

ISMAGI

Rapport d'Examen de fin de module d'UML

Gestion de rendez-vous Médical

Table des matières

Partie I – Introduction et Présentation du projet

- 1.1. Contexte du projet
- 1.2. Problématique
- 1.3. Objectifs du projet
- 1.4. Technologies et méthodologie utilisées

Partie II – Présentation de 2TUP et application des étapes d'analyse

- 2.1. Présentation de la démarche 2TUP
- 2.2. Étape 1 : Modélisation métier
 - Diagramme d'activité global
 - Mini cahier des charges fonctionnel
- 2.3. Étape 2 : Spécification des exigences par les cas d'utilisation
 - Identification des acteurs
 - Cas d'utilisation par acteur
 - Structuration en packages
 - Priorisation et planification des itérations
- 2.4. Étape 3 : Réalisation des cas d'utilisation – Classes d'analyse
 - Modèle de domaine
 - Diagramme de classes métier
 - Diagramme d'état du RendezVous
 - Classes participantes
- 2.5. Étape 4 : Modélisation de la navigation
 - Interfaces identifiées
 - Diagramme de navigation
 - Maquettes Swing

Partie III – Application des étapes de conception et du code réalisé

- 3.1. Architecture technique de l'application
 - Modèle MVC + DAO
 - Organisation en couches
- 3.2. Conception détaillée par itération
 - Itération 1 : Réserver un rendez-vous
 - Itération 2 : Gérer un rendez-vous (Scheduler)
 - Itération 3 : Ajouter un commentaire (Docteur)
- 3.3. Diagramme de classes de conception

3.4. Difficultés rencontrées et ajustements

3.5. Résultat fonctionnel

- Fonctionnalités validées
- État du système
- Pistes d'évolution

Conclusion générale

Partie I – Introduction et Présentation du projet

1.1. Contexte du projet

Dans le cadre du cours d'UML et de modélisation orientée objet, il nous est demandé de concevoir et de réaliser une application Java en suivant une démarche rigoureuse de conception basée sur des diagrammes UML. Le projet retenu est une application de **gestion des visites médicales au sein d'une clinique**.

La gestion des rendez-vous en clinique peut rapidement devenir complexe lorsqu'il s'agit de jongler entre plusieurs patients, plusieurs médecins, des salles et des horaires à organiser. Ce projet vise à centraliser ces processus en une seule application claire, bien structurée et facile d'utilisation, permettant aux différents acteurs d'interagir efficacement.

1.2. Problématique

Dans un contexte de forte affluence ou d'organisation manuelle, plusieurs problèmes peuvent survenir :

- Double réservation de créneaux horaires ;
- Erreurs humaines dans la saisie ou le suivi des rendez-vous ;
- Difficulté à suivre les commentaires médicaux post-visite ;
- Gestion confuse des modifications et annulations.

D'où le besoin d'un **système informatisé**, centralisé et orienté objet, facilitant la **gestion des visites** en intégrant plusieurs rôles (patient, médecin, personnel administratif).

1.3. Objectifs du projet

L'application développée a pour objectifs de :

- Permettre aux **patients** de réserver ou annuler un rendez-vous selon les disponibilités ;
- Permettre au **scheduler** (chargé de gestion) de modifier ou annuler des rendez-vous ;
- Permettre au **docteur** de consulter ses rendez-vous et d'ajouter des commentaires à la fin d'une visite ;
- Garantir l'intégrité des créneaux horaires grâce à une gestion centralisée des *TimeSlots* ;

- Suivre une architecture claire et évolutive en Java, utilisant le modèle **MVC avec DAO** et des **diagrammes UML** pour la conception.

1.4. Technologies et méthodologie utilisées

L'application repose sur les outils et concepts suivants :

- **Langage** : Java (version standard)
- **Interface graphique** : Swing
- **Architecture** : MVC (Model-View-Controller) avec DAO (Data Access Object)
- **Modélisation UML** : diagrammes de classes, cas d'utilisation, séquence, composants et états
- **Méthode de conception** : 2TUP (Two-Track Unified Process), qui permet une modélisation parallèle fonctionnelle et technique.

Partie II – Présentation de 2TUP et application des différentes étapes d'analyse

2.1 Présentation de la démarche 2TUP

La démarche **2TUP** (*Two Track Unified Process*) est une méthode de développement logiciel itérative et incrémentale, fondée sur la modélisation UML. Elle propose une approche structurée en **deux pistes parallèles** :

- Une **piste fonctionnelle**, centrée sur la compréhension des besoins utilisateurs à travers des **cas d'utilisation**, des **maquettes** et des **scénarios métiers** ;
- Une **piste technique**, orientée sur la **structure logicielle**, les **architectures**, et les **composants** nécessaires à la réalisation du système.

L'un des grands principes de 2TUP est de **retarder le codage** pour d'abord **modéliser en profondeur** :

- ce que doit faire le système (analyse),
- puis comment il va être structuré (conception).

Cette méthode favorise une construction **progressive et contrôlée du système**, en découpant les besoins en **itérations successives**, chacune produisant une version enrichie du logiciel.

2.2 Étape 1 : Modélisation métier

Objectif de l'étape

L'objectif de cette étape est de comprendre et représenter le fonctionnement métier de la clinique sans entrer encore dans les détails techniques. Il s'agit de **décrire le processus métier** global : de la réservation à la réalisation du rendez-vous, en passant par sa gestion et son suivi.

Démarche appliquée

- Identification des **principaux acteurs métier** : patients, médecins, personnel chargé de la planification (scheduler).
- Analyse des **flux d'actions métier** : qui fait quoi, quand et pourquoi.
- Représentation du **processus global** via un **diagramme d'activité UML**.
- Extraction d'un **mini cahier des charges fonctionnel** à partir du scénario général.

Résultats obtenus

- Élaboration d'un **diagramme d'activité** représentant les grandes étapes :
 - Consultation des créneaux horaires ;
 - Réservation d'un rendez-vous ;
 - Validation ou modification par le scheduler ;
 - Réalisation de la visite par le médecin ;
 - Saisie d'un commentaire éventuel ;
 - Archivage ou annulation.
- Rédaction d'un **cahier des charges fonctionnel** synthétique, présentant les attentes de chaque acteur, sous forme de tableau ou fiche par rôle.

✓ Diagramme d'activité (représentation textuelle)

Ce diagramme représente **le processus métier global** de gestion des visites dans la clinique, depuis la prise de rendez-vous jusqu'à sa finalisation.

Activité principale : Gestion des rendez-vous

Début



[Patient] → Consulter les créneaux disponibles



Choisir un créneau



Envoyer une demande de réservation



[Scheduler] → Valider ou modifier le rendez-vous

└─> Si indisponibilité : proposer autre créneau



Rendez-vous confirmé



[Docteur] → Effectuer la consultation



[Docteur] → Ajouter un commentaire (facultatif)



Fin du rendez-vous



[Scheduler] → Marquer le rendez-vous comme terminé



Fin

Exceptions possibles :

- Le patient annule avant la confirmation → Fin
 - Le scheduler annule en cas d'indisponibilité → Notification au patient
-

✓ Mini cahier des charges fonctionnel

Ce tableau synthétise **les fonctionnalités attendues** du système selon les différents **acteurs** identifiés.

Acteur	Objectif métier	Fonctionnalités attendues
Patient	Réserver un rendez-vous médical	- Consulter les créneaux horaires disponibles
		- Réserver un rendez-vous
		- Annuler un rendez-vous réservé
Scheduler	Superviser les rendez-vous	- Valider ou refuser les réservations
		- Modifier la date/heure d'un rendez-vous
		- Annuler un rendez-vous (problème ou absence de médecin)
		- Marquer un rendez-vous comme terminé
Docteur	Réaliser la visite médicale	- Visualiser la liste des rendez-vous à venir
		- Ajouter un commentaire après consultation

2.3 Étape 2 : Spécification des exigences par les cas d'utilisation

Objectif de l'étape

Cette étape vise à formaliser **les besoins fonctionnels du système** sous forme de **cas d'utilisation**, en se basant sur le processus métier décrit précédemment. Elle permet de préciser **les interactions entre les acteurs et le système**, et constitue la base des futures itérations de développement.

Démarche appliquée

1. Identification des acteurs

À partir du processus métier et du cahier des charges fonctionnel, trois acteurs principaux ont été identifiés :

- **Patient** : utilisateur du système qui souhaite réserver une visite médicale.

- **Scheduler** : personnel chargé de la gestion des rendez-vous.
- **Docteur** : professionnel de santé réalisant les consultations.

2. Définition des cas d'utilisation

Pour chaque acteur, les cas d'utilisation suivants ont été retenus :

Acteur	Cas d'utilisation
Patient	- Réserver un rendez-vous
	- Annuler un rendez-vous
Scheduler	- Valider / Refuser un rendez-vous
	- Modifier la date/heure d'un rendez-vous
	- Annuler un rendez-vous
	- Marquer un rendez-vous comme terminé
Docteur	- Visualiser les rendez-vous programmés
	- Ajouter un commentaire post-visite

3. Structuration des cas d'utilisation

Les cas d'utilisation ont été organisés par **acteurs** et **priorités** dans des **packages logiques** :

- Package GestionRendezVous
- Package ConsultationMedecin
- Package AdministrationPlanning


4. Ajout de relations entre cas d'utilisation

- Le cas *"Ajouter un commentaire"* est un **extend** du cas *"Effectuer une consultation"*
- Le cas *"Modifier un rendez-vous"* **inclut** *"Valider un rendez-vous"*

5. Priorisation des cas d'utilisation

Une priorité a été assignée à chaque cas, selon :

- **La valeur fonctionnelle** (importance métier)
- **Le risque technique** (complexité à implémenter)

Cas d'utilisation	Priorité fonctionnelle	Risque technique	Choisi pour itération
Réserver un rendez-vous	Haute	Faible	 Oui

Annuler un rendez-vous	Moyenne	Faible	✓ Oui
Ajouter un commentaire	Moyenne	Moyenne	✓ Oui
Modifier un rendez-vous	Moyenne	Moyenne	✗ Pas encore
Marquer un rendez-vous terminé	Basse	Faible	✗ Pas encore

6. Planification des itérations

Trois cas d'utilisation ont été retenus pour la première itération :

- Réservation (Patient)
- Annulation (Patient)
- Ajout de commentaire (Docteur)

2.4 Étape 4 : Réalisation des cas d'utilisation – Classes d'analyse

Objectif de l'étape

L'objectif de cette étape est de transformer les exigences fonctionnelles modélisées dans les cas d'utilisation en **structures analytiques orientées objet**, c'est-à-dire :

- Identifier les **concepts du domaine** (classes principales),
- Construire un **modèle de domaine** (diagramme de classes métier),
- Repérer les classes ayant un **comportement dynamique complexe**,
- Détailler les **classes participantes** à chaque cas d'utilisation.

Cela permet de poser les bases solides pour la **conception technique** dans la prochaine phase.

Démarche appliquée

1. Identification des concepts métier

À partir des cas d'utilisation, nous avons extrait les objets principaux manipulés dans le système :

Concept réel	Classe UML correspondante
Patient de la clinique	Patient
Médecin en consultation	Docteur
Rendez-vous planifié	RendezVous
Créneau horaire	TimeSlot
Gestionnaire de planning	Scheduler
Salle de consultation	Salle

Chaque concept a été traduit en **classe d'analyse** avec ses attributs essentiels (ex. nom, date, état...).

2. Modélisation du domaine

Un **diagramme de classes UML métier** a été construit pour représenter :

- Les classes identifiées ;
- Les **relations entre elles** (association, multiplicité) ;
- Les **rôles métier** de chaque classe.

Exemple :

- Un RendezVous est associé à un Patient, un Docteur, un TimeSlot, une Salle, et potentiellement modifié par un Scheduler.

3. Ajout des attributs et associations

Chaque classe a été enrichie avec :

- des **attributs métier** pertinents (nom, email, spécialité, etc.),
- des **associations** précisant les dépendances entre classes.

4. Diagramme d'état

La classe RendezVous a été identifiée comme ayant un **comportement dynamique complexe**. Un **diagramme d'états-transitions** a été réalisé pour modéliser sa vie :

État	Événement déclencheur
Disponible	Création du créneau (TimeSlot)
Réservé	Réservation par le patient
Annulé	Annulation par patient ou scheduler
Reporté	Report par la patiente ou scheduler
Terminé	Visite effectuée, commentaire ajouté

Cela permet de **vérifier les transitions possibles** et de **contrôler la validité du système**.

5. Identification des classes participantes

Pour chaque **cas d'utilisation retenu**, les **classes participantes** ont été listées.

Exemple pour : Réserver un rendez-vous

- Patient : acteur déclencheur
- TimeSlot : créneau ciblé
- RendezVous : objet métier créé
- Scheduler : éventuellement notifié

2.5 Étape 5 : Modélisation de la navigation

Objectif de l'étape

L'objectif de cette étape est de représenter la **navigation entre les interfaces utilisateur** du système, en particulier dans le cadre d'une **application Swing**. Cela permet de simuler **l'expérience utilisateur**, en anticipant les différents **écrans, transitions et actions** accessibles selon le profil de l'utilisateur (patient, docteur, scheduler).

Même si l'approche est plus souvent associée au développement web, la **navigation dans une application Java Swing** peut être modélisée avec les **mêmes principes** : états, transitions, événements, conditions.

Démarche appliquée

1. Identification des écrans ou interfaces

- Pour chaque acteur, nous avons identifié les écrans nécessaires pour interagir avec le système.

Acteur	Interfaces principales
Patient	- Écran de connexion- Liste des créneaux disponibles- Formulaire de réservation- Historique des rendez-vous
Scheduler	- Liste des demandes- Édition / suppression d'un rendez-vous- Consultation des créneaux
Docteur	- Liste des rendez-vous à venir- Détail d'un rendez-vous- Zone de commentaire post-visite

2. Identification des actions (événements IHM)

- Chaque transition entre les interfaces est déclenchée par un **événement utilisateur** (ex. clic sur un bouton, sélection d'un rendez-vous, validation d'un formulaire).

3. Création d'un diagramme de navigation

- Représentation des interfaces comme des **états** ;
- Transitions entre interfaces associées à des **actions explicites** ;
- Certaines transitions conditionnelles (ex. accès restreint au rôle, validation préalable...).

Résultat obtenu

Un **diagramme d'état de navigation** a été réalisé (ou peut l'être graphiquement) représentant :

Exemple de navigation pour un patient :

[Menu Principal]

↓ "Réserver un RDV"

[Liste des créneaux]

↓ "Choisir un créneau"

[Formulaire réservation]

↓ "Valider"

[Confirmation] → retour au menu

Exemple pour le docteur :

[Accueil Médecin]

↓ "Voir mes rendez-vous"

[Liste RDV] → [Détail RDV] → "Ajouter commentaire" → [Validation]

Exemple pour le scheduler :

[Accueil Scheduler]

↓ "Gérer les rendez-vous"

[Liste des rendez-vous]

↓ "Modifier" / "Annuler"

[Formulaire modification]

Partie III – Application des étapes de conception et du code réalisé

3.1 Architecture technique de l'application

Objectif

L'objectif de cette étape est de **transformer les modèles d'analyse** (issus de la phase précédente) en une **architecture logicielle concrète**, structurée, maintenable et extensible. Cette architecture servira de socle pour la réalisation effective du système.

Architecture choisie : MVC + DAO

Le système a été construit selon une architecture en **couches**, suivant le modèle **MVC (Modèle-Vue-Contrôleur)** enrichi par une couche **DAO (Data Access Object)**.

Cette séparation permet :

- une meilleure organisation du code,

- une indépendance entre l'interface et la logique métier,
- une facilité de maintenance et d'évolution.

Couches principales de l'application

1. Modèle (Model)

Contient les **entités métier** directement issues du diagramme de classes d'analyse :

- Patient, Docteur, Scheduler, RendezVous, TimeSlot, Salle

Chaque classe contient :

- des **attributs métier** (nom, date, spécialité, etc.)
- des **méthodes simples** : getters, setters, méthodes d'état

2. Vue (View)

Interface utilisateur **développée avec Java Swing**.

Chaque rôle (acteur) possède une interface propre :

- PatientView : consultation des créneaux, formulaire de réservation
- DoctorView : liste des rendez-vous, zone de commentaire
- SchedulerView : gestion de la file d'attente, modification/suppression

Ces vues sont construites à l'aide de composants Swing (JFrame, JPanel, JButton, etc.).

3. Contrôleur (Controller)

Composants intermédiaires entre les vues et les services métier.

Responsables de :

- récupérer les **actions utilisateur** (ex : clic, saisie),
- valider les données si besoin,
- appeler les services métier appropriés.

Exemples :

- PatientController, DoctorController, SchedulerController

4. Service

C'est la **couche métier**, où se situe la **logique applicative** (vérification des créneaux, changement d'état, règles de gestion).

Exemples :

- AppointmentService : réserver, annuler, terminer un rendez-vous
- PatientService, DoctorService : appels spécifiques liés aux rôles

5. DAO (Data Access Object)

Couche responsable de **l'interaction avec la base de données**, via JDBC.

Chaque entité persistante possède un DAO correspondant :

- PatientDAO, DoctorDAO, AppointmentDAO, TimeSlotDAO

Chaque DAO propose des méthodes CRUD :

- findById, findAll, save, update, delete



Schéma des dépendances (simplifié)

[Vue Swing]



[Contrôleur]



[Service Métier]



[DAO (JDBC)]



[Base de données MySQL]



Avantages de cette architecture

- **Modularité** : chaque composant est indépendant des autres
- **Testabilité** : on peut tester les services sans interface
- **Réutilisabilité** : les DAO et services peuvent servir à d'autres interfaces (ex: Web)
- **Facilité de maintenance** : un bug dans la vue ne touche pas la base, etc.

3.2 Conception détaillée par itération

Itération 1 : Réserver un rendez-vous

Objectif de l'itération

Implémenter le cas d'utilisation prioritaire permettant à un **patient** de :

- consulter les créneaux disponibles,
- choisir un créneau,
- réserver un rendez-vous.

Ce cas est fondamental pour tester l'ossature complète du système : **vue → contrôleur → service → DAO → base de données**.

Classes impliquées

Couche	Classes utilisées
Vue	PatientView (interface Swing)
Contrôleur	PatientController
Service	AppointmentService
Modèle	RendezVous, Patient, TimeSlot
DAO	AppointmentDAO, TimeSlotDAO, PatientDAO

Scénario de séquence technique (boîte blanche)

Voici les **étapes de l'interaction complète**, sous forme de séquence technique :

1. PatientView affiche la liste des TimeSlot disponibles via TimeSlotDAO.findAvailable().
2. Le patient sélectionne un créneau et clique sur "Réserver".
3. PatientController récupère le créneau sélectionné.
4. Il appelle AppointmentService.bookRendezVous(patientId, slotId).
5. Le service vérifie la disponibilité du TimeSlot.
6. Il crée un objet RendezVous, l'associe au patient et au créneau.
7. Il appelle AppointmentDAO.save(rendezVous).
8. TimeSlotDAO.updateAvailability(slotId, false) est appelé pour bloquer le créneau.
9. Confirmation envoyée à la vue.

Itération 2 : Gérer un rendez-vous (Scheduler)



Objectif de l'itération

Implémenter les fonctionnalités destinées au **scheduler**, c'est-à-dire la personne chargée de :

- **modifier la date/heure** d'un rendez-vous existant,
- **annuler** un rendez-vous en cas de problème ou d'indisponibilité.

Ces opérations modifient l'état d'un objet RendezVous et nécessitent la **mise à jour du créneau associé** (TimeSlot).



Classes impliquées

Couche	Classes utilisées
Vue	SchedulerView (liste des rendez-vous)
Contrôleur	SchedulerController
Service	AppointmentService
Modèle	RendezVous, TimeSlot, Scheduler
DAO	AppointmentDAO, TimeSlotDAO

Scénario 1 : Modifier un rendez-vous

1. Le SchedulerView affiche tous les rendez-vous.
2. L'utilisateur sélectionne un rendez-vous et un nouveau créneau.
3. SchedulerController appelle AppointmentService.modifyRendezVous(idRdv, newSlotId).
4. Le service :
 - vérifie la disponibilité du nouveau TimeSlot,
 - met à jour l'objet RendezVous (champ slot),
 - libère l'ancien créneau,
 - bloque le nouveau créneau.
5. Les DAO AppointmentDAO et TimeSlotDAO assurent la persistance.

Scénario 2 : Annuler un rendez-vous

1. Le scheduler sélectionne un rendez-vous à annuler.
2. SchedulerController appelle AppointmentService.cancelRendezVous(idRdv).
3. Le service :
 - passe le state du rendez-vous à ANNULÉ,
 - libère le TimeSlot associé.
4. Mise à jour effectuée via les DAO.

Diagramme d'état du RendezVous

[Disponible]

↓ (réservation)

[Réservé]

↓ (annulation)

[Annulé]

↓ (replanification)

[Réservé] (nouveau créneau)

↓ (consultation)

[Terminé]

Itération 3 : Ajouter un commentaire (Docteur)

Objectif de l'itération

Permettre au **docteur** de :

- consulter la liste de ses rendez-vous à venir,
- accéder aux détails d'un rendez-vous,
- ajouter un **commentaire post-visite**, une fois le rendez-vous terminé.

Cette fonctionnalité complète le cycle de vie du RendezVous en permettant au médecin de **laisser une trace écrite** de la consultation.

Classes impliquées

Couche	Classes utilisées
Vue	DoctorView (liste et détail des rendez-vous)
Contrôleur	DoctorController
Service	DoctorService, AppointmentService
Modèle	RendezVous, Docteur
DAO	AppointmentDAO

Scénario de séquence : Ajouter un commentaire

1. DoctorView affiche la liste des rendez-vous du médecin connecté.
2. Le médecin sélectionne un rendez-vous terminé.

3. Une zone de texte permet de saisir un commentaire.
4. En cliquant sur "Valider", DoctorController appelle DoctorService.addComment(rdvId, texte).
5. Le service vérifie que le rendez-vous est terminé (ou force le passage à l'état TERMINÉ si prévu).
6. Le commentaire est enregistré dans le champ comment de l'objet RendezVous.
7. AppointmentDAO.update() applique la mise à jour en base.

Diagramme d'état (mis à jour)

[Réservé]

↓ (consultation + commentaire)

[Terminé] ← — ajout du commentaire

3.3 Diagramme de classes de conception

Objectif de l'étape

L'objectif est de représenter, de façon détaillée et technique, **l'architecture orientée objet** de l'application telle qu'elle sera **implémentée en Java**.

Contrairement au diagramme métier de l'analyse, celui-ci **intègre les méthodes, les types de données, et les relations concrètes** entre les classes réparties selon les **couches de l'architecture MVC + DAO**.

Démarche appliquée

1. Reprendre les classes du **modèle d'analyse** (Patient, RendezVous, TimeSlot, etc.).
2. Ajouter :
 - les méthodes métier utiles (getters/setters, méthodes de service),
 - les **interfaces DAO** et leurs implémentations,
 - les **services métiers** (logique d'application),
 - les **contrôleurs** (liaison entre vue et service),
 - les **vues Swing** associées à chaque acteur.

3. Respecter les règles de visibilité :
 - + public, - private, # protected
4. Organiser les dépendances :
 - Vue → Contrôleur → Service → DAO → BDD

Contenu du diagramme de classes de conception

1. Classes du modèle (entités métier)

- Patient : id, nom, email, dateNaissance
- Docteur : id, nom, specialite
- Scheduler : id, nom
- RendezVous : id, state, comment, date, patient, docteur, timeSlot
- TimeSlot : start, end, isAvailable
- Salle : id, numero

2. DAO (accès aux données)

- PatientDAO, AppointmentDAO, TimeSlotDAO, etc.
 - Méthodes : findById(), findAll(), save(), update(), delete()

3. Services métiers

- AppointmentService
 - bookRendezVous()
 - cancelRendezVous()
 - modifyRendezVous()
- DoctorService
 - addComment()
- PatientService

✓ 4. Contrôleurs

- PatientController
- DoctorController
- SchedulerController

Responsables de recevoir les actions de l'IHM, appeler les services et mettre à jour les vues.

✓ 5. Vues Swing

- PatientView
- DoctorView
- SchedulerView

Composées de JFrame, JPanel, JButton, JTable, etc.

Elles affichent les données du modèle, sans logique métier.

Exemple de dépendances (entre couches)

PatientView



PatientController



AppointmentService



AppointmentDAO



Base de données

✓ Résultats obtenus

- **Structure technique complète et cohérente**, prête pour l'implémentation.
- Application entièrement **découplée** : vue, logique métier, persistance.
- Chaque classe a une **responsabilité unique** (principe SOLID).
- Le diagramme sert de **référence principale** pour les phases de codage et de tests.

3.4 Difficultés rencontrées et ajustements

Objectif de l'étape

Cette partie vise à présenter de manière honnête et constructive les **obstacles rencontrés** durant la conception et la réalisation du projet, ainsi que les **solutions apportées** pour les surmonter. Cela témoigne d'une capacité à appliquer la démarche UML de façon agile et réaliste.

Principales difficultés rencontrées

1. Surcharge de la classe RendezVous

Au début du projet, la classe RendezVous concentrait toutes les relations (patient, médecin, salle, créneau, état, commentaire, etc.), ce qui la rendait difficile à gérer et à maintenir.

✓ **Solution** : clarification du rôle des entités :

- Ajout d'un lien clair vers TimeSlot comme entité autonome ;
- Utilisation de services pour déléguer la logique métier.

2. Modélisation dynamique complexe

Il a été difficile d'identifier les classes qui nécessitaient un **diagramme d'états**, car toutes ne présentaient pas un comportement évolutif pertinent.

✓ **Solution** : seul RendezVous a été retenu comme classe à comportement dynamique justifiant un diagramme d'état.

3. Navigation dans l'interface utilisateur (Swing)

La gestion de plusieurs vues Swing et leurs transitions n'était pas évidente à modéliser (surtout en l'absence d'un framework web ou de routes).

✓ **Solution** : modélisation de la navigation via un **diagramme d'état de navigation** adapté à Swing, avec des JFrame et JPanel comme "états" visuels.

4. Couplage entre les couches

Des erreurs de conception sont apparues lorsque certaines vues accédaient directement aux modèles ou DAO sans passer par les contrôleurs ou services.

✓ **Solution** :

- Renforcement du découplage entre les couches ;
- Respect strict de l'architecture MVC : Vue → Contrôleur → Service → DAO.

5. Intégration de la persistance (JDBC)

La création manuelle des requêtes SQL via JDBC et la gestion des exceptions ont nécessité plus de temps que prévu.

✓ **Solution** :

- Factorisation du code DAO (classe utilitaire pour la connexion) ;
- Test unitaire de chaque méthode DAO indépendamment.

✓ Enseignements et ajustements







- L'analyse UML a permis d'**anticiper les problèmes** de conception avant le codage.
- Le passage par une **phase d'analyse bien structurée** a facilité la répartition claire des responsabilités.
- Les difficultés ont été résolues par des **ajustements progressifs** sans remettre en cause l'ensemble de l'architecture.

3.5 Résultat fonctionnel

Objectif de l'étape

Cette section présente les **fonctionnalités réellement implémentées** lors du projet, en lien avec les cas d'utilisation choisis, et fait le point sur l'**état d'achèvement du système**. Elle permet aussi de vérifier que les objectifs initiaux sont atteints.

Fonctionnalités implémentées

Cas d'utilisation	Statut	Remarques
Réserver un rendez-vous (Patient)	 Terminé	Créneau sélectionné et bloqué automatiquement
Annuler un rendez-vous (Patient ou Scheduler)	 Terminé	Créneau libéré automatiquement
Ajouter un commentaire (Docteur)	 Terminé	Accessible uniquement après la visite
Modifier un rendez-vous (Scheduler)	 En cours (partiel)	Fonctionnalité codée mais pas encore testée
Visualiser les rendez-vous (Docteur)	 Terminé	Liste des rendez-vous affichée
Gérer les créneaux (Scheduler)	 Terminé	Affichage, suppression, libération

Aperçu technique

- L'application est **fonctionnelle en console et en Swing**.
- Elle repose sur :
 - une **structure MVC + DAO** solide,
 - une **connexion JDBC** à une base de données MySQL,
 - un découpage propre en **couches et packages**.

Captures (à intégrer dans le rapport)

- Formulaire de réservation d'un rendez-vous (vue Swing)

- Affichage des rendez-vous pour un médecin
- Modification ou annulation par le scheduler
- Exemple de commentaire ajouté à un rendez-vous terminé

Conformité au cahier des charges

Les objectifs fonctionnels du **mini cahier des charges** ont été **globalement atteints** :

Acteur	Besoins initiaux	Réponse apportée
Patient	Réserver, consulter, annuler	✅ Implémenté
Docteur	Consulter et commenter	✅ Implémenté
Scheduler	Gérer et modifier les rendez-vous	✅ Implémenté (modification en test)

Possibilités d'évolution

- Ajout d'un système d'**authentification** (par rôle)
- Export des rendez-vous au format PDF
- Envoi de **notifications automatiques** (mail, SMS)
- Interface web via JavaFX ou Spring Boot
- Meilleure gestion des erreurs (messages utilisateur + logs)

Conclusion générale

Ce projet de gestion des rendez-vous médicaux dans une clinique a permis de mettre en pratique une démarche de modélisation rigoureuse fondée sur l'approche UML et la méthode 2TUP. En suivant une progression claire, de l'analyse fonctionnelle jusqu'à la conception technique, nous avons pu développer une application Java structurée et modulaire, intégrant une interface Swing, une logique métier centralisée, et une couche de persistance JDBC.

La première partie du travail a permis de bien comprendre les besoins métier à travers les cas d'utilisation, les diagrammes d'activité, et les interactions entre les acteurs (patients, médecins, personnel administratif). Cette phase d'analyse a fourni une base solide pour la conception.

La seconde phase a consisté à transformer ces modèles en architecture logicielle concrète. Le choix d'une architecture MVC enrichie d'une couche DAO a favorisé la clarté, la réutilisabilité et la maintenabilité du code. Les itérations de développement ont été organisées de manière progressive, en se concentrant d'abord sur les fonctionnalités essentielles : réservation, annulation et suivi de rendez-vous.

Malgré certaines difficultés rencontrées (comme le couplage excessif initial ou la gestion de la navigation dans Swing), des ajustements ont été faits tout au long du processus, illustrant l'intérêt d'une démarche itérative et modulaire.

Ce projet a donc permis de consolider des compétences clés en modélisation UML, en architecture logicielle Java, et en conception orientée objet. Il ouvre la voie à de nombreuses évolutions possibles, et constitue une base stable pour aller vers des systèmes plus complexes ou multi-plateformes.