qu'il a un excellent support pour les bibliothèques .NET et les outils de développement de Microsoft. Cela peut être particulièrement utile pour intégrer l'application avec d'autres systèmes ou technologies utilisés par GSB.

ASP.NET Core MVC :

- Architecture MVC : Le modèle MVC (Modèle-Vue-Contrôleur) sépare les préoccupations de l'application, ce qui rend le code plus propre et plus facile à gérer. Cela permet également de faciliter les tests unitaires et l'entretien du code.
- Performance : ASP.NET Core est connu pour ses performances élevées et son faible encombrement, ce qui en fait un excellent choix pour les applications web modernes.
- Cross-Platform : Contrairement à la version précédente d'ASP.NET, ASP.NET Core est multiplateforme, ce qui signifie que l'application peut être déployée sur Windows, Linux ou macOS.
- Sécurité : ASP.NET Core offre de nombreuses fonctionnalités de sécurité intégrées, telles que la protection contre les attaques CSRF et XSS, ainsi que l'authentification et l'autorisation basées sur les revendications.

Entity Framework :

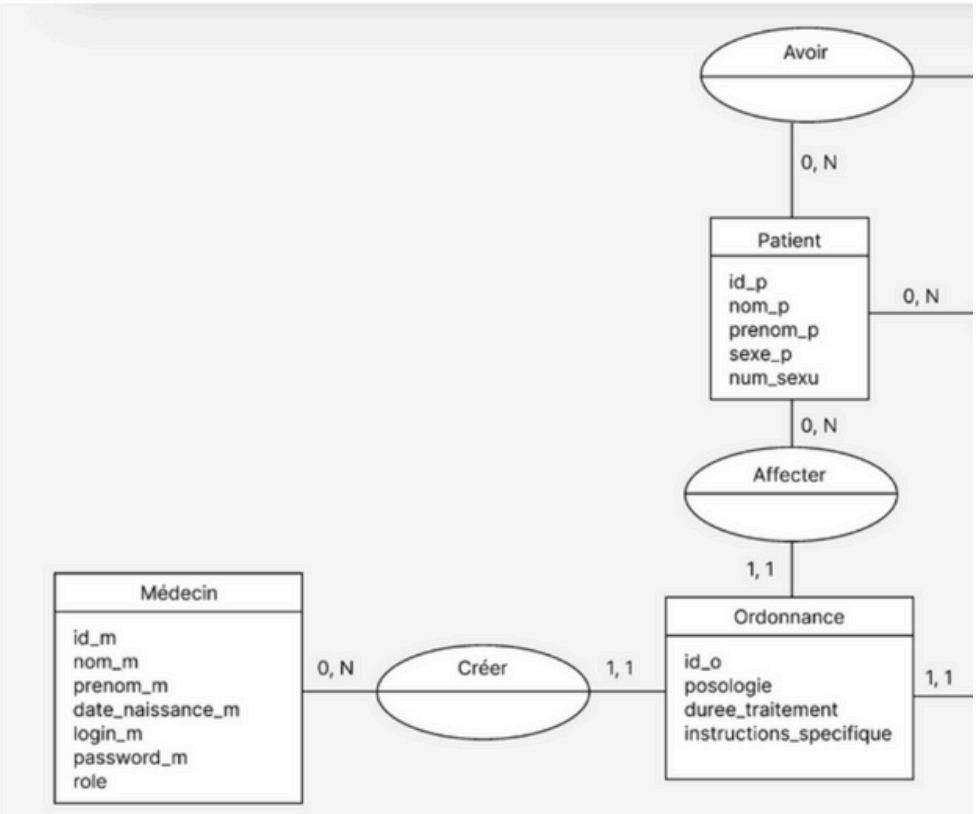
- ORM (Object-Relational Mapping) : Entity Framework est un ORM qui permet aux développeurs de travailler avec des données sous forme d'objets .NET, sans avoir à écrire de code SQL brut. Cela peut accélérer le développement et réduire le risque d'erreurs.
- Productivité : Entity Framework permet la génération automatique de schémas de base de données à partir de modèles de données, ce qui peut augmenter la productivité des développeurs.
- Maintenance : En utilisant Entity Framework, les modifications apportées aux modèles de données sont automatiquement reflétées dans la base de données, ce qui facilite la maintenance de l'application.

Ces technologies sont bien adaptées pour le développement d'une application de gestion des ordonnances médicales, offrant à la fois sécurité, performance et facilité de maintenance.

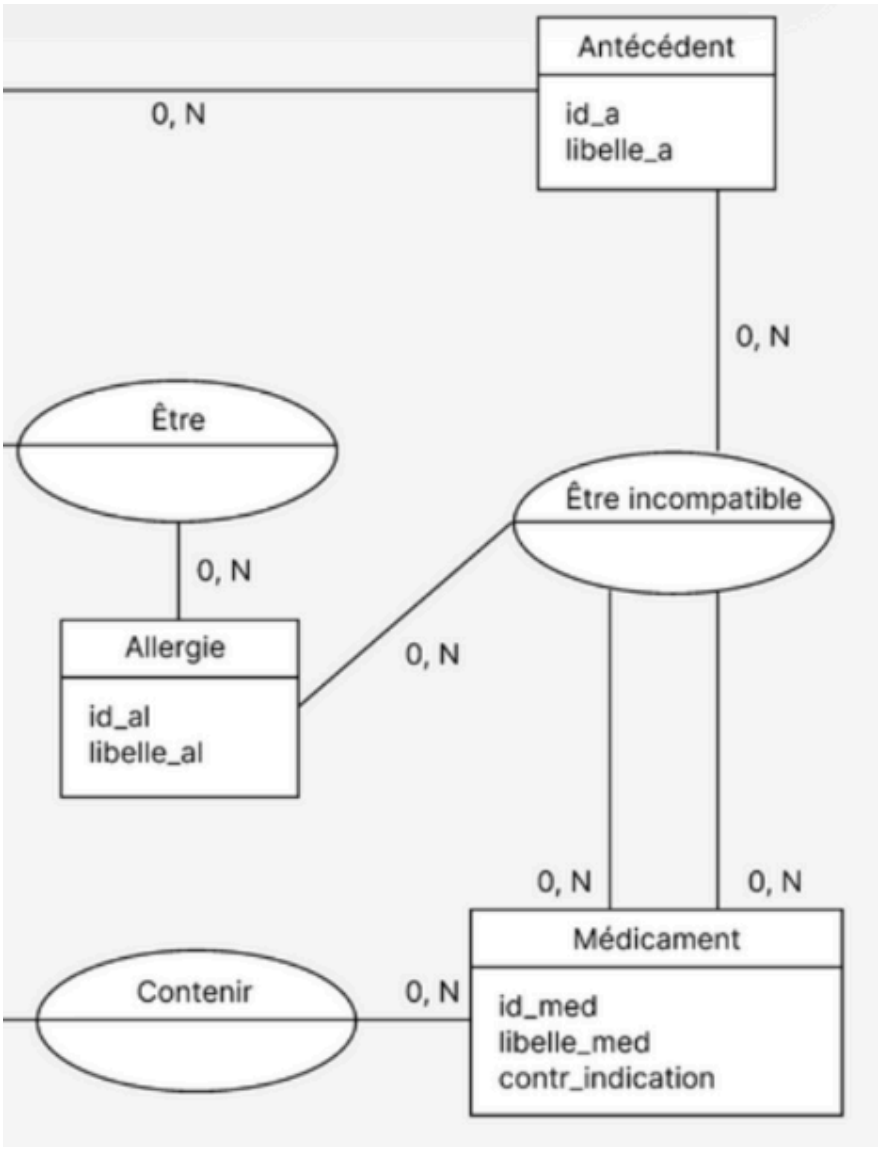
04

MA BDD

MCD



1. Medecin: Cette entité représente les utilisateurs du système. Chaque utilisateur a un identifiant unique (id_m), ainsi que son nom, prénom, date de naissance, login, mot de passe et rôle
2. Ordonnance: Cette entité contient les informations relatives aux ordonnances. Chaque fiche a un identifiant unique (id_o), la posologie, la durée du traitement et les instructions spécifiques
3. Patient: Cette entité contient les informations des patients. Chaque patient a un identifiant unique (id_p), ainsi que son nom, son prénom, son sexe et son numéro de sécurité.



1. Allergie : Cette entité donne tous les types d'allergies existants
2. Antécédent: Cette entité donne tous les types d'allergies existants
3. Médicament: Cette entité contient les informations des médicaments. Chaque médicamenta un identifiant unique (id_med), ainsi que son libelle et sa contre indication

Les relations entre ces entités sont également dépeintes, montrant comment elles interagissent au sein du système pour la gestion des fiches de frais :

- Médecin a une relation 0-à-plusieurs avec Ordonnance.
- Ordonnance a une relation un-à-un avec Patient ainsi qu'avec Médecin
- Allergies et Antécédents ont des relation a une relation 0-à-plusieurs avec Patient et Médicament

MLD

Medicament (id_medicament, nom,
interaction_médicamenteuses)

Ordonnance(id_ordonnance, posologie,
durée_taitement, instruction_spéciale,, #id_patient,
#id_medecin)

PATIENT(id_patient, nom, âge, sexe)

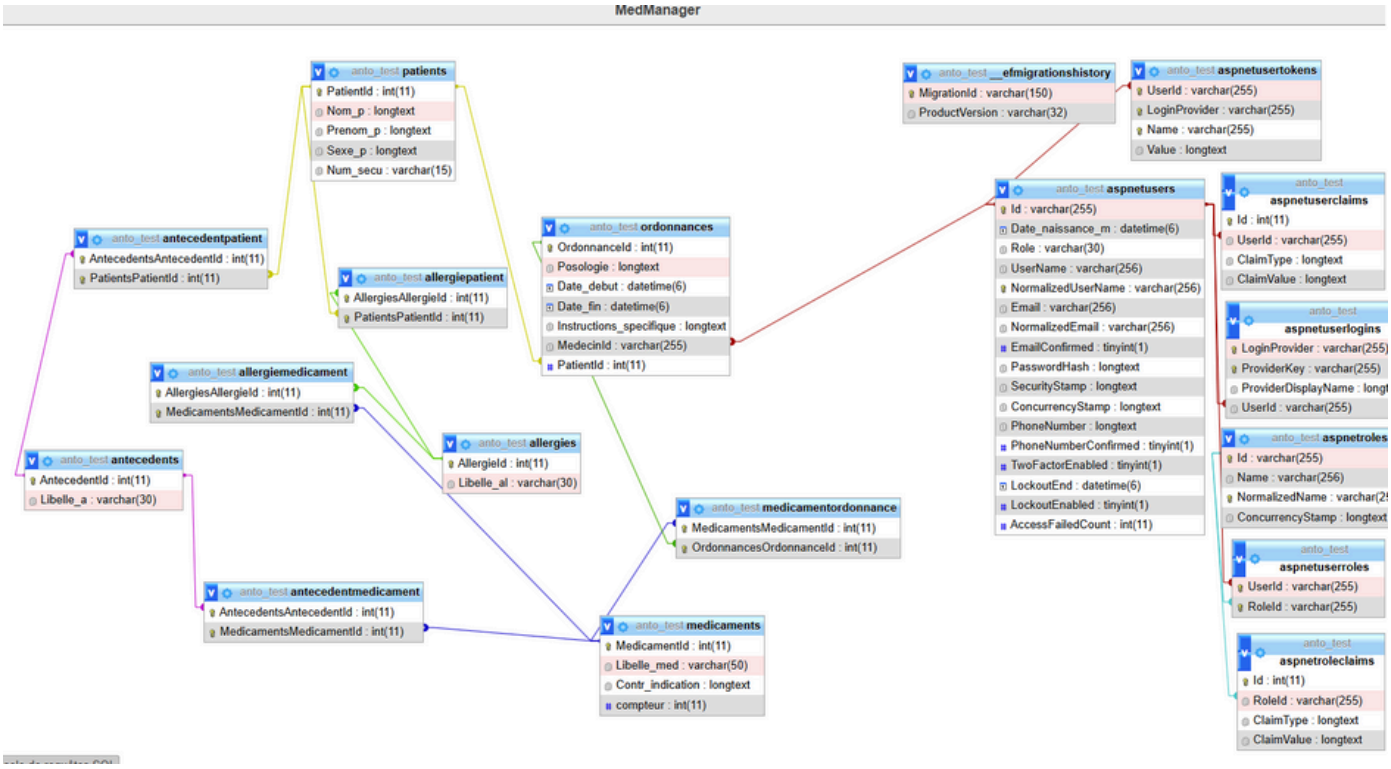
MEDECIN(id_medecin, identifiant, mdp)

POSSEDE(#id_ordonnance, #id_medicament)

INCOMPTABILTE(id_comptabilité, nom ,
#id_medicament, #id_patient)

Une transformation du MCD en MLD permet de mieux structurer les entités pour la création dans la base de donnée, les identifiants sont mis en évidence en étant soulignés et les # permettent d'identifier les clés étrangères, on peut donc remarquer les dépendances entre les entités comme par exemple avec l'id_patient qui a été mis en clé étrangère dans ordonnance.

Schéma conceptuel PHPMyAdmin



Après les 2 dernières étapes, une création de la base de donnée est possible, toutes les informations renseignés avec le MLD sont mises avec les bons types .
Les relations sont bien mis évidence pour faciliter la compréhension et la maintenance pour le futur. Le fait qu'il y ait plus de tables que dans le schéma peut être expliqué par l'ajout de tables de liaisons (intermédiaires) ou encore par l'utilisation de l'Asp Net User qui lui même a beaucoup de tables associés

05

ORGANISTAION DU CODE

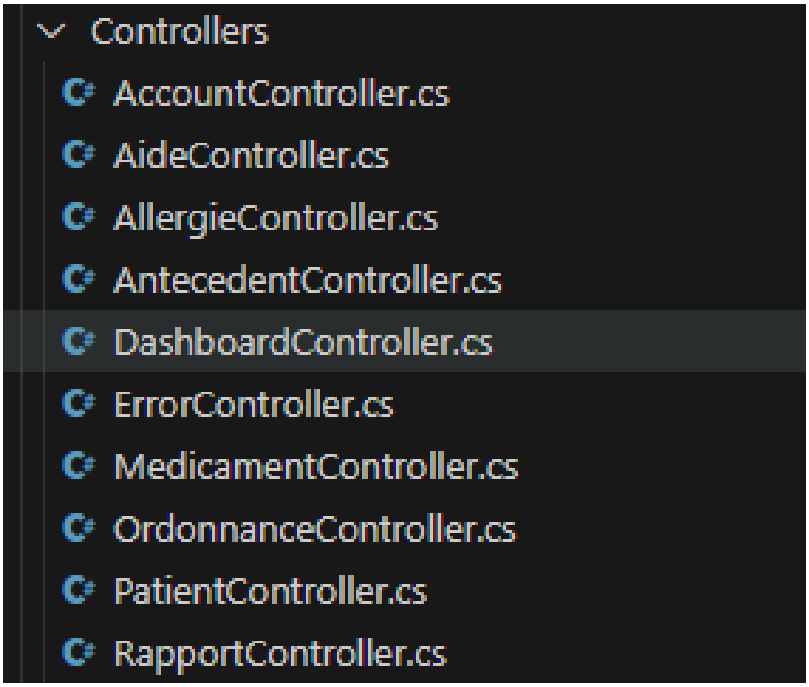
Organisation du code

MVC

L'utilisation du modèle MVC (models / controllers /views) complémenté des ViewModels ainsi que l'applicationContext permet une organisation simple et efficace pour le projet

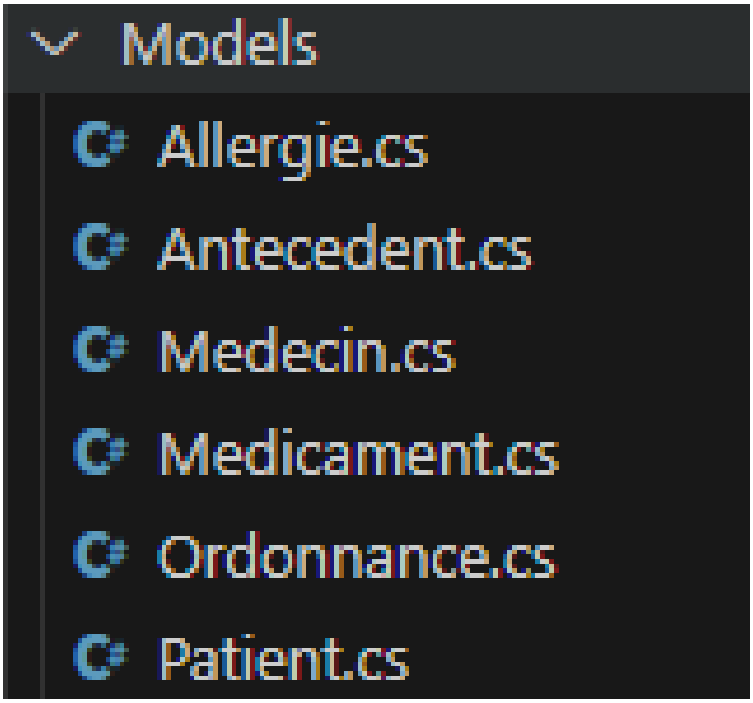
Controllers

Les controllers sont les éléments qui contiennent tous le back end du code, toutes les logiques côtés serveurs sont mis ici.



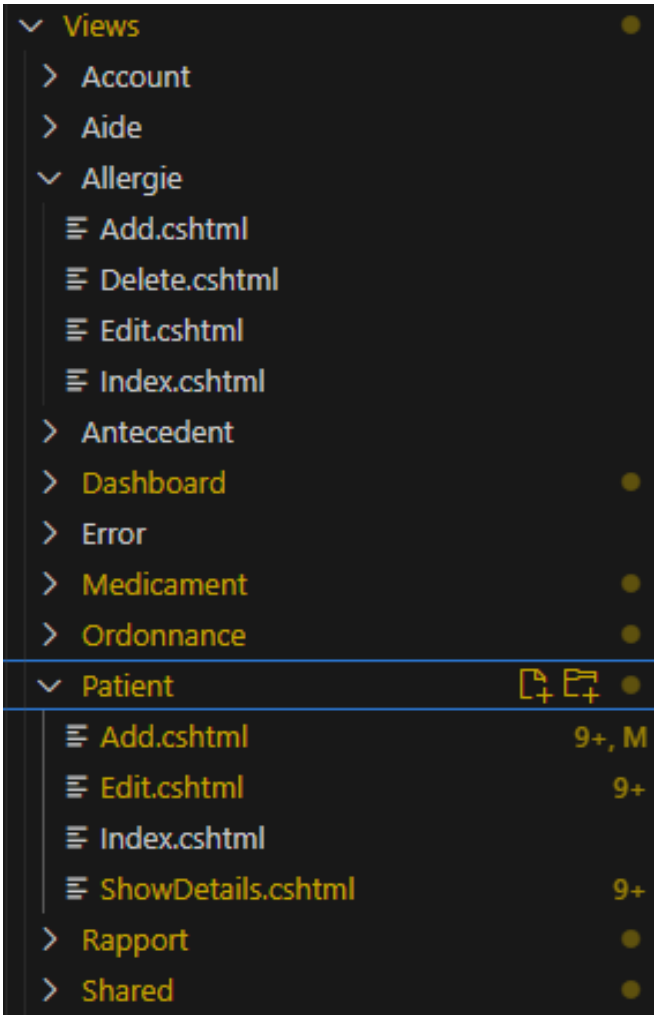
Models

Les models dans mon projet sont les entités nécessaires pour la base de donnée



Views

Les Views sont les éléments affichés en web, elles sont composés en dossier avec le nom du controller ainsi que ce qu'elles retournent



06

PRÉSENTATION DU CODE

Présentation du code

LOGIN

La gestion de l'authentification utilise AspNetCore Identity qui redirige ensuite sur le Dashboard

```
public IActionResult Login()
{
    return View();
}

[HttpPost]
0 references
public async Task<IActionResult> Login(LoginViewModel model)
{
    if (ModelState.IsValid)
    {
        var result = await _signInManager.PasswordSignInAsync(model.UserName, model.Password, model.RememberMe, false);

        if (result.Succeeded)
        {
            return RedirectToAction("Index", "Dashboard");
        }

        ModelState.AddModelError(string.Empty, "Erreur lors du login");
    }

    return View(model);
}
```

La barre de recherche et affichage de l'index

Cette fonction permet en utilisant un input dans la vue de faire une recherche par rapport à un des éléments demandés dans l'exemple, un des attributs du patient, de plus il retourne aussi la liste de patient dans le contexte

```
public IActionResult Index(string searchString)
{
    var patients = from p in _context.Patients select p;
    if (!string.IsNullOrEmpty(searchString))
    {
        patients = patients.Where(p => p.Nom_p.Contains(searchString) || p.Prenom_p.Contains(searchString
        || p.Sexe_p.Contains(searchString) || p.Num_secu.Contains(searchString));
        return View(patients);
    }
    return View(patients);
}
```

ShowDetails()

Dans la majorité des fonctionnalités, il y a un bouton pour afficher le détail, dans le cas de l'exemple il affiche les détails du patient grâce à un ViewModel, cette action est réalisée après avoir affiché la vue index, l'id est d'ailleurs envoyé grâce ay asp-route-id

```
@if (Model.Count() > 0)
{
    <div class="container text-center mt-4 mb-4">
        <div class="row">
            <div class="col-1"></div>
            <table class="table table-dark table-hover">
                <thead>
                    <tr>
                        <th>Nom de famille</th>
                        <th>Prénom</th>
                        <th>Sexe_p</th>
                        <th>Num_secu</th>
                        <th>Modifier</th>
                    </tr>
                </thead>
                <tbody>
                    @foreach (var Patient in Model)
                    {
                        <tr>
                            <td>@Html.DisplayFor(m => Patient.Nom_p)</td>
                            <td>@Html.DisplayFor(m => Patient.Prenom_p)</td>
                            <td>@Html.DisplayFor(m => Patient.Sexe_p)</td>
                            <td>@Html.DisplayFor(m => Patient.Num_secu)</td>

                            @* <TD>@Html.ActionLink("details", "ShowDetails", new { id = @Patient.PatientId })</TD> *@
                            <td><a asp-action="Edit" asp-route-id="@Patient.PatientId" id="accueil" class="btn"> Modifier</a>
                            <a asp-action="ShowDetails" asp-route-id="@Patient.PatientId" id="accueil" class="btn">Détails</a></td>
                        </tr>
                    }
                </tbody>
            </table>
            <div class="col-1"></div>
```

```
public async Task<IActionResult> ShowDetails(int id)
{
    var patient = await _context.Patients
        .Include(p => p.Antecedents)
        .Include(p => p.Allergies)
        .FirstOrDefaultAsync(p => p.PatientId == id);

    if (patient == null)
        return NotFound();

    var viewModel = new PatientEditViewModel
    {
        Patient = patient,
        Antecedents = patient.Antecedents.ToList(),
        Allergies = patient.Allergies.ToList()

    };

    return View(viewModel);
}
```

int Patient.PatientId { get; set; }

Edit()

Pour toutes les fonctionnalités ayant la possibilité de modifier un élément, une logique de vue avec un "Post" est nécessaire.

Il est d'abord nécessaire de charger les données correspondante

```
public async Task<IActionResult> Edit(int id)
{
    var patient = await _context.Patients
        .Include(p => p.Antecedents)
        .Include(p => p.Allergies)
        .FirstOrDefaultAsync(p => p.PatientId == id);

    if (patient == null)
        return NotFound();

    var viewModel = new PatientEditViewModel
    {
        Patient = patient,
        Antecedents = await _context.Antecedents.ToListAsync(),
        Allergies = await _context.Allergies.ToListAsync(),
        SelectedAntecedentIds = patient.Antecedents.Select(a => a.AntecedentId).ToList() ?? new List<int>(),
        SelectedAllergieIds = patient.Allergies.Select(a => a.AllergieId).ToList() ?? new List<int>()
    };

    return View(viewModel);
}
```

Puis de prendre les informations modifiés par l'utilisateur et d'enregistrer les changements

```
[HttpPost, ValidateAntiForgeryToken]
0 references
public async Task<IActionResult> Edit(int id, PatientEditViewModel viewModel)
{
    if (id != viewModel.Patient.PatientId)
    {
        return NotFound();
    }
    if (!ModelState.IsValid)
    {
        viewModel.Antecedents = await _context.Antecedents.ToListAsync();
        viewModel.Allergies = await _context.Allergies.ToListAsync();
        return View(viewModel);
    }
    if (ModelState.IsValid)
    {
        try
        {
            var patient = await _context.Patients
                .Include(p => p.Antecedents)
                .Include(p => p.Allergies)
                .FirstOrDefaultAsync(p => p.PatientId == id);


            if (patient == null)
            {
                return NotFound();
            }

            patient.Nom_p = viewModel.Patient.Nom_p;
            patient.Prenom_p = viewModel.Patient.Prenom_p;
            patient.Sexe_p = viewModel.Patient.Sexe_p;
            patient.Age_p = viewModel.Patient.Age_p;
            patient.Num_secu = viewModel.Patient.Num_secu;
```

```
// Mise à jour des allergies
patient.Allergies.Clear();
if (viewModel.SelectedAllergieIds != null)
{
    var selectedAllergies = await _context.Allergies
        .Where(a => viewModel.SelectedAllergieIds.Contains(a.AllergieId))
        .ToListAsync();
    foreach (var allergie in selectedAllergies)
    {
        patient.Allergies.Add(allergie);
    }
}

// Mise à jour des antécédents
patient.Antecedents.Clear();
if (viewModel.SelectedAntecedentIds != null)
{
    var selectedAntecedents = await _context.Antecedents
        .Where(a => viewModel.SelectedAntecedentIds.Contains(a.AntecedentId))
        .ToListAsync();
    foreach (var antecedent in selectedAntecedents)
    {
        patient.Antecedents.Add(antecedent);
    }
}
_context.Entry(patient).State = EntityState.Modified;
await _context.SaveChangesAsync();
return RedirectToAction(nameof(Index));
```

Add()

Cette fonction fonction comme pour l'edit se déroule en 2 temps, l'entrée des informations de l'utilisateur puis d'une récupération de ces données pour les envoyer à la base de donnée, il y a bien sûr une vérification sur les données envoyées, la fonction appelé du contexte est de plus différente

```
    _context.Patients.Add(patient);
    await _context.SaveChangesAsync();
```

L'export en PDF

Cette fonction pour les ordonnances utilise le nugget Rotativa.AspNetCore

```
    _context.Patients.Add(patient);
    await _context.SaveChangesAsync();
    var viewResult = await PdfOrdonnance(id) as ViewResult;

    if (viewResult == null)
    {
        return NotFound();
    }

    var pdfResult = new ViewAsPdf("PdfOrdonnance", viewResult.Model)
    {
        FileName = "Ordonnance.pdf"
    };

    return pdfResult;
}
```

La vérification des incomptabilités

Cette fonction renvoie false si une allergie ou un antécédent est détecté à la fois dans la liste du patient et dans celle du médicament. L'une des fonctions qui récupère une de ces listes, nommée GetAllergiesMed, s'apparente à ceci :

```
public List<Allergie> GetAllergiesMed(Ordonnance ordonnance)
{
List<Allergie> allergies = new List<Allergie>();
if (ordonnance == null)
{
    throw new ArgumentNullException(nameof(ordonnance), "Une erreur imprévu sur l'ordonnace a été trouvé.");
}

if (ordonnance.Medicaments != null)
{
    foreach (var medicament in ordonnance.Medicaments)
    {
        if (medicament.Allergies != null)
        {
            foreach(var allergie in medicament.Allergies)
                allergies.Add(allergie);
        }
    }

}
return allergies;
```

```
public bool VerifyImpossibility(Ordonnance ordonnance)
{
    List<Allergie> allergiesMedicaments = GetAllergiesMed(ordonnance);
    List<Antecedent> antecedentsMedicaments = GetAntecedentsMed(ordonnance);
    if(ordonnance.Patient == null)
    {
        throw new ArgumentNullException(nameof(ordonnance.Patient), "Une erreur imprévu sur le patient a été trouvé.");
    }
     List<Allergie> allergiesPatient = ordonnance.Patient.Allergies;
     foreach (Allergie allergieM in allergiesMedicaments)
    {
        foreach (Allergie allergieP in allergiesPatient)
        {
            if (allergieM == allergieP)
            {
                return false;
            }
        }
    }
    List<Antecedent> antecedentsPatient = ordonnance.Patient.Antecedents;
     foreach (Antecedent antecedentM in antecedentsMedicaments)
    {
        foreach (Antecedent antecedentP in antecedentsPatient)
        {
            if (antecedentM == antecedentP)
            {
                return false;
            }
        }
    }
    return true;
}
```

Voici la vérification grâce à des boucles for dans des boucles for

PeriodPrescription

Une fonctionnalité permettant d'afficher toutes les ordonnances selon des dates précisés

```
    [Authorize]
    0 references
    public IActionResult PeriodPrescription()
    {

    PeriodPrescriptionViewModel viewModel = new PeriodPrescriptionViewModel {
        DateDebut = DateTime.Now,
        DateFin = DateTime.Now.AddDays(1)
    };
        return View(viewModel);
    }

    [HttpPost]
    [ValidateAntiForgeryToken]
    0 references
    public async Task<IActionResult> PeriodPrescription(PeriodPrescriptionViewModel viewModel)
{
if (viewModel.DateDebut > viewModel.DateFin)
{
    ModelState.AddModelError("", "Vérifiez les dates.");
    return View(viewModel);
}
string? medecinId = _userManager.GetUserId(User);
List<Ordonnance> ordonnancesPeriod = await _context.Ordonnances
    .Where(o => o.Date_debut >= viewModel.DateDebut && o.Date_fin <= viewModel.DateFin)
    .Include(o => o.Patient)
    .Include(o => o.Medecin)
    .Include(o => o.Medicaments).Where(m => m.MedecinId == medecinId)
    .ToListAsync();

return View("PeriodPrescriptionShow", ordonnancesPeriod);
}
```

07

PRÉSENTATION APPLICATION

Présentation de l'application

LOGIN

Au lancement de l'application, la page de Login s'affiche avec 2 zones de textes dont 1 en masquée et 2 boutons.

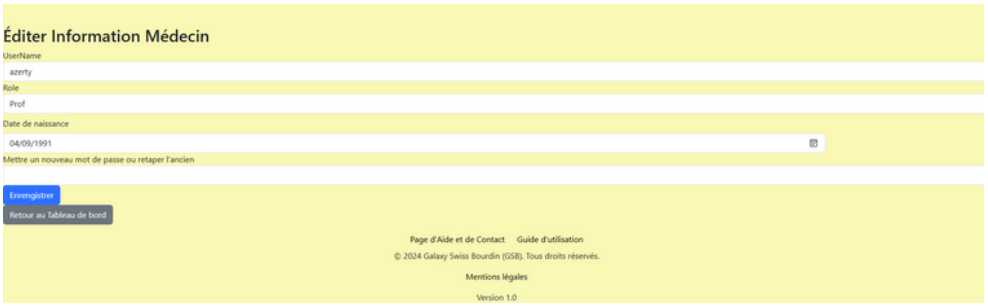
Un click sur le bouton login redirigera vers le bonne page si les données sont bonnes, il y de plus une option pour créer un compte

Dashboard

Après être connecté, une page du tableau de bord est affiché avec toutes les fonctionnalités possibles ainsi que les ordonnances en cours et les patients des 30 derniers jours

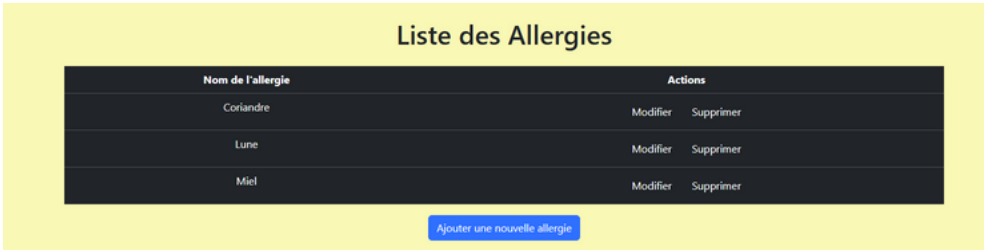
Modifier le profil

Il est possible en interagissant avec la navbar de modifier le profil



Gestion des allergies / antécédents

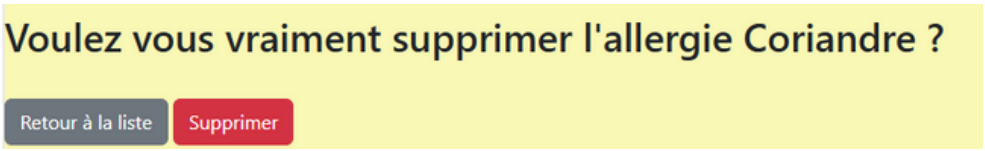
Dans cette fonctionnalité qui est la même dans les 2 cas, les éléments déjà existants sont montrés l'utilisateur peut ajouter, modifier ou supprimer un élément



Une création a juste besoin du libelle

De la même manière que le modifier

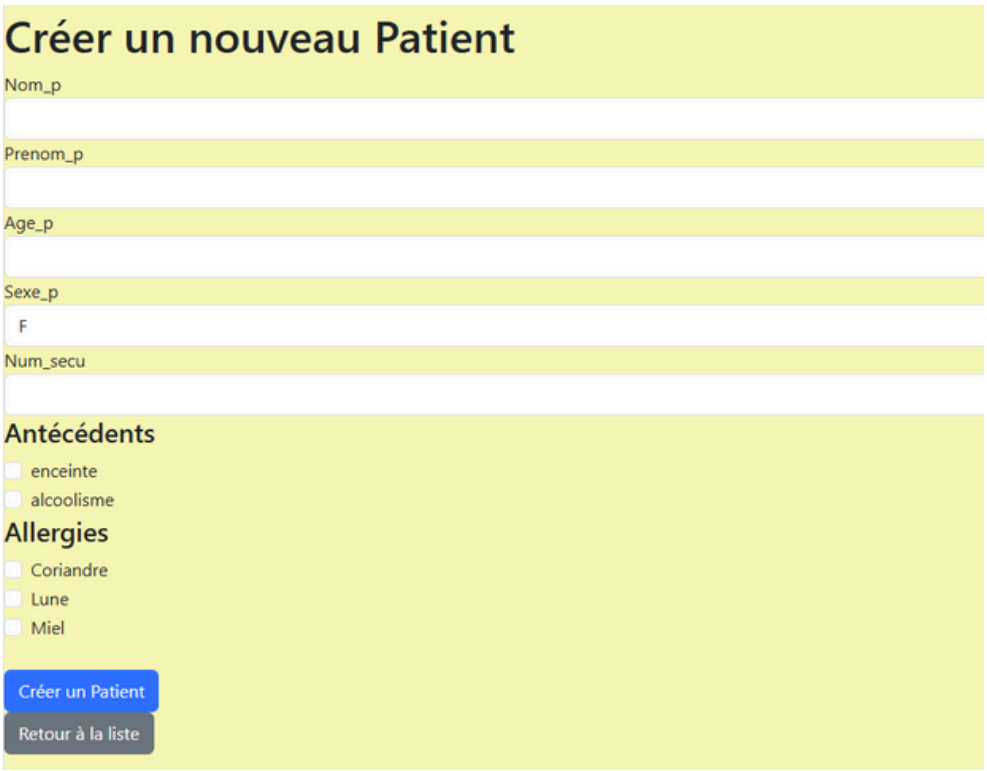
Une représentation du supprimer ressemblerait à cela



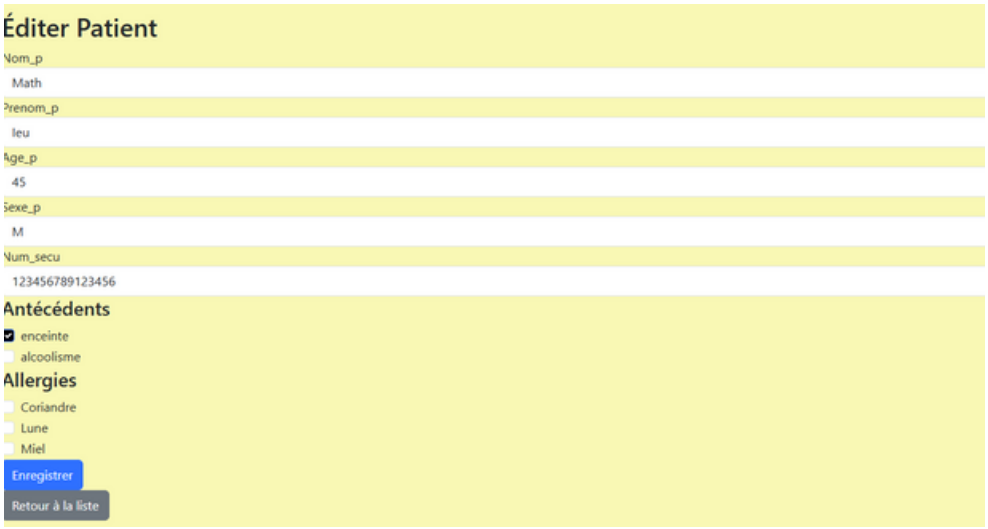
Gestion des patients

De la même manière, il y est affiché tous les patients déjà existant avec une possibilité d'ajouter, de modifier ou de montrer le détail

En comparaison aux allergies et aux antécédents l'utilisateur lors de l'addition d'un patient, se voit la possibilité de choisir quels allergies ou antécédents le patient possède



Le modifier permet de la même manière pour les allergies et les antécédents de modifier les informations de l'entité



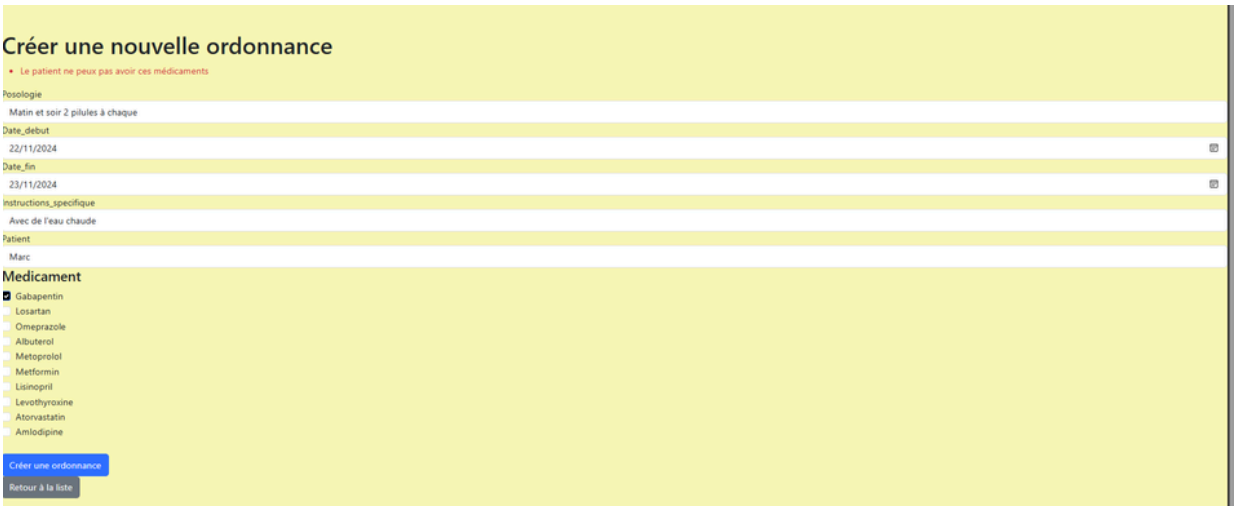
Le détail de la même manière que le modifier montre les informations du patient mais dans ce cas sans la possibilité de les modifier

Gestion des médicaments

Cette fonctionnalité sert à ajouter, modifier ou montrer les détails d'un médicament, les propriétés sont différentes mais il y a la logique des antécédents et des allergies avec lesquelles la maladie ne convient pas, le reste garde les mêmes logiques que pour le patient

Gestion d'ordonnances

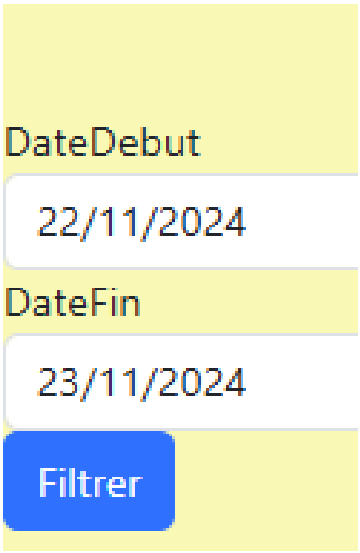
Cette fonctionnalité sert à ajouter, modifier ou montrer les détails d'une ordonnance, la logique étant la même à part que les éléments sélectionnés sont les médicaments, nous pouvons observer un message d'erreur si un des des médicaments ne convient pas au patient



- Le patient ne peux pas avoir ces médicaments

Afficher les prescriptions sur une période

Il est possible d'afficher toutes les ordonnances dans un écart de temps demandé

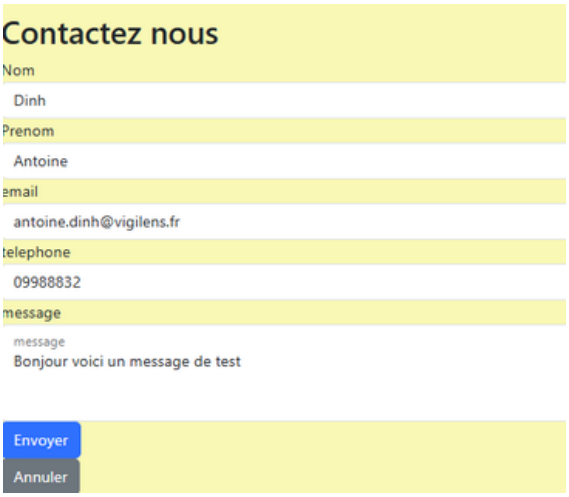


Afficher l'utilisation des médicaments

Il est aussi possible d'afficher le nombre d'utilisation d'un médicament en tout

Contact

Il est possible d'envoyer un message qui sera envoyé sur un mail préparé pour l'application



Page de Faq

Une page de faq est de plus disponible en cliquant sur le footer ainsi que ce rapport qui répondra au questions "de base".

08

ANNEXES

GITHUB

Lien github dans l'éventualité où il y a un problème avec le code envoyé :

https://github.com/Goyef/MedManager

ANTOINE DINH

Isitech

BTS SIO SLAM

Novembre 2024