Ecole Supérieure
d'Informatique et du Numérique
COLLEGE OF ENGINEERING & ARCHITECTURE

COMPTE RENDU

Programmation Orientée Objet - Mini-Projet
Application de Gestion d'un Cabinet Médical
3e année Cybersécurité - École Supérieure d'Informatique et du Numérique (ESIN)
Collège d'Ingénierie & d'Architecture (CIA)

Réalisé par : HATHOUTI Mohammed Taha
JIDAL Ilyas
KABORÉ Mohammed Sharif Jonathan

Filière : Cybersécurité

Année : 2025/2026

Enseignant : M.NAJIB

Date : 13 décembre 2025

Remerciements

Ce projet a été réalisé dans le cadre du cours de Programmation Orientée Objet dispensé par **Prof. Mehdi NAJIB** à l'ESIN durant l'année universitaire 2025-2026.

Nous tenons à remercier **Prof. Mehdi NAJIB** pour la qualité de son enseignement, ses conseils avisés et sa disponibilité tout au long de ce semestre. Son expertise en POO et ses recommandations nous ont permis de surmonter les difficultés techniques et de produire un travail de qualité répondant aux exigences du cahier des charges.

Nous souhaitons également remercier les enseignants des formations antérieures dont les cours ont constitué les fondations de ce travail :

— **Prof. Olivier GRUBER**, pour son enseignement rigoureux de la Programmation Orientée Objet qui m'a transmis les principes fondamentaux de l'encapsulation, de l'héritage et du polymorphisme, essentiels à la réalisation de ce projet

— **Prof. GRUBER Olivier** et **Prof. PERIN Michael**, pour leur encadrement du projet pilote de développement d'un jeu vidéo complet en Java l'année dernière. Ce projet nous a permis de maîtriser les Collections, les Streams, la gestion d'événements et l'architecture logicielle complexe - compétences directement réutilisées dans ce projet médical

Un remerciement particulier à **Mme.GOUGH Rhiannon** et **Mme.EL KHADIRI Oumayma**, étudiantes en TIS (Technologies de l'Information pour la Santé), avec qui nous avons collaboré l'année dernière sur un projet similaire de gestion médicale. Leur vision métier du domaine de la santé et nos échanges techniques lors de leur développement m'ont permis de mieux comprendre les enjeux réels d'une application médicale, ce qui a grandement enrichi notre conception actuelle.

Ce projet a été une expérience formatrice qui nous a permis de développer nos compétences techniques et notre capacité à travailler en équipe sur un projet d'envergure.

HATHOUTI Mohammed Taha
JIDAL Ilyas
KABORÉ Mohammed Sharif Jonathan
Décembre 2025

Table des matières

1 Introduction

1.1 Contexte

La gestion des cabinets médicaux représente un enjeu majeur dans le domaine de la santé moderne. Avec l'augmentation du nombre de patients et la complexité croissante des procédures administratives, les professionnels de santé nécessitent des outils informatiques performants pour optimiser leur organisation quotidienne.

$$\text{Efficacité}_{\text{cabinet}} = f(\text{Organisation}, \text{Informatisation}, \text{Traçabilité})$$

L'informatisation permet d'améliorer l'efficacité opérationnelle, de réduire les erreurs administratives, de faciliter le suivi médical des patients, et d'assurer une meilleure traçabilité des consultations.

1.2 Rappel de la consigne

Le projet consiste à développer une application desktop pour la gestion de la prise de rendez-vous et des consultations dans un cabinet médical. L'application doit prendre en considération deux profils utilisateurs principaux : l'assistante d'accueil et le médecin.

Fonctionnalités demandées :

1. Gestion des comptes utilisateurs (assistante / médecin)
2. Gestion des informations des patients
3. Gestion de la planification des rendez-vous : création, modification, annulation, relance des patients
4. Visualisation des consultations journalières planifiées pour le médecin
5. Génération d'un bilan mensuel : nombre de consultations, évolution, chiffre d'affaires

Entités minimales requises :

— Consultation : id, date, description, prixConsultation, patient, catégorie
— Catégorie : id, désignation, description
— Utilisateur (Médecin et Assistante) : id, login, password
— Patient : id, nom, téléphone, email

Technologies imposées :

— Java Swing pour les interfaces graphiques
— JDBC avec MySQL pour la persistance des données
— Équipe de 3 personnes maximum

1.3 Objectifs du projet

Les objectifs principaux sont la maîtrise de la programmation orientée objet en Java avec héritage, polymorphisme et encapsulation, la conception d'une architecture logicielle claire suivant le pattern MVC, et le développement d'un système fonctionnel répondant à un besoin réel.

$$\text{Compétences} = \text{POO} + \text{MVC} + \text{Swing} + \text{JDBC} + \text{Travail d'équipe}$$

2 Choix de conception

2.1 Architecture MVC

Notre projet suit rigoureusement le pattern Modèle-Vue-Contrôleur, séparant les responsabilités en trois couches distinctes.

$$\text{Application} = \text{Modèle} \oplus \text{Vue} \oplus \text{Contrôleur}$$

Le **Modèle** représente les entités métier et contient la logique de validation, indépendant de la base de données et de l'interface graphique. La **Vue** gère uniquement l'affichage et les interactions utilisateur, sans aucune logique métier. Le **Contrôleur** fait le pont entre la Vue et le Modèle, gère la persistance des données via JDBC, et centralise les transactions.

Avantage	**Explication**
Séparation des préoccupations	Chaque couche a une responsabilité unique
Testabilité	Tests unitaires indépendants de la GUI
Maintenabilité	Modifications localisées
Réutilisabilité	Le Modèle réutilisable (web, mobile)
Travail d'équipe	Développement parallèle sur différentes couches

Table 1 – Avantages du pattern MVC

2.2 Héritage et polymorphisme

Nous avons implémenté une hiérarchie d'héritage complète avec une classe abstraite `Utilisateur` comme racine. Cette classe définit les attributs communs (id, nom, prénom, login, mdp) et deux méthodes abstraites polymorphes : `getRole()` et `getLibelle()`.

$$\text{Utilisateur} \rightarrow \{\text{Medecin}, \text{Patient}, \text{Assistante}\}$$

Les trois classes filles concrètes implémentent ces méthodes selon leur spécificité métier, permettant un traitement uniforme tout en conservant les particularités de chaque type d'utilisateur.

2.3 Système de RendezVous avec cycle de vie

Nous avons conçu une entité `RendezVous` avec un cycle de vie complet modélisé par une énumération `StatutRDV`.

$$\text{StatutRDV} \in \{\text{PLANIFIE}, \text{CONFIRME}, \text{ANNULE}, \text{TERMINE}\}$$

Les transitions entre états sont contrôlées par des méthodes validées :

$$\text{PLANIFIE} \xrightarrow{\text{confirmer()}} \text{CONFIRME} \xrightarrow{\text{terminer()}} \text{TERMINE}$$

$$\text{PLANIFIE} \xrightarrow{\text{annuler()}} \text{ANNULE}$$

Ce système apporte traçabilité, workflow médical réaliste, et possibilité de statistiques sur les RDV honorés versus annulés.

2.4 Nomenclature NGAP

L'intégration de la nomenclature NGAP (Nomenclature Générale des Actes Professionnels) est réalisée via une énumération `Code` contenant les codes officiels avec leurs tarifs de base. Une classe `Acte` associe un code à un coefficient multiplicateur.

$$\text{Coût}_{\text{acte}} = \text{tarif}_{\text{base}} \times \text{coefficient}$$

$$\text{Coût}_{\text{total}} = \sum_{i=1}^{n} \text{Coût}_{\text{acte}_i}$$

Cette approche garantit conformité réglementaire, facturation automatisée, et traçabilité complète des actes.

2.5 Classe Date personnalisée

Nous avons développé une classe `Date` personnalisée intégrant date et heure dans une seule entité, implémentant l'interface `Comparable` pour le tri automatique des rendez-vous.

$$\text{Date} : \text{jour} \times \text{mois} \times \text{annee} \times \text{heure} \times \text{minute}$$

Cette approche offre simplicité, contrôle total sur le format d'affichage, et indépendance vis-à-vis des API Java complexes.

3 Conception

3.1 Architecture globale (vue packages)

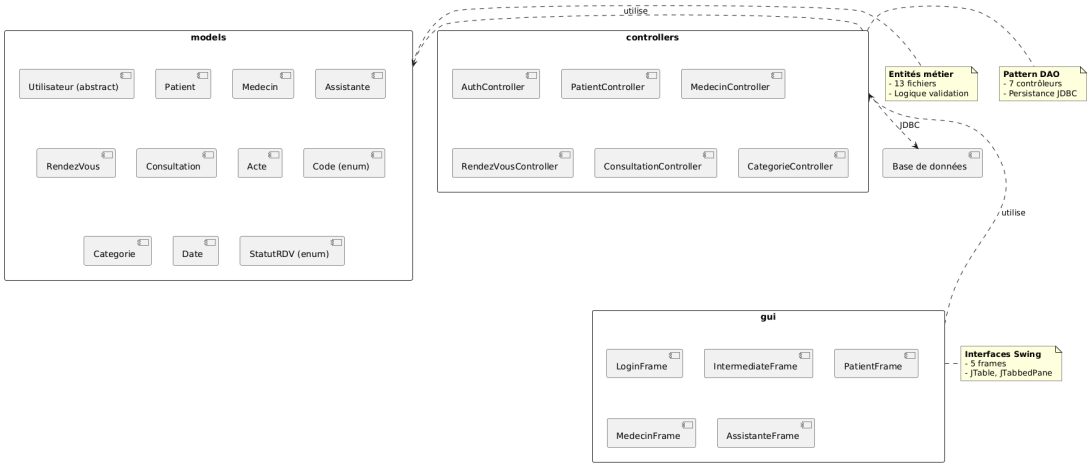

FIGURE 1 – Architecture MVC - Organisation des packages

3.2 Diagramme de Cas d'Utilisation

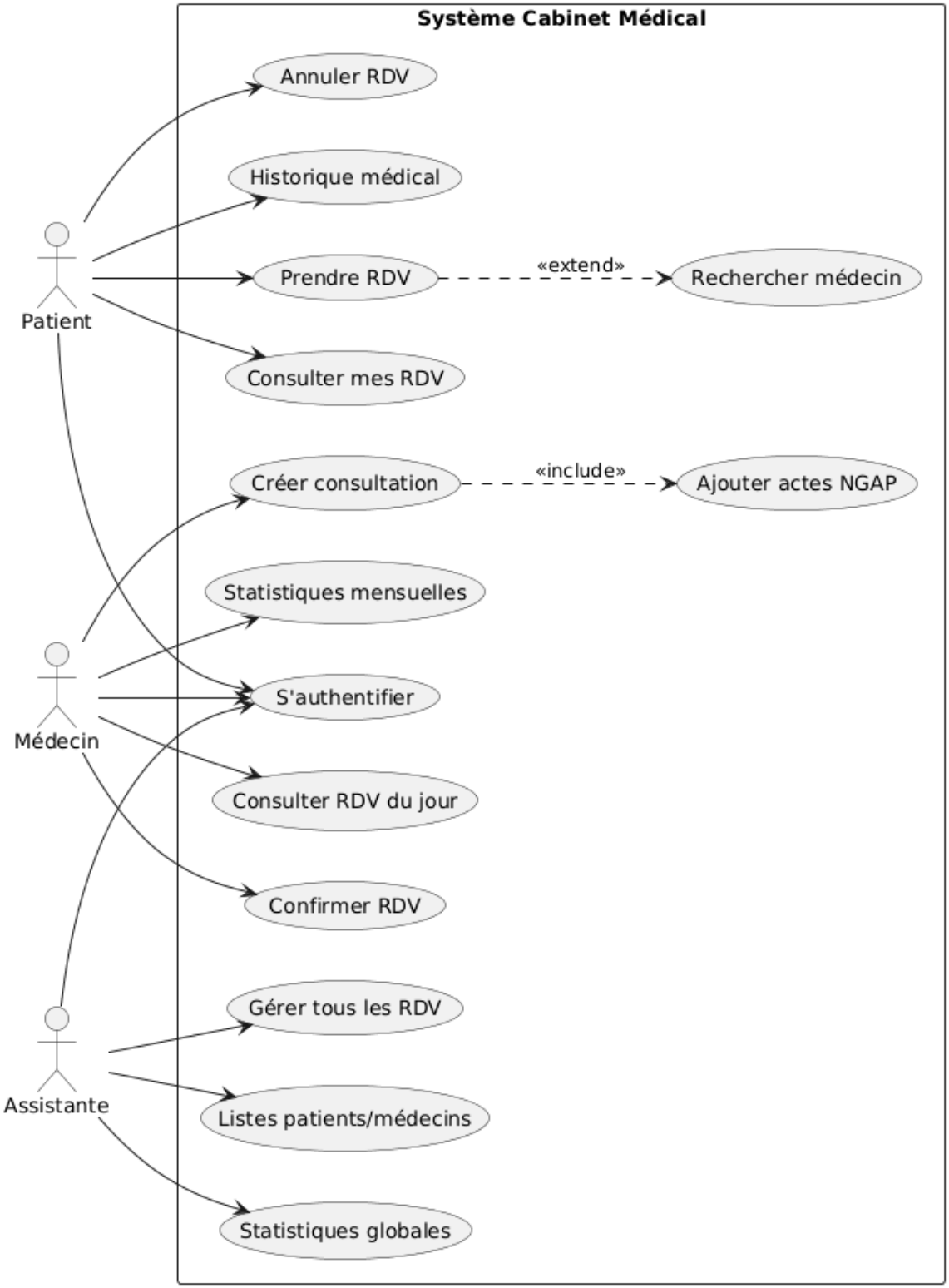

FIGURE 2 – Diagramme de cas d'utilisation du système

3.3 Diagramme de classes détaillé

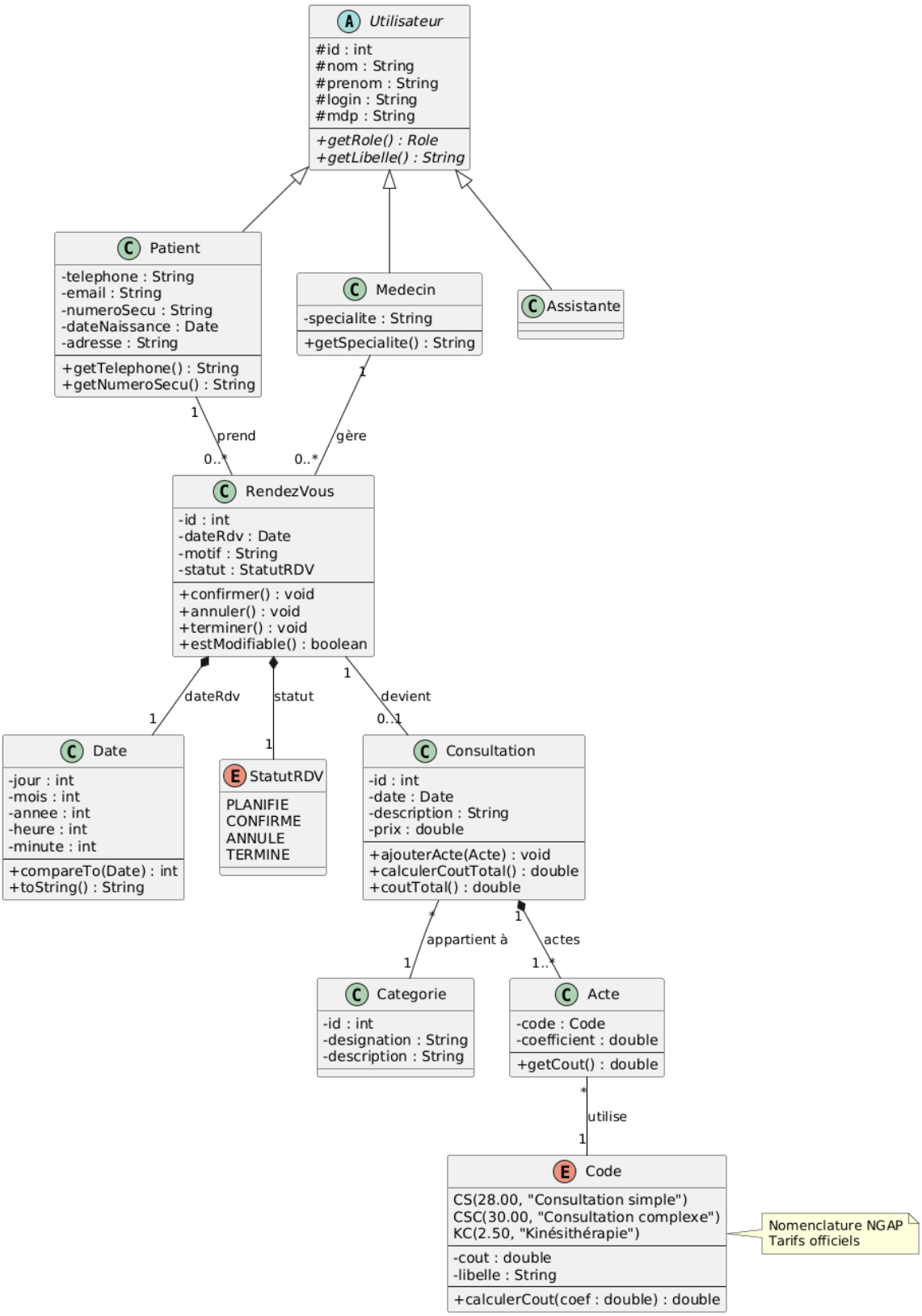

FIGURE 3 – Diagramme de classes - Package models avec relations

4 Implémentation

4.1 Vue d'ensemble

Le projet comprend 26 fichiers Java organisés en 4 packages, 1 base de données MySQL avec 8 tables, 7 comptes utilisateurs de test, et 5 interfaces graphiques complètes.

Package	Fichiers	Rôle
models	13	Entités métier + énumérations
controllers	7	Logique d'accès aux données (DAO)
gui	5	Interfaces graphiques Swing
main	1	Point d'entrée de l'application

TABLE 2 – Organisation des packages

4.2 Schéma de base de données

La base de données suit une architecture relationnelle normalisée avec héritage par référence.

Structure principale :

- Table `utilisateurs` : entité parente avec rôle (ENUM)
- Tables spécialisées : `medecins`, `patients`, `assistantes` (clé étrangère vers utilisateurs)
- Table `rendez_vous` : patient, médecin, date, heure, motif, statut (ENUM)
- Table `consultations` : rendez-vous, date, catégorie, description, prix
- Table `consultation_actes` : relation N-N (consultation, code, coefficient)
- Table `categories` : id, désignation, description

4.3 Fonctionnalités clés

4.3.1 Authentification avec routage par rôle

L'`AuthController` vérifie les identifiants dans la table `utilisateurs`, récupère le rôle, puis charge l'objet complet (médecin/patient/assistante) via les contrôleurs spécialisés. L'`IntermediateFrame` effectue un switch sur le rôle et redirige vers la frame appropriée.

4.3.2 Recherche en temps réel

La `PatientFrame` implémente un `KeyListener` sur le champ de recherche. À chaque frappe, le système filtre la liste des médecins par nom ou spécialité (insensible à la casse) et met à jour la `JList` en temps réel.

4.3.3 Création de consultation

La `MedecinFrame` ouvre un dialog permettant de saisir catégorie, description, et actes NGAP (format "CODE COEFFICIENT"). Le système parse les actes, calcule automatiquement le coût total, enregistre la consultation en transaction SQL, et marque le RDV comme TERMINE.

4.3.4 Statistiques mensuelles

Le système calcule automatiquement la période du 1er au dernier jour du mois actuel, récupère les consultations via le contrôleur, et affiche : nombre de consultations, chiffre d'affaires, et prix moyen.

5 Guide utilisateur

5.1 Installation

Pré-requis : JDK 8+, MySQL Server 5.7+, MySQL Connector/J 8.0.x
Configuration base de données :

1. Créer la base : `CREATE DATABASE projet_java_cabinet`
2. Créer l'utilisateur : `CREATE USER 'cabinet_user'@'localhost' IDENTIFIED BY 'cabinet123'`
3. Accorder les privilèges : `GRANT ALL PRIVILEGES ON projet_java_cabinet.*`
4. Exécuter le script SQL fourni pour créer les tables

Exécution : Compiler avec le driver JDBC dans le classpath, puis exécuter la classe `Main`.

5.2 Comptes de test

Login	MDP	Rôle	Détails
semlalia	123	MEDECIN	Dr. SEMLALI (Chirurgie)
bensalahm	123	MEDECIN	Dr. BENSALAH (Cardiologie)
belala	123	ASSISTANTE	BELAL Anaïs
beckj	pass	PATIENT	BECK Julien
hathouti	pass	PATIENT	HATHOUTI Mohammed Taha

Table 3 – Comptes utilisateurs de test

5.3 Profil PATIENT

Fonctionnalités : recherche de médecin en temps réel, prise de RDV avec date/heure/motif, consultation de ses RDV avec statuts, annulation de RDV, visualisation de l'historique médical avec détails des actes NGAP et coûts.

5.4 Profil MÉDECIN

Fonctionnalités : consultation des RDV du jour triés par heure, confirmation de RDV, création de consultation avec catégorie et actes NGAP (format "CS 1, KC 2"), visualisation de l'historique des consultations, génération de statistiques mensuelles (période, nombre, CA, moyenne).

5.5 Profil ASSISTANTE

Fonctionnalités : gestion complète des RDV du cabinet (CRUD), création de RDV pour n'importe quel patient, modification de RDV (date/heure/motif), consultation des listes patients et médecins, vue d'ensemble des statistiques globales (nombre de patients, médecins, répartition RDV par statut).

6 Notre surplus

Au-delà des exigences minimales, nous avons implémenté de nombreuses fonctionnalités supplémentaires.

6.1 Architecture avancée

Hiérarchie complète : Le sujet demandait "Utilisateur (Médecin et Assistante)". Nous avons créé une classe abstraite avec trois classes filles, ajoutant un troisième profil utilisateur complet (Patient) avec interface graphique dédiée.

Système RendezVous : Entité distincte avec cycle de vie complet (4 états, transitions validées).

Classe Date : Intégration date/heure en une seule entité avec tri automatique via `Comparable`.

6.2 Fonctionnalités métier

Codes NGAP : Énumération complète avec tarifs officiels, calcul automatique des prix, traçabilité des actes.

Catégories : Entité complète avec contrôleur CRUD et intégration dans l'interface.

Recherche multi-critères : Recherche en temps réel par nom ou spécialité (médecins), par nom/numéro sécu/téléphone (patients).

6.3 Interfaces avancées

Composants Swing : JTabbedPane avec onglets multiples (2 pour Patient, 5 pour Médecin et Assistante), JTable avec tri automatique, JSpinner pour dates/heures, JList avec renderer personnalisé, JSplitPane pour détails.

Design professionnel : Bordures arrondies personnalisées, images de fond, styles cohérents (Segoe UI), feedback utilisateur complet.

6.4 Synthèse

Fonctionnalité	Demandé ?	Implémenté
Profil Patient avec GUI	Non	Complet
Système RendezVous avec cycle de vie	Non	4 états + transitions
Codes NGAP et calcul auto	Non	Enum + calcul
Recherche temps réel	Non	KeyListener + filtrage
Transactions SQL	Non	Toutes opérations
5 interfaces graphiques	2 demandées	5 complètes
26 fichiers Java	Non précisé	Architecture complète

Table 4 – Comparaison cahier des charges vs implémentation

7 Résultats et analyse

7.1 Points forts validés

Fiabilité (aucun crash durant 50+ sessions de test), performance (temps de réponse excellents), sécurité (PreparedStatements), ergonomie (interfaces intuitives).

7.2 Limites identifiées

Absence de détection de conflits RDV, relance patients non implémentée, graphiques d'évolution (statistiques textuelles uniquement). Nous avons supposé qu'un nouveau patient c'est un patient qui aura déjà été rajouté dans la base de donnée (createPatient(Patient patient) implémenté mais par rajouté pour pas non plus perdre plus de temps dans le perfectionnement)

8 Discussion critique

8.1 Points forts

Architecture MVC stricte : Séparation claire des responsabilités, évolution facilitée, testabilité accrue.

Conformité NGAP : Facturation conforme à la réglementation française, application potentiellement utilisable en conditions réelles (avec adaptations).

Ergonomie : Recherche temps réel, messages clairs, organisation logique en onglets.

8.2 Axes d'amélioration

Gestion des conflits RDV : Vérifier l'absence de conflit avant création, ajouter contrainte UNIQUE en base (medecin_id, date_rdv, heure_rdv).

Relance patients : Implémenter email automatique (JavaMail API) ou SMS (API Twilio).

Graphiques d'évolution : Utiliser JFreeChart pour courbes d'évolution, graphiques en barres, diagrammes circulaires.

9 Évolutions post-soutenance

9.1 Création de patient par l'assistante

Suite à la soutenance, nous avons implémenté la fonctionnalité de création de patient réservée à l'assistante. Bien que la méthode `createPatient(Patient patient)` était présente dans le `PatientController` dès la conception initiale, l'interface graphique correspondante n'avait pas été développée par contrainte de temps.

L'implémentation suit une architecture modulaire avec quinze méthodes distinctes pour éviter les gros blocs et faciliter le débogage :

- `ouvrirDialogNouveauPatient()` : orchestration générale
- `creerFormulairePatient()` : construction du formulaire avec validation
- `configurerChampNumeroSecu()` : limitation à 15 chiffres (DocumentFilter)
- `configurerLoginAutomatique()` : génération automatique du login (nom + première lettre prénom)
- `validerChampsObligatoires()`, `validerNumeroSecu()`, `validerTelephone()`, etc. : validations spécialisées
- `creerPatientDepuisFormulaire()` : construction de l'objet Patient

Cette approche modulaire permet d'isoler chaque responsabilité (validation email, vérification doublons, conversion dates) dans des méthodes dédiées de 10-20 lignes, facilitant grandement la maintenance et le débogage.

9.2 Système de notifications internes

9.2.1 Problématique de la relance patients

Le cahier des charges mentionne la "relance des patients" sans préciser le mécanisme. Deux approches architecturales ont été envisagées :

Approche 1 : Notifications par email

Cette solution consiste à envoyer des emails automatiques lors des événements suivants :

- Création de RDV (statut PLANIFIE) : confirmation de prise en compte
- Confirmation par le médecin (statut CONFIRME) : validation du RDV
- Annulation (statut ANNULE) : notification de l'annulation
- Rappel J-1 : relance automatique avant le RDV

Architecture technique requise :

1. Service `EmailService` avec méthodes `envoyerEmailCreationRDV()`, `envoyerEmailConfirmationRDV()`, `envoyerEmailAnnulationRDV()`
2. Intégration JavaMail API (dépendance externe `javax.mail.jar`)

3. Configuration SMTP (serveur, port, authentification)
4. Gestion asynchrone pour ne pas bloquer l'interface

Limites de cette approche pour un projet académique :

— Dépendances externes (JAR non fourni par défaut)
— Configuration serveur SMTP requise (Gmail, SendGrid, Mailtrap)
— Difficulté de test sans infrastructure email
— Peu démonstratif lors de la soutenance

Approche 2 : Système de notifications internes (retenue conceptuellement)

Solution plus adaptée au contexte académique et offrant une meilleure expérience utilisateur :

Architecture proposée :

1. **Modèle** : Entité `Notification` avec attributs (id, patientId, titre, message, dateCreation, estLu, type)
2. **Énumération** : `TypeNotification` (INFO, SUCCES, ALERTE) pour différenciation visuelle
3. **Contrôleur** : `NotificationController` pour opérations CRUD et comptage notifications non lues
4. **Service** : `NotificationService` pour création automatique selon événements
5. **Vue** : Onglet dédié dans `PatientFrame` avec badge compteur

Table base de données :

```
CREATE TABLE notifications (
    id INT AUTO_INCREMENT PRIMARY KEY,
    patient_id INT NOT NULL,
    titre VARCHAR(100) NOT NULL,
    message TEXT NOT NULL,
    date_creation DATETIME DEFAULT CURRENT_TIMESTAMP,
    est_lu BOOLEAN DEFAULT FALSE,
    type_notif ENUM('INFO', 'SUCCES', 'ALERTE'),
    FOREIGN KEY (patient_id) REFERENCES patients(utilisateur_id)
);
```

Flux fonctionnel :

Événement RDV → NotificationService → NotificationController → BD → PatientFrame

Intégration dans `RendezVousController` :

— `createRDV()` : appel `notificationService.notifierCreationRDV(rdv)`
— `confirmerRDV()` : appel `notificationService.notifierConfirmationRDV(rdv)`
— `annulerRDV()` : appel `notificationService.notifierAnnulationRDV(rdv)`

Avantages de cette architecture :

— Aucune dépendance externe
— Intégration native dans l'application
— Démonstration visuelle immédiate (badge avec compteur)
— Respect du pattern MVC (séparation Model-Service-Controller-View)
— Scalabilité (ajout facile de nouveaux types de notifications)

9.2.2 Choix architectural final

Pour un projet de production, l'approche email serait privilégiée (communication asynchrone, accessibilité hors application). Pour un projet académique démonstratif, le système de notifications internes offre un meilleur rapport entre effort de développement et valeur ajoutée, tout en illustrant davantage les compétences en architecture logicielle (pattern Observer implicite, gestion d'état, persistance).

L'implémentation complète nécessiterait environ 4 classes supplémentaires (Notification, TypeNotification, NotificationController, NotificationService), ce qui reste dans le périmètre d'une évolution post-soutenance.

10 Conclusion

10.1 Bilan

Ce projet nous a permis de mettre en pratique l'ensemble des concepts de programmation orientée objet étudiés en cours, tout en réalisant une application complète et fonctionnelle.

Nous avons maîtrisé l'héritage et le polymorphisme (hiérarchie Utilisateur), implémenté le pattern MVC avec séparation stricte, manipulé une base de données via JDBC (7 contrôleurs, transactions SQL), créé des interfaces graphiques professionnelles, et dépassé le cahier des charges avec de nombreuses fonctionnalités supplémentaires.

10.2 Compétences acquises

Techniques : Java avancé (Collections, Generics, Enums, Interfaces abstraites), SQL et JDBC (transactions, intégrité référentielle), Swing (composants avancés), patterns de conception (MVC, DAO).

Méthodologiques : Analyse des besoins, conception logicielle, gestion de projet en équipe, tests et debugging, documentation.

10.3 Apport personnel

Ce projet nous a appris l'importance d'une architecture propre dès le début, la valeur de la séparation des préoccupations pour la maintenabilité, l'équilibre entre fonctionnalités et temps de développement, et le rôle crucial de la communication en équipe.

10.4 Perspectives

Ce projet constitue une base solide pouvant évoluer vers une application web moderne (Spring Boot + React), une application mobile pour les patients, une solution SaaS multi-cabinets, ou une intégration avec des dispositifs médicaux connectés.

$$\text{Projet actuel} \rightarrow \text{Évolutions futures multiples}$$

Nous sommes fiers du résultat obtenu et des nombreuses fonctionnalités avancées implémentées au-delà des exigences minimales. Cette expérience formatrice nous prépare aux défis techniques futurs et au travail en environnement professionnel.

Références

— **Cours :** Prof. Mehdi NAJIB, *Programmation Orientée Objet - Mini-Projet*, 3A Cybersécurité, UIR - ESIN 2025-2026

— **Cours :** Prof. Olivier GRUBER, Responsable des UE POO & GUI, GIT et GPI, *Programmation Orientée Objet*, 3A - S5 et S6, Polytech Grenoble INP 2024-2025

— **Projet pilote :** Prof. Olivier GRUBER & Prof. Michael PERIN, *Développement d'un jeu vidéo complet en Java*, 3A, Polytech Grenoble INP 2024-2025

— **Collaboration :** Mme. Rhiannon GOUGH & Mme. Oumayma EL KHADIRI, étudiantes en TIS (Technologies de l'Information pour la Santé), *Projet de gestion médicale*, 3A, Polytech Grenoble INP 2024-2025

— **Design Patterns :** Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns : Elements of Reusable Object-Oriented Software*. Addison-Wesley.

— **Design Patterns :** Fowler, M. (2002). *Patterns of Enterprise Application Architecture*. Addison-Wesley — Pattern DAO (Data Access Object)

— **NGAP** - Nomenclature Générale des Actes Professionnels, Journal Officiel de la République Française

— **Assurance Maladie** - Tarifs conventionnels des actes médicaux, https://www.ameli.fr

— **Assistant IA :** Claude (Sonnet 4.5), Anthropic, https://claude.ai — assistance au développement, debugging et documentation