Medilab

Projet d'intégration de développement

Professeur : Mr Eric Poncin

Tistan Jacqmard & Vincent Vanhees

Année scolaire 2023/2024

ECI
Ecole de Commerce et d'Informatique

Table des matières

1 Rapport de l'application

1.1 Introduction

Dans le cadre de la modernisation des pratiques de gestion au sein d'un cabinet médical, une application spécialement conçue pour automatiser et rationaliser les processus est devenue au centre de nos occupations. Cette application de gestion de cabinet médical vise à centraliser les tâches administratives et cliniques, améliorant ainsi l'efficacité et la qualité des soins offerts aux patients.

1.2 Objectif de l'Application

L'objectif principal de l'application est de fournir une plateforme intégrée qui assiste le personnel médical et administratif dans la gestion quotidienne des activités du cabinet. Cela inclut la gestion des dossiers des patients, la planification des rendez-vous, le suivi des traitements médicaux, et la gestion administrative du cabinet.

1.3 Fonctionnalités Détaillées

1.3.1 Gestion des Utilisateurs

- Authentification et Autorisation : L'application propose un système robuste d'authentification et de gestion des rôles, permettant différentes niveaux d'accès (administrateur, médecin, secrétaire). Chaque rôle a des permissions spécifiques qui contrôlent l'accès aux données et aux fonctionnalités de l'application.
- Gestion de Profil : Les utilisateurs peuvent gérer leur propre profil, y compris les informations personnelles et les paramètres de compte, ce qui permet une personnalisation et une sécurité accrues.

1.3.2 Tableaux de Bord Multifonctionnels

- Vue Administrateur : Offre une vue d'ensemble sur toutes les opérations du cabinet, incluant des rapports détaillés sur l'activité des médecins, l'utilisation des ressources, et les finances.
- Vue Médecin : Permet aux médecins d'accéder rapidement à leur emploi du temps, aux dossiers des patients, et aux alertes médicales importantes.
- Vue Secrétaire : Facilite la gestion des rendez-vous, l'accueil des patients, et la coordination des agendas des médecins.

1.3.3 Gestion des Patients

- Enregistrement et Suivi : Permet un enregistrement complet des patients, de leurs informations médicales et de leurs antécédents. L'application permet également le suivi continu des conditions des patients et de leurs traitements.
- Accès et Sécurité des Données : Les données des patients sont protégées par des protocoles de sécurité stricts, assurant la confidentialité et l'intégrité des informations.

1.3.4 Planification des Rendez-vous

- Interface Intuitive : Les patients peuvent prendre rendez-vous en ligne via une interface facile à utiliser. Les médecins et les secrétaires peuvent également gérer ces rendez-vous directement depuis leurs tableaux de bord.
- Gestion des Horaires : L'application permet une gestion efficace des horaires des médecins, évitant les surréservations et optimisant le temps de chaque praticien.

1.3.5 Gestion Médicale et Administrative

- Congés et Absences : Gère les demandes de congés des médecins, assurant une couverture adéquate du personnel en tout temps.
- Facturation et Paiements : Intègre des fonctionnalités de facturation automatique et de traitement des paiements, réduisant les erreurs et améliorant la rapidité des transactions financières.

1.3.6 Architecture Technique

L'architecture de l'application est basée sur une séparation claire entre le backend (PHP et MySQL pour la gestion des données et la logique applicative) et le frontend (HTML, CSS, et JavaScript pour l'interface utilisateur). Le système utilise un modèle de routage centralisé, ce qui simplifie la maintenance et l'évolution de l'application. Le fichier index.php joue le rôle de routeur frontal, dirigeant les requêtes vers les contrôleurs appropriés en fonction de l'action demandée.

1.3.7 Sécurité

La sécurité est une priorité absolue dans la gestion des données de santé. L'application implémente des sessions sécurisées, une validation et un assainissement rigoureux des entrées pour prévenir les injections SQL et les attaques XSS, ainsi qu'un cryptage fort des données sensibles pour garantir la confidentialité et la sécurité des informations des patients.

1.3.8 Conclusion

Cette application de gestion de cabinet médical est une solution complète qui répond aux besoins modernes des établissements de santé. Elle offre non seulement des outils de gestion efficaces mais garantit aussi une expérience utilisateur optimale pour le personnel et les

patients. Avec son approche modulaire et son architecture flexible, elle est prête à s'adapter aux évolutions futures de la technologie et aux besoins du secteur médical.

2 Cahier des charges

2.1 Exigences fonctionnelles

2.1.1 Gestion des utilisateurs (Tristan)

2.1.1.1 Conception de la base de données

Tristan établit la structure de la base de données nécessaire pour stocker les informations des utilisateurs, en s'assurant de la normalisation pour éviter la redondance et faciliter la maintenance.

2.1.1.2 Création des modèles

Il développe des modèles suivant le pattern MVC, qui servent à manipuler les données des utilisateurs (création, lecture, mise à jour, suppression - CRUD).

2.1.1.3 Mise en œuvre des contrôleurs

Tristan implémente les contrôleurs qui interagissent avec les modèles pour traiter la logique métier et répondre aux requêtes de l'interface utilisateur, garantissant ainsi une gestion sécurisée des utilisateurs, avec authentification et autorisation.

2.1.2 Gestion des rendez-vous (Vincent)

2.1.2.1 Planification des rendez-vous

Vincent développe une interface permettant aux utilisateurs de planifier des rendez-vous, avec des options pour choisir les dates, les heures et le médecin.

2.1.2.2 Modification et annulation

Il permet également aux utilisateurs de modifier ou d'annuler des rendez-vous existants, avec des vérifications pour s'assurer que les changements respectent les règles d'affaires.

2.1.3 Gestion des patients (Vincent et Tristan)

2.1.3.1 Interface utilisateur

Ils collaborent pour créer des interfaces intuitives pour l'ajout, la modification et la visualisation des dossiers patients.

2.1.3.2 Historique médical

L'application permet aussi de suivre et d'accéder à l'historique médical complet des patients, facilitant ainsi l'accès rapide à ces informations cruciales par le personnel médical.
Tableaux de bord

2.1.3.3 Structure (Tristan)

Tristan configure la structure des tableaux de bord, intégrant des visualisations des données pour une compréhension rapide de la situation du cabinet.

2.1.4 Gestion des médecins (Tristan)

Tristan met en place les fonctionnalités pour gérer les informations des médecins, incluant leurs spécialisations, horaires et disponibilités, facilitant ainsi la planification des rendez-vous et la gestion des ressources.

2.2 Exigences non fonctionnelles

2.2.1 Sécurité (Tristan)

Tristan met en œuvre des mesures de sécurité robustes, telles que le cryptage des données, les contrôles d'accès basés sur les rôles, et la protection contre les vulnérabilités Web communes.

2.2.2 Performance (Tristan et Vincent)

Ils s'assurent que l'application est optimisée pour gérer un volume élevé de transactions sans dégradation des performances, en utilisant des techniques comme la mise en cache et l'optimisation des requêtes de base de données.

2.2.3 Fiabilité (Vincent)

Vincent assure que l'application est stable et fiable, avec peu ou pas de temps d'arrêt, et capable de récupérer rapidement des erreurs ou des pannes.

2.2.4 Maintenabilité (Tristan)

Tristan se concentre sur la maintenabilité en écrivant un code clair, bien commenté, et en suivant les meilleures pratiques de développement pour faciliter les mises à jour futures et la maintenance.

2.2.5 Architecture du système (Tristan)

2.2.5.1 Modèle MVC

Tristan décrit comment l'architecture MVC aide à séparer les préoccupations dans l'application, rendant le code plus modulaire, plus facile à tester et à maintenir.

2.2.5.2 Base de données

Il détaille la conception de la base de données, les types de données stockées, et comment les différentes tables se rapportent les unes aux autres pour soutenir les processus métier de l'application.

2.2.6 Technologies utilisées

2.2.6.1 Frontend (Vincent et Tristan)

Ils utilisent HTML, CSS, et JavaScript pour créer une interface utilisateur réactive. Le framework Bootstrap a été employé pour améliorer l'expérience utilisateur et faciliter le développement.

2.2.6.2 Backend (Tristan)

Tristan utilise le langage de programmation PHP pour gérer la logique côté serveur et l'interaction avec la base de données.

2.2.6.3 Base de données (Tristan)

Il sélectionne un système de gestion de base de données MySQL pour stocker et gérer les données de manière efficace et sécurisée.

3 Vue d'Ensemble des Opérations Système

3.1 Index.php

3.1.1 Contenu

3.1.1.1 Rôle

Agit comme un routeur frontal, décidant quel contrôleur à utiliser basé sur l'action demandée. Il initie les contrôleurs et dirige les appels aux méthodes appropriées, en fonction de la logique définie.

3.1.1.2 Initialisation et configuration

Démarrage de la session : session_start() permet de démarrer une nouvelle session ou de reprendre une session existante. Ceci est crucial pour gérer les sessions utilisateurs à travers différentes pages.

- Inclusion des contrôleurs : Les fichiers des contrôleurs principaux (UserController, AdminController, MedecinController, SecretaireController) sont inclus. Ces contrôleurs contiennent la logique métier de l'application.

3.1.1.3 Création des instances des contrôleurs

Des instances de chaque contrôleur sont créées. Ces instances permettent d'appeler les méthodes spécifiques à chaque action déclenchée par l'utilisateur.

3.1.1.4 Gestion des actions

Détermination de l'action : La variable $action est utilisée pour déterminer quelle action doit être exécutée. Elle est récupérée de la superglobale $_GET. Si aucune action n'est spécifiée, une valeur par défaut 'default' est utilisée.

- Gestion des droits d'accès : Une fonction checkAccess est définie pour vérifier si l'utilisateur courant a le droit d'accéder à une fonctionnalité spécifique basée sur son rôle. Elle compare le rôle de l'utilisateur avec le rôle requis pour l'action.

3.1.1.5 Dispatching des actions

Le routeur utilise une structure switch pour dispatcher l'action à la méthode correspondante dans le contrôleur approprié.

- Actions Utilisateur : Connexion, inscription, profil, et déconnexion.
- Actions Admin : Gestion de tableaux de bord, utilisateurs, médecins, patients, etc.
- Actions Médecin : Gestion des congés, des historiques médicaux, des patients, et des rendez-vous médicaux.

Actions Secrétaire : Gestion des patients, des rendez-vous, et modification des horaires de travail.

3.1.1.6 Sécurité et contrôle d'accès

Pour chaque action nécessitant une vérification de rôle, checkAccess est appelé pour s'assurer que l'utilisateur a les droits nécessaires pour exécuter l'action.

Les actions sensibles comme la modification ou la suppression de données vérifient également le rôle avant d'exécuter l'opération.

3.1.2 Examen de la fonction 'checkAccess'

3.1.2.1 Rôle

La fonction checkAccess sert de mécanisme de contrôle d'accès pour restreindre l'exécution de certaines actions aux utilisateurs disposant des rôles appropriés. Elle s'assure que seuls les utilisateurs autorisés peuvent accéder à certaines parties de l'application. Elle aide à prévenir les accès non autorisés et les éventuelles failles de sécurité qui pourraient en résulter.

3.1.2.2 Définition de la fonction

```
function checkAccess($role, $requiredRole) {
    if ($role !== $requiredRole) {
        echo "Accès refusé";
        exit();
    }
}
```

3.1.2.3 Paramètres

- $role : Le rôle actuel de l'utilisateur, qui est récupéré de la session utilisateur. Ce rôle est utilisé pour vérifier si l'utilisateur a le droit d'accéder à une fonctionnalité donnée.
- $requiredRole : Le rôle requis pour accéder à une fonctionnalité spécifique. Ce rôle est défini statiquement dans le code pour chaque action qui nécessite une vérification d'accès.

3.1.3 Fonctionnalité

3.1.3.1 Comparaison des rôles

La fonction compare le rôle actuel de l'utilisateur ($role) au rôle requis ($requiredRole).
Si les rôles ne correspondent pas, l'accès à la fonctionnalité demandée est refusé.

3.1.3.2 Gestion de l'accès refusé

Si un utilisateur tente d'accéder à une fonctionnalité pour laquelle il n'a pas le bon rôle, un message "Accès refusé" est affiché.

- La fonction exit() est appelée immédiatement après pour arrêter l'exécution du script. Cela empêche toute exécution ultérieure de code qui pourrait compromettre la sécurité de l'application ou accéder à des données sensibles.

3.1.3.3 Utilisation dans le routeur

La fonction checkAccess est utilisée dans le routeur frontal (index.php) pour sécuriser les routes qui nécessitent une certaine autorisation basée sur le rôle de l'utilisateur.

3.2 Controllers

3.2.1 AdminController

3.2.1.1 Rôle

Ce contrôleur agit comme le cerveau de la partie administrateur de votre application, coordonnant les interactions entre les données stockées (modèles) et leur représentation (vues), tout en s'assurant que seuls les utilisateurs autorisés peuvent effectuer certaines opérations.

3.2.1.2 Initialisation et Construction (__construct)

Initialise les instances des modèles et de la vue : Chaque modèle spécifique à des entités comme UserModel, MedecinModel, PatientModel, etc., est instancié. Cela permet au contrôleur d'interagir directement avec la base de données à travers ces modèles pour manipuler les données.
Création de la vue administrative (AdminView) : Permet de rendre les interfaces utilisateur spécifiques à l'administration.

3.2.1.3 Authentification et Accès (isUserAdmin)

Vérifie le rôle de l'utilisateur : S'assure que l'utilisateur actuel a le rôle 'Admin' avant de permettre l'accès aux fonctions administratives.

3.2.1.4 Affichage des Tableaux de Bord

Gestion des tableaux de bord : Des méthodes comme dashboard, dashboardUser, dashboardMedecin, etc., vérifient d'abord les privilèges de l'utilisateur avant de rendre les tableaux de bord respectifs. Ils utilisent les données récupérées via les modèles et les affichent en utilisant AdminView.

3.2.1.5 Gestion CRUD (Créer, Lire, Mettre à jour, Supprimer)

- Manipulation des entités : Les méthodes editUser, updateUser, deleteUser, etc., gèrent les opérations CRUD pour différentes entités (utilisateurs, médecins, patients, etc.). Ces fonctions interagissent avec les modèles correspondants pour effectuer des opérations sur la base de données et refléter les changements dans l'interface utilisateur.
- Formulaires et Mise à jour : Les fonctions d'édition (editUser par exemple) préparent et affichent les formulaires pour la modification, qui sont ensuite traitées par des méthodes de mise à jour (updateUser) qui appliquent les modifications dans la base de données.
- Suppression : Les méthodes de suppression (deleteUser) suppriment les entités de la base de données et redirigent vers le tableau de bord approprié.

3.2.1.6 Gestion Spécifique (Congés, Factures, Historique Médical, etc.)

- Tableaux de bord spécifiques : Des méthodes comme dashboardConges, dashboardFactures, et dashboardHistorique affichent des informations spécifiques liées à des congés, des factures, et des historiques médicaux.
- Fonctions de mise à jour spécialisées : Ces méthodes gèrent également l'édition, la mise à jour, et la suppression pour ces catégories spécifiques.

3.2.1.7 Redirection et Gestion des Erreurs

- Redirections : Après une mise à jour ou une suppression, l'utilisateur est redirigé vers le tableau de bord approprié pour voir les résultats de l'opération.
- Gestion des erreurs : Si des erreurs surviennent lors de la récupération ou la modification des données, elles sont gérées et des messages d'erreur sont affichés.

3.2.2 BaseController

3.2.2.1 Rôle

Simplifie la gestion des tâches communes telles que la vérification des permissions d'accès, la redirection des utilisateurs, le chargement des vues, et la gestion des erreurs. Par sa conception, il permet aux contrôleurs dérivés de se concentrer sur des tâches plus spécifiques sans se soucier des aspects répétitifs de la manipulation des sessions, des redirections, ou des erreurs, tout en assurant une cohérence à travers l'application.

3.2.2.2 Constructeur (__construct)

Initialisation de la connexion à la base de données : Prend une connexion à la base de données ($dbConnection) comme paramètre et l'assigne à la propriété $db de la classe pour être utilisée par les classes héritées.

3.2.2.3 Gestion des Sessions et des Rôles

- Vérification de la connexion (isLoggedIn) : Vérifie si l'utilisateur est actuellement connecté en consultant l'existence d'une variable de session spécifique (user_id). Cette méthode est utilisée pour contrôler l'accès à certaines parties de l'application qui nécessitent une authentification.
- Vérification du rôle administrateur (isAdmin) : En plus de vérifier si l'utilisateur est connecté, cette méthode vérifie également si le rôle de l'utilisateur correspond à 'admin'. Cela est souvent utilisé pour restreindre l'accès aux fonctionnalités administratives.

3.2.2.4 Redirection

Redirection de l'utilisateur (redirect) : Cette méthode prend un emplacement (URL) en paramètre et redirige l'utilisateur vers cet emplacement. C'est un outil utile pour la

navigation post-action, comme après la soumission d'un formulaire ou une opération CRUD réussie.

3.2.2.5 Chargement des Vues

Chargement des vues (loadView) : Charge un fichier de vue basé sur le nom de la vue donné et passe des données à cette vue. Utilise extract pour transformer les éléments de l'array $data en variables, rendant ces données accessibles dans la vue sous forme de variables individuelles.

3.2.2.6 Gestion des Erreurs

Gestion des erreurs (handleError) : Prend un message d'erreur en paramètre, puis redirige vers une page d'erreur ou log l'erreur, dépendant de l'implémentation souhaitée. Cette méthode est cruciale pour une gestion d'erreur cohérente à travers l'application.

3.2.3 CategoryController

3.2.3.1 Rôle

Encapsule et sépare les préoccupations liées à la gestion des catégories. Il interagit avec le modèle pour obtenir ou modifier les données et avec la vue pour présenter ces données, suivant ainsi le principe de séparation des préoccupations du modèle MVC. Ce contrôleur facilite la maintenance et l'évolution de la partie catégorie de l'application, car il centralise la logique de gestion des catégories en un seul endroit, rendant le code plus organisé et facile à gérer.

3.2.3.2 Initialisation (__construct)

Initialisation des modèles et des vues : À la construction, le contrôleur initialise les instances de CategoryModel et CategoryView. Cela permet au contrôleur de manipuler les données de catégorie et d'interagir avec la couche de présentation.

3.2.3.3 Listage des Catégories (listCategories)

Récupération et Affichage des Catégories : Cette méthode fait appel à getAllCategories du modèle pour récupérer toutes les catégories disponibles. Les données récupérées sont ensuite passées à displayCategories de la vue pour être affichées à l'utilisateur.

3.2.3.4 Visualisation d'une Catégorie (viewCategory)

Affichage Détail d'une Catégorie : Après avoir récupéré une catégorie spécifique par son ID via getCategoryById, cette méthode vérifie si la catégorie existe. Si c'est le cas, displaySingleCategory de la vue est appelée pour afficher les détails de la catégorie. En cas d'échec, un message d'erreur est affiché.

3.2.3.5 Création d'une Catégorie (createCategory)

Création d'une Nouvelle Catégorie : Cette fonction prend en entrée un nom et une description pour créer une nouvelle catégorie. Elle fait appel à createCategory du modèle pour insérer les données dans la base. En cas de succès, un message de confirmation est affiché, sinon un message d'erreur.

3.2.4 MedecinController

3.2.4.1 Rôle

Centralise la gestion des activités des médecins, simplifiant ainsi l'interaction avec les données médicales, les congés, et les patients à travers des interfaces dédiées. Chaque méthode est conçue pour faciliter une interaction spécifique avec la base de données ou l'interface utilisateur, en maintenant une séparation claire entre la logique métier et la présentation, ce qui est au cœur de l'architecture MVC. Ce contrôleur assure également que les données sensibles sont manipulées de manière sécurisée et que les médecins ont un accès facile à toutes les fonctionnalités nécessaires pour leur pratique quotidienne.

3.2.4.2 Initialisation (__construct)

Initialise les modèles et la vue : Crée des instances pour divers modèles comme MedecinModel, PatientModel, CongesMedecinsModel, etc., ainsi que pour MedecinView. Cela permet au contrôleur de manipuler des données spécifiques aux médecins et d'interagir avec la vue pour la présentation.

3.2.4.3 Gestion des Médecins

Liste des médecins (listMedecins) : Récupère et affiche la liste de tous les médecins.
Affichage d'un médecin (viewMedecin) : Récupère les détails d'un médecin spécifique par son ID et les affiche, avec une gestion d'erreur si le médecin n'est pas trouvé.

3.2.4.4 Gestion des Congés

Affichage des congés (viewCongesMed) : Affiche les congés d'un médecin spécifique, en utilisant l'ID du médecin stocké en session.
Ajout de congé (addCongeMed) : Crée un nouveau congé pour le médecin connecté, avec gestion des erreurs en cas d'échec de création.
Modification de congé (editCongeMed, updateCongeMed) : Permet d'éditer et de mettre à jour les informations d'un congé spécifique.
Suppression de congé (deleteCongeMed) : Supprime un congé spécifié par son ID.
Gestion de l'Histoire Médicale
Affichage de l'historique médical (viewMedicalHistory) : Affiche tous les historiques médicaux liés à un médecin.

Ajout d'un historique médical (addMedicalHistory) : Permet de créer un nouvel historique médical pour un patient spécifique.

3.2.4.5 Gestion des Patients

- Liste des patients (listPatients) : Affiche tous les patients d'un médecin.
- Ajout de patient (addPatient) : Permet de créer un nouveau patient avec les détails fournis par formulaire.

3.2.4.6 Gestion des Rendez-Vous

Affichage des rendez-vous confirmés (viewConfirmedAppointments) : Affiche les rendez-vous confirmés pour un médecin, ce qui est crucial pour la gestion quotidienne des activités médicales.

3.2.4.7 Fonctions Diverses

Formulaires de création (createCongeForm, renderCreateMedicalHistoryForm, CreatePatientForm) : Ces méthodes génèrent des formulaires pour la création de nouveaux congés, historiques médicaux, ou patients, facilitant la saisie des données par les médecins.

3.2.5 SecretaireController

3.2.5.1 Rôle

Centralise la gestion des activités des secrétaires médicales, simplifiant ainsi l'interaction avec les données des patients, les rendez-vous, et les horaires de travail à travers des interfaces dédiées. Chaque méthode est conçue pour faciliter une interaction spécifique avec la base de données ou l'interface utilisateur, en maintenant une séparation claire entre la logique métier et la présentation, ce qui est au cœur de l'architecture MVC. Ce contrôleur assure également que les opérations sont menées de manière sécurisée et efficace, facilitant la gestion quotidienne des tâches administratives dans un environnement médical.

3.2.5.2 Initialisation (__construct)

Initialise les modèles et la vue : Crée des instances pour SecretaireModel, PatientModel, RendezVousModel, HorairesTravailModel, et SecretaireView. Cette structure permet de manipuler les données et d'interagir avec les vues associées.

3.2.5.3 Gestion des Secrétaires

- Affichage et création de secrétaires : Des méthodes comme homeSecretaire, listSecretaires, et createSecretaire facilitent la visualisation et la gestion des secrétaires dans la base de données.

- Modification et suppression : Les méthodes montrerFormulaireModification, mettreAJourSecretaire, et supprimerSecretaire permettent de modifier et de supprimer des entrées de secrétaires.

3.2.5.4 Gestion des Patients

- Liste et création de patients : listePatients et CreePatientForm affichent les patients existants et permettent la création de nouveaux patients.
- Détails et mise à jour de patients : showPatientDetails et updatePatient permettent de visualiser et de mettre à jour les informations des patients.

3.2.5.5 Gestion des Rendez-Vous

- Affichage et création de rendez-vous : montrerFormulaireRendezVous et ajouterRendezVous gèrent la planification des rendez-vous.
- Visualisation des rendez-vous par statut : viewAllRdv, viewConfirmedRdv, viewPendingRdv, et viewCancelledRdv affichent les rendez-vous selon différents statuts pour une gestion facile.
- Modification du statut des rendez-vous : changerStatutRendezVous permet de changer le statut des rendez-vous planifiés.

3.2.5.6 Gestion des Horaires de Travail

- Visualisation et modification des horaires : secretaireHoraires, modifHoraires, et changeHoraires facilitent la visualisation et la modification des horaires de travail des secrétaires ou des médecins.
- Suppression des horaires : supprimeHoraires permet de supprimer des horaires de travail spécifiques.

3.2.6 UserController

3.2.6.1 Rôle

Centralise la gestion des utilisateurs, offrant une interface claire pour l'authentification, l'inscription, et la gestion de session. Il facilite l'interaction sécurisée avec les données des utilisateurs et assure une expérience utilisateur cohérente et sécurisée à travers l'application. Chaque méthode est conçue pour faciliter une interaction spécifique avec la base de données ou l'interface utilisateur, en respectant les principes de l'architecture MVC. Ce contrôleur joue donc un rôle crucial dans la gestion des accès et des identités au sein de l'application.

3.2.6.2 Initialisation (__construct)

Initialise les modèles et les vues : Crée des instances de UserModel, UserView, MedecinView, et ContactView. Ce setup permet une gestion flexible des différents aspects des interactions utilisateur, incluant l'affichage et la gestion des données utilisateur.

3.2.6.3 Fonctionnalités de Base

- Contact (contact) : Affiche la page de contact en utilisant ContactView.
- Accueil (home) : Affiche la page d'accueil via UserView.

3.2.6.4 Authentification et Gestion de Session

- Formulaire de Connexion et Connexion (showLoginForm, login) : Affiche le formulaire de connexion et gère la vérification des identifiants de l'utilisateur. En cas de succès, il stocke les informations de l'utilisateur en session et redirige en fonction de son rôle.
- Formulaire d'Inscription et Inscription (showRegistrationForm, register) : Affiche le formulaire d'inscription et enregistre un nouvel utilisateur avec les informations fournies. Les mots de passe sont sécurisés via hashing.
- Déconnexion (logout) : Termine la session utilisateur et redirige vers la page d'accueil.

3.2.6.5 Gestion des Rôles et Redirections

Gestion dynamique des rôles : Après la connexion, le système vérifie le rôle de l'utilisateur (médecin, secrétaire, admin) pour charger des informations spécifiques et diriger l'utilisateur vers des tableaux de bord appropriés.

3.2.6.6 Fonctionnement Détail des Méthodes

- Connexion (login) : Vérifie l'existence de l'email et du mot de passe. Utilise authenticateUser pour valider les identifiants. En cas de réussite, il initialise des variables de session spécifiques et redirige l'utilisateur en fonction de son rôle.
- Inscription (register) : Vérifie que tous les champs nécessaires sont remplis, crypte le mot de passe et tente d'enregistrer l'utilisateur. En cas de succès, redirige vers la page d'accueil.
- Déconnexion (logout) : Efface les variables de session et détruit la session, puis redirige vers la page d'accueil.

3.3 Models

3.3.1 BaseModel

3.3.1.1 Rôle :

Fournit un cadre pour la gestion des connexions à la base de données et l'exécution de requêtes SQL dans votre application. En centralisant la connexion à la base de données et les fonctionnalités de base pour l'exécution des requêtes, il facilite la réutilisation du code et la maintenance. Ce modèle assure également que toutes les interactions avec la base de données sont gérées de manière sécurisée, prévenant les injections SQL et gérant les erreurs de connexion ou de requête de manière efficace.

3.3.1.2 Initialisation (__construct)

Connexion à la base de données : Établit une connexion au serveur de base de données MySQL en utilisant les paramètres spécifiés (hôte, utilisateur, mot de passe, nom de la base). Gère également les erreurs de connexion pour éviter des problèmes d'exécution ultérieurs sans gestion appropriée des erreurs.

3.3.1.3 Fonctionnalité de Requête

Exécution de requêtes avec jointures et conditions (lireAvecJointure) :

- Construction de la requête : Commence par une requête de base pour sélectionner toutes les colonnes d'une table spécifique. Des jointures sont ajoutées si elles sont fournies, permettant la combinaison de plusieurs tables dans une seule requête.
- Gestion des conditions : Ajoute des conditions à la requête pour filtrer les résultats, si des conditions sont spécifiées.
- Préparation et exécution de la requête : La requête est préparée pour éviter les injections SQL et exécutée. Les paramètres sont liés à la requête si nécessaires, ce qui est crucial pour la sécurité des données.
- Récupération des résultats : Les résultats sont obtenus et retournés pour être utilisés par les modèles dérivés.

3.3.2 CongesMedecinsModel

3.3.2.1 Rôle

Offre une interface pour la gestion des congés des médecins, en utilisant des méthodes de base de données sécurisées et efficaces pour manipuler les données. En encapsulant les opérations de base de données dans des méthodes bien définies, il facilite la maintenance et l'évolutivité de l'application, tout en garantissant que les interactions avec la base de données sont sécurisées et optimisées.

3.3.2.2 Initialisation (__construct)

Hérite de BaseModel : En étendant BaseModel, ce modèle utilise la connexion à la base de données établie dans BaseModel et bénéficie de toutes ses méthodes et propriétés utilitaires.

3.3.2.3 Gestion des Congés des Médecins

- Création de congé (creer) : Insère un nouveau congé dans la base de données avec des détails tels que l'ID du médecin, les dates de début et de fin, et la raison du congé. Les données sont nettoyées et échappées pour prévenir les injections SQL.
- Suppression de congé (supprimer) : Supprime un congé spécifié par son ID. Cela implique une opération directe de suppression dans la base de données après la préparation et la liaison des paramètres pour assurer la sécurité.
- Lecture de tous les congés (lire) : Récupère tous les enregistrements de congés depuis la base de données, utilisant une requête simple sans paramètres.
- Lecture d'un congé par ID (lireUn) : Récupère les détails d'un congé spécifique en utilisant son ID. Cela nécessite la préparation de la requête pour éviter les injections SQL et pour assurer que le bon enregistrement est retourné.
- Lecture des congés par ID de médecin (getByMedecinId) : Récupère tous les congés associés à un médecin spécifique, permettant de visualiser facilement tous les congés d'un médecin particulier.
- Mise à jour d'un congé (mettreAJour) : Met à jour les détails d'un congé existant. Comme pour l'insertion, les données sont nettoyées et liées à la requête préparée pour empêcher les erreurs et les injections SQL.

Sécurité et Nettoyage des Données

- Nettoyage des données d'entrée : Chaque méthode qui interagit avec la base de données utilise htmlspecialchars et strip_tags pour nettoyer les données d'entrée et éviter les injections XSS et SQL. Cela est crucial pour la sécurité de l'application.
- Préparation et liaison des requêtes : L'utilisation de requêtes préparées avec bind_param pour lier dynamiquement les paramètres à la requête SQL est une pratique de sécurité essentielle, minimisant le risque d'injection SQL.

3.3.3 FacturesModel

3.3.3.1 Role

Offre une interface et sécurisée pour la gestion des factures dans votre application. En intégrant la sécurité et la gestion des erreurs directement dans les méthodes de manipulation de données, il garantit que les opérations sur les factures sont à la fois sûres et efficaces, permettant une gestion précise des transactions financières dans le contexte médical. Ce modèle facilite également l'audit et le suivi des paiements pour les services médicaux rendus, jouant un rôle crucial dans l'administration financière de l'établissement médical.

3.3.3.2 Initialisation (__construct)

Héritage de BaseModel : Utilise la connexion à la base de données et les fonctionnalités de gestion de la base de données de BaseModel, ce qui simplifie la gestion des requêtes et assure la cohérence des opérations de base de données à travers l'application.

3.3.3.3 Gestion des Factures

- Lecture des factures (lire) : Récupère toutes les factures enregistrées dans la base de données, en utilisant une requête simple qui extrait tous les éléments de la table des factures.
- Lecture d'une facture par ID (lireUn) : Récupère les détails spécifiques d'une facture en utilisant son ID. Cela inclut la préparation de la requête pour éviter les injections SQL, et l'utilisation de paramètres liés pour la requête.
- Création d'une nouvelle facture (creer) : Ajoute une nouvelle entrée dans la table des factures avec des détails tels que l'ID du rendez-vous, le montant, le statut du paiement, et la date de la facture. Les données sont nettoyées et échappées pour assurer la sécurité.
- Mise à jour d'une facture (mettreAJour) : Met à jour les détails d'une facture existante. Les champs pouvant être mis à jour comprennent le montant, le statut de paiement, et potentiellement la date de la facture, selon que cette dernière est fournie ou non.
- Suppression d'une facture (supprimer) : Efface une facture de la base de données en utilisant son ID. Cette méthode prépare la requête, lie l'ID et exécute l'opération, gérant également les erreurs potentielles lors de chaque étape.

3.3.3.4 Sécurité et Gestion des Erreurs

- Préparation et exécution des requêtes : Toutes les requêtes sont préparées pour éviter les injections SQL. Les données sont nettoyées via htmlspecialchars et strip_tags avant d'être liées à la requête, ce qui prévient les attaques XSS et les manipulations malveillantes des données.
- Gestion des erreurs : Les erreurs de préparation, de liaison des paramètres, et d'exécution des requêtes sont capturées et traitées immédiatement, ce qui empêche l'exécution de requêtes erronées et informe l'utilisateur ou le système d'éventuelles défaillances.

3.3.4 HistoriqueMedicalModel

3.3.4.1 Role

Offre une interface pour la gestion des dossiers médicaux des patients, assurant une manipulation sécurisée et efficace des données sensibles. Il simplifie les interactions complexes avec la base de données et assure que les données médicales sont gérées de manière conforme aux normes de sécurité et d'efficacité. Ce modèle est essentiel pour maintenir une trace précise et accessible de l'historique médical des patients, facilitant ainsi les diagnostics et les suivis médicaux.

3.3.4.2 Initialisation (__construct)

Héritage de BaseModel : Ce modèle étend BaseModel, utilisant ainsi la connexion à la base de données établie pour exécuter les requêtes nécessaires.

3.3.4.3 Gestion des Données d'Historique Médical

- Lecture de toutes les entrées (lire) : Récupère toutes les entrées d'historique médical enregistrées dans la base de données.
- Lecture par ID (lireUn) : Obtient les détails spécifiques d'une entrée d'historique médical en utilisant son ID.
- Création d'une nouvelle entrée (creer) : Ajoute un nouvel historique médical avec les informations détaillées du patient, du médecin, de la visite, du diagnostic, du traitement, et des commentaires associés.
- Mise à jour d'une entrée (mettreAJour) : Met à jour les informations d'une entrée d'historique médical existante en modifiant les détails pertinents comme le médecin, le diagnostic, et le traitement.
- Suppression d'une entrée (supprimer) : Efface une entrée d'historique médical spécifiée par son ID.

3.3.4.4 Méthodes Supplémentaires

- Recherche par médecin (getByMedecinId) : Récupère toutes les entrées d'historique médical associées à un médecin particulier.
- Recherche par patient (rechercherParPatient) : Obtient toutes les entrées d'historique médical pour un patient donné, facilitant l'accès rapide aux dossiers médicaux du patient.

3.3.4.5 Sécurité et Gestion des Erreurs

- Préparation et exécution sécurisées des requêtes : Toutes les requêtes sont préparées pour éviter les injections SQL. Les paramètres sont nettoyés et échappés avant d'être liés à la requête.
- Gestion des erreurs : Les erreurs de préparation de la requête, de liaison des paramètres, et d'exécution sont systématiquement gérées pour prévenir l'exécution de requêtes erronées et fournir des diagnostics clairs en cas de problème.

3.3.5 HorairesTravailModel

3.3.5.1 Role

Offre une interface pour la gestion des horaires des médecins dans votre application. En assurant une manipulation sécurisée et efficace des données, il facilite la gestion des plannings de travail et de congé, rendant les opérations quotidiennes plus fluides et moins susceptibles d'erreur. Ce modèle est essentiel pour la planification et la coordination au sein

des établissements médicaux, permettant un suivi précis et accessible des horaires de travail des professionnels de santé.

3.3.5.2 Initialisation (__construct)

Héritage de BaseModel : Utilise la connexion à la base de données établie par BaseModel, permettant une intégration et une gestion efficace des requêtes SQL.

3.3.5.3 Gestion des Horaires de Travail

- Lecture de tous les horaires (lire) : Récupère tous les horaires de travail enregistrés dans la base de données, permettant un accès facile à l'ensemble des horaires.
- Lecture par ID (lireUn) : Obtient les détails spécifiques d'un horaire de travail en utilisant son ID, utile pour les consultations ou les mises à jour spécifiques.
- Création d'un nouvel horaire (creer) : Ajoute un nouvel horaire de travail avec des informations détaillées comme le jour de la semaine, les heures de début et de fin, le type (travail ou congé), et des commentaires éventuels.
- Mise à jour d'un horaire (mettreAJour) : Met à jour les informations d'un horaire de travail existant, modifiant des éléments comme les heures et les commentaires.
- Suppression d'un horaire (supprimer) : Efface un horaire de travail de la base de données en utilisant son ID, important pour la gestion des changements dans les plannings des médecins.

3.3.5.4 Fonctionnalités Additionnelles

Recherche par médecin (rechercherParMedecin) : Récupère tous les horaires associés à un médecin spécifique, ce qui est crucial pour visualiser ou ajuster les plannings individuels des médecins.

3.3.5.5 Sécurité et Gestion des Erreurs

- Préparation et exécution des requêtes sécurisées : Les requêtes sont systématiquement préparées pour éviter les injections SQL. Les paramètres tels que les ID des médecins ou les détails des horaires sont nettoyés et échappés avant d'être liés aux requêtes.
- Gestion des erreurs : Les erreurs potentielles dans la préparation des requêtes, la liaison des paramètres, ou l'exécution sont gérées immédiatement, empêchant les opérations erronées et fournissant des diagnostics précis en cas de problème.

3.3.6 MedecinModel

3.3.6.1 Role

Offre une interface pour la gestion des informations des médecins dans votre application, assurant une manipulation sécurisée et efficace des données médicales personnelles. En simplifiant les interactions complexes avec la base de données et en assurant que les

données des médecins sont gérées de manière conforme aux normes de sécurité et d'efficacité, ce modèle joue un rôle crucial dans la gestion des ressources humaines médicales de l'établissement.

3.3.6.2 Initialisation (__construct)

Héritage de BaseModel : Utilise la connexion à la base de données établie par BaseModel, permettant une intégration et une gestion efficace des requêtes SQL.

3.3.6.3 Gestion des Informations des Médecins

- Lecture de tous les médecins (lire) : Récupère tous les médecins enregistrés dans la base de données. Utilise une requête simple pour extraire toutes les entrées de la table des médecins.
- Lecture par ID (lireUn) : Obtient les détails spécifiques d'un médecin en utilisant son ID. Cette méthode est utile pour des consultations ou des mises à jour spécifiques.
- Création d'un nouveau médecin (creer) : Ajoute un nouveau médecin avec des informations détaillées telles que l'ID utilisateur, le nom, le prénom, la spécialité, et le téléphone.
- Mise à jour d'un médecin (mettreAJour) : Met à jour les informations d'un médecin existant, modifiant des éléments comme le nom, le prénom, la spécialité, et le téléphone.
- Suppression d'un médecin (supprimer) : Efface un médecin de la base de données en utilisant son ID.

3.3.6.4 Sécurité et Gestion des Erreurs

- Préparation et exécution sécurisées des requêtes : Les requêtes sont systématiquement préparées pour éviter les injections SQL. Les paramètres tels que les ID des médecins ou les détails personnels sont nettoyés et échappés avant d'être liés aux requêtes.
- Gestion des erreurs : Les erreurs potentielles dans la préparation des requêtes, la liaison des paramètres, ou l'exécution sont gérées immédiatement, empêchant les opérations erronées et fournissant des diagnostics précis en cas de problème.

3.3.6.5 Fonctionnalités Additionnelles

Recherche par médecin : Bien que non explicitement détaillée, des méthodes supplémentaires pourraient inclure la recherche de médecins par spécialité ou par nom, augmentant ainsi la flexibilité et l'utilité du modèle dans l'application.

3.3.7 MedecinServicesModel

3.3.7.1 Role

Offre une interface pour la gestion des associations entre médecins et services dans votre application, assurant une manipulation sécurisée et efficace des données de service. En

simplifiant les interactions complexes avec la base de données et en assurant que les données des associations sont gérées de manière conforme aux normes de sécurité et d'efficacité, ce modèle joue un rôle crucial dans la gestion des ressources humaines médicales et des services offerts de l'établissement.

3.3.7.2 Initialisation (__construct)

Héritage de BaseModel : Utilise la connexion à la base de données établie par BaseModel, permettant une intégration et une gestion efficace des requêtes SQL.

3.3.7.3 Gestion des Associations Médecin-Service

Lecture de toutes les associations (lire) : Récupère toutes les associations médecin-service enregistrées dans la base de données, permettant un accès facile à l'ensemble des associations.

- Lecture par IDs médecin et service (lireUn) : Obtient les détails spécifiques d'une association médecin-service en utilisant l'ID du médecin et l'ID du service. Cette méthode est utile pour des consultations ou des mises à jour spécifiques.
- Création d'une nouvelle association (creer) : Ajoute une nouvelle association médecin-service avec l'ID du médecin et l'ID du service.
- Mise à jour d'une association (mettreAJour) : Met à jour une association médecin-service existante, potentiellement en modifiant l'activation de cette association (champ active).
- Suppression d'une association (supprimer) : Efface une association médecin-service de la base de données en utilisant son ID.

3.3.7.4 Sécurité et Gestion des Erreurs

- Préparation et exécution sécurisées des requêtes : Les requêtes sont systématiquement préparées pour éviter les injections SQL. Les paramètres tels que les IDs des médecins, des services, et les détails de l'association sont nettoyés et échappés avant d'être liés aux requêtes.
- Gestion des erreurs : Les erreurs potentielles dans la préparation des requêtes, la liaison des paramètres, ou l'exécution sont gérées immédiatement, empêchant les opérations erronées et fournissant des diagnostics précis en cas de problème.

Initialisation (__construct)

Héritage de BaseModel : Ce modèle étend BaseModel, utilisant ainsi la connexion à la base de données établie pour exécuter des requêtes SQL nécessaires à la gestion des notes.

3.3.7.5 Gestion des Notes

Lecture de toutes les notes (lire) : Récupère toutes les notes stockées dans la base de données, permettant un accès et une révision faciles de toutes les entrées.

- Lecture par ID (lireUn) : Obtient les détails spécifiques d'une note en utilisant son ID. Cette fonction est utile pour la consultation détaillée ou la mise à jour spécifique d'une note.
- Création d'une nouvelle note (creer) : Ajoute une nouvelle note avec les informations fournies telles que l'ID du rendez-vous, le contenu de la note, et la date de création (automatiquement générée à la création).
- Mise à jour d'une note (mettreAJour) : Met à jour le contenu d'une note existante.
- Suppression d'une note (supprimer) : Efface une note de la base de données en utilisant son ID, important pour la gestion des données obsolètes ou incorrectes.

3.3.7.6 Fonctionnalités Additionnelles

- Recherche par rendez-vous (rechercherParRendezVous) : Récupère toutes les notes associées à un rendez-vous spécifique, facilitant l'accès aux informations détaillées pour des suivis médicaux.
- Recherche par médecin (getByMedecinId) : Bien que cette méthode soit mentionnée, elle nécessiterait que chaque note soit associée à un médecin spécifique pour fonctionner correctement, ce qui n'est pas indiqué dans la structure actuelle de la base de données.

3.3.7.7 Sécurité et Gestion des Erreurs

- Préparation et exécution sécurisées des requêtes : Les requêtes sont préparées pour prévenir les injections SQL. Les données sont nettoyées et échappées avant d'être utilisées dans les requêtes.
- Gestion des erreurs : Les erreurs lors de la préparation, de la liaison des paramètres, ou de l'exécution des requêtes sont traitées immédiatement, empêchant les opérations erronées et offrant un diagnostic en cas de problème.

3.3.8 NotesModel

3.3.8.1 Role

Offre une interface pour la gestion des notes médicales dans votre application, assurant une manipulation sécurisée et efficace des notes liées aux rendez-vous. Ce modèle joue un rôle crucial dans le suivi des détails des consultations, améliorant la qualité des soins médicaux et la communication entre les professionnels de santé. En facilitant l'accès et la mise à jour des notes, ce modèle aide à maintenir une documentation précise et accessible des interactions médicales.

3.3.8.2 Initialisation (__construct)

Héritage de BaseModel : Ce modèle étend BaseModel, utilisant ainsi la connexion à la base de données établie pour exécuter des requêtes SQL nécessaires à la gestion des notes.

3.3.8.3 Gestion des Notes

- Lecture de toutes les notes (lire) : Récupère toutes les notes stockées dans la base de données, permettant un accès et une révision faciles de toutes les entrées.
- Lecture par ID (lireUn) : Obtient les détails spécifiques d'une note en utilisant son ID. Cette fonction est utile pour la consultation détaillée ou la mise à jour spécifique d'une note.
- Création d'une nouvelle note (creer) : Ajoute une nouvelle note avec les informations fournies telles que l'ID du rendez-vous, le contenu de la note, et la date de création (automatiquement générée à la création).
- Mise à jour d'une note (mettreAJour) : Met à jour le contenu d'une note existante.
- Suppression d'une note (supprimer) : Efface une note de la base de données en utilisant son ID, important pour la gestion des données obsolètes ou incorrectes.

3.3.8.4 Fonctionnalités Additionnelles

- Recherche par rendez-vous (rechercherParRendezVous) : Récupère toutes les notes associées à un rendez-vous spécifique, facilitant l'accès aux informations détaillées pour des suivis médicaux.
- Recherche par médecin (getByMedecinId) : Bien que cette méthode soit mentionnée, elle nécessiterait que chaque note soit associée à un médecin spécifique pour fonctionner correctement, ce qui n'est pas indiqué dans la structure actuelle de la base de données.

3.3.8.5 Sécurité et Gestion des Erreurs

- Préparation et exécution sécurisées des requêtes : Les requêtes sont préparées pour prévenir les injections SQL. Les données sont nettoyées et échappées avant d'être utilisées dans les requêtes.
- Gestion des erreurs : Les erreurs lors de la préparation, de la liaison des paramètres, ou de l'exécution des requêtes sont traitées immédiatement, empêchant les opérations erronées et offrant un diagnostic en cas de problème.

3.3.9 PatientModel

3.3.9.1 Role

Offre une interface pour la gestion des informations des patients dans votre application, assurant une manipulation sécurisée et efficace des données personnelles et médicales. En simplifiant les interactions complexes avec la base de données et en assurant que les données des patients sont gérées de manière conforme aux normes de sécurité et d'efficacité, ce modèle joue un rôle crucial dans la gestion des dossiers médicaux. Il facilite l'accès et la mise à jour des informations des patients, aidant à maintenir une documentation précise et accessible, essentielle pour la prestation de soins de santé de qualité.

3.3.9.2 Initialisation (__construct)

Héritage de BaseModel : Ce modèle étend BaseModel, utilisant ainsi la connexion à la base de données établie pour exécuter les requêtes SQL nécessaires à la gestion des données des patients.

3.3.9.3 Gestion des Informations des Patients

- Lecture de tous les patients (lire) : Récupère tous les patients enregistrés dans la base de données, utilisant une requête simple pour extraire toutes les entrées de la table des patients.
- Lecture par ID (lireUn) : Obtient les détails spécifiques d'un patient en utilisant son ID. Cette fonction est utile pour la consultation détaillée ou la mise à jour spécifique d'un patient.
- Création d'un nouveau patient (creer) : Ajoute un nouveau patient avec des informations complètes telles que nom, prénom, date de naissance, email, téléphone, adresse et historique médical.
- Mise à jour d'un patient (mettreAJour) : Met à jour les informations d'un patient existant. Cela peut inclure des changements dans n'importe quel aspect des données personnelles du patient.
- Suppression d'un patient (supprimer) : Efface un patient de la base de données en utilisant son ID, important pour la gestion des données et la protection de la vie privée.

3.3.9.4 Sécurité et Gestion des Erreurs

- Préparation et exécution sécurisées des requêtes : Toutes les requêtes sont systématiquement préparées pour éviter les injections SQL. Les données des patients sont nettoyées et échappées avant d'être utilisées dans les requêtes.
- Gestion des erreurs : Les erreurs potentielles dans la préparation des requêtes, la liaison des paramètres, ou l'exécution sont traitées immédiatement, empêchant les opérations erronées et offrant un diagnostic en cas de problème.

3.3.10 RendezVousModel

3.3.10.1 Role

Offre une interface pour la gestion des rendez-vous médicaux dans votre application, assurant une manipulation sécurisée et efficace des données de rendez-vous. Ce modèle joue un rôle crucial dans la planification et la gestion des consultations médicales, facilitant l'accès, la mise à jour, et le suivi des rendez-vous, et améliorant ainsi l'efficacité opérationnelle de l'établissement médical.

3.3.10.2 Initialisation (__construct)

Héritage de BaseModel : Ce modèle étend BaseModel, utilisant ainsi la connexion à la base de données établie pour exécuter les requêtes SQL nécessaires à la gestion des rendez-vous.

3.3.10.3 Gestion des Rendez-vous

- Lecture de tous les rendez-vous (lire) : Récupère tous les rendez-vous enregistrés dans la base de données, utilisant une requête simple pour extraire toutes les entrées de la table des rendez-vous.
- Lecture par ID (lireUn) : Obtient les détails spécifiques d'un rendez-vous en utilisant son ID. Cette fonction est utile pour la consultation détaillée ou la mise à jour spécifique d'un rendez-vous.
- Création d'un nouveau rendez-vous (creer) : Ajoute un nouveau rendez-vous avec des informations complètes telles que l'ID du médecin, l'ID du patient, la date et l'heure, la raison, le statut, et des commentaires éventuels.
- Mise à jour d'un rendez-vous (mettreAJour) : Met à jour les informations d'un rendez-vous existant, incluant des modifications sur la date, l'heure, la raison, le statut, et les commentaires.
- Suppression d'un rendez-vous (supprimer) : Efface un rendez-vous de la base de données en utilisant son ID, ce qui est crucial pour la gestion des annulations et des modifications de planification.

3.3.10.4 Fonctionnalités Additionnelles

- Recherche par ID (rechercherParID) : Récupère un rendez-vous spécifique par son ID, permettant une récupération rapide pour des consultations ou des vérifications.
- Obtenir les rendez-vous confirmés par ID de médecin (getConfirmedAppointmentsByMedecinId) : Récupère tous les rendez-vous confirmés pour un médecin spécifique, aidant à visualiser les engagements futurs du médecin.
- Lire les rendez-vous confirmés avec détails du patient (lireConfirmesParMedecin): Cette méthode avancée joint la table des patients pour enrichir les informations des rendez-vous confirmés avec les noms et prénoms des patients, offrant une vue complète pour la gestion des rendez-vous.

3.3.10.5 Sécurité et Gestion des Erreurs

- Préparation et exécution sécurisées des requêtes : Les requêtes sont préparées pour prévenir les injections SQL. Les données sont nettoyées et échappées avant d'être utilisées dans les requêtes.
- Gestion des erreurs : Les erreurs lors de la préparation, de la liaison des paramètres, ou de l'exécution des requêtes sont traitées immédiatement, empêchant les opérations erronées et offrant un diagnostic en cas de problème.

3.3.11 SecretaireModel

3.3.11.1 Role

Offre une interface pour la gestion des informations des secrétaires médicaux dans votre application, assurant une manipulation sécurisée et efficace des données de secrétariat. En facilitant l'accès et la mise à jour des informations des secrétaires, ainsi que la gestion des rendez-vous et des dossiers patients, ce modèle joue un rôle crucial dans la gestion administrative des établissements de soins de santé.

3.3.11.2 Initialisation (__construct)

Héritage de BaseModel : Ce modèle étend BaseModel, utilisant ainsi la connexion à la base de données établie pour exécuter les requêtes SQL nécessaires à la gestion des données des secrétaires.

3.3.11.3 Gestion des Informations des Secrétaires

- Lecture de tous les secrétaires (lire) : Récupère tous les secrétaires enregistrés dans la base de données, utilisant une requête simple pour extraire toutes les entrées de la table des secrétaires.
- Lecture par ID (lireUn) : Obtient les détails spécifiques d'un secrétaire en utilisant son ID. Cette fonction est utile pour la consultation détaillée ou la mise à jour spécifique d'un secrétaire.
- Création d'un nouveau secrétaire (creer) : Ajoute un nouveau secrétaire avec des informations telles que l'ID utilisateur, le nom, le prénom, et le téléphone.
- Mise à jour d'un secrétaire (mettreAJour) : Met à jour les informations d'un secrétaire existant, incluant des modifications sur le nom, le prénom, et le téléphone.
- Suppression d'un secrétaire (supprimer) : Efface un secrétaire de la base de données en utilisant son ID.

3.3.11.4 Gestion des Rendez-vous et des Patients

- Lister les rendez-vous (listerRendezVous) : Récupère tous les rendez-vous programmés, fournissant une vue d'ensemble pour les planifications.
- Créer un rendez-vous (creerRendezVous) : Ajoute un nouveau rendez-vous avec les détails nécessaires comme les IDs de médecin et de patient, la date et heure, la raison du rendez-vous, le statut, et les commentaires éventuels.
- Rechercher un rendez-vous par ID (rechercherRdvParID) : Obtient les détails spécifiques d'un rendez-vous pour une gestion ou une modification précise.
- Gérer les statuts des rendez-vous : Méthodes comme getConfirmedRdv, getPendingRdv, et getCancelledRdv permettent de filtrer les rendez-vous selon leur statut, aidant à la gestion efficace des engagements.

3.3.11.5 Fonctionnalités Additionnelles

- Gestion dynamique des rendez-vous : Permet la modification des statuts des rendez-vous, crucial pour la gestion des changements de dernière minute dans les plannings.
- Interaction avec les données des patients (editPatient) : Permet de mettre à jour les informations des patients, facilitant la gestion des dossiers médicaux par les secrétaires.

3.3.11.6 Sécurité et Gestion des Erreurs

- Préparation et exécution sécurisées des requêtes : Les requêtes sont systématiquement préparées pour éviter les injections SQL. Les données sont nettoyées et échappées avant d'être utilisées dans les requêtes.
- Gestion des erreurs : Les erreurs lors de la préparation, de la liaison des paramètres, ou de l'exécution des requêtes sont traitées immédiatement, empêchant les opérations erronées et offrant un diagnostic en cas de problème.

3.3.12 ServiceModel

3.3.12.1 Role

Offre une interface pour la gestion des services médicaux dans votre application, assurant une manipulation sécurisée et efficace des informations sur les services. En simplifiant les interactions complexes avec la base de données et en assurant que les données des services sont gérées de manière conforme aux normes de sécurité et d'efficacité, ce modèle joue un rôle crucial dans la gestion des offres de services de l'établissement. Il aide à maintenir une documentation précise et accessible des services disponibles, ce qui est essentiel pour la prestation de services médicaux efficaces.

3.3.12.2 Initialisation (__construct)

Héritage de BaseModel : Le modèle étend BaseModel, utilisant ainsi la connexion à la base de données établie pour exécuter les requêtes SQL nécessaires à la gestion des services.

3.3.12.3 Gestion des Services

- Lecture de tous les services (lire) : Récupère tous les services enregistrés dans la base de données, permettant un accès facile à l'ensemble des services disponibles.
- Lecture par ID (lireUn) : Obtient les détails spécifiques d'un service en utilisant son ID, utile pour des consultations ou des mises à jour spécifiques.
- Création d'un nouveau service (creer) : Ajoute un nouveau service avec le nom et la description fournis, facilitant l'extension des offres de services de l'établissement.
- Mise à jour d'un service (mettreAJour) : Met à jour les informations d'un service existant, telles que le nom et la description, permettant de maintenir à jour les détails des services offerts.

- Suppression d'un service (supprimer) : Efface un service de la base de données en utilisant son ID, important pour la gestion des services obsolètes ou inutiles.

3.3.12.4 Fonctionnalités Additionnelles

Recherche par ID (rechercherParID) : Permet de récupérer rapidement un service spécifique par son ID pour vérification ou traitement ultérieur.

3.3.12.5 Sécurité et Gestion des Erreurs

- Préparation et exécution sécurisées des requêtes : Les requêtes sont préparées pour éviter les injections SQL. Les paramètres comme le nom, la description, et l'ID sont nettoyés et échappés avant d'être utilisés dans les requêtes.
- Gestion des erreurs : Les erreurs lors de la préparation, de la liaison des paramètres, ou de l'exécution des requêtes sont traitées immédiatement, offrant un diagnostic en cas de problème et empêchant les opérations erronées.

3.3.13 UserModel

3.3.13.1 Role

Offre une interface pour la gestion des utilisateurs dans votre application, assurant une manipulation sécurisée et efficace des informations des utilisateurs. Ce modèle joue un rôle crucial en facilitant les interactions complexes avec la base de données, en garantissant que les données des utilisateurs sont gérées conformément aux normes de sécurité et d'efficacité les plus strictes.

3.3.13.2 Initialisation (__construct)

Héritage de BaseModel : Le modèle étend BaseModel, utilisant ainsi la connexion à la base de données établie pour exécuter les requêtes SQL nécessaires à la gestion des utilisateurs.

3.3.13.3 Gestion des Utilisateurs

- Lecture de tous les utilisateurs (getUsers) : Récupère tous les utilisateurs enregistrés dans la base de données. Cela permet une vue d'ensemble pour des tâches administratives ou des vérifications de conformité.
- Lecture par ID (getUserById) : Récupère les informations détaillées d'un utilisateur spécifique en utilisant son ID, crucial pour des opérations telles que la modification des profils ou l'assistance utilisateur.
- Création d'un utilisateur (registerUser) : Ajoute un nouvel utilisateur avec les informations de base nécessaires telles que le nom, le mot de passe (haché pour la sécurité), l'email, et un rôle prédéfini.

- Authentification d'un utilisateur (authenticateUser) : Vérifie les informations de connexion d'un utilisateur, utilisant le hachage et la vérification des mots de passe pour assurer une connexion sécurisée.
- Mise à jour d'un utilisateur (updateById) : Permet de mettre à jour les informations d'un utilisateur, y compris son nom, son email, et son rôle selon l'ID.
- Suppression d'un utilisateur (deleteById) : Efface un utilisateur de la base de données en utilisant son ID, une opération nécessaire pour la gestion des droits d'accès ou la conformité aux politiques de données.

3.3.13.4 Intégration avec d'autres rôles

- Récupération des informations du médecin par ID utilisateur (getMedecinByUserId) : Permet d'obtenir des détails spécifiques liés à un médecin associé à un utilisateur, facilitant ainsi l'accès aux informations professionnelles pour la gestion des horaires ou des responsabilités cliniques.
- Récupération des informations de la secrétaire par ID utilisateur (getSecretaireByUserId) : Similar to the above, it retrieves details about a secretary associated with a specific user, essential for managing administrative tasks.

3.3.13.5 Sécurité et Gestion des Erreurs

- Préparation et exécution sécurisées des requêtes : Chaque requête est préparée pour éviter les injections SQL, et les paramètres sont systématiquement nettoyés et échappés pour préserver la sécurité des données.
- Gestion des erreurs : Les erreurs de préparation, de liaison des paramètres, ou d'exécution des requêtes sont immédiatement gérées, offrant des diagnostics précis en cas de problèmes pour permettre des corrections rapides.

3.4 Views

3.4.1 admin_view

3.4.1.1 Rôle

Gère la présentation et l'interaction avec les vues administratives. Elle contient plusieurs méthodes pour afficher les différentes sections de l'interface d'administration, y compris les utilisateurs, les médecins, les patients, et d'autres éléments de gestion.

3.4.1.2 Gestion de l'interface utilisateur

- renderHeader et renderFooter : Ces méthodes incluent les fichiers d'en-tête et de pied de page standard pour les pages administratives.
- renderAdminDashboard : Affiche le tableau de bord principal de l'administrateur avec des liens vers les gestionnaires de contenu spécifiques comme les utilisateurs, les médecins, et les patients.

3.4.1.3 Affichage dynamique de tableaux

- renderTable : Affiche un tableau HTML basé sur les données fournies, utilisant un callback pour générer les lignes du tableau. Cela permet une flexibilité dans la présentation des données spécifiques à chaque type d'information (utilisateurs, médecins, etc.).
- renderDataTableScript : Injecte le JavaScript nécessaire pour transformer les tableaux HTML standards en DataTables interactives avec pagination, recherche et tri.

3.4.1.4 Tableaux de bord spécialisés

adminDashboardUser, adminDashboardMedecin, adminDashboardPatient : Ces méthodes génèrent des vues spécifiques pour les différents types d'acteurs dans le système, chacune avec sa propre logique de présentation et ses boutons d'action pour la modification et la suppression.

3.4.1.5 Fonctionnalités d'édition

renderEditUserForm, renderEditMedecinForm, renderEditPatientForm : Ces fonctions affichent des formulaires pré-remplis pour la modification des détails des utilisateurs, médecins, et patients. Ils intègrent la validation côté client pour assurer que les champs requis sont remplis avant la soumission.

3.4.1.6 Scripts de validation JavaScript

Ces scripts assurent que les formulaires sont correctement remplis avant leur envoi, augmentant la robustesse de l'interface utilisateur et prévenant les erreurs côté serveur.

3.4.1.7 Gestion de contenu additionnel

La classe gère également d'autres types de données, comme les congés, les factures, les horaires de travail, les associations médecin-service, les notes, les rendez-vous et les services, en fournissant des méthodes pour afficher et modifier ces informations.

3.4.1.8 Méthodes utilitaires

isSelected : Une méthode utilitaire pour marquer un élément sélectionné dans les listes déroulantes, améliorant l'expérience utilisateur lors de la modification des données existantes.

3.4.2 base_view

3.4.2.1 Rôle

Fournit des méthodes de base pour charger des templates, ainsi que pour afficher les en-têtes, les conteneurs, et les pieds de page communs à toutes les pages.

3.4.2.2 Chargement de templates

loadTemplate($templateName, $data = []) : Cette méthode charge un fichier template spécifique situé dans le répertoire "templates". Elle utilise la fonction extract() pour transformer les éléments du tableau $data en variables locales. Cela permet de passer facilement des données au template, où elles peuvent être utilisées directement comme variables.

3.4.2.3 Gestion des éléments de base de la page

- renderHeader() : Inclut le fichier d'en-tête général pour les pages (header.php). Cet en-tête peut inclure des éléments comme la barre de navigation, les liens CSS, et d'autres ressources partagées entre toutes les pages.
- renderContainer() : Charge un conteneur principal (container.php) qui peut être utilisé pour envelopper le contenu principal de la page. Cela peut être utile pour maintenir une cohérence structurelle et stylistique à travers l'application.
- renderFooter() : Inclut le pied de page (footer.php). Le pied de page peut contenir des informations comme des liens vers des informations légales, des informations de contact, ou des scripts JavaScript qui doivent être chargés à la fin de la page.

3.4.2.4 Utilisation

La classe BaseView est typiquement héritée par d'autres classes de vue plus spécifiques. Ces classes peuvent utiliser les méthodes de BaseView pour s'occuper des parties communes de la page tout en ajoutant leur propre logique et présentation spécifique pour différents segments de l'application. Par exemple, une classe UserView pourrait étendre BaseView et utiliser loadTemplate() pour charger des templates spécifiques à la gestion des utilisateurs.

3.4.2.4.1 Avantages

- Réutilisation du code : Centralise le code commun de manipulation des vues, réduisant la redondance à travers l'application.
- Organisation : Aide à maintenir une structure claire pour les fichiers de templates et les rend facilement accessibles grâce à une méthode centralisée.
- Flexibilité : Permet de passer des données dynamiques aux templates, facilitant la création de pages dynamiques.

3.4.3 category_view

3.4.3.1 Rôle

Conçue pour gérer l'affichage des données liées aux catégories dans une application web. Elle fournit des méthodes spécifiques pour afficher les catégories, une catégorie individuelle, et pour afficher un formulaire d'édition de catégorie.

3.4.3.2 Méthodes

- displayCategories($categories): Affiche une liste de catégories. Pour chaque catégorie, elle affiche le nom et la description dans un format de bloc.
- displaySingleCategory($category): Affiche les détails d'une seule catégorie, incluant son nom et sa description. Cette méthode pourrait être utilisée pour une page de détails où l'utilisateur clique sur une catégorie spécifique pour en voir plus.
- renderEditCategoryForm($category): Affiche un formulaire pour éditer une catégorie existante. Utilise une instance de FormView (non définie dans ce fragment de code mais probablement une autre classe gérant la construction de formulaires HTML) pour créer le formulaire. Le formulaire est ensuite inséré dans une structure de page HTML. Des validations JavaScript sont également ajoutées pour s'assurer que le champ du nom n'est pas laissé vide lors de la soumission.
- showCategories($categories): Une autre méthode pour afficher une liste de catégories, mais avec un lien cliquable sur chaque nom de catégorie qui redirige vers une page détaillée pour chaque catégorie. Cela peut être utile pour une vue d'ensemble où les utilisateurs peuvent cliquer sur une catégorie pour voir tous les articles ou produits associés à cette catégorie.

3.4.3.3 Utilisation

La classe est clairement structurée pour séparer les responsabilités de l'affichage des données de catégorie des autres aspects de l'application, en s'appuyant sur l'héritage de BaseView pour les fonctionnalités de base de la vue comme l'affichage des en-têtes et des pieds de page. Cela aide à garder le code organisé et réutilisable.

3.4.3.4 Avantages

- Modularité: En héritant de BaseView, CategoryView peut utiliser des méthodes communes tout en ajoutant des fonctionnalités spécifiques pour les catégories, ce qui rend le code plus modulaire et plus facile à gérer.
- Réutilisabilité: Les méthodes comme displayCategories et displaySingleCategory peuvent être réutilisées dans différents contextes de l'application, par exemple, dans des listes de pages ou des dashboards administratifs.
- Maintenabilité: La séparation claire des responsabilités facilite la maintenance du code et les éventuelles mises à jour ou modifications des fonctionnalités liées aux catégories.

3.4.4 contact_view

3.4.4.1 Rôle

Gére l'affichage de la page de contact.

3.4.4.2 Méthodes

- renderHeader() et renderFooter(): Ces méthodes incluent les fichiers header et footer respectivement. Ces fichiers contiendraient probablement le HTML commun en en-tête et en pied de page de toutes les pages du site.
- renderContact(): Cette méthode est le cœur de ContactView et est responsable de la mise en page de la page de contact. Elle procède comme suit :
- Appelle renderHeader() pour afficher l'en-tête de la page.
- Utilise FormContainer, une classe conçue pour gérer des conteneurs de formulaire ou des sections structurées sur la page. Elle est utilisée ici pour ajouter des éléments non spécifiquement liés à un formulaire mais à la structuration du contenu.
- Fil d'Ariane: Ajoute un chemin de navigation pour améliorer l'expérience utilisateur et permettre une navigation facile. Cela commence par un lien vers la page d'accueil suivi d'un item non cliquable pour "Contact".
- Titre et descriptions: Définit un titre principal pour la page et ajoute des informations de contact telles que l'adresse email, le téléphone, et l'adresse physique, améliorant ainsi la transparence et la facilité de contact pour l'utilisateur.
- Rend le contenu construit via form_container->render(), qui générerait le HTML final basé sur les éléments ajoutés.

3.4.5 form_container

3.4.5.1 Rôle

Structure de gestion de contenu pour les éléments d'interface dans l' application, permet de structurer les pages de l'application de manière logique et esthétique en ajoutant des titres, des descriptions, des champs pour saisir les informations, et des boutons pour soumettre le formulaire, tout en gardant le code organisé et lisible.
Cette approche modulaire et structurée est un exemple classique de bonnes pratiques de développement en PHP, favorisant la réutilisabilité et la maintenabilité du code.

3.4.5.2 Méthodes et Fonctionnalités

- setBreadcrumbSeparator($separator): Permet de personnaliser le séparateur utilisé dans le fil d'Ariane.

- addMainTitle($text): Ajoute un titre principal à la page.
- addSubTitle($text): Ajoute un sous-titre.
- addDescription($text): Ajoute un paragraphe de texte descriptif.
- addBreadcrumb($text, $link = null): Ajoute un élément au fil d'Ariane, qui peut être un lien ou simplement du texte.
- addElement($title, $content): Ajoute un élément avec un titre et un contenu, ce qui peut être utile pour des sections informatives supplémentaires.
- addField($name, $type, $label, $value = '', $attributes = []): Ajoute un champ de formulaire, où $type pourrait être 'text', 'email', etc.
- addButton($type, $text, $attributes = []): Ajoute un bouton, par exemple un bouton de soumission.

3.4.5.3 Processus de Rendu

- La méthode render() assemble tous les éléments ajoutés dans un format HTML structuré :
- Commence par créer un conteneur principal avec des classes Bootstrap pour la mise en page.
- Construit un fil d'Ariane basé sur les éléments ajoutés, supprimant le dernier séparateur pour une présentation propre.
- Intègre les titres, descriptions, et autres éléments dans un ordre logique, en s'assurant que le titre principal et le fil d'Ariane apparaissent en premier.
- Les champs et les boutons de formulaire sont ajoutés avec des attributs personnalisés et des classes pour la gestion de l'affichage et de la validation.

Ferme toutes les balises ouvertes pour maintenir la structure HTML propre et valide.

3.4.5.4 Utilisation

Cette classe est extrêmement utile dans notre système MVC où la vue doit rester dynamique et facilement modifiable. Elle permet de centraliser la création de formulaires et d'autres éléments de page, rendant les vues plus faciles à gérer et à maintenir. De plus, elle permet une personnalisation avancée grâce à des méthodes dédiées à chaque type d'élément de page.

3.4.6 form_view

3.4.6.1 Rôle

Conçue pour faciliter la création et la gestion de formulaires HTML dynamiques. Elle permet de structurer les formulaires en ajoutant des champs, des boutons de soumission, et d'autres éléments interactifs, tout en assurant une intégration facile avec des scripts backend grâce à des méthodes bien définies.

3.4.6.2 Principales Caractéristiques et Fonctions

- Constructeur: Initialise le formulaire avec une action spécifique (URL de traitement du formulaire) et un libellé par défaut pour le bouton de soumission.
- addField($id, $name, $type, $label, $value = '', $attributes = []): Permet d'ajouter un champ de formulaire. Les types de champs peuvent inclure 'text', 'email', 'password', etc. Les attributs supplémentaires (comme class, style, etc.) peuvent également être spécifiés.
- addSelectField($id, $name, $label, $options, $selectedValue = '', $attributes = []): Ajoute un champ de sélection (menu déroulant) avec des options prédéfinies. Cela inclut la gestion de l'option sélectionnée et des attributs HTML supplémentaires.
- addButton($type, $label, $attributes = []): Ajoute un bouton au formulaire, par exemple un bouton de soumission. Les attributs additionnels peuvent être ajoutés pour une personnalisation avancée.
- build(): Compile et retourne le code HTML du formulaire. Cette méthode organise les champs et les boutons dans le format HTML approprié, prêt à être affiché sur une page web.

3.4.7 medecin_view

3.4.7.1 Rôle

Sert à gérer l'affichage des données liées aux médecins, comme lister les médecins, leurs congés, et les historiques médicaux. Cette classe utilise FormContainer pour structurer l'affichage et FormView pour les formulaires. Elle est conçue pour faciliter la représentation des informations de manière structurée et esthétique sur le site web.

3.4.7.2 Principales Fonctions de la Classe MedecinView

- renderHeader() et renderFooter(): Ces méthodes gèrent l'inclusion des en-têtes et pieds de page, encapsulant les vues dans un template cohérent.
- renderMedecinList($medecins): Affiche une liste de médecins avec des détails comme la spécialité et le téléphone. Utilise FormContainer pour un affichage structuré.
- renderConges($conges): Présente une liste des congés des médecins avec des options pour modifier ou supprimer chaque entrée. Intègre des interactions comme les boutons de formulaire pour les actions.
- renderEditCongeForm($conge): Génère un formulaire pour éditer un congé spécifique, permettant la modification des dates et de la raison du congé.
- renderCreateCongeForm($medecin_id): Fournit un formulaire pour créer un nouveau congé pour un médecin donné.
- renderMedicalHistories($histories): Affiche les historiques médicaux associés à un médecin, offrant une visualisation sous forme de tableau des diverses entrées.
- renderCreateMedicalHistoryForm($medecin_id): Crée un formulaire pour ajouter un nouvel historique médical pour un médecin spécifié.

- renderPatientList($patients): Liste les patients associés à un médecin, avec des informations détaillées pour chaque patient.
- renderCreatePatientForm(): Offre un formulaire pour enregistrer un nouveau patient dans le système.
- renderConfirmedAppointments($appointments): Montre une liste de rendez-vous confirmés pour un médecin, potentiellement intégrée avec une visualisation de calendrier pour une meilleure gestion des rendez-vous.

3.4.7.3 Avantages

- Cohérence: L'utilisation de FormContainer et FormView assure une cohérence dans l'affichage des formulaires et des listes, facilitant la maintenance et les mises à jour du code.
- Modularité: Chaque partie de l'affichage est gérée par des méthodes distinctes, ce qui permet de réutiliser et de modifier facilement les composants individuels sans perturber le reste du code.
- Intégration Facile: Peut être facilement intégré avec d'autres vues ou modèles pour étendre les fonctionnalités, comme la gestion des rendez-vous ou des urgences médicales.

3.4.8 secretaire_view

3.4.8.1 Rôle

Conçue pour gérer l'affichage des données relatives aux secrétaires et aux patients dans une application de gestion médicale. Elle utilise des composants tels que BaseView, AdminView, FormView, et FormContainer pour structurer et présenter les informations de manière claire et interactive.

3.4.8.2 Fonctionnalités de SecretaireView

- renderHeader() et renderFooter(): Ces méthodes chargent les en-têtes et les pieds de page, encapsulant les vues dans un template cohérent.
- renderSecretaireList($secretaires): Affiche une liste de secrétaires avec des options pour interagir, telles que voir des détails ou modifier des informations. Utilise FormContainer pour un affichage structuré.
- renderPatientList($patients): Présente une liste de patients avec des options d'action comme afficher, modifier, ou prendre rendez-vous. Cette méthode intègre également des scripts JavaScript pour améliorer l'interaction utilisateur.
- renderPatientDetails($patientDetails): Affiche les détails complets d'un patient spécifique, fournissant des informations détaillées et des actions possibles comme la modification ou la prise de rendez-vous.
- renderCreePatientForm(): Fournit un formulaire pour ajouter un nouveau patient, utilisant FormView pour la création du formulaire.

- renderEditPatientForm($patient): Génère un formulaire pour éditer les informations d'un patient existant, permettant de modifier les détails comme l'adresse, l'email, etc.
- renderCreateRdvForm($medecins, $patients): Offre un formulaire pour créer un nouveau rendez-vous, listant les médecins et patients disponibles pour sélection.
- secretaireRdv($rdv) et autres méthodes similaires pour rendez-vous confirmés, en attente, ou annulés: Ces méthodes affichent les rendez-vous dans différents statuts, permettant des actions comme confirmer ou annuler des rendez-vous.
- secretaireHoraires($horaires): Affiche les horaires de travail des médecins, avec des options pour modifier ou supprimer des horaires.
- renderChangeHorairesForm($horaire): Permet de modifier un horaire de travail spécifique, avec des champs pour ajuster les heures et les jours.

3.4.8.3 Avantages

Cohérence et réutilisabilité: L'utilisation de FormContainer et FormView assure une uniformité et une modularité dans l'affichage des formulaires et des listes, facilitant ainsi les mises à jour et la maintenance du code.
Interaction utilisateur améliorée: L'intégration de JavaScript pour des actions comme l'affichage des détails des patients directement sur la page améliore l'expérience utilisateur en rendant l'interface plus dynamique.
Flexibilité et extensibilité: Peut être facilement étendue ou modifiée pour intégrer d'autres fonctionnalités, comme des notifications ou des rappels pour les rendez-vous.

3.4.9 user_view

3.4.9.1 Rôle

Partie du système de gestion des vues pour l' application se concentrant sur les fonctionnalités liées aux utilisateurs, telles que l'affichage de la page d'accueil, la gestion des formulaires de connexion et d'inscription, ainsi que l'affichage d'un lien de déconnexion. Elle hérite de BaseView et utilise FormContainer pour structurer et afficher les contenus de manière organisée et modulaire.

3.4.9.2 Fonctionnalités de UserView

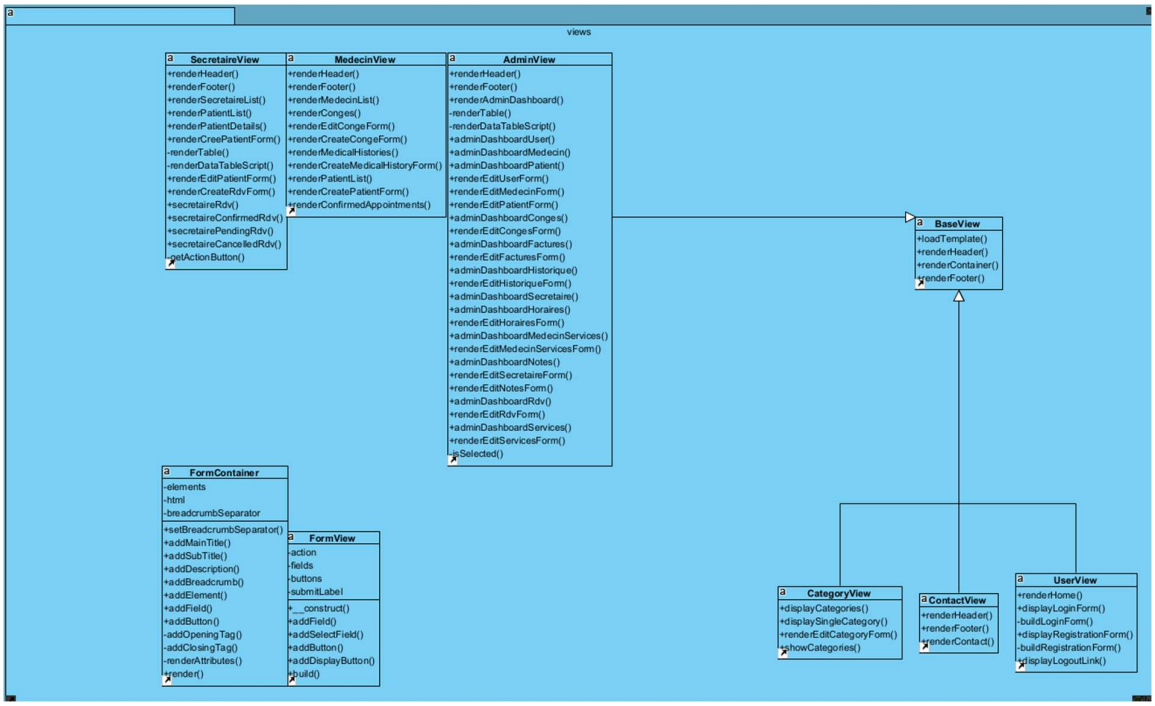
- renderHome(): Affiche la page d'accueil avec un fil d'Ariane et une description générale, utilisant FormContainer pour un affichage structuré et clair.
- displayLoginForm(): Présente un formulaire de connexion pour les utilisateurs, intégrant des méthodes pour construire le formulaire de manière dynamique avec validation et sécurité des données.
- displayRegistrationForm(): Offre un formulaire d'inscription pour de nouveaux utilisateurs, permettant la saisie de nom, email, et mot de passe, avec des options pour soumettre le formulaire.

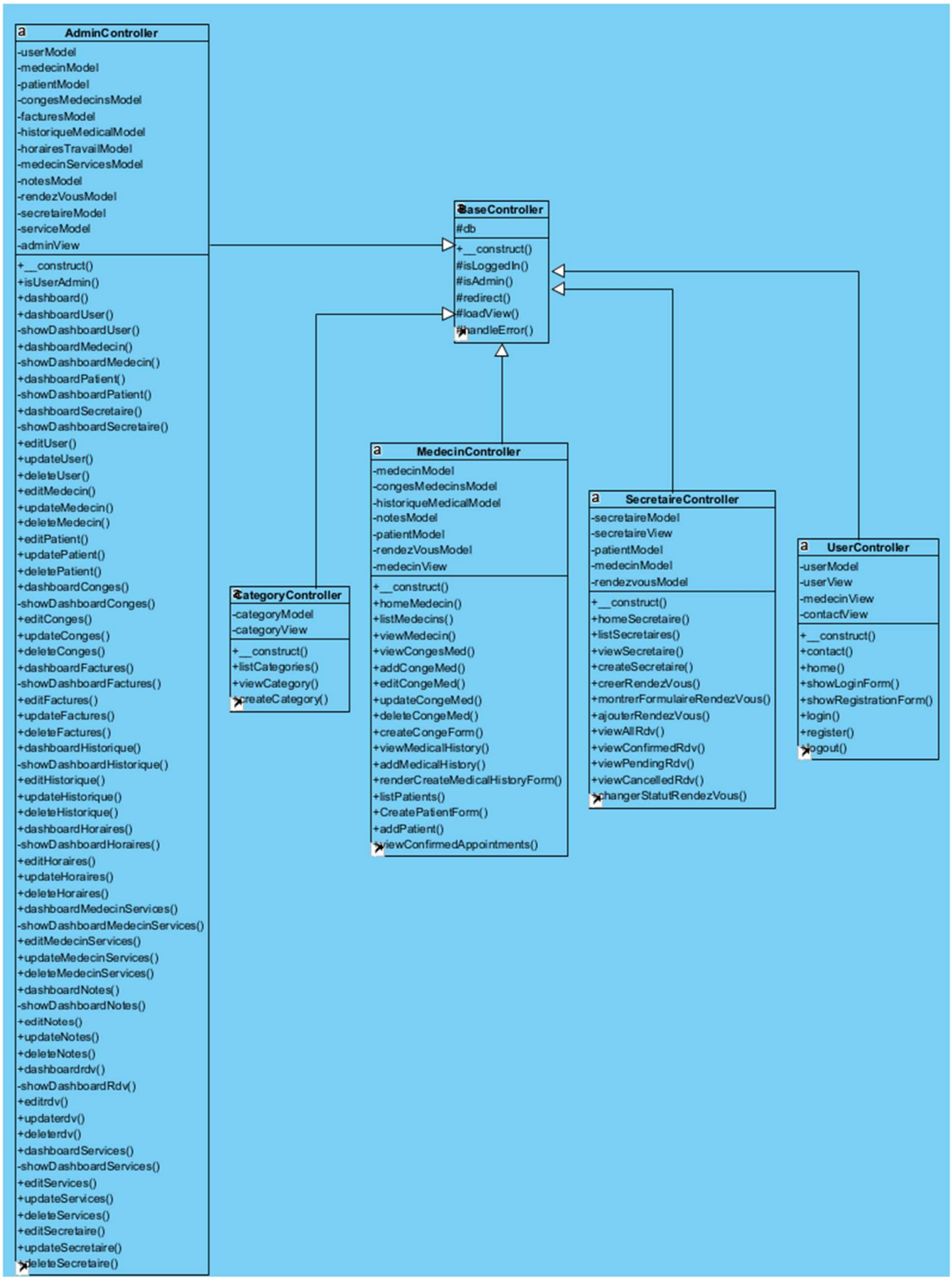
- displayLogoutLink(): Affiche un lien pour permettre à l'utilisateur de se déconnecter de l'application, ce qui est une pratique courante pour la gestion des sessions utilisateurs.

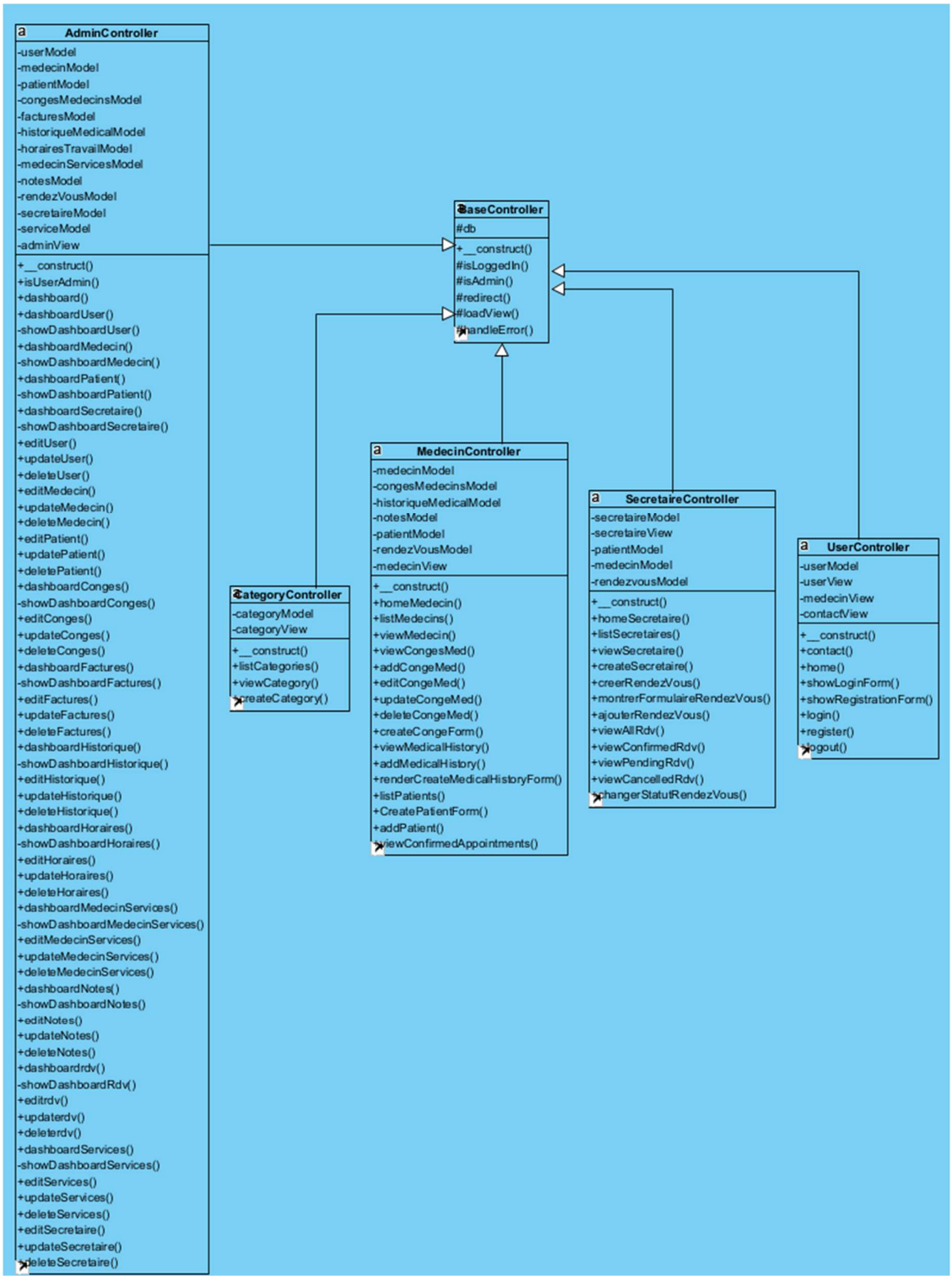
3.4.9.3 Avantages

- Modularité et réutilisation du code: Utilise FormContainer et FormView pour réduire la redondance du code et augmenter la réutilisabilité à travers différentes parties de l'application.
- Cohérence de l'interface utilisateur: Maintient une interface cohérente pour les utilisateurs lors de la navigation à travers différentes pages de l'application, améliorant ainsi l'expérience utilisateur.
- Sécurité améliorée: Les formulaires sont construits de manière à inclure des mesures de sécurité de base, telles que la validation du côté serveur, minimisant les risques de failles de sécurité.

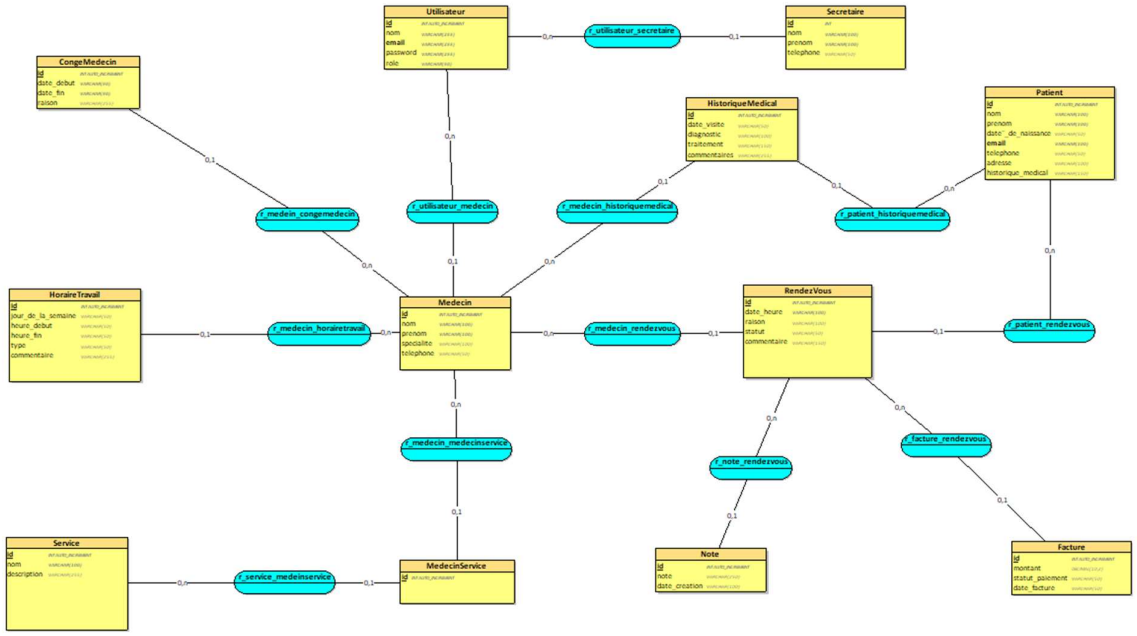
4 Schéma uml




AdminController
-userModel
-medecinModel
-patientModel
-congesMedecinsModel
-facturesModel
-historiqueMedicalModel
-horairesTravailModel
-medecinServicesModel
-notesModel
-rendezVousModel
-secretaireModel
-serviceModel
-adminView
+__construct()
+isUserAdmin()
+dashboard()
+dashboardUser()
-showDashboardUser()
+dashboardMedecin()
-showDashboardMedecin()
+dashboardPatient()
-showDashboardPatient()
+dashboardSecretaire()
-showDashboardSecretaire()
+editUser()
+updateUser()
+deleteUser()
+editMedecin()
+updateMedecin()
+deleteMedecin()
+editPatient()
+updatePatient()
+deletePatient()
+dashboardConges()
-showDashboardConges()
+editConges()
+updateConges()
+deleteConges()
+dashboardFactures()
-showDashboardFactures()
+editFactures()
+updateFactures()
+deleteFactures()
+dashboardHistorique()
-showDashboardHistorique()
+editHistorique()
+updateHistorique()
+deleteHistorique()
+dashboardHoraires()
-showDashboardHoraires()
+editHoraires()
+updateHoraires()
+deleteHoraires()
+dashboardMedecinServices()
-showDashboardMedecinServices()
+editMedecinServices()
+updateMedecinServices()
+deleteMedecinServices()
+dashboardNotes()
-showDashboardNotes()
+editNotes()
+updateNotes()
+deleteNotes()
+dashboardrdv()
-showDashboardRdv()
+editrdv()
+updaterdv()
+deleterdv()
+dashboardServices()
-showDashboardServices()
+editServices()
+updateServices()
+deleteServices()
+editSecretaire()
+updateSecretaire()
+deleteSecretaire()
BaseController
#db
+__construct()
#isLoggedIn()
#isAdmin()
#redirect()
#loadView()
#handleError()
CategoryController
-categoryModel
-categoryView
+__construct()
+listCategories()
+viewCategory()
+createCategory()
MedecinController
-medecinModel
-congesMedecinsModel
-historiqueMedicalModel
-notesModel
-patientModel
-rendezVousModel
-medecinView
+__construct()
+homeMedecin()
+listMedecins()
+viewMedecin()
+viewCongesMed()
+addCongeMed()
+editCongeMed()
+updateCongeMed()
+deleteCongeMed()
+createCongeForm()
+viewMedicalHistory()
+addMedicalHistory()
+renderCreateMedicalHistoryForm()
+listPatients()
+CreatePatientForm()
+addPatient()
+viewConfirmedAppointments()
SecretaireController
-secretaireModel
-secretaireView
-patientModel
-medecinModel
-rendezvousModel
+__construct()
+homeSecretaire()
+listSecretaires()
+viewSecretaire()
+createSecretaire()
+creerRendezVous()
+montrerFormulaireRendezVous()
+ajouterRendezVous()
+viewAllRdv()
+viewConfirmedRdv()
+viewPendingRdv()
+viewCancelledRdv()
+changerStatutRendezVous()
UserController
-userModel
-userView
-medecinView
-contactView
+__construct()
+contact()
+home()
+showLoginForm()
+showRegistrationForm()
+login()
+register()
+logout()


AdminController
-userModel
-medecinModel
-patientModel
-congesMedecinsModel
-facturesModel
-historiqueMedicalModel
-horairesTravailModel
-medecinServicesModel
-notesModel
-rendezVousModel
-secretaireModel
-serviceModel
-adminView
+__construct()
+isUserAdmin()
+dashboard()
+dashboardUser()
-showDashboardUser()
+dashboardMedecin()
-showDashboardMedecin()
+dashboardPatient()
-showDashboardPatient()
+dashboardSecretaire()
-showDashboardSecretaire()
+editUser()
+updateUser()
+deleteUser()
+editMedecin()
+updateMedecin()
+deleteMedecin()
+editPatient()
+updatePatient()
+deletePatient()
+dashboardConges()
-showDashboardConges()
+editConges()
+updateConges()
+deleteConges()
+dashboardFactures()
-showDashboardFactures()
+editFactures()
+updateFactures()
+deleteFactures()
+dashboardHistorique()
-showDashboardHistorique()
+editHistorique()
+updateHistorique()
+deleteHistorique()
+dashboardHoraires()
-showDashboardHoraires()
+editHoraires()
+updateHoraires()
+deleteHoraires()
+dashboardMedecinServices()
-showDashboardMedecinServices()
+editMedecinServices()
+updateMedecinServices()
+deleteMedecinServices()
+dashboardNotes()
-showDashboardNotes()
+editNotes()
+updateNotes()
+deleteNotes()
+dashboardrdv()
-showDashboardRdv()
+editrdv()
+updaterdv()
+deleterdv()
+dashboardServices()
-showDashboardServices()
+editServices()
+updateServices()
+deleteServices()
+editSecretaire()
+updateSecretaire()
+deleteSecretaire()
BaseController
#db
+__construct()
#isLoggedIn()
#isAdmin()
#redirect()
#loadView()
#handleError()
CategoryController
-categoryModel
-categoryView
+__construct()
+listCategories()
+viewCategory()
+createCategory()
MedecinController
-medecinModel
-congesMedecinsModel
-historiqueMedicalModel
-notesModel
-patientModel
-rendezVousModel
-medecinView
+__construct()
+homeMedecin()
+listMedecins()
+viewMedecin()
+viewCongesMed()
+addCongeMed()
+editCongeMed()
+updateCongeMed()
+deleteCongeMed()
+createCongeForm()
+viewMedicalHistory()
+addMedicalHistory()
+renderCreateMedicalHistoryForm()
+listPatients()
+CreatePatientForm()
+addPatient()
+viewConfirmedAppointments()
SecretaireController
-secretaireModel
-secretaireView
-patientModel
-medecinModel
-rendezvousModel
+__construct()
+homeSecretaire()
+listSecretaires()
+viewSecretaire()
+createSecretaire()
+creerRendezVous()
+montrerFormulaireRendezVous()
+ajouterRendezVous()
+viewAllRdv()
+viewConfirmedRdv()
+viewPendingRdv()
+viewCancelledRdv()
+changerStatutRendezVous()
UserController
-userModel
-userView
-medecinView
-contactView
+__construct()
+contact()
+home()
+showLoginForm()
+showRegistrationForm()
+login()
+register()
+logout()

5 Schéma MCD


Utilisateur
id
nom
email
password
role
r_utilisateur_secretaire
Secretaire
id
nom
prenom
telephone
CongeMedecin
id
date_debut
date_fin
raison
HistoriqueMedical
id
date_visite
diagnostic
traitement
commentaires
Patient
id
nom
prenom
date_de_naissance
email
telephone
adresse
historique_medical
r_medein_congemedecin
r_utilisateur_medecin
r_medecin_historiquemedical
r_patient_historiquemedical
HoraireTravail
id
jour_de_la_semaine
heure_debut
heure_fin
type
commentaire
Medecin
id
nom
prenom
specialite
telephone
RendezVous
id
date_heure
raison
statut
commentaire
r_medecin_horairetravail
r_medecin_rendezvous
r_patient_rendezvous
r_medecin_medecinservice
r_note_rendezvous
r_facture_rendezvous
Service
id
nom
description
r_service_medeinservice
MedecinService
id
Note
id
note
date_creation
Facture
id
montant
statut_paiement
date_facture