

Institut Saint-Laurent

Enseignement Supérieur Economique de type court et de promotion sociale
Rue Saint-Laurent, 33 – 4000 Liège



UN OUTIL NUMÉRIQUE AU SERVICE DU PARCOURS OPÉRATOIRE EN CHIRURGIE DE LA CATARACTE

Vincent Vanhees

Année scolaire 2024-2025

Travail de fin d'étude présenté en vue
de l'obtention du diplôme de

Bachelier en informatique

Je tiens à remercier l'ensemble des personnes qui ont contribué, directement ou indirectement, à la réalisation de ce Travail de Fin d'Études.

Merci à l'Institut Saint-Laurent, à son corps enseignant et à sa direction pour l'encadrement offert dans le cadre de ce projet. Leur disponibilité et leur accompagnement ont permis de mener ce travail dans des conditions favorables, malgré un parcours académique partiellement réalisé dans un autre établissement.

Je remercie également les enseignants impliqués dans les unités d'intégration et de développement, pour l'encadrement technique, les échanges pertinents et les conseils apportés tout au long de l'année.

L'entreprise partenaire a joué un rôle essentiel dans la concrétisation de ce projet. Je remercie l'ensemble des intervenants pour leur disponibilité, leur implication et la confiance accordée. Cette collaboration m'a permis d'évoluer dans un cadre professionnel stimulant, en lien direct avec les réalités du métier.

Le commanditaire du projet a fourni une vision claire des besoins et des objectifs à atteindre. Ses retours réguliers ont permis d'ajuster la solution de manière efficace, en maintenant une orientation métier forte.

Je remercie également mon employeur et mes responsables hiérarchiques pour leur compréhension et la flexibilité dont ils ont fait preuve afin de faciliter l'organisation de ce travail.

Enfin, je souhaite exprimer ma reconnaissance à mes proches pour leur soutien constant, leur patience et leur présence tout au long de ce parcours.

Table des matières

1.	Introduction	2
1.1	Contexte général et justification du projet.....	2
1.2	Présentation de la société d'accueil – JORAMI.....	2
1.3	Objectifs, enjeux et méthode appliquée.....	3
1.4	Plan du travail.....	4
2.	Analyse fonctionnelle	5
2.1	Présentation du besoin et du cahier des charges.....	5
2.2	Objectifs fonctionnels du projet	6
2.3	Acteurs et rôles du système	7
2.4	Parcours patient dans le système	7
2.5	Contraintes métier et exigences spécifiques	7
2.6	Résultat attendu	8
3.	Choix technologiques et architecture de l'application EyeWeb.....	8
3.1	Objectifs techniques de la solution et contexte métier	8
3.2	Stack technologique.....	10
4.	Réalisation technique des modules d'EyeWeb.....	18
4.1	Méthodologie de développement.....	18
4.2	Gestion des rôles et des utilisateurs (Role-Based Access Control)	18
4.3	Gestion des erreurs, des performances et des logs	21
4.4	Organisation du développement : méthode agile, outils et qualité logicielle.....	24
4.5	Composants Angular : Structure, Communication et Intégration UI	43
4.6	Système de formulaires réactifs : structure, validation , interactions.....	48
4.7	Importation de Données Patients : Excel, PDF et Encodage Manuel	53
4.8	Flux de données et évolutivité du modèle relationnel	56
4.9	Évolutivité et Scalabilité.....	66
5.	Sécurité, tests et conformité RGPD.....	68

5.1	Sécurité applicative : principes et mise en œuvre.....	68
5.2	Gestion des rôles et des droits d'accès	69
5.3	Tests et validation fonctionnelle.....	69
5.4	Conformité au RGPD et protection des données.....	70
5.5	Traçabilité , logs et audit	71
6.	Conclusion.....	72
7.	Ressources	73
7.1	Technologies, outils et références techniques	73
7.2	Ophthalmologie et médecine	74
7.3	Sécurité des applications et RGPD.....	74
7.4	Bibliographie – Méthodologie Agile et SCRUM	74
8.	ANNEXES	75
8.1	Annexe I – Modèle Conceptuel de Données (MCD)	76
8.2	Annexe II - Modèle Logique de Données (MLD).....	77
8.3	Annexe III - Modèle LogiqUnified Modeling Language (UML)	78
8.4	Annexe III – Thésaurus - Dictionnaire des données.....	79

1. Introduction

1.1 Contexte général et justification du projet

L'intégration des technologies numériques dans le domaine médical connaît une accélération constante, tant en matière de gestion administrative que de suivi clinique. Dans ce contexte, les outils de planification opératoire dédiés à l'ophtalmologie restent encore largement sous-exploités, souvent dispersés entre plusieurs logiciels hétérogènes, avec des interfaces peu ergonomiques et des processus difficilement automatisables.

C'est dans cette réalité de terrain qu'est né le projet EyeWeb, une application web innovante destinée à structurer, centraliser et fiabiliser le parcours préopératoire des patients ophtalmologiques. La demande initiale a été formulée par le DR FRANÇOIS-XAVIER CRAHAY, chirurgien ophtalmologue, qui souhaitait disposer d'un outil entièrement personnalisé, intuitif et sécurisé, capable de couvrir l'ensemble des étapes menant à l'intervention chirurgicale de la cataracte.

Le projet a ainsi été conçu comme un dispositif logiciel professionnel complet, à la fois robuste techniquement et adapté aux usages du personnel médical. Il vise à optimiser la gestion des patients, à automatiser les calculs biométriques, à sécuriser les données sensibles, et à fluidifier la collaboration entre médecins, secrétaires et administrateurs.

Par ailleurs, ce travail s'inscrit dans une approche professionnalisante, en lien direct avec le programme de Bachelier en Informatique de gestion. Il a été mené dans un cadre réel, au sein de la société JORAMI, dans le cadre d'un stage intensif permettant de mettre en œuvre l'ensemble des compétences acquises en développement logiciel, conception de bases de données, sécurité, UX/UI, gestion de projet agile et documentation technique.

1.2 Présentation de la société d'accueil – JORAMI

Fondée sur des valeurs de rigueur, d'innovation et de proximité, JOMARI est une entreprise belge spécialisée dans le développement de solutions logicielles sur mesure. Située au cœur du territoire wallon, elle se positionne comme un partenaire technologique de confiance pour les professionnels de santé, les entreprises et les collectivités en quête d'outils numériques performants, évolutifs et adaptés à leurs besoins métier.

JOMARI rassemble une équipe pluridisciplinaire de développeurs, analystes, UX designers et chefs de projet, unis par une même exigence de qualité et par la volonté d'apporter des solutions pragmatiques à des problématiques concrètes. Leur démarche repose sur l'écoute active des clients, une forte culture de l'itératif, et l'intégration continue des technologies les plus pertinentes, qu'il s'agisse de frameworks modernes comme Spring Boot, Angular, ou encore de pratiques de développement agiles.

C'est dans ce contexte que s'est inscrit le projet EyeWeb, une application métier développée au sein de l'équipe JOMARI, en étroite collaboration avec le Dr FRANÇOIS-XAVIER CRAHAY, ophtalmologue. Il visait à digitaliser, sécuriser et optimiser le parcours opératoire des patients en ophtalmologie, depuis l'anamnèse¹ jusqu'à la synthèse pré-chirurgicale. L'équipe JORAMI a joué un rôle central à chaque étape du projet : cadrage des besoins, architecture logicielle, conception de la base de données, développement full-stack, intégration de composants spécifiques, validation métier, et accompagnement du client.

La culture d'entreprise chez JOMARI repose également sur la transmission, l'autonomie et le partage de savoir-faire, autant d'éléments qui ont permis de m'intégrer efficacement dans les phases de développement du projet EyeWeb au cours de mon stage. Encadré par une équipe expérimentée et bienveillante, j'ai pu m'impliquer de manière significative dans la conception et l'implémentation de modules critiques du système.

Ce projet, qui s'inscrit pleinement dans l'ADN de la société, illustre sa capacité à allier expertise technique et compréhension métier pour livrer des applications robustes, utiles, et réellement centrées sur l'utilisateur.

1.3 Objectifs, enjeux et méthode appliquée

Le projet poursuit plusieurs objectifs fonctionnels et techniques majeurs, articulés autour des axes suivants :

- a) Fiabiliser et centraliser les données patient, tout au long de leur parcours opératoire ;
- b) Permettre le calcul automatisé des implants IOL² à partir de mesures biométriques précises ;
- c) Générer des résumés opératoires clairs, validés et exploitables en contexte clinique ;

¹ L'anamnèse désigne le recueil structuré d'informations auprès du patient par le professionnel de santé. Elle comprend ses antécédents médicaux, familiaux, ses habitudes de vie et les symptômes motivant la consultation.

² Les implants IOL (IntraOcular Lens) sont des lentilles artificielles implantées dans l'œil, généralement pour remplacer le cristallin naturel lors d'une chirurgie de la cataracte. Leur puissance est calculée sur base des données biométriques du patient afin d'optimiser la vision post-opératoire.

- d) Offrir une interface responsive, utilisable tant par les secrétaires que par les médecins ;
- e) Garantir la sécurité des données de santé (conformité RGPD, authentification JWT, traçabilité) ;
- f) S'appuyer sur une architecture moderne et modulaire, évolutive vers d'autres spécialités médicales.

La méthodologie choisie repose sur le cadre Agile Scrum, avec une organisation du travail en sprints, des points réguliers avec le commanditaire, et une démarche d'amélioration continue. L'ensemble des fonctionnalités a été prototypé, validé, testé puis itéré, en suivant une logique incrémentale et collaborative.

L'architecture logicielle est basée sur un découpage frontend-backend-database, avec usage des technologies suivantes :

- Spring Boot (Java 17) pour le backend RESTful,
- Angular 17 et PrimeNG pour le frontend dynamique et ergonomique,
- PostgreSQL comme base de données relationnelle robuste,
- JWT (JSON Web Token) pour la gestion des accès et la sécurité,
- Stripe pour la gestion des paiements dans les futures versions.

Le présent document décrit la genèse, la conception, le développement et les résultats de ce projet dans un cadre réaliste et professionnalisant. Il est structuré selon une logique progressive, documentant à la fois les aspects fonctionnels, techniques et organisationnels de la solution.

1.4 Plan du travail

Le mémoire est structuré selon les axes suivants :

- a) Analyse fonctionnelle et cahier des charges : présentation du besoin métier, modélisation des cas d'utilisation, description des acteurs, analyse des flux, et planification.
- b) Choix technologiques et architecture : justification des outils, schéma d'architecture, structure des modules.
- c) Réalisation technique : implémentation du backend, du frontend, des APIs, des modèles de données et de la logique métier.
- d) Sécurité, tests et validation : intégration des mécanismes de sécurité, traçabilité, tests manuels et automatisés.

- e) Perspectives d'évolution : extensions possibles, généralisation à d'autres spécialités, intégration hospitalière.
- f) Conclusion : bilan du projet, retour d'expérience, évaluation personnelle.

2. Analyse fonctionnelle

2.1 Présentation du besoin et du cahier des charges

Le projet EyeWeb, développé pour le Docteur FRANÇOIS-XAVIER CRAHAY, répond à une problématique concrète et actuelle dans le domaine de l'ophtalmologie : l'absence d'un outil numérique unifié, ergonomique et sécurisé, capable de centraliser et fiabiliser l'ensemble des étapes du parcours préopératoire des patients atteints de cataracte.

Actuellement, la préparation à une chirurgie implique l'utilisation parallèle d'Excel, de logiciels de biométrie, de formulaires papier ou de mails internes, ce qui rend le processus fragmenté, lent, sujet aux erreurs humaines, et difficilement traçable. Ce manque d'intégration génère une charge cognitive importante pour les médecins et un risque accru de perte d'informations médicales sensibles.

L'objectif du projet est donc de concevoir une plateforme web complète, avec une interface moderne, qui facilite le travail des médecins, des secrétaires médicales et des assistantes tout en garantissant la sécurité des données de santé. Le tout doit être modulaire, extensible, et adaptable à d'autres spécialités médicales à moyen terme.

Les spécificités attendues dans le cahier des charges sont les suivantes :

- a) Centraliser les informations administratives et médicales du patient dans un dossier unique.
- b) Suivre chaque patient tout au long du parcours opératoire, de l'encodage à l'impression du résumé opératoire.
- c) Permettre aux médecins de valider les examens, interpréter les résultats biométriques et de choisir les implants adaptés.
- d) Fournir aux assistantes et secrétaires un environnement de travail simple, intuitif et collaboratif.
- e) Sécuriser l'ensemble du système à travers une authentification JWT, une gestion des rôles, et une journalisation des actions critiques.

2.2 Objectifs fonctionnels du projet

Les objectifs fonctionnels ont été définis en collaboration avec le client, sur base de plusieurs itérations agiles et de démonstrations régulières de l'outil.

Les objectifs finaux sont les suivants :

- a) Gestion des patients : création, consultation, édition, suppression d'un dossier patient, avec ses données personnelles, médicales et organisationnelles.
- b) Encodage biométrique ophtalmologique : saisie de données spécifiques comme la kératométrie³, la longueur axiale (AL)⁴, la profondeur de chambre antérieure (ACD)⁵, etc.
- c) Calculs IOL personnalisés: génération automatique de propositions d'implants via les formules ophtalmiques (Haigis⁶, SRK/T⁷, Hoffer Q⁸, Holladay⁹, etc.), avec sélection du modèle final.
- d) Résumé opératoire automatisé : génération d'un PDF structuré contenant les données essentielles validées par le médecin.
- e) Module utilisateur : distinction claire entre profils (médecins, secrétaires, administrateurs) avec des permissions spécifiques.
- f) Traçabilité avancée : historique des connexions, log des modifications et actions critiques, compatibilité avec les exigences RGPD.
- g) Interface responsive : accessibilité sur écran large ou tablette, en environnement chirurgical ou administratif.
- h) Connexion sécurisée : gestion JWT, gestion multisessions, révocation des tokens, encodage des mots de passe, vérification par code.

³ Kératométrie : mesure de la courbure de la cornée, exprimée en dioptries, essentielle pour le calcul de la puissance de l'implant intraoculaire.

⁴ Longueur axiale (AL) : distance entre la surface antérieure de la cornée et la rétine, mesurée en millimètres ; c'est un paramètre déterminant pour les formules de calcul d'implants.

⁵ Profondeur de chambre antérieure (ACD) : espace entre la face postérieure de la cornée et la face antérieure du cristallin ; elle permet d'ajuster les calculs pour certains types de lentilles.

⁶ La formule Haigis utilise trois constantes spécifiques (a_0 , a_1 , a_2) et prend en compte la longueur axiale et la profondeur de chambre antérieure pour prédire la puissance de l'implant intraoculaire.

⁷ La formule SRK/T (Sanders-Retzlaff-Kraff/Theoretical) combine des éléments théoriques et empiriques, idéale pour les yeux de longueur axiale normale à longue.

⁸ Hoffer Q est particulièrement adaptée aux yeux courts ; elle intègre des paramètres optiques fins pour améliorer la précision du calcul.

⁹ La formule Holladay repose sur des mesures de longueur axiale et de kératométrie, et peut être affinée par l'ajout d'un facteur de chirurgien personnalisé.

2.3 Acteurs et rôles du système

Acteur	Rôle dans le système
Médecin	Supervise le dossier médical, valide les examens et les calculs, génère le résumé final.
Secrétaire médicale	Crée les dossiers patients, planifie les rendez-vous, assure la saisie des données.
Assistante médicale	Complète les examens ophtalmiques et assure la préparation des données de calcul.
Administrateur	Gère les utilisateurs, les rôles, les accès, les logs, et la configuration du système.
Patient	Sujet des données traitées. Il n'accède pas directement au système.

Chaque profil est défini dans la base de données et associé à un rôle distinct, géré via une stratégie de contrôle d'accès basée sur JWT (avec Spring Security 6), garantissant à la fois la sécurité, la traçabilité et le respect des normes médicales.

2.4 Parcours patient dans le système

Le parcours numérique du patient au sein de la plateforme suit une séquence logique et complète :

- Création du dossier patient par la secrétaire.
- Planification et encodage des examens : kératométrie, biométrie, visus, etc.
- Calculs IOL personnalisés : choix de la formule, affichage des options, validation par le médecin.
- Rédaction automatique du résumé opératoire, avec génération d'un PDF sécurisé.
- Archivage, exportation et suivi du dossier dans un format exploitable.

Ce cheminement respecte les étapes réelles du parcours chirurgical et assure la cohérence des données à chaque instant, tout en laissant à chaque acteur un rôle clair et circonscrit.

2.5 Contraintes métier et exigences spécifiques

Type de contrainte	Détail
Scientifique	Implémentation rigoureuse des formules ophtalmiques ; respect des valeurs cliniques validées.
Légale / RGPD	Cryptage des données sensibles, logs de connexion, droits d'accès cloisonnés.
Ergonomique	Interface épurée, typée médical, compatible avec l'environnement opératoire.

Évolutive	Architecture modulaire permettant l'extension à d'autres domaines médicaux.
Technique	Séparation stricte frontend/backend, code typé et testé.

Ces contraintes ont été intégrées dès les premières phases du développement afin d'assurer une cohérence continue entre les exigences métier et les choix techniques (comme l'utilisation de PostgreSQL, JWT, Spring Boot 3, Angular 16, PrimeNG, etc.).

2.6 Résultat attendu

À la fin du développement, l'application EyeWeb devait fournir :

- Une prise en charge fluide du parcours opératoire.
- Une amélioration significative de la productivité pour le médecin et son personnel.
- Une réduction des erreurs de saisie, via des formulaires dynamiques et des contrôles rigoureux.
- Une architecture sécurisée, modulaire et durable, prête à évoluer.

3. Choix technologiques et architecture de l'application EyeWeb

3.1 Objectifs techniques de la solution et contexte métier

3.1.1 Contexte métier

La chirurgie de la cataracte est aujourd'hui l'une des interventions les plus fréquentes en ophtalmologie. Pourtant, son bon déroulement dépend d'un enchaînement rigoureux et précis de consultations, d'examens biométriques et de calculs ophthalmiques, souvent répartis entre plusieurs outils, logiciels spécialisés, documents papier et fichiers Excel.

Dans la pratique du DR FRANÇOIS-XAVIER CRAHAY, chirurgien ophtalmologue, ces tâches essentielles, de la création du dossier patient à la décision opératoire, étaient encore largement fragmentées, sans véritable plateforme centralisée, ce qui pouvait entraîner :

- Une perte de temps dans la gestion des dossiers.
- Une charge cognitive importante pour le personnel médical.
- Un risque élevé d'erreurs humaines dans la transmission ou la saisie des données.
- Une traçabilité insuffisante des actions critiques.

Face à ce constat, l'application EyeWeb a été pensée comme une réponse numérique complète, capable de digitaliser, structurer et sécuriser l'ensemble du parcours préopératoire du patient. Elle vise également à fluidifier la communication interprofessionnelle entre les différents acteurs de la chaîne médicale : chirurgiens, assistantes, secrétaires, et à terme, le personnel de bloc opératoire.

3.1.2 Objectifs techniques

Afin de répondre efficacement à ces enjeux métier, la solution technique retenue se devait d'être à la fois robuste, modulaire, évolutive et sécurisée. EyeWeb repose sur une architecture full-stack moderne, articulée autour d'une séparation stricte entre le frontend (Angular 16) et le backend (Spring Boot 3), avec PostgreSQL comme socle de persistance.

Les orientations techniques majeures du projet sont les suivantes :

- a) Architecture en couches (multi-tier) : séparation claire des responsabilités entre les couches Contrôleur, Service, Données et Sécurité, facilitant la maintenance et l'évolutivité.
- b) Standardisation des API RESTful : tous les échanges entre le frontend et le backend passent par des end points REST documentés, typés, et visionnables.
- c) Sécurité dès la conception (Security by Design) :
 - o Authentification basée sur JWT (stateless, traçable, révoquable).
 - o Contrôle des rôles avec Spring Security (@PreAuthorize, GrantedAuthority).
 - o Encodage des mots de passe (BCryptPasswordEncoder).
 - o Réinitialisation sécurisée avec code de vérification temporaire.
 - o Journalisation des actions critiques (création, modification, suppression).
- d) Base de données relationnelle :
 - o Modèle structuré en entités relationnelles cohérentes.
 - o Relations 1-N, N-N modélisées avec JPA/Hibernate.
 - o Prise en charge du versioning des entités via Hibernate Envers.
 - o Optimisation via indexation et requêtes spécifiques pour les cas métiers.
- e) Ergonomie et accessibilité de l'interface :
 - o Utilisation de PrimeNG et Bootstrap pour une interface responsive et intuitive.
 - o Affichage en temps réel des calculs et résultats biométriques.
 - o Navigation fluide entre les étapes du parcours patient.

- f) Technologies modernes de développement :
 - Java 17, Angular 16, TypeScript, SCSS.
 - Développement en local avec PostgreSQL, et Spring Tools Suite.
 - Utilisation de Lombok pour réduire la verbosité du code.
 - Projet prêt pour des tests unitaires et d'intégration, et pour un futur déploiement via conteneurisation (Docker).
- g) Interopérabilité et extensibilité :
 - Architecture modulaire permettant l'ajout de nouveaux modules médicaux (ex. : glaucome, chirurgie réfractive).
 - Conformité aux standards techniques facilitant un potentiel raccordement à un Dossier Patient Informatisé (DPI) dans le futur.

En résumé, EyeWeb est bien plus qu'une simple application web. C'est une plateforme médicale spécialisée, conçue pour répondre à des besoins cliniques concrets, tout en reposant sur un socle technique solide, conforme aux bonnes pratiques de l'ingénierie logicielle moderne et aux exigences réglementaires du domaine de la santé.

3.2 Stack technologique

Le choix des technologies repose sur trois piliers : stabilité, maintenabilité, et compatibilité avec le cloud ou un hébergement classique.

Couche	Technologie	Raison du choix
Frontend	Angular 17 + PrimeNG	Framework moderne, SPA, interopérable, évolutif
Backend	Spring Boot 3 (Java 17)	Framework Java standard, robuste, orienté REST
Base de données	PostgreSQL	SGBD relationnel open-source, fiable et performant
Sécurité	JWT, BCrypt	Authentification stateless, hashing sécurisé
Outils	Git, Postman	Dev et test local, gestion de version
UI/UX	SCSS, Bootstrap, PrimeNG	Design responsive, expérience utilisateur fluide

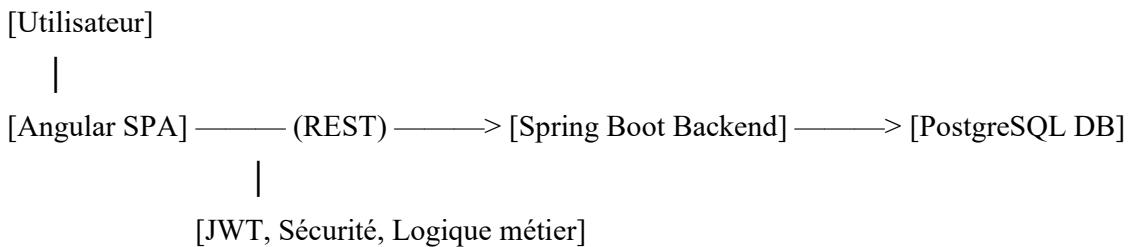
3.2.1 Architecture logicielle générale

EyeWeb adopte une architecture logicielle modulaire basée sur une séparation nette des responsabilités (Clean Architecture¹⁰) :

Frontend Angular (Single Page Application¹¹) : gestion de l'interface utilisateur et des appels API ;

- Backend Spring Boot : logique métier, sécurisation, gestion des rôles, couche d'accès aux données ;
- Base de données PostgreSQL : persistance relationnelle des entités métier ;
- API RESTful : interface de communication standardisée entre frontend et backend.

3.2.2 Schéma simplifié



3.2.3 Backend – Structure, sécurité et logique métier

3.2.3.1 Environnement et languages

- Langage principal : Java 17 (LTS)
- Framework : Spring Boot 3, prenant en charge Jakarta EE 10
- Outils de build : Apache Maven (pom.xml)
- Configuration : fichiers application.yaml, application-dev.yaml, et .env
- Gestion des dépendances : via Maven (JPA, Spring Security, JWT, PostgreSQL, MapStruct...)

¹⁰ La Clean Architecture est un modèle d'architecture logicielle proposé par Robert C. Martin. Elle repose sur une organisation en couches concentriques, où le cœur métier (domain) est indépendant des technologies extérieures (frameworks, base de données, interface utilisateur). Elle favorise la testabilité, la maintenabilité et l'évolutivité du code.

¹¹ Une Single Page Application (SPA) est une application web qui se charge en une seule page HTML et qui met à jour dynamiquement le contenu sans recharger la page entière. Cela permet une expérience utilisateur fluide, proche d'une application native, notamment grâce à l'utilisation de frameworks comme Angular ou React.

3.2.3.2 Architecture des couches

Le backend suit une structure classique en couches :

- Controller Layer : réception des requêtes via `@RestController`
- Service Layer : logique métier avec `@Service`, `@Transactional`
- Repository Layer : persistance via `JpaRepository`, requêtes `@Query`
- Entity Layer : entités `@Entity`, gestion ORM via JPA/Hibernate
- DTOs & Mappers : objets de transfert (SummaryDto, ExamDto...), mappés avec `MapStruct`
- Chaque couche est isolée et respecte le principe de séparation des responsabilités, facilitant les tests et la maintenance.

3.2.3.3 ORM et persistance

Spring Data JPA utilisé avec Hibernate pour faire le lien entre les entités Java et la base PostgreSQL :

- Relations `@OneToMany`, `@ManyToOne`
- Gestion automatique du schéma avec `ddl-auto: update`
- Indexation sur les colonnes critiques
- Chargement lazy (fetch = `FetchType.LAZY`) par défaut

3.2.3.4 Sécurité JWT avec spring security

La sécurité repose sur :

- Authentification stateless avec JWT
- Filtres personnalisés (`JwtAuthenticationFilter`)
- Gestion des rôles (`ROLE_ADMIN`, `ROLE_DOCTOR`, `ROLE_SECRETARY`, `ROLE_USER`)
- Stockage des tokens en base (`Token`, `VerificationCode`)
- Services de sécurité dédiés (`JwtServiceImpl`, `AuthenticationServiceImpl`...)

Le token est transmis dans le header `Authorization` à chaque requête, validé via le filtre JWT, et inséré dans le `SecurityContext`.

3.2.3.5 Outils complémentaires

- Lombok : annotations pour alléger le code (@Getter, @Setter, @Builder)
- Logger : journalisation des actions critiques dans les services
- Tests : intégration de JUnit 5, @SpringBootTest, testabilité assurée dans chaque couche
- CI/CD : en attente d'implémentation via GitHub Actions

3.2.4 Frontend – Angular et ergonomie

Le frontend de l'application EyeWeb repose sur Angular 16, en combinaison avec TypeScript, RxJS, SCSS, et PrimeNG. L'architecture est modulaire, orientée composants, et respecte les bonnes pratiques de développement SPA (Single Page Application).

3.2.4.1 Environnement et outils

- Langage principal : TypeScript (configuré dans tsconfig.json)
- Framework : Angular 16 avec Angular CLI
- Styles : SCSS (modulaire) + Bootstrap pour le responsive design
- Bibliothèques UI : PrimeNG (p-table, p-tree, p-dialog, etc.)
- Gestion des dépendances : package.json (Angular, RxJS, PrimeNG, Bootstrap)

3.2.4.2 Structure du projet Angular

Organisation des répertoires :

- /pages/ : vues principales routables (home, summary, calculator...)
- /components/ : composants fonctionnels (résumé, calculs, forms)
- /services/ : accès aux endpoints backend via HttpClient
- /models/ : interfaces TypeScript (Patient, Exam, etc.)
- /shared/ : composants réutilisables, pipes, configuration

Chaque entité métier (ex : Patient) dispose de son propre service, composant, et modèle.

3.2.4.3 Routing angular

- Géré via app-routing.module.ts
- Navigation dynamique par ID (ex. /patient/123/summary)
- Protection des routes via AuthGuard
- Modules chargés à la demande (lazy loading)

- Réactive forms
- Utilisation de FormGroup, FormBuilder, Validators
- Validation dynamique des champs
- Affichage d'erreurs personnalisées

3.2.4.4 Formulaires typés pour : création patient, saisie Exam, IOL, etc.

- PrimeNG : UI professionnelle

Composants principaux utilisés :

- p-table : listes d'examens, patients
- p-tree : antécédents médicaux
- p-dialog : modales d'ajout/modif
- p-toast : notifications utilisateur
- p-dropdown, p-calendar, etc.

PrimeNG permet une interface fluide, moderne et accessible (dark mode, navigation clavier).

3.2.4.5 RXJS et gestion asynchrone

- Services Angular renvoient des Observable<T>
- Utilisation d'opérateurs pipe(), map(), catchError()
- Flux unidirectionnel : le service récupère → le composant affiche
- Gestion globale des erreurs via interceptors

3.2.4.6 Sécurité frontend

- Injection automatique du token JWT dans chaque requête via HttpInterceptor
- Redirection vers le login en cas de 401
- Protection dynamique des routes

3.2.4.7 Expérience utilisateur

- Responsive design (PrimeFlex + Bootstrap)
- Toasts pour toutes les actions utilisateurs (ajout, modif, suppression)
- Formulaires réactifs fluides
- Structure modulaire et composants réutilisables

3.2.5 Base de données – PostgreSQL, normalisation et conformité RGPD

Le stockage des données dans EyeWeb repose sur PostgreSQL 15, couplé à JPA (Hibernate) pour l'ORM (Object-Relational Mapping¹²). La base a été pensée pour garantir la cohérence, la performance, la sécurité et la traçabilité des données médicales sensibles.

3.2.5.1 Choix de PostgreSQL

PostgreSQL a été retenu pour sa :

- Robustesse et fiabilité transactionnelle (ACID)
- Richesse relationnelle (relations N-N, contraintes, triggers)
- Capacité à indexer finement et à gérer des types avancés (jsonb, tsvector...)
- Compatibilité parfaite avec Spring Data JPA

3.2.5.2 Structure et relations

Entités principales :

- Patient, Exam, Calcul, Lens
- User, Token, VerificationCode (sécurité)
- Link, Appointment, Material (fonctionnalités avancées)

Relations modélisées :

1 Patient → N Exams

1 Exam → N Calculs

1 Calcul → 1 Lentille (Lens)

Les clés étrangères sont définies via @JoinColumn, avec cascade, contrainte ON DELETE SET NULL ou CASCADE selon les cas.

3.2.5.3 ORM avec JPA et Hibernate

- Chaque entité Java est annotée avec @Entity, @Table
- Relations avec @ManyToOne, @OneToMany, @OneToOne

¹² L'Object-Relational Mapping (ORM) est une technique de programmation permettant de faire le lien entre les objets d'une application (en programmation orientée objet) et les tables d'une base de données relationnelle. Elle simplifie l'interaction avec la base en manipulant des objets plutôt que des requêtes SQL directes.

- Mapping automatique des attributs via `@Column`, `@Enumerated`, `@JoinColumn`
- Chargement LAZY par défaut pour éviter les surcoûts
- Génération automatique du schéma via `ddl-auto: update` (dev uniquement)

[3.2.5.4 Audit et journalisation](#)

- Utilisation de `@Audited` (Hibernate Envers) pour conserver un historique
- Suivi de `createdAt`, `modifiedAt`, `deleted` sur toutes les entités sensibles
- Table d'audit générée automatiquement pour chaque entité annotée

Cette traçabilité répond aux exigences RGPD et au contexte médical.

[3.2.5.5 Sécurité des données](#)

- Stockage sécurisé des tokens JWT (entité `Token`)
- Révocation dynamique via champ `revoked`
- Utilisateur en base restreint (`EyeWeb_user`) avec droits contrôlés
- Données sensibles encryptées (reset password, vérification, etc.)

[3.2.5.6 Optimisations](#)

- Indexation sur les colonnes critiques : `id`, `patient_id`, `exam_id`, etc.
- Requêtes personnalisées JPQL avec `@Query` si besoin
- Utilisation de `@Where(clause = "deleted = FALSE")` pour les entités supprimées logiquement
- Partitionnement logique par entité (`Exam`, `Calcul...`)

[3.2.5.7 Migration et initialisation](#)

- Environnement de dev : création auto via Spring Boot
- Possibilité d'ajouter Flyway ou Liquibase pour la gestion des versions en production
- Pas de dépendance forte à un outil d'admin graphique

3.2.5.8 Principales forces de cette architecture

3.2.5.8.1 Une structuration efficace des données médicales

La base garantit une traçabilité complète des patients, examens, calculs biométriques et implants intraoculaires. L'organisation des données assure un accès optimisé aux informations les plus critiques.

3.2.5.8.2 Une intégrité et cohérence renforcées

Gestion robuste des relations entre les entités (via les clés étrangères et les contraintes d'intégrité).

Réduction des risques d'erreurs et garantie de la cohérence des données médicales.

3.2.5.8.3 Des performances optimales

Utilisation d'index et partitionnement pour améliorer la rapidité des requêtes SQL.

Possibilité d'implémenter des vues matérialisées pour précalculer des données complexes et accélérer leur restitution.

3.2.5.8.4 Interopérabilité et évolutivité

Compatibilité HL7/FHIR¹³ pour échanger les données avec d'autres systèmes hospitaliers.

Connexion à des APIs médicales pour enrichir les données via l'importation des équipements médicaux.

Support des calculs avancés via des algorithmes de Machine Learning pour améliorer la précision des diagnostics et des prescriptions de lentilles intraoculaires.

En conclusion, cette base de données pose les bases solides d'un système de gestion avancé pour les données médicales ophtalmologiques. Son architecture modulaire et sécurisée permet une intégration fluide avec d'autres solutions et garantit une expérience utilisateur optimale pour les médecins, secrétaires et administrateurs du système.

¹³ HL7 (Health Level Seven) et FHIR (Fast Healthcare Interoperability Resources) sont des standards internationaux permettant l'échange électronique structuré de données médicales entre applications. FHIR, développé par HL7, facilite l'interopérabilité en s'appuyant sur les technologies web modernes comme REST et JSON.

4. Réalisation technique des modules d'EyeWeb

4.1 Méthodologie de développement

Le développement d'EyeWeb s'est appuyé sur une approche agile, avec des sprints hebdomadaires, des validations régulières du produit par le commanditaire, et une amélioration continue selon les retours métier.

Chaque fonctionnalité a été développée de manière modulaire, en respectant la séparation des couches backend et frontend. Les composants critiques ont été traités en priorité : gestion des utilisateurs, dossiers patients, examens biométriques, calculs IOL, et génération du résumé opératoire.

4.2 Gestion des rôles et des utilisateurs (Role-Based Access Control)

4.2.1 Objectif du module de sécurité

- a) Le module de gestion des utilisateurs et des rôles dans EyeWeb assure :
- b) L'authentification sécurisée (inscription, connexion, mot de passe oublié)
- c) L'attribution dynamique des rôles (ADMIN, USER, SECRETARY, DOCTOR) via une table dédiée.
- d) Le contrôle d'accès aux endpoints sensibles via Spring Security et RBAC
- e) La traçabilité des sessions grâce au stockage des tokens JWT
- f) La protection du cycle de vie des comptes (création, activation, révocation, suppression)
- g) Structure technique des utilisateurs

4.2.2 Authentification et cycle de vie utilisateur

4.2.2.1 Composants backend

Composant	Rôle
AuthenticationServiceImpl	Gère login, register, logout
JwtServiceImpl	Génère et valide les tokens JWT
LogoutServiceImpl	Révoque les tokens actifs
UserDetailsService	Charge les utilisateurs pour Spring Security

TokenRepository	Stocke les JWT et leur statut (révoqué, expiré)
AuthenticationController	Expose les endpoints /auth/**

4.2.2.2 Cycle d'authentification complet

- a) Inscription (/auth/register) :

Création du compte utilisateur + attribution de rôle + génération d'un token JWT stocké en base.

- b) Connexion (/auth/authenticate) :

Vérification des identifiants → émission d'un nouveau token JWT.

- c) Utilisation du système : Le frontend Angular envoie automatiquement le JWT à chaque appel API (Authorization: Bearer <token>).

- d) Déconnexion (/auth/logout) : Révocation du token en base (revoked = true).

4.2.3 Sécurité des accès : Contrôle RBAC (Role-Based Access Control)

4.2.3.1 Définition des droits

Chaque utilisateur dispose d'un rôle unique contrôlé par Spring Security via :

- Annotations @PreAuthorize dans les contrôleurs
- Filtres dans la configuration SecurityConfig
- Vérification dynamique via SecurityContextHolder dans les services

4.2.3.2 Protection des routes (SecurityConfig)

Dans la classe SecurityConfig.java, la protection est définie comme suit :

```

@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    http // CORS : configuration personnalisée pour autoriser le frontend Angular
        .cors(cors -> cors.configurationSource(corsConfigurationSource()))
        .csrf(AbstractHttpConfigurer::disable) // CSRF désactivé car l'application est stateless (pas de sessions serveur)
        // Configuration des autorisations par route
        .authorizeHttpRequests(authorize -> {
            authorize.requestMatchers(...patterns: "/auth/**").permitAll(); // Auth public : login, register
            authorize.requestMatchers(...patterns: "/admin/**").hasRole("ADMIN"); // Accès restreint à ADMIN
            authorize.requestMatchers(...patterns: "/doctor/**").hasAnyRole(...roles: "DOCTOR", "ADMIN");
            authorize.requestMatchers(...patterns: "/secretary/**").hasAnyRole(...roles: "SECRETARY", "ADMIN");
            authorize.requestMatchers(...patterns: "/user/**").hasAnyRole(...roles: "USER", "SECRETARY", "DOCTOR", "ADMIN");
            authorize.anyRequest().authenticated(); // Toute autre requête nécessite une authentification
        })
        // Le mode stateless évite de stocker l'état utilisateur côté serveur (important pour les JWT)
        .sessionManagement(session -> session.sessionCreationPolicy(STATELESS))
        // Fournisseur d'authentification utilisé
        .authenticationProvider(authenticationProvider)
        // Ajout du filtre JWT AVANT le filtre standard d'authentification
        .addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class)
        // Configuration du logout
        .logout(logout -> logout
            .logoutUrl("/auth/logout") // URL de déconnexion
            .addLogoutHandler(logoutHandler) // Comportement personnalisé à la déconnexion
            .logoutSuccessHandler((request, response, authentication) -> SecurityContextHolder.clearContext()) // Nettoyage du contexte
        );
        // On retourne la configuration de sécurité complète
    return http.build();
}

```

Routes /auth/** : accessibles publiquement (inscription, authentification)

Routes /admin/** : accessibles uniquement aux administrateurs (ADMIN)

Routes /doctor/** : accessibles aux médecins (DOCTOR) et administrateurs (ADMIN)

Routes /secretary/** : accessibles aux secrétaires (SECRETARY) et administrateurs (ADMIN)

Routes /user/** : accessibles à tous les rôles authentifiés (USER, SECRETARY, DOCTOR, ADMIN)

4.2.3.3 Table Token pour la traçabilité

Champ	Description
token	JWT émis
revoked	Statut de révocation
expired	Statut d'expiration
user	Lien vers l'utilisateur propriétaire

La table assure une traçabilité complète des sessions actives et permet la révocation immédiate en cas de besoin.

4.2.3.4 Gestion des rôles et sessions côté Frontend Angular

Dans l'application EyeWeb, le rôle utilisateur est intégré dans le token JWT (via un claim role) et décodé côté Angular à l'aide de la bibliothèque jwt-decode.

L'injection automatique du token dans les appels HTTP est assurée via un HttpInterceptor, et le rôle de l'utilisateur est extrait et exploité dans le service AuthService.

Exemple (extrait de auth.service.ts) :

```
getUserRole(): string {
  const token = sessionStorage.getItem('token');
  if (!token) return '';
  const decoded = jwtDecode(token.split(' ')[1]) as User;
  return decoded.role.name;
}

isAdmin(): boolean {
  return this.getUserRole() === 'ADMIN';
}
```

Ainsi, le frontend Angular applique dynamiquement des restrictions d'accès ou des comportements d'affichage selon le rôle de l'utilisateur connecté.

4.2.3.5 Avantages de cette architecture RBAC

- a) Sécurité renforcée : accès strictement contrôlé selon les rôles
- b) Évolutivité : ajout de nouveaux rôles/fonctionnalités facilité
- c) Traçabilité : historique des sessions JWT complet
- d) Centralisation : règles de sécurité concentrées dans Spring Security

4.3 Gestion des erreurs, des performances et des logs

L'application EyeWeb met en œuvre une stratégie robuste de gestion des erreurs et de journalisation, assurant stabilité, lisibilité et traçabilité des incidents survenant dans le système.

4.3.1 Gestion centralisée des erreurs dans le backend

L'API REST du backend utilise un système global de capture d'exceptions grâce à la classe `@GlobalExceptionHandler`, annotée avec `@RestControllerAdvice`. Ce composant intercepte

automatiquement toutes les exceptions levées dans l'application et retourne une réponse HTTP structurée, en journalisant l'événement via la bibliothèque org.jboss.logging.Logger.

La logique repose sur une gestion fine par type d'erreur, avec un code HTTP adapté :

Code	Exception interceptée	Signification
400	MethodArgumentNotValidException, IllegalArgumentException, IOException, MethodArgumentTypeMismatchException	Requête mal formée ou invalidité des données
401	BadCredentialsException, ResponseStatusException	Échec d'authentification
403	AccessDeniedException	Accès refusé
404	NoSuchElementException	Ressource non trouvée
409	OptimisticLockingFailureException	Conflit de modification concurrente
500	RuntimeException, Exception	Erreur interne inattendue

4.3.2 Format des erreurs : DTO personnalisés

Les erreurs retournées vers le frontend sont encapsulées dans des objets Java simples, comme ErrorResponse. Ce DTO contient typiquement :

```
//404 NOT FOUND-----  
/**  
 * Indique que l'objet recherché n'existe pas.  
 * @param e L'identifiant de l'objet recherché.  
 * @return Un message d'erreur et le code HTTP 404.  
 */  
no usages  
@ExceptionHandler(NoSuchElementException.class)  
public ResponseEntity<?> handleNoSuchElementException(NoSuchElementException e) {  
    //TODO: Nouvelle manière de faire?  
    LOGGER.error( message: HttpStatus.NOT_FOUND + " - " + e.getMessage(), e);  
    return ResponseEntity.status(HttpStatus.NOT_FOUND)  
        .body(new ErrorResponse(HttpStatus.NOT_FOUND.value(), e.getMessage()));  
}
```

statusCode : le code HTTP de l'erreur

message : une description lisible de l'erreur

éventuellement une map de champs invalides pour les erreurs de validation

Cela permet au frontend d'afficher dynamiquement des messages clairs pour l'utilisateur.

4.3.3 Journalisation avec SLF4J & Logback

Le projet utilise SLF4J¹⁴ comme interface de logging, avec Logback comme implémentation concrète. Tous les événements notables (erreurs, exceptions métier, conflits de ressources, accès non autorisés, etc.) sont enregistrés via le logger statique :

La granularité des logs peut être définie via application.yaml ou un fichier logback-spring.xml,

```
private final static Logger LOGGER = Logger.getLogger(GlobalExceptionHandler.class);
```

permettant de filtrer les logs par environnement (dev, prod) ou par niveau (DEBUG, INFO, ERROR...).

4.3.4 Logs de sécurité et audit

Des événements critiques de sécurité sont systématiquement journalisés, notamment :

- les tentatives d'authentification échouées (BadCredentialsException)
- Les accès non autorisés (AccessDeniedException)
- les actions sensibles (création, modification ou suppression de données médicales)

Le tout est géré automatiquement par la combinaison de filtres Spring Security, des contrôleurs REST et du GlobalExceptionHandler.

4.3.5 Pratiques de performance backend

L'optimisation des performances a été prise en compte dès la phase de développement via :

- l'usage ciblé de @Transactional(readOnly = true) pour toutes les opérations en lecture simple, améliorant le rendement des transactions
- l'utilisation de Pageable dans les endpoints de type liste pour limiter la charge mémoire et favoriser la pagination dynamique côté client
- l'indexation des colonnes clés dans PostgreSQL (telles que : id, patient_id, exam_id, role_id), optimisant la vitesse des requêtes

¹⁴ SLF4J (Simple Logging Facade for Java) est une façade de journalisation qui permet d'abstraire l'implémentation du logging. Elle offre une interface unique pour différentes bibliothèques de log comme Logback, Log4j ou java.util.logging.

Bien qu'aucun outil de monitoring (type Prometheus, Grafana ou Actuator) ne soit encore intégré, l'application respecte les bonnes pratiques de performance sur Spring Boot et PostgreSQL.

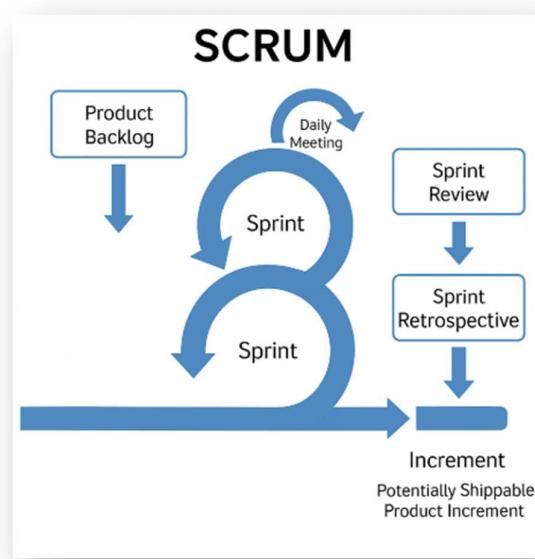
4.4 Organisation du développement : méthode agile, outils et qualité logicielle

4.4.1 Méthodologie Agile et Sprints SCRUM

4.4.1.1 Cadre méthodologique appliqué

Le développement d'EyeWeb s'est inscrit dans une méthodologie agile SCRUM adaptée. Ce cadre itératif et incrémental s'est avéré particulièrement adapté à la nature évolutive du projet, impliquant des ajustements fréquents et une interaction constante avec le client final, le DR FRANÇOIS-XAVIER CRAHAY. Chaque sprint hebdomadaire visait à livrer une version opérationnelle de l'application, testée et validée lors des réunions de sprint review.

Le suivi du projet a été rythmé par :



- 1 à 2 daily sprints hebdomadaires (30 minutes avec l'équipe JORAMI ou en autonomie) ;
- 1 sprint review chaque samedi avec le client pour démonstration, retours métier et réajustement du backlog.

4.4.1.2 Daily Scrum & Sprint Planning

Chaque semaine débutait par une priorisation des tâches lors d'un daily meeting (souvent le lundi), au cours duquel les modules à implémenter ou à améliorer étaient définis. Le backlog, géré via Trello, était structuré en colonnes (À faire / En cours / À valider / Terminé). Ces échanges ont permis de fluidifier les développements, d'anticiper les blocages techniques et de s'adapter en temps réel aux remarques du médecin.

Le sprint planning était fusionné au besoin avec le daily du début de semaine, compte tenu de la petite taille de l'équipe. Il définissait les objectifs de la semaine, les entités à modéliser, les composants à coder et les priorités techniques ou cliniques.

4.4.1.3 Sprint Review & Présentation des Avancées

Chaque samedi, une réunion de revue de sprint était organisée avec le commanditaire, permettant de présenter les développements achevés, de recueillir des retours cliniques précis, et de prioriser les fonctionnalités pour le sprint suivant. Ces sessions ont permis :

- d'ajuster la logique métier des calculs ophtalmiques,
- d'affiner l'ergonomie des interfaces (summary, patient-form, calculator),
- de valider ou corriger la structuration des données (OD/OS¹⁵, implants, etc.).

Ces rendez-vous hebdomadaires ont garanti un alignement constant entre les exigences fonctionnelles du métier et la réalité technique de l'implémentation.

4.4.2 Réalité des Sprints (du 17 février au 12 mai 2025)

Le projet a été découpé en 13 sprints hebdomadaires réels, chacun documenté dans un journal de bord intégré au TFE. Les grandes étapes de développement ont été :

- Semaine 1 (17–23/02) : refonte complète de l'analyse fonctionnelle et restructuration du backend et frontend.
- Semaine 2 : mise en place de la sécurité (JWT, rôles), premières pages Angular.
- Semaine 3 : logique métier des examens et calculs biométriques.
- Semaine 4-5 : gestion complète des patients avec recherche dynamique.
- Semaine 6-7 : résumé opératoire, simulation des implants, refactoring.
- Semaine 8 : finalisation du front-back et du module résumé.
- Semaine 9-12 : développement avancé du module secrétaire et du planning.
- Semaine 13-14 : relecture, finalisation technique, simulation de la soutenance.

Chaque sprint a été clos par une démo concrète (live ou vidéo), validée par le médecin.

¹⁵ OD (*oculus dexter*) désigne l'œil droit, et OS (*oculus sinister*) désigne l'œil gauche. Ces abréviations sont issues du latin et couramment utilisées en ophtalmologie pour différencier les deux yeux.

4.4.2.1 Phase de Prototypage et Analyse (Semaine 1)

La première semaine a été consacrée à une phase de co-construction du besoin, intégrant :

- Des entretiens approfondis avec le client pour formaliser le parcours patient.
- La rédaction d'un cahier des charges fonctionnel orienté ophtalmologie.
- La création d'un prototype basse fidélité sur Figma¹⁶, rapidement transposé en HTML/SCSS pour validation ergonomique.

Cette phase fondatrice a permis d'ancrer le projet dans un cycle itératif solide.

4.4.2.2 Construction de la Base de Données et Sécurité (Semaine 2)

Une modélisation relationnelle adaptée à PostgreSQL a été mise en place, couvrant les entités : Patient, Exam, Calcul, Lens, User, Token.

En parallèle, la sécurité applicative a été configurée :

Intégration de Spring Security 6,

Authentification JWT,

Encodage des mots de passe,

Gestion stricte des rôles utilisateurs (ADMIN, DOCTOR SECRETARY).

4.4.2.3 Développement UI/UX et Intégration PrimeNG (Semaine 3)

Cette étape a permis de construire les premières interfaces Angular :

Formulaires patients,

Tableaux d'examens,

Résumé opératoire avec composants PrimeNG (p-table, p-dialog, p-tree, etc.).

Les tests manuels ont été réalisés à chaque incrément, avec une logique OD/OS réactive.

4.4.2.4 Tests, Refactoring et Retours Client (Semaines 4 à 14)

À partir de la semaine 4, le projet EyeWeb est entré dans une phase de montée en puissance fonctionnelle. Chaque sprint hebdomadaire était orienté vers l'enrichissement métier, l'amélioration de l'expérience utilisateur et la stabilisation de la plateforme. L'enchaînement suivant décrit les évolutions majeures sprint par sprint.

¹⁶ Figma est un outil de conception d'interfaces (UI/UX) basé sur le cloud, qui permet de créer des maquettes interactives collaboratives, en temps réel, sans installation locale.

Semaine 4 – 5 : Finalisation du module patient et recherche dynamique

Ces deux semaines ont permis de :

Finaliser toutes les fonctionnalités CRUD pour la gestion des patients ;

Implémenter une recherche dynamique en direct sur les champs nom, prénom, date de naissance ;

Afficher les résultats dans des tableaux filtrables (p-table) avec déclenchement à la frappe (startsWith) ;

Gérer proprement les messages d'erreurs et validations côté formulaire ;

Étendre les tests unitaires sur les services PatientService et UserService.

Semaine 6 – 7 : Calculs ophtalmiques, logique métier et résumé opératoire

Durant cette phase centrale :

Les formules biométriques (SRK/T, Haigis, Hoffer Q) ont été testées et validées par le DR CRAHAY ;

Le service de calcul (CataractServiceImpl) a été enrichi pour traiter les deux yeux indépendamment ;

Le composant Résumé opératoire a été finalisé, incluant tous les blocs OD/OS, implants recommandés, historique ;

Un export PDF a été simulé et validé pour archivage futur ;

Le routing Angular a été structuré pour garantir la navigation fluide.

Semaine 8 – 9 : Consolidation UI, validations métier, internationalisation

Les sprints suivants ont permis :

La consolidation de l'interface utilisateur, avec des feedbacks visuels lors des actions sensibles (loading, confirmation, erreur) ;

L'intégration des validations métier spécifiques, comme les seuils de longueur axiale, les incompatibilités de formules, ou les erreurs de saisie ;

L'uniformisation des messages, labels et placeholders dans toute l'interface.

Semaine 10 – 11 : Module secrétaire et planning opératoire

Ces sprints ont été consacrés à :

La création du module de planification des examens et interventions, avec un calendrier interactif ;

L'implémentation d'une gestion complète des rendez-vous, affichables et modifiables par date et par médecin ;

Le développement du composant consultation pour créer une trace médicale structurée avant opération ;

La navigation entre les modules a été rendue contextuelle : le système conserve l'état du patient sélectionné.

Semaine 12 – 13 : Finalisation, optimisation, corrections

En phase de finition :

Une revue de code complète a été effectuée : renommage des fichiers, nettoyage, factorisation des blocs communs ;

Un effort a été porté sur la centralisation des services Angular (patient.service.ts, exam.service.ts, summary.service.ts, etc.) ;

Des corrections ont été apportées sur :

la gestion des droits (filtrage UI en fonction des rôles) ;

les transitions entre composants ;

les erreurs de casting côté DTOs backend.

Un tableau récapitulatif des anomalies résolues a été dressé sur base des retours hebdomadaires du DR CRAHAY.

Semaine 14 : Documentation, support utilisateur et simulation de soutenance

La dernière semaine a permis de :

Finaliser la documentation technique intégrée dans le TFE (API REST, base de données, flux métier, sécurité) ;

Créer un guide utilisateur simplifié avec captures d'écran et scénarios d'usage ;

Simuler la soutenance complète devant un tiers technique, avec déroulé du parcours patient, démonstration live, et réponses aux questions critiques ;

Organiser les dossiers de livraison : code source, base PostgreSQL, livrables PDF, export JSON de configuration, vidéo de démonstration.

4.4.2.5 Communication asynchrone

En l'absence de réunions physiques régulières, la communication s'est effectuée via :
GitHub (commentaires de commit),
Discord (questions rapides),
Teams (revues de code ponctuelles).
Ce dispositif a permis de respecter les deadlines du sprint tout en conservant une flexibilité de travail.

4.4.3 Outils DevOps et CI/CD

4.4.3.1 Github et gestion du code source

Le projet a été découpé en deux dépôts GitHub séparés :
un dépôt privé pour le backend Spring Boot,
un dépôt pour le frontend Angular.
Chacun possédait ses propres branches (main, dev, features/...) avec gestion des pull requests.
Une politique de merge avec revue obligatoire a été mise en place à partir du sprint 3.

4.4.3.2 CI/CD (Continuous integration / Delivery)

Un pipeline de CI/CD GitHub Actions a été amorcé, avec pour objectifs à terme :
Lancement automatique des tests unitaires à chaque push sur main,
Vérification du build Angular (ng build --prod),
Déploiement automatisé sur un serveur de préproduction.

4.4.3.3 Bonnes pratiques mises en place

Conventions de commit (Angular Commit Message Convention),
Analyse de code statique via ESLint (frontend) et SonarLint (backend),
Validation croisée des fonctionnalités lors des merge requests.

4.4.4 Tests et validation

4.4.4.1 Testes unitaires

Des tests unitaires JUnit ont été implémentés pour les services critiques du backend :

AuthenticationServiceImpl

CataractServiceImpl

SummaryService

Ces tests ont couvert des scénarios métier (calcul SRK/T, récupération du patient, etc.) avec des mocks pour isoler les dépendances.

4.4.4.2 Tests manuels et validation fonctionnelle

Chaque fonctionnalité validée en sprint a été testée manuellement par l'équipe :

Vérification du cycle complet "Patient → Exam → Calcul → Résumé"

Tests en conditions réelles avec les données du Dr Crahay

Retours hebdomadaires pour ajustements UX ou logique métier

4.4.4.3 Tests sur le frontend

Les formulaires ont été vérifiés avec des cas limites (champs vides, formats invalides).

Les composants Angular ont été testés pour leur réactivité (Reactive Forms, comportement des modales, navigation sécurisée).

4.4.4.4 Stratégie de non régression

Un plan de test évolutif a été mis en place dans Trello pour prévenir les régressions sur les modules critiques (cal-culs, résumé opératoire, utilisateurs).

4.4.5 Système de routing Angular dans eyeweb

Le système de routing Angular est au cœur de la navigation de l'application EyeWeb, garantissant une expérience fluide et modulaire propre aux applications SPA (Single Page Application).

Il permet de naviguer entre les vues sans rechargement complet de la page, tout en assurant la sécurité via des guards et une organisation modulaire optimisée.

4.4.5.1 Structure générale du routing

La configuration principale est définie dans le fichier AppRoutingModule, qui centralise :

- Les routes principales de l'application
- Le lazy loading des modules métiers
- La protection des routes par AuthGuard et AccessGuard
- Le support des routes imbriquées avec AppLayoutComponent comme composant parent

4.4.5.2 Configuration principale (app-routing.module.ts)

Le fichier app-routing.module.ts centralise la configuration des routes principales :

```
const routes: Routes = [
  {
    path: '',
    component: AppLayoutComponent,
    canActivate: [AuthGuard],
    children: [
      {
        path: '',
        loadChildren: () =>
          import('./body/components/dashboard/dashboard.module').then(
            (m) => m.DashboardModule
          ),
        canActivate: [AuthGuard],
        data: { breadcrumb: 'Home' }
      },
      ...
    ],
    {
      path: 'auth',
      data: { breadcrumb: 'Auth' },
      loadChildren: () =>
        import('./body/components/auth/auth.module').then(
          (m) => m.AuthModule
        ),
    },
    {
      path: 'notfound',
      loadChildren: () =>
        import('./body/components/notfound/notfound.module').then(
          (m) => m.NotFoundModule
        ),
    },
    { path: '**', redirectTo: '/notfound' },
  ];
]
```

4.4.5.3 Routage par modules fonctionnels

EyeWeb repose sur un découpage modulaire strict. Chaque module fonctionnel (ex : Patient, Cataract, Summary) possède :

Un *.module.ts pour ses déclarations

Un *.routing.module.ts dédié à sa propre configuration de routes internes

Cela permet :

Une meilleure maintenabilité

Une séparation claire des responsabilités

Le lazy loading automatique à la navigation

4.4.5.4 AuthGuard et AccessGuard

Deux gardes de sécurité protègent les routes sensibles :

AuthGuard : vérifie que l'utilisateur est connecté (JWT valide)

AccessGuard : vérifie les droits d'accès selon le rôle (ex. : accès restreint au Back-Office au rôle ADMIN)

```
// Sélection de patient
selectPatient(id: number) {
    this.patientService.getPatientById(id, this.idOrga).subscribe({
        next: (patient) => {
            patient.fullname = `${patient.firstname} ${patient.lastname} (${patient.birthDate})`;
            this.selectionService.setPatient(patient);
            this.router.navigate([NAVIGATION.patientBiometrics(patient.id)]);
        },
        error: () => {
            this.loadingPatients = false;
        },
    });
}
```

4.4.5.5 Navigation dynamique et paramètres

La navigation peut être :

- a) Statique via [routerLink]

```
<div class="z-1 text-center">
    <div class="text-900 font-bold text-8xl mb-4">Access Denied</div>
    <p class="line-height-3 mt-0 mb-5 text-xl font-medium text-700">You don't have the permissions to access this page</p>
    <p-button [routerLink]="/" styleclass="p-button-help font-medium p-button-raised">Go to Dashboard</p-button>
</div>
```

- b) Dynamique via le service Router

Les paramètres comme :idPatient sont récupérés via ActivatedRoute :

```
const id = this.route.snapshot.paramMap.get('idPatient');
```

Gestion des redirections et erreurs

- La racine ("") redirige vers la page dashboard
- Une route wildcard ** redirige vers /notfound pour gérer les erreurs 404
- Le module NotFoundModule affiche une page claire pour l'utilisateur

4.4.5.6 Breadcrumbs et navigation contextuelle

Chaque route contient un champ data.breadcrumb pour générer dynamiquement un fil d'Ariane (breadcrumb), améliorant la lisibilité de la navigation dans l'interface utilisateur.

```
{  
  path: 'back-office',  
  loadChildren: () =>  
    import(  
      './body/components/back-office/back-office.module'  
    ).then((m) => m.BackOfficeModule),  
  canActivate: [AccessGuard],  
  data: { breadcrumb: 'Back-Office' }  
},
```

4.4.5.7 Avantages de cette architecture de routing

Fonctionnalité	Mise en œuvre dans EyeWeb
SPA fluide	via Angular Router
Routing dynamique et paramétré	:idPatient, :examId
Sécurité via Guards	AuthGuard + AccessGuard
Modularité	Lazy loading par module
Expérience utilisateur	Breadcrumb + router-outlet
Gestion des erreurs	route ** vers NotFound

4.4.6 Services Angular et gestion des données via RXJS et HTTPCLIENT

La gestion de la logique métier côté frontend dans EyeWeb repose sur des services Angular injectables. Ces services assurent la communication avec le backend Spring Boot, encapsulent les traitements métier, gèrent la réactivité via RxJS, la sécurité des appels API avec JWT, et la gestion centralisée des erreurs. Ils constituent une brique fondamentale de l'architecture modulaire d'Angular.

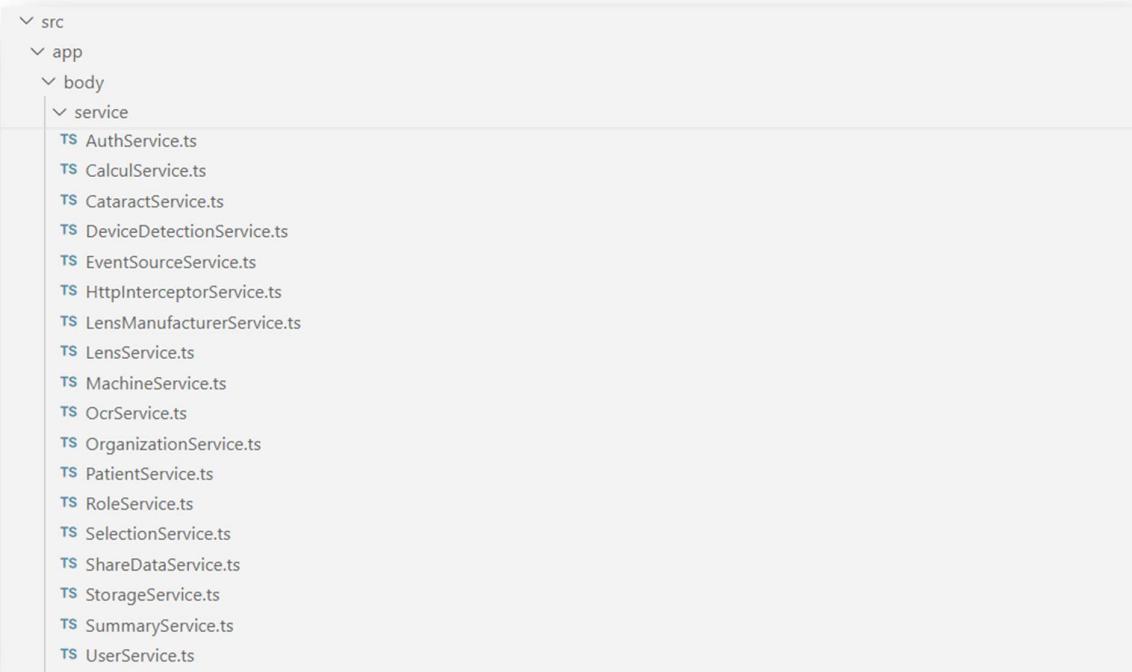
4.4.6.1 Rôle des services Angular dans l'architecture EyeWeb

Les services Angular sont des classes décorées avec `@Injectable({ providedIn: 'root' })`, ce qui garantit une instance unique partagée dans toute l'application. Leur rôle est multiple :

- Centraliser les appels HTTP vers l'API REST.
- Encapsuler la logique métier spécifique à chaque entité.
- Permettre une réutilisation du code entre plusieurs composants.
- Offrir une séparation stricte entre interface (composants) et données (services).
- Faciliter la communication entre composants via des observables, notamment les Subject ou BehaviorSubject.
- Améliorer la testabilité et le découplage grâce à l'injection de dépendance.

4.4.6.2 Structure et organisation des services

Les services Angular sont organisés dans un dossier dédié :



Chaque fichier représente une entité métier ou une fonctionnalité transverse. Cette organisation respecte le principe de modularité et de séparation des responsabilités.

4.4.6.3 Communication HTTP avec HttpClient

Le module HttpClientModule est importé dans AppModule. Tous les services utilisent HttpClient pour consommer les endpoints du backend.

Exemple d'injection :

```
constructor(private httpClient: HttpClient) {}
```

Méthodes typiques utilisées :

```
get<T>()
post<T>()
put<T>()
delete<T>()
```

Chaque méthode retourne un Observable<T>, assurant une gestion asynchrone fluide des données.

4.4.6.4 Exemple de service métier : PatientService

```
@Injectable({ providedIn: "root" })
export class PatientService { readonly API_URL = API_URL.DEV; readonly ENDPOINT_PATIENTS = "/patients";
  constructor(private httpClient: HttpClient) {}

  /**
   * Met à jour un patient existant en base de données.
   * @param patient - Objet patient avec les modifications
   * @param organizationsId - Liste des IDs des organisations associées
   * @returns Observable contenant le patient mis à jour
   */
  updatePatient(patient: Patient, organizationsId: number[]): Observable<Patient> {
    if (!patient.id) { throw new Error("ERROR : Patient ID is required for update."); }
    if (!organizationsId || organizationsId.length === 0) { throw new Error("ERROR : At least one organization ID is required for update"); }
    const url = `${this.API_URL}${this.ENDPOINT_PATIENTS}/${patient.id}/organizations/${organizationsId.join(',')}`;
    return this.httpClient.put<Patient>(url, patient).pipe(
      tap((updatedPatient: any) => {
        if (!updatedPatient) {
          throw new Error("The API did not return an updated patient!"));
        }
        catchError(() => { return throwError(() => new Error("Error updating the patient")));})
    );
  }

  /**
   * Supprime un patient en mettant son attribut `deleted` à `true`.
   * @param idPatient - ID du patient à supprimer
   * @param organizationsId - Liste des IDs des organisations associées
   * @returns Observable<Patient> contenant le patient marqué comme supprimé
   */
  deletePatient(idPatient: number, organizationsId: number[]): Observable<Patient> {
    if (!idPatient) { throw new Error("ERROR : Patient ID is required for deletion."); }
    if (!organizationsId || organizationsId.length === 0) { throw new Error("ERROR : At least one organization ID is required for deletion."); }
    const url = `${this.API_URL}${this.ENDPOINT_PATIENTS}/${idPatient}/organizations/${organizationsId.join(',')}/delete`;
    return this.httpClient.put<Patient>(url, {}).pipe(
      tap((deletedPatient: any) => {
        if (!deletedPatient) { throw new Error("The API did not return a deleted patient!");}
      })
      catchError(() => { return throwError(() => new Error("Error deleting the patient"));})
    );
  }
}
```

4.4.6.5 Gestion des erreurs : catchError et résilience

a) Traitement asynchrone avec RxJS et catchError

Les services Angular du projet EyeWeb, notamment PatientService, intègrent la gestion d'erreurs grâce à l'opérateur catchError de la bibliothèque RxJS. Ce mécanisme permet d'intercepter les erreurs HTTP (réseaux, statuts 4xx/5xx, corps de réponse vide) et d'en produire une version lisible et maîtrisée, tout en évitant un crash du composant abonné.

La méthode handleError centralise ce comportement :

```
/**  
 * Méthode générique de gestion des erreurs HTTP.  
 * @param error - Erreur renournée par le backend  
 * @returns Observable avec message d'erreur personnalisé  
 */  
private handleError(error: any): Observable<never> {  
  console.error('HTTP Error :', error);  
  const message = error?.error?.message || 'Error communicating with the server.';  
  return throwError(() => new Error(message));  
}
```

Les avantages sont :

- Prévention des plantages de l'interface utilisateur ;
- Personnalisation des messages pour affichage contextuel ou debug ;
- Préparation à l'intégration future de systèmes de suivi type Sentry, Datadog, ou OpenTelemetry.

b) Étendue et limitations actuelles

Les appels GET (getAllPatients, getPatientById) sont également sécurisés par catchError, ce qui renforce la robustesse globale. Chaque méthode importante utilise une vérification systématique du contenu de la réponse (tap) et un message d'erreur explicite.

```
/**  
 * @returns Observable contenant un tableau de patients  
 * Cette méthode interroge le backend via HTTP GET et retourne la liste complète des patients disponibles.  
 * Elle inclut un mécanisme de résilience : en cas d'erreur HTTP,  
 * un message d'erreur personnalisé est renvoyé grâce au catchError().  
 */  
getAllPatients(): Observable<Patient[]> {  
  return this.httpClient  
    .get<Patient[]>(`${this.API_URL}${this.ENDPOINT_PATIENTS}`)  
    .pipe(  
      catchError(this.handleError.bind(this))  
    );  
}
```

4.4.6.6 Sécurité des appels API avec JWT et HttpInterceptor

Dans EyeWeb, la sécurité des appels API repose sur un HttpInterceptor centralisé, injecté globalement via le AppModule. Ce mécanisme permet d'assurer que toutes les requêtes HTTP

sortantes sont enrichies du jeton JWT et qu'aucune erreur technique n'échappe au contrôle de l'interface.

a) Injection du jeton JWT dans les requêtes

Le service `HttpInterceptorService` vérifie, à chaque requête, si l'utilisateur est authentifié via `authService.isAuthenticated()`. Si c'est le cas, le jeton est automatiquement ajouté à l'en-tête `Authorization` :

```
if (this.authService.isAuthenticated()) {
    // Si l'utilisateur est authentifié, on ajoute le token à l'en-tête
    const request = req.clone({
        headers: new HttpHeaders({
            Authorization: this.authService.getToken(),
        }),
    });
    return next.handle(request).pipe(
        catchError((error: HttpErrorResponse) => {
            return this.manageErrors(error);
        })
    );
} else {
    // Si l'utilisateur n'est pas authentifié, on gère quand même les erreurs
    return next.handle(req).pipe(
        catchError((error: HttpErrorResponse) => {
            return this.manageErrors(error);
        })
    );
}
```

Ce fonctionnement garantit une authentification transparente et systématique sur tous les endpoints sécurisés, sans duplication de logique dans les services.

b) Gestion centralisée des erreurs HTTP

L'intercepteur implémente un système de résilience globale : toute erreur interceptée (`catchError`) est traduite en un message clair, à la fois côté console développeur (`console.log(...)`) et côté interface utilisateur via un `MessageService` (PrimeNG).

c) Traduction des erreurs techniques

Les erreurs du backend sont classifiées par code HTTP. Lorsqu'aucun message explicite n'est renvoyé, le système applique un libellé par défaut selon le statut :

Code HTTP	Message affiché
400	Bad Request. Veuillez réessayer plus tard.
401	Unauthorized. Veuillez réessayer plus tard.
403	Forbidden. Veuillez réessayer plus tard.

404	Not Found. Veuillez réessayer plus tard.
409	Conflict. Veuillez réessayer plus tard.
Autre	Impossible d'atteindre le serveur.

Cela permet d'éviter tout comportement silencieux côté interface, en informant toujours l'utilisateur de manière contextualisée.

d) Affichage des erreurs en temps réel

Grâce à PrimeNG et son MessageService, chaque erreur est visuellement restituée dans l'interface via un toast :

```
if (this.resetPasswordForm.get('resetPassword').value !== this.resetPasswordForm.get('confirmPassword').value) {
    this.messageService.add({
        severity: 'error',
        summary: 'Confirm password',
        detail: 'confirmation password not the same as password.'
    });
}
```

L'UX est ainsi respectée même dans les cas d'échec technique.

e) Résilience globale et anticipation

Ce système :

- Unifie la sécurité et les erreurs dans un point central (le HttpInterceptor) ;
- Prépare l'intégration de logs distants comme Sentry, en concentrant toutes les anomalies dans une fonction unique manageErrors() ;
- Maintient une architecture propre, en allégeant les services métier.

4.4.6.7 Gestion des états avec RxJS : Observable, Subject, BehaviorSubject

RxJS est au cœur de la gestion des flux dans EyeWeb. Les services exposent des données réactives via des Observable<T>.

```
private patientSource = new BehaviorSubject<Patient | null>(null);
```

Cette logique permet de propager un état partagé entre plusieurs composants sans liaison directe.

4.4.6.8 Utilisation des services dans les composants

Dans EyeWeb, chaque composant Angular s'appuie sur l'injection de services métier pour communiquer avec l'API backend, centraliser la logique de traitement, et déléguer les responsabilités non liées à la présentation. Ce découplage est fondamental dans le respect du principe SOLID de responsabilité unique.

a) Injection des services

Le constructeur du composant reçoit les services nécessaires à son bon fonctionnement :

```
constructor(  
  private fb: FormBuilder,  
  private selectionService: SelectionService,  
  private messageService: MessageService,  
  private patientService: PatientService,  
  private storageService: StorageService,  
  private confirmationService: ConfirmationService,  
  private router: Router  
) {}
```

Chaque service injecté répond à une responsabilité bien définie :

- PatientService : gestion des données patients (CRUD, validations)
- SelectionService : gestion du patient courant en mémoire
- MessageService : affichage des messages visuels (erreurs, succès)
- StorageService : récupération des IDs d'organisation stockés localement
- ConfirmationService : boîtes de dialogue interactives
- Router : redirection après actions réussies

b) Récupération et initialisation des données

Lors du `ngOnInit()`, le composant récupère le patient actuellement sélectionné par le `SelectionService`, et adapte son état local :

```
ngOnInit() {
    this.patient = this.selectionService.getPatient();

    if (this.patient) {
        if (this.patient.birthDate) {
            this.birthDateValue = new Date(this.patient.birthDate);
        }

        this.idOrga = this.patient.organizations ? this.patient.organizations.map(org => org.id) : [];
        if (!this.idOrga.length) {
            this.idOrga = this.storageService.get('organization') || [];
        }

        if (!this.patient.address) {
            this.patient.address = this.createEmptyAddress();
        }
    }
}
```

Cette logique garantit une préparation de l'interface cohérente, quelle que soit la provenance de l'utilisateur dans le parcours de navigation.

c) Mise à jour des patients via le service

Lorsqu'un utilisateur valide la modification d'un patient, le composant appelle `PatientService.updatePatient(...)`, tout en contrôlant la qualité des données et les cas d'erreur :

```
sendUpdateRequest() {
    this.patientService.updatePatient(this.patient, this.idOrga).subscribe({
        next: (updatedPatient) => {
            this.isEditMode = false;
            this.patient = updatedPatient;

            this.messages = [
                { severity: 'success', detail: `The patient ${updatedPatient.firstname} ${updatedPatient.lastname} has been successfully updated. ` }
            ];

            setTimeout(() => {
                this.selectionService.resetPatient();
                this.router.navigate(['/']);
            }, 2000);
        },
        error: (err) => {
            this.messageService.add({ severity: 'error', summary: 'Error', detail: 'Failed to update patient.' });
        }
    });
}
```

Le service gère les aspects techniques : construction de l'URL, envoi HTTP, gestion du `catchError` (résilience), typage strict. Le composant, quant à lui, reste centré sur l'UX et les états de formulaire.

d) Suppression d'un patient

La logique de suppression repose également sur une confirmation utilisateur (via ConfirmationService) puis sur un appel au service métier :

```
confirmDeletePatient() {
    if (!this.patient || !this.patient.id) return;

    this.patientService.deletePatient(this.patient.id, this.idOrga).subscribe({
        next: () => {
            this.messageService.add({ severity: 'success', summary: 'Success', detail: 'Patient successfully deleted!' });

            this.selectionService.setPatient(null);
            this.patient = null;

            setTimeout(() => {
                this.router.navigate(['/']);
            }, 1500);
        },
        error: () => {
            this.messageService.add({ severity: 'error', summary: 'Error', detail: 'Failed to delete patient. Please try again.' });
        }
    });
}
```

4.4.7 Appels spécialisés : login, registration, résumé opératoire, calcul

Chaque domaine dispose de son propre service :

Service	Rôle
AuthService	login(), logout(), register(), gestion des tokens
CalculatorService	calcul IOL personnalisé
SummaryService	récupération du résumé opératoire
ExamService	gestion des examens liés à un patient

4.4.7.1 Typage strict avec les interfaces TypeScript

Les services utilisent des interfaces typées pour renforcer l'intelliSense et la cohérence avec le backend :

```

export interface Patient {
    id: number;
    createdBy: string;
    createdAt: string;
    modifiedBy: string;
    modifiedAt: string;
    deleted: boolean;
    version: number;

    lastname: string | null;
    firstname: string | null;
    fullname: string | null;
    birthDate: string | null;
    niss: string | null;
    gender: string | null;
    phone: string | null;
    mail: string | null;
    job: string | null;
    hobbies: string | null;
    address: Address | null;
    organizations: Organization[] | null;
    exams: Exam[] | null;
}

```

Ce typage facilite la transformation des réponses API et réduit les erreurs de manipulation des données.

Ce design garantit une séparation des responsabilités exemplaire entre la logique métier (dans le service) et la logique d’interaction utilisateur (dans le composant).

a) Avantages de cette approche

Aspect	Bénéfices
Lisibilité	Le composant est clair, linéaire, focalisé sur la présentation
Réutilisabilité	Les services peuvent être appelés par d’autres composants (ex. création, liste, résumé...)
Sécurité	Les appels API sont encapsulés, typés et protégés contre les erreurs
Testabilité	Les services sont facilement mockables dans les tests unitaires du composant
Évolutivité	Ajouter des logs, un cache ou une instrumentation ne nécessite aucun changement dans le composant

4.4.7.2 Architecture RESTful : respect des conventions http

Chaque verbe HTTP est utilisé selon les standards REST :

Verbe	Rôle	Exemple
GET	Lecture	/patients, /summary/5
POST	Création	/auth/register
PUT	Mise à jour	/patients/5
DELETE	Suppression	/patients/5

EyeWeb respecte strictement cette convention dans tous ses services.

4.4.7.3 Services transversaux et comportement UI

Certains services ont un rôle transverse dans la gestion de l'interface :

- ToastService ou NotificationService : messages de confirmation
- LoadingService : affichage d'un loader global
- StorageService : persistance dans localStorage ou sessionStorage

4.4.7.4 Résumé des avantages des services dans EyeWeb

Atout	Description
Réutilisabilité	Services injectables globalement
Typage fort	Interfaces pour chaque entité
Sécurité	JWT avec HttpInterceptor
Architecture claire	Séparation composants / services
Réactivité	RxJS et Observables
Maintenance facile	Tests unitaires, lisibilité
Compatibilité REST	Respect des verbes http

4.5 Composants Angular : Structure, Communication et Intégration UI

L'architecture frontend de l'application EyeWeb repose entièrement sur le paradigme composant d'Angular. Chaque fonctionnalité métier est encapsulée dans un composant autonome, interconnecté avec le backend via des services typés, et organisé au sein de modules fonctionnels. Ce chapitre présente l'ensemble des aspects liés à la structure, la communication,

l'intégration UI, la réactivité et les bonnes pratiques mises en œuvre dans les composants Angular de l'application EyeWeb.

4.5.1 Structure d'un composant Angular

Chaque composant Angular se compose de quatre fichiers principaux :

- .component.ts : définit la logique de gestion de l'interface
- .component.html : décrit le rendu de l'interface utilisateur
- .component.scss : contient le style local appliqué au composant

L'initialisation se fait dans ngOnInit(), la récupération des données via les services injectés, et la vue est automatiquement mise à jour par les observables exposés.

4.5.1.1 Organisation modulaire des composants

EyeWeb adopte une architecture modulaire basée sur les Feature Modules. Chaque domaine fonctionnel dispose de son propre dossier, module, routing, composant principal et service associé.

Exemples de domaines :

- /patient/ (liste, détails, formulaire)
- /calculator/ (formulaire de calcul biométrique)
- /summary/ (résumé opératoire)
- /auth/ (connexion, inscription)

Types de composants :

- Pages : home, calculator, summary
- Métier : patient-summary, exam-detail
- Techniques/UI : toast, modal-confirm, loading
- Modaux dynamiques : edit-exam-dialog, create-patient

4.5.1.2 Cycle de vie et réactivité des composants

Les principaux hooks Angular utilisés sont :

- ngOnInit() : initialisation et chargement des données
- ngOnDestroy() : nettoyage des subscriptions et ressources
- ngAfterViewInit() : accès au DOM (cas rares)

Tous les composants critiques (calcul, résumé, formulaire) sont réactifs grâce à l'utilisation de RxJS.

4.5.1.3 Communication entre composants

Deux stratégies sont employées pour l'échange de données :

- `@Input()` et `@Output()` avec `EventEmitter` : communication parent → enfant et remontée d'événements.
- `BehaviorSubject` : partages globaux d'état via les services (ex. : patient sélectionné).

```
@Input() visible!: boolean;
@Input() side!: string;
@Input() patient!: Patient;
@Input() idOrga!: number[];
@Output() visibleModified = new EventEmitter<boolean>();
```

4.5.1.4 Liaison de données et interaction avec la vue

Les différents types de data binding Angular sont utilisés :

- `{{ expression }}` : interpolation
- `[property]` : liaison unidirectionnelle vers la vue
- `(event)` : capture d'événements (clic, input...)
- `[(ngModel)]` : liaison bidirectionnelle (cas simples)

Directives structurelles :

`*ngIf`, `*ngFor`, `ngSwitch`

```
<ul *ngIf="uploadedFiles.length">
  <li *ngFor="let file of uploadedFiles">
    {{ file.name }} - {{ file.size }} bytes
  </li>
</ul>
```

4.5.1.5 Affichage des composants modaux avec DialogModule (PrimeNG)

Dans EyeWeb, l'affichage des composants modaux repose sur l'utilisation du module DialogModule de PrimeNG. Les modales sont déclarées directement dans le template HTML à l'aide de la balise <p-dialog> et leur visibilité est contrôlée par une propriété booléenne liée au composant TypeScript.

```
<!-- Dialog de chargement -->
<p-dialog [(visible)]="query" [modal]="true" [style]="{{ width: '90vw' }}" [draggable]="false" [resizable]="false"
   [maximizable]="true" [closable]="false">
  <ng-template pTemplate="header">
    <div class="inline-flex align-items-center justify-content-center gap-2">
      <div class="flex justify-content-between mb-4">
        <div class="flex align-items-center justify-content-end bg-blue-100 border-round-right relative p-3"
            style="left: -1.5rem">
          <span class="font-bold text-xl text-blue-600 pl-4">Loading..</span>
        </div>
      </div>
    </div>
  </ng-template>
  <p-progressBar mode="indeterminate" [style]="{{ height: '10px' }}></p-progressBar>
</p-dialog>
```

Ce mode de fonctionnement offre une expérience utilisateur fluide, sans manipulation manuelle du DOM. Il permet également de centraliser la logique d'ouverture des boîtes de dialogue dans le composant parent, tout en gardant un contrôle total sur les données passées à la modale via des propriétés @Input().

4.5.1.6 Gestion des états et flux de données

L'application EyeWeb applique une architecture unidirectionnelle des données, inspirée des bonnes pratiques Angular. Ce modèle repose sur trois principes :

- Le service est la source de vérité, exposant les données via des observables (Observable<T> ou Behavior-Subject<T>),
- Le composant s'abonne à ces données pour les consommer,
- La vue HTML s'actualise automatiquement, généralement grâce au pipe | async.

Cette approche permet de :

- Réduire les effets de bord, en centralisant la logique dans les services,
- Simplifier la synchronisation des composants, qui réagissent aux changements de manière déclarative,
- Faciliter les tests unitaires des composants et des services.

Contrairement à d'autres applications Angular, EyeWeb n'utilise pas switchMap ni combineLatest dans les composants, car les paramètres d'URL ne sont pas injectés via ActivatedRoute. Les appels API sont déclenchés directement dans le ngOnInit() ou à partir d'événements utilisateurs.

4.5.1.7 Intégration UI, accessibilité et responsive design

Les composants EyeWeb sont stylisés avec :

- SCSS local
- PrimeNG : p-table, p-dialog, p-tree, p-toast, etc.
- Bootstrap personnalisé

Accessibilité assurée par :

- Attributs aria-*
- Navigation clavier avec tabindex
- Comportements adaptés au mobile (toggle menu, responsive layout)

4.5.1.8 Tests des composants et bonnes pratiques

Les tests unitaires Angular sont réalisés avec Jasmine + Karma. Chaque composant critique dispose d'un .spec.ts pour vérifier :

- Son instanciation
- L'affichage conditionnel
- La réaction aux services mockés

Bonnes pratiques appliquées :

- Architecture modulaire
- Aucun accès direct à l'API depuis le composant
- Services injectés via DI
- Modèle réactif exclusif (Observable + | async)
- Composants réutilisables, séparés de leur logique

4.5.1.9 Cas d'usage concrets dans EyeWeb

Composant	Fonctionnalité principale
summary.component.ts	Affichage du résumé opératoire en temps réel

calculator.component.ts	Calcul biométrique des implants ophthalmiques avec envoi API
administrative.component.ts	Formulaire de modification des patients avec validations dynamiques
app.topbar.component.ts	Affichage du menu responsive, changement de thème couleur, déconnexion

4.6 Système de formulaires réactifs : structure, validation , interactions

4.6.1 Choix stratégique : Reactive Forms

Dans une application comme EyeWeb, les formulaires sont un élément clé pour interagir avec l'utilisateur. Ils servent à saisir des données importantes comme les informations des patients, les calculs IOL ou encore les mots de passe. Il est donc essentiel que la saisie soit fiable et bien validée.

Angular propose deux façons de gérer les formulaires : les formulaires basés sur le template et les formulaires réactifs. J'ai choisi les formulaires réactifs car ils sont plus souples à manipuler dans le code, mieux adaptés aux validations complexes, et parfaitement compatibles avec des composants comme p-dropdown, p-inputNumber ou p-calendar utilisés avec PrimeNG.

4.6.1.1 Déclaration programmatique via FormBuilder

Les formulaires sont construits intégralement dans le fichier .ts du composant à l'aide du service FormBuilder, ce qui permet :

- un typage fort des champs via TypeScript .
- une initialisation explicite de chaque FormControl .
- une association directe avec les règles de validation .

Exemple dans profile.component.ts :

```
this.userForm = this.fb.group({
  firstname: ['', [Validators.required, Validators.pattern("^[A-zÀ-Ú-]+(?:[-\\s][A-zÀ-Ú-]+)*$")]],
  lastname: ['', [Validators.required, Validators.pattern("^[A-zÀ-Ú-]+(?:[-\\s][A-zÀ-Ú-]+)*$")]],
  email: ['', [Validators.required, Validators.email]],
  phone: ['', [Validators.required, Validators.pattern("^\\d{8,15}$")]],
  birthDate: [null],

  address: this.fb.group({
    street: ['', Validators.required],
    streetNumber: ['', Validators.required, Validators.pattern("^[A-z0-9/-]+$")],
    boxNumber: ['', Validators.pattern("^[A-z0-9/-]*$")],
    city: ['', [Validators.required, Validators.pattern("^[A-zÀ-Ú\\s-]+$")]],
    zipcode: ['', [Validators.required, Validators.pattern("\\d{4}$")]],
    country: ['', [Validators.required, Validators.pattern("^[A-zÀ-Ú\\s-]+$")]]
  })
});
```

Chaque champ est ainsi contrôlé individuellement, permettant une validation précise dès la saisie.

4.6.1.2 Connexion au template et liaison dynamique

Dans les fichiers .html (ex. patient-form.html, new-calcul-form.html), le formulaire est lié au modèle via l'attribut [formGroup] :

```
<form [formGroup]="patientForm" novalidate>
  <div class="p-fluid p-formgrid grid">
    <div class="field col-12 md:col-4">
      <label for="firstname" class="block font-medium mb-2">Firstname
        <span class="font-bold mr-3 text-red-500 w-2">*</span></label>
        <input id="firstname" type="text" placeholder="Firstname" class="p-inputtext p-component p-element"
               pInputText formControlName="firstname" />
        <small class="p-error block"
              *ngIf="patientForm.get('firstname') ?.invalid && patientForm.get('firstname')?.dirty">
          The firstname is not available.
        </small>
    </div>
  </div>
</form>
```

Cette liaison permet un binding réactif : toute modification de l'état ou de la valeur d'un champ est immédiatement reflétée dans le modèle.

4.6.1.3 Validation intégrée et personnalisée

La validation des champs repose principalement sur les validateurs natifs d'Angular comme Validators.required, Validators.min, Validators.max, ou encore Validators.pattern. Ces règles permettent de vérifier immédiatement que l'utilisateur a bien rempli les champs obligatoires, et que les formats saisis (numériques, dates, structures de chaînes...) sont valides.

Pour certains cas plus spécifiques (ex. : format du NISS, numéro de téléphone, code postal), des expressions régulières personnalisées sont directement intégrées dans les composants, sans passer par des validateurs Angular réutilisables. Cette approche locale reste efficace dans le contexte actuel du projet.

Ce système de validation améliore la qualité des données dès la saisie, tout en fournissant un retour clair et immédiat à l'utilisateur.

4.6.1.4 Validation et fiabilité

EyeWeb ne repose pas uniquement sur les validateurs intégrés d'Angular. Les vérifications sont faites manuellement dans le code TypeScript, selon les règles métier :

- Champs obligatoires (ex. prénom, nom, date de naissance).
- Regex personnalisées pour les formats (SSN, téléphone, email...).
- Validité du genre : uniquement MALE ou FEMALE.

Les erreurs sont ensuite affichées de manière conditionnelle en HTML :

```
<small class="p-error" *ngIf="!isValidFirstname && isEditMode">
    Firstname is invalid. Only letters, spaces, hyphens, or apostrophes are allowed.
</small>
```

Cette gestion manuelle garantit une personnalisation complète, adaptée aux exigences médicales du projet.

4.6.1.5 Soumission contrôlée

La validation ne repose pas sur form.invalid ou markAllAsTouched(), mais sur des indicateurs booléens définis dans le composant (ex. isValidFirstname). Lors de l'appel à updatePatient(), ces indicateurs sont recalculés à partir de fonctions de validation spécifiques, et une notification est émise si des erreurs sont détectées.

```
// Vérification des champs obligatoires
if (!this.isValidFirstname || !this.isValidLastname || !this.isValidSSN || !this.isValidBirthDate) {
    this.messageService.add({ severity: 'error', summary: 'Validation Error', detail: 'Required fields are invalid.' });
}
```

Cette méthode offre une maîtrise plus fine des conditions de validation, tout en permettant une gestion contextuelle via isEditMode.

4.6.1.6 Patching et édition

Lors de l'édition d'un patient ou d'un examen, les formulaires sont préremplis avec les données existantes grâce à patchValue() ou l'attribution directe des valeurs :

```
this.patient = this.selectionService.getPatient();

if (this.patient) {
    if (this.patient.birthDate) {
        this.birthDateValue = new Date(this.patient.birthDate);
```

Ce mécanisme garantit une compatibilité même si certains champs sont absents ou conditionnels.

4.6.1.7 Dynamisme et champs conditionnels

Certaines valeurs déclenchent l'activation ou la désactivation d'autres champs. Par exemple, le champ allergyDetails peut être activé uniquement si hasAllergy est coché. Ce type de logique est géré à travers valueChanges, mais peut également être implémenté dans des méthodes validateInputs() comme c'est le cas dans AdministrativeComponent.

4.6.1.8 Intégration avec PrimeNG

EyeWeb utilise intensivement les composants PrimeNG dans ses formulaires pour améliorer l'expérience utilisateur :

- p-dropdown pour les listes déroulantes (genre, praticien...)
- p-calendar pour les dates
- p-inputMask pour les numéros de sécurité sociale
- p-toast pour les notifications

Exemple d'intégration :

```
<div class="field col-12 md:col-4">
    <label for="birthDate" class="block font-medium mb-2">Date of birth
        <span class="font-bold mr-3 text-red-500 w-2">*</span></label>
        <p-calendar formControlName="birthDate" [iconDisplay]="'input'" [showIcon]="true" [minDate]="${minDate}"
            [maxDate]="${maxDate}" dateFormat="dd-mm-yy" placeholder="DD-MM-YY" appendTo="body" id="birthDate">
    </p-calendar>
</div>
```

La mise en page est également optimisée grâce à p-formgrid, p-fluid et les classes utilitaires de PrimeFlex.

4.6.1.9 Interaction utilisateur et messages

Le service MessageService est injecté dans les composants et permet l'affichage de messages clairs suite aux actions utilisateur (erreurs, succès, confirmation).

```
this.userService.updateVerifiedUser(updatedUser).subscribe({
  next: () => {
    this.query = false;
    this.messageService.add({
      severity: 'success',
      summary: 'User Modified',
      detail: 'User data has been successfully updated.'
    });
    this.refreshUsers();
  },
  error: (err) => {
    this.query = false;
    this.messageService.add({
      severity: 'error',
      summary: 'ERROR',
      detail: 'An error occurred while updating the user.'
    });
  }
});
```

Chaque feedback est contextualisé : succès après une sauvegarde, avertissement sur champ invalide, ou message d'échec lors d'un appel API.

4.6.1.10 Application dans plusieurs composants

Le système de formulaire réactif est utilisé dans de nombreux contextes dans EyeWeb :

- patient-form.ts : création complète d'un patient
- login.component.ts, forgotpassword.component.ts : saisie des identifiants ou mails
- new-calcul-form.ts : calculs biométriques
- second-eye-form.ts : second œil du patient

Chacun de ces composants exploite la souplesse de FormGroup, la validation personnalisée, et la gestion d'état pour offrir une saisie fluide, fiable et ergonomique.

4.6.1.11 Récapitulatif des bonnes pratiques formulaires dans EyeWeb

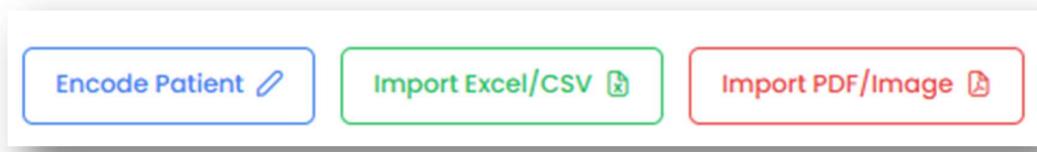
L'application EyeWeb applique les standards Angular en matière de gestion de formulaires, tout en intégrant des pratiques avancées adaptées au contexte médical et ergonomique. Le tableau ci-dessous :

Bonne pratique	Mise en œuvre dans EyeWeb
Déclaration explicite et structurée des FormGroup	Tous les formulaires sont construits via FormBuilder dans le .ts, avec une organisation claire : patientForm, examForm, calculForm, etc.
Validation typée et immédiate	Les champs critiques sont validés via des fonctions manuelles ou des expressions régulières métier, avec affichage conditionnel d'erreurs dans le HTML.
Gestion manuelle des erreurs	L'état des champs est contrôlé par des indicateurs booléens (ex. isValidFirstname) pour un contrôle fin de la validation.
Pré-remplissage avec patchValue()	En mode édition, les formulaires sont mis à jour à partir de l'API grâce à patchValue() ou à une injection directe des données.
Affichage dynamique et contextuel des champs	Certains champs sont activés/désactivés selon les réponses précédentes via valueChanges() ou une logique dédiée dans le composant.
Intégration complète avec PrimeNG	Tous les champs utilisent des composants UI ergonomiques : p-calendar, p-dropdown, p-toast, p-inputMask, etc.
Réutilisabilité des composants de formulaire	Les formulaires sont encapsulés dans des composants dédiés et réutilisables : SecondEyeForm, BiometryDisplay, PatientForm, etc.
Soumission conditionnée par validation	Chaque appel au backend est déclenché uniquement après vérification manuelle des champs requis, et validité du formulaire.
Communication propre avec les services	Les valeurs des formulaires (form.value) sont transmises aux services avec typage strict (Patient, Exam, CalculDto).
Affichage clair des messages d'erreur ou de succès	MessageService de PrimeNG est utilisé pour notifier l'utilisateur (succès, échec ou erreur de validation).

4.7 Importation de Données Patients : Excel, PDF et Encodage Manuel

Une fonctionnalité métier essentielle demandée par le commanditaire était de permettre l'intégration rapide et fiable de données patient, que ce soit par fichiers numériques fournis par les appareils médicaux, documents scannés ou saisie directe.

Pour cela, trois modalités distinctes ont été développées et intégrées au composant



DashboardComponent :

4.7.1 Importation Excel

Objectif

Permettre le chargement massif de données patient via un fichier .xls ou .xlsx, typiquement exporté depuis un logiciel d'appareillage biométrique (type Zeiss IOLMaster).

Implémentation technique

- a) Utilisation de la librairie xlsx de SheetJS (Angular) pour parser les fichiers Excel.
- b) Boîte de dialogue p-dialog avec p-fileUpload pour sélectionner le fichier.
- c) Mappage automatique des colonnes vers les champs cibles (e.g. K1, K2, AL, birthdate).
- d) Envoi des données en JSON via HttpClient vers une route POST /api/import/excel.
- e) Vérifications backend : cohérence, absence de doublon, validité des dates.
- f) Fenêtre d'import Excel ouverte, fichier sélectionné
- g) Aperçu des données analysées et prêtes à être validées

Validation utilisateur

- L'utilisateur reçoit un récapitulatif des données importées
- En cas d'erreurs : messages bloquants par p-messages

4.7.2 Importation PDF (OCR)

Objectif

Extraire automatiquement les données d'un fichier PDF scanné d'un examen biométrique.

Implémentation technique

- a) Utilisation de l'API OcrService personnalisée basée sur Tesseract.js (Angular).
- b) Upload du PDF via un p-fileUpload dans un p-dialog modal.
- c) Envoi vers le backend avec FormData, traitement via une fonction OCR qui extrait les blocs de texte, puis regex pour repérer les champs médicaux (K1, AL, ACD, etc.).
- d) Nettoyage du texte : suppression des artefacts visuels, reformattage des nombres.
- e) Pré-remplissage automatique d'un FormGroup Angular pour validation
- f) Fenêtre de dialogue PDF import, nom du fichier affiché
- g) Tableau de valeurs extraites automatiquement affiché

Validation utilisateur

- L'utilisateur peut corriger manuellement les données OCRisées avant validation.
- En cas de doute : possibilité de relancer l'analyse OCR ou de switcher vers l'encodage manuel.

4.7.3 Encodage manuel assisté

Objectif

Saisir à la main les données pour un patient, dans les cas où aucune exportation n'est disponible.

Implémentation technique

- a) Formulaire dynamique Angular réparti en plusieurs sections :
- b) Données administratives (nom, prénom, date de naissance, etc.)
- c) Données biométriques (ACD, AL, K1, K2, WTW, HOA, etc.)
- d) Validation intégrée dans le FormBuilder avec contraintes médicales (ex. : $AL > 18.0 \&& AL < 32.0$)
- e) Affichage des erreurs en direct via p-messages
- f) Sauvegarde via POST /api/patient/:id/examen
- g) Formulaire prérempli avec suggestions de valeurs standards
- h) Messages de validation affichés dynamiquement

Validation utilisateur

- L'utilisateur doit confirmer l'exactitude des données via un bouton « Valider et enregistrer ».
- Les données sont ensuite liées au patient sélectionné via id_patient.

4.7.4 Intégration technique dans EyeWeb

Les flags importExcel, importPDF, importEncode sont gérés dans le DashboardComponent pour contrôler les boîtes de dialogue Angular.

Les services PatientService, OcrService, StorageService et SelectionService sont injectés pour centraliser la logique et conserver la séparation des responsabilités.

4.7.5 Perspectives évolutives

- OCR amélioré avec entraînement sur des jeux de données spécifiques (OCR médical).
- Historique des imports lié à chaque patient pour tracer les modifications.
- Intégration directe avec les API des fabricants (ex. Zeiss, Haag-Streit).
- Import FHIR/HL7 via connecteurs pour interopérabilité avec des DME (dossiers médicaux électroniques).

4.8 Flux de données et évolutivité du modèle relationnel

4.8.1 Inscription et gestion des utilisateurs

Création d'un compte utilisateur

Un nouvel utilisateur (ex. médecin, administrateur, secrétaire) crée un compte et s'authentifie pour accéder au système.

Étapes du processus

- L'utilisateur remplit le formulaire d'inscription (nom, prénom, email, rôle, etc.).
- Le système génère une entrée dans la table _User avec un identifiant unique (id_user).
- Un code de vérification est généré et stocké dans VerificationCode, puis envoyé à l'email fourni.
- L'utilisateur reçoit l'email et saisit son code pour valider son inscription.

Le système vérifie le code :

- Si valide → Active le compte (User.verified = TRUE).
- Si invalide → Affiche un message d'erreur.

L'utilisateur se connecte en saisissant ses identifiants :

- Le système vérifie les informations (email et password).
- Un token JWT est généré (Token) et stocké en base.
- L'utilisateur peut maintenant naviguer dans l'application selon son rôle (UserOrganization).

Flux des données et relations utilisées

- User → Stocke l'utilisateur.
- VerificationCode → Assure la validation de l'inscription.
- Token → Authentifie l'utilisateur.
- UserOrganization → Définit le rôle et les permissions dans une organisation

4.8.2 Ajout d'un Patient et gestion des organisations

Enregistrement d'un nouveau patient

Objectif : Ajouter un patient dans la base et l'associer à une organisation médicale.

Étapes du processus

- Un utilisateur (ex. médecin) accède à l'interface d'ajout de patient.
- Il saisit les informations du patient (nom, prénom, date de naissance, etc.).
- Le système crée une entrée dans Patient avec un identifiant unique (id_patient).
- Le patient est affilié à une ou plusieurs organisations médicales (Organizations).
- L'association est enregistrée dans la table UserOrganization.
- Le patient peut maintenant avoir des examens biométriques et des calculs ophtalmologiques associés.

Flux des données et relations utilisées

- Patient → Stocke les données du patient.
- Organizations → Définit les centres médicaux où le patient est suivi.

- UserOrganization → Lie un patient à une organisation médicale.

4.8.3 Enregistrement des Examens Biométriques

Un patient réalise un examen ophtalmologique

Objectif : Stocker les mesures biométriques d'un patient après un examen médical.

Étapes du processus

Un spécialiste réalise un examen biométrique avec des instruments ophtalmologiques.

Il renseigne les valeurs biométriques dans le système :

- Courbure de la cornée
- Longueur axiale
- Profondeur de la chambre antérieure
- Épaisseur du cristallin
- etc.

Le système crée une entrée dans Examen et la lie au patient via id_patient.

Une analyse des données est réalisée.

L'examen peut être utilisé pour faire des calculs intraoculaires (Calcul).

Flux des données et relations utilisées

- Examen → Stocke les valeurs mesurées.
- Patient → Permet d'identifier le patient concerné.
- Calcul → Utilise ces données pour simuler l'implantation d'une lentille.

4.8.4 Calcul et Simulation des Implants Intraoculaires

Génération d'un Calcul Intraoculaire

Objectif : Analyser les données biométriques pour déterminer la meilleure lentille pour le patient.

Étapes du processus

Un médecin sélectionne un examen biométrique existant (Examen).

Il lance un calcul intraoculaire (Calcul) basé sur :

- La réfraction cible
- Les valeurs biométriques
- Les formules mathématiques utilisées (Constant)
- Le système génère plusieurs simulations de correction :
 - Différentes Diopter (puissances de correction) sont testées.
 - Différentes Constant (paramètres optiques) sont appliquées.
 - Une ou plusieurs lentilles intraoculaires (Lens) sont proposées.
 - Le médecin choisit l'implant optimal, qui est alors associé au calcul.

Flux des données et relations utilisées

- Calcul → Contient les valeurs calculées.
- Diopter → Stocke les dioptries testées.
- Constant → Applique les constantes mathématiques aux formules.
- Lens → Propose différentes lentilles intraoculaires adaptées.

Objectifs cliniques et fonctionnels

Les calculs biométriques sont au cœur de la planification chirurgicale de la cataracte. Ils permettent, à partir de mesures anatomiques précises de l'œil, de prédire la puissance de l'implant intraoculaire nécessaire pour corriger la vision.

L'application automatise ce processus, en exploitant :

- Les données mesurées lors de l'examen (biométrie),
- Les formules ophtalmiques standardisées (SRK/T, Haigis, Barrett Universal II, etc.),
- Les données associées aux implants disponibles.

Données biométriques traitées

Les champs biométriques pertinents sont intégrés à l'entité Exam. Parmi les plus utilisés :

Donnée	Colonne Exam	Description
Longueur axiale (AL)	al	Distance du pôle antérieur à la rétine
Profondeur de chambre	acd, internalAcd	Distance entre la cornée et le cristallin
Courbure cornéenne	k1, k2, k1Axis, k2Axis	Kératométrie
Diamètre pupillaire	pupilDia, pupilMin, pupilMax	Taille pupille
Épaisseur lentille	lensThickness	Indice clé pour Haigis

CCT (épaisseur cornée)	cct	Corrélé à l'astigmatisme et fiabilité des mesures
WTW (blanc à blanc)	wtw	Largeur de la cornée mesurée horizontalement

Ces champs sont initialisés automatiquement lors de l'import d'un examen depuis une machine biométrique (Zeiss, IOLMaster, etc.) via l'entité Exam.

Entité calcul et prédition

Chaque examen peut produire plusieurs Calcul contenant :

- Une formule de calcul choisie (champ formulaUsed)
- Une lentille associée (champ lens)
- Une réfraction prédictive (champ predictedRefraction)
- Un indicateur de sélection (selected)

Fonctionnement d'un calcul

Le calcul peut être déclenché via :

- Le backend (appel API REST : /calculs)
- Une interaction utilisateur dans le frontend (sélection manuelle ou auto-suggestion)

Le service métier applique alors la formule sélectionnée.

Voici une simplification du workflow :

```

Float AL = exam.getAl();
Float ACD = exam.getAcd();
Float K1 = exam.getK1();
Float K2 = exam.getK2();
Float Kavg = (K1 + K2) / 2;
Float Aconstant = lens.getAConstant();
Float predictedPower = (1336 / (AL - 0.05)) - (Kavg); // SRK/T simplifié
Calcul c = new Calcul();
c.setFormulaUsed("SRK/T");
c.setPredictedRefraction(predictedPower - lens.getPower());
c.setSelected(false);

```

Sélection du meilleur calcul

Le Calcul sélectionné est celui :

- Le plus proche de la réfraction cible (targetRefrSph)
- Associé à un implant disponible
- Compatible avec les contraintes du patient
- Ce calcul est marqué via calcul.setSelected(true).
- L'IOL retenue s'affiche ensuite dans :
- Le composant Summary côté frontend
- Le SummaryDto côté backend

Services métier associés.

CalculService :

- Génération de plusieurs calculs à partir d'un examen
- Application de formules ophtalmiques
- Sélection automatique du meilleur calcul

CataractService :

- Intègre les calculs dans le parcours patient
- Fournit les valeurs à la couche Summary

Interface utilisateur

Le module Calculator affiche :

Élément	Composant PrimeNG utilisé
Liste des calculs	p-table
Sélection IOL manuelle	p-dropdown
Résultat sélectionné	p-card, p-tag

Interaction :

- L'utilisateur peut sélectionner manuellement un calcul.
- Le calcul est alors mis à jour en BDD et marqué comme selected.

Sécurité et audit

- Chaque modification de calcul est auditee
- Seuls les utilisateurs autorisés (chirurgien, assistant) peuvent modifier les choix d'IOL
- Historique des modifications possible via Envers (Audited)

Perspectives d'évolution

- Intégration de l'algorithme Barrett Universal II (API payante ou implémentation propre)
- Apprentissage automatique basé sur les cas historiques pour ajuster les prédictions
- Calcul en batch lors de l'import de mesures (préchargement)
- Gestion des contraintes cornéennes spécifiques (astigmatisme, toricité¹⁷⁾)

4.8.5 Gestion des Lentilles Intraoculaires

Sélection et commande d'une lentille

Objectif : Confirmer le choix de la lentille à implanter et gérer son fabricant.

Contexte médical et finalité logicielle

Dans la chirurgie de la cataracte, le cristallin opacifié est retiré, puis remplacé par une lentille intraoculaire (IOL). Le choix de l'implant repose sur :

- Des mesures biométriques précises (kératométrie, AL, ACD...),
- Un modèle de calcul ophtalmologique (Haigis, Barrett, SRK/T...),
- Le profil du patient (monovision, presbytie, myopie...).

L'application prend en charge tout ce workflow via une architecture intégrée et modulable.

Étapes du processus

- Une lentille intraoculaire est sélectionnée pour un patient.
- L'association est enregistrée dans LensManufacturer.
- Le fabricant de la lentille est identifié.
- La commande est validée et envoyée au fournisseur.

Flux des données et relations utilisées

- Lens → C'est la représentation métier de l'implant. Elle comprend des données essentielles pour le calcul, la traçabilité, et la prescription.

¹⁷ La toricité désigne une caractéristique optique selon laquelle une lentille (ou la cornée) possède deux courbures différentes selon les axes principaux. Elle permet de corriger l'astigmatisme en adaptant la puissance optique à chaque axe.

- LensManufacturer → Fournisseur de la lentille.
- Calcul → Elle relie un examen à un implant suggéré ou choisi, en y associant des détails de calcul.

Relations entre examen, calcul et lens

Un Exam peut générer plusieurs Calcul (ex : simulation de plusieurs formules).

Chaque Calcul peut référencer une et une seule Lens.

Une Lens peut être réutilisée dans plusieurs Calcul.

Représentation logique :

Exam (1) —— (0..n) Calcul (n) —— (1) Lens

Sécurité et audit

Chaque sélection de lentille est enregistrée avec trace d'utilisateur (via audit ou token).

Les implants obsolètes (active = false) sont cachés au frontend.

Évolutions possibles

Intégration d'un moteur de calcul IOL personnalisé (algorithme maison).

Ajout de la gestion des toricités (axe, puissance cylindrique).

Historique complet des implants posés chez un patient (multi-yeux / multi-séjours).

Suggestions intelligentes via un moteur d'aide à la décision (basé sur l'IA ou Règles métier).

4.8.6 Gestion des Accès et Sécurité

Authentification et Permissions

Objectif : Contrôler l'accès des utilisateurs selon leur rôle.

Étapes du processus

- Un utilisateur tente de se connecter avec son email et son mot de passe.
- Le système vérifie les informations d'identification (_User).
- Un token JWT est généré (Token) et stocké pour gérer la session.
- Le rôle de l'utilisateur est récupéré (UserOrganization) :
 - Si ADMIN → accès à tout le système.

- Si MEDECIN → accès aux patients et calculs.
- Si SECRETAIRE → accès limité aux dossiers patients.
- L'utilisateur navigue dans l'application selon ses droits.

Flux des données et relations utilisées

- User → Gère l'identité de l'utilisateur.
- Token → Authentifie la session.
- UserOrganization → Définit les permissions selon l'organisation.

4.8.7 Module Summary (résumé opératoire du patient)

Objectif fonctionnel

Le module Summary permet de centraliser toutes les données cliniques nécessaires à la planification de l'intervention chirurgicale. Il s'agit d'un point d'entrée essentiel pour le chirurgien, lui offrant une vision synthétique et fiable du patient, de son état ophtalmologique, et des décisions prises en amont (calcul de la lentille, traitements, antécédents...).

Il fonctionne comme une fiche opératoire digitale, à usage médical.

Étapes d'accès

L'utilisateur accède au module Summary à partir de la liste des patients, puis en sélectionnant un patient ayant au moins un Exam associé. L'ID du patient est transmis au composant Summary via le routeur Angular : ‘this.router.navigate(['/summary', patient.id]);’;

Structure Angular du module

- component.ts : logique de récupération et de binding des données.
- component.html : mise en page visuelle avec PrimeNG.
- service.ts : communication HTTP avec le backend.
- model.ts : définition stricte du DTO Summary.
- module.ts : déclaration des dépendances Angular.

Backend – Génération du summary

La méthode principale est :

SummaryDto getSummaryByPatientId(Long patientId)

Elle est définie dans le service CataractServiceImpl et agrège des données issues de :

Source	Données utilisées
Patient	Nom, prénom, profession, date de naissance
Exam	Antécédents, traitements, œil dominant, œil opéré, date opération, type chirurgie
Calcul	Lentille sélectionnée, fabricant, puissance
Lens	Paramètres optiques
Autres champs	Valeurs mockées en attendant les entités (matériel, traitements post-op...)

Interface utilisateur Angular

Le composant summary.component.html affiche les données dans une interface utilisateur propre et accessible, en utilisant :

- PrimeNG Card pour la mise en page générale.
- GridLayout (p-grid) pour l'alignement des informations.
- p-badge pour signaler les informations urgentes.
- p-timeline ou p-accordion pour organiser les informations chirurgicales.

Exemple d'extrait HTML :

```
<div class="grid" *ngIf="patient">
  <div class="col">
    <div class="card mb-0">
      <div class="p-fluid p-formgrid grid">

        <!-- Identité patient -->
        <div class="field col-12 md:col-4">
          <label for="lastname">
            Lastname
            <i class="pi pi-exclamation-triangle text-orange-500"
                pTooltip="Non editable field"
                tooltipPosition="top"
                style="font-size: 1rem;"></i>
          </label>
          <div class="flex align-items-center gap-2">
            <input pInputText id="lastname" type="text" [(ngModel)]="patient.lastname" [disabled]="true" />
          </div>
        </div>
        <div class="field col-12 md:col-4">
          <label for="firstname">
            Firstname
            <i class="pi pi-exclamation-triangle text-orange-500"
                pTooltip="Non editable field"
                tooltipPosition="top"
                style="font-size: 1rem;"></i>
          </label>
          <div class="flex align-items-center gap-2">
            <input pInputText id="firstname" type="text" [(ngModel)]="patient.firstname" [disabled]="true" />
          </div>
        </div>
      </div>
    </div>
  </div>
</div>
```

Problématiques techniques rencontrées

Problème	Solution
Champs manquants (matériel, traitements)	Valeurs mockées ou arrays vides
Erreurs de sérialisation	Correction via Jackson (@JsonProperty)
Multiples Exam sans sélection	Ajout de la notion selected = true
Surcharge du DTO	Ajout progressif des champs avec rétrocompatibilité

Améliorations prévues

Amélioration	Objectif
Liaison aux vrais traitements post-opératoires	Intégration d'une entité Treatment
Refonte graphique	Timeline opératoire, export PDF
Ajout du chirurgien connecté	Lier la session à l'utilisateur opérant
Synthèse vocale ou résumé imprimable	Accessibilité patient

4.9 Évolutivité et Scalabilité

4.9.1 Modularité et Extensibilité

Ajout de nouvelles fonctionnalités métier :

Grâce à la structure relationnelle, l'extension du modèle est facilitée. De nouveaux types de calculs peuvent être intégrés simplement en ajoutant des entrées dans Constant et Diopter, sans impacter les structures existantes.

Évolutivité de l'examen biométrique :

La table Examen peut être enrichie pour inclure d'autres mesures biométriques (par exemple, de nouveaux paramètres liés aux diagnostics ophtalmologiques).

Ajout de nouvelles méthodes d'analyse via la table Calcul en intégrant de nouvelles formules ou modèles de prédiction.

Gestion avancée des organisations :

Possibilité d'ajouter facilement de nouveaux types d'organisations médicales dans Organizations.

Flexibilité pour gérer plusieurs établissements pour un même utilisateur grâce à UserOrganization.

Interopérabilité avec des systèmes tiers :

La base de données permet la connexion à des APIs médicales externes pour importer automatiquement les données des appareils médicaux (OCT, topographes cornéens, etc.).

Support de FHIR/HL7 pour assurer une compatibilité avec d'autres systèmes hospitaliers.

4.9.2 *Intégration avec des outils externes*

API REST sécurisée :

Exposition des données via une API RESTful pour permettre aux médecins et administrateurs d'accéder aux informations en temps réel.

Authentification via JWT avec gestion des rôles utilisateurs (Role et UserOrganization).

4.9.3 *Exportation des données médicales :*

Génération de rapports compatibles HL7/FHIR pour assurer la standardisation et l'interopérabilité avec d'autres applications médicales.

Fonctionnalité d'exportation des examens et calculs sous format PDF ou JSON pour intégration avec les dossiers médicaux électroniques (DME).

4.9.4 *Connexion à des services de calcul avancés :*

Possibilité d'intégrer des algorithmes d'IA pour l'analyse prédictive des résultats biométriques. Déploiement d'outils de Machine Learning pour optimiser le choix des implants intraoculaires (Lens).

5. Sécurité, tests et conformité RGPD

5.1 Sécurité applicative : principes et mise en œuvre

EyeWeb manipule des données médicales hautement sensibles. Dès les premières itérations, la sécurité applicative a été intégrée au cœur de l'architecture, dans une approche Security by Design.¹⁸

5.1.1 Backend sécurisé – Spring Security 6

Authentification stateless avec tokens JWT, portés dans les headers des requêtes.

Génération du JWT au login contenant des claims (ID, rôle, langue, etc.).

Filtre personnalisé JwtAuthenticationFilter interceptant les requêtes entrantes pour valider l'authenticité du token.

Centralisation de la configuration dans SecurityConfig avec distinction claire des endpoints publics (/auth/**) et protégés (**).

```
http
    // CORS : configuration personnalisée pour autoriser le frontend Angular
    .cors(cors -> cors.configurationSource(corsConfigurationSource()))
    // CSRF désactivé car l'application est stateless (pas de sessions serveur)
    .csrf(AbstractHttpConfigurer::disable)
    // Configuration des autorisations par route
    .authorizeHttpRequests(authorize -> {
        // Autorisations selon les profils utilisateurs
        //...
    })
    // Le mode stateless évite de stocker l'état utilisateur côté serveur (important pour les JWT)
    .sessionManagement(session -> session.sessionCreationPolicy(STATELESS))
    // Fournisseur d'autentification utilisé (JWT)
    .authenticationProvider(authenticationProvider)
    // Ajout du filtre JWT AVANT le filtre standard d'autentification
    .addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class)
    // Configuration du logout
    .logout(logout -> logout
        .logoutUrl("/auth/logout") // URL de déconnexion
        .addLogoutHandler(logoutHandler) // Comportement personnalisé à la déconnexion
        .logoutSuccessHandler((request, response, authentication) -> SecurityContextHolder.clearContext()) // Nettoyage du contexte de sécurité
    );
}
```

Stockage et contrôle des tokens

Contrairement à une approche entièrement stateless, les tokens JWT sont stockés dans la base (table token) pour :

¹⁸ *Security by Design* : approche de développement logiciel qui consiste à intégrer la sécurité dès la conception de l'application, en anticipant les risques et en renforçant la robustesse des composants critiques, plutôt que d'ajouter des mesures correctives a posteriori.

- Révocation immédiate (déconnexion, changement de rôle) ;
- Historisation des connexions ;
- Application de règles métier (ex. accès temporaire).

5.1.2 Sécurité des mots de passe

- Hachage des mots de passe avec BCrypt (salt intégré) ;
- Pas de retour du mot de passe ni de stockage en clair ;
- Validation forte côté backend.

5.2 Gestion des rôles et des droits d'accès

Mécanisme de RBAC (Role-Based Access Control)

La sécurité repose sur une gestion dynamique des rôles, via les entités User, UserOrganization et Role.

Rôles métiers disponibles :

- ADMIN : gestion complète de la plateforme, audit, utilisateurs, organisations ;
- DOCTOR : saisie d'examens, création de calculs IOL, consultation de dossiers ;
- SECRETARY : ajout de patients, planification ;
- CLIENT / PATIENT : prévu pour un futur accès en lecture.

Filtrage des accès

Backend : protection avec `@PreAuthorize("hasRole('ROLE_DOCTOR')")`, ou `hasAnyRole(...)` ;

Frontend : AuthGuard, RoleGuard, récupération du rôle via un claim contenu dans le token JWT ;

Aucune route protégée n'est accessible sans rôle défini.

5.3 Tests et validation fonctionnelle

5.3.1 Outils utilisés :

Postman : tests des endpoints REST (login, CRUD, sécurités JWT, 401/403) ;

Angular DevTools : vérification des appels HTTP, console et logs ;

Tests manuels documentés par scénario.

5.3.2 Exemples de cas testés :

Fonction testée

Résultat attendu

Login correct

JWT généré, rôle reconnu

Login incorrect

HTTP 401 Unauthorized

Accès à une API sans token

HTTP 403 Forbidden

Création patient incomplète

Message d'erreur bloquant

Validation d'un calcul ophtalmique

Résultat conforme, persisté

Suppression logique d'un utilisateur

Données non supprimées physiquement

Limites actuelles

Pas encore de tests automatisés (Junit, Cypress) ;

Prévision : automatisation des tests E2E via Cypress / Playwright (v2).

5.4 Conformité au RGPD et protection des données

Données considérées comme sensibles :

- Nom, prénom, email, date de naissance, examens médicaux, biométries, etc. ;
 - Aucune donnée sensible n'est exposée en clair ni retournée inutilement par les APIs ;
 - Seuls les champs utiles à l'utilisateur authentifié sont visibles.
 - Consentement
 - Implémenté implicitement lors de la création d'un patient dans le cadre d'un workflow médical ;
 - Une version 2 intégrera une case d'acceptation explicite des CGU/RGPD.
-
- Suppression logique
 - Utilisation d'un champ deleted: boolean sur les entités sensibles (User, Patient, Exam) ;
 - La suppression physique est désactivée pour garantir l'historique médical.

Anonymisation (à venir)

- Objectif : masquer automatiquement les identifiants personnels tout en conservant les données statistiques ;
- Masquage prévu : nom, prénom, ID, email, etc.

5.5 Traçabilité , logs et audit

Journalisation et audit

- Champs createdAt, createdBy, modifiedAt, modifiedBy présents dans toutes les entités principales ;
- Journalisation des connexions via l'entité Token ;
- Traçabilité complète de toutes les actions utilisateur critiques.

Logs

- Intégration avec SLF4J et loggers Spring ;
- Possibilité future d'exposer les logs d'audit via une interface Angular réservée aux ADMIN ;
- Suivi prévu : connexions, modifications de patients, calculs créés, rôles modifiés, etc.

Cette architecture garantit un haut niveau de sécurité, tout en restant modulaire, évolutive, et conforme aux standards européens en matière de protection des données médicales.

6. Conclusion

La réalisation de l'application EyeWeb constitue l'aboutissement d'un projet technique et humain d'envergure, pen-sé dès son origine pour répondre à des enjeux cliniques concrets et critiques en ophtalmologie. Plus qu'une simple application, EyeWeb incarne la rencontre entre les exigences du terrain médical et la rigueur du développement logiciel moderne.

Ce projet a mobilisé une large palette de compétences : analyse des besoins métier, modélisation de données cliniques sensibles, conception d'une base de données relationnelle optimisée, développement sécurisé de services web, intégration d'interfaces utilisateurs ergonomiques, et structuration du code selon les principes d'architecture modulaire et scalable. Chaque étape a été pensée pour garantir à la fois robustesse, clarté et maintenabilité à long terme.

Sur le plan technologique, EyeWeb s'appuie sur un socle solide et cohérent : Spring Boot, Java 17, JPA, JWT, Post-greSQL côté backend ; Angular 17, PrimeNG, RxJS côté frontend. Le choix de ces technologies n'a jamais été arbitraire. Il répond à une logique de pertinence fonctionnelle, de sécurité, de fiabilité et de compatibilité avec les standards du secteur médical. L'authentification sécurisée, la séparation stricte des rôles, la traçabilité des opérations et la gestion fluide du parcours opératoire ne sont que quelques illustrations de l'attention portée à chaque détail métier.

Mais au-delà de la performance technique, EyeWeb témoigne d'une ambition plus vaste : fournir un outil réellement utile aux professionnels de santé. L'interface a été pensée avec les utilisateurs finaux (chirurgien, secrétaire, personnel médical) pour qu'elle s'adapte à leur pratique quotidienne. Le système permet de centraliser l'information, de sécuriser la communication et d'optimiser la préparation préopératoire, dans une logique de qualité des soins et de réduction des risques cliniques.

EyeWeb n'est pas figé. Il offre une base extensible pour l'intégration de modules futurs : systèmes de messagerie sécurisée, intégration HL7/FHIR avec les hôpitaux, tableaux de bord décisionnels, intelligence artificielle pour l'aide au diagnostic... autant de pistes qui laissent entrevoir un potentiel d'évolution significatif, au service d'une médecine de plus en plus connectée, personnalisée et performante.

En définitive, ce travail démontre qu'un projet de fin d'études peut répondre à une problématique métier réelle, tout en mettant en œuvre les bonnes pratiques de l'ingénierie logicielle. Il illustre aussi combien l'informatique médicale peut être un levier d'amélioration tangible pour le personnel de santé et, par conséquent, pour les patients eux-mêmes.

7. Ressources

7.1 Technologies, outils et références techniques

- Auth0. (n.d.). JSON Web Tokens – Introduction. Retrieved March 2025, from <https://auth0.com/learn/json-web-tokens/>
- Bootstrap. (n.d.). Bootstrap · The most popular HTML, CSS, and JS library. Retrieved January 2025, from <https://getbootstrap.com>
- Google. (n.d.). Angular – The modern web developer's platform. Retrieved March 2025, from <https://angular.io>
- JWT.io. (n.d.). JWT Debugger & Library. Retrieved March 2025, from <https://jwt.io>
- Microsoft. (n.d.). TypeScript: JavaScript With Syntax for Types. Retrieved March 2025, from <https://www.typescriptlang.org>
- Oracle. (n.d.). Java SE 17 Documentation. Retrieved February 2025, from <https://docs.oracle.com/en/java/javase/17/>
- PostgreSQL Global Development Group. (n.d.). PostgreSQL Documentation. Retrieved March 2025, from <https://www.postgresql.org/docs/>
- Postman. (2024). Postman API Platform. Retrieved March 2025, from <https://www.postman.com/>
- PrimeFaces. (n.d.). PrimeNG UI Components for Angular. Retrieved March 2025, from <https://www.primefaces.org/primeng/>
- ReactiveX. (n.d.). RxJS: Reactive Extensions for JavaScript. Retrieved March 2025, from <https://rxjs.dev>
- Spring. (n.d.-a). Spring Boot Project. Retrieved February 2025, from <https://spring.io/projects/spring-boot>
- Spring. (n.d.-b). Spring Initializr. Retrieved February 2025, from <https://start.spring.io>
- Stack Overflow. (n.d.). Where Developers Learn, Share, & Build Careers. Retrieved March 2025, from <https://stackoverflow.com>
- Swagger. (n.d.). OpenAPI Specification. Retrieved March 2025, from <https://swagger.io/specification/>
- Visual Studio Code. (n.d.). Code editing. Redefined. Retrieved January 2025, from <https://code.visualstudio.com>

7.2 Ophtalmologie et médecine

- American Academy of Ophthalmology. (n.d.). AAO.org. Retrieved January 2025, from <https://www.aao.org>
- EyeWiki. (n.d.). EyeWiki – Collaborative Ophthalmic Knowledge. Retrieved May 2025, from <https://www.eyewiki.org>
- Gatinel, L. (n.d.). Ophtalmologie, chirurgie de la cataracte et de la cornée. Retrieved March 2025, from <https://www.gatinel.com>
- National Institutes of Health. (n.d.-a). PubMed. Retrieved February 2025, from <https://pubmed.ncbi.nlm.nih.gov>
- National Institutes of Health. (n.d.-b). PMC – PubMed Central. Retrieved February 2025, from <https://www.ncbi.nlm.nih.gov/pmc/>
- Ophthalmic Calculators. (n.d.). IOL Calculator. Retrieved April 2025, from <https://iolcalculator.com>

7.3 Sécurité des applications et RGPD

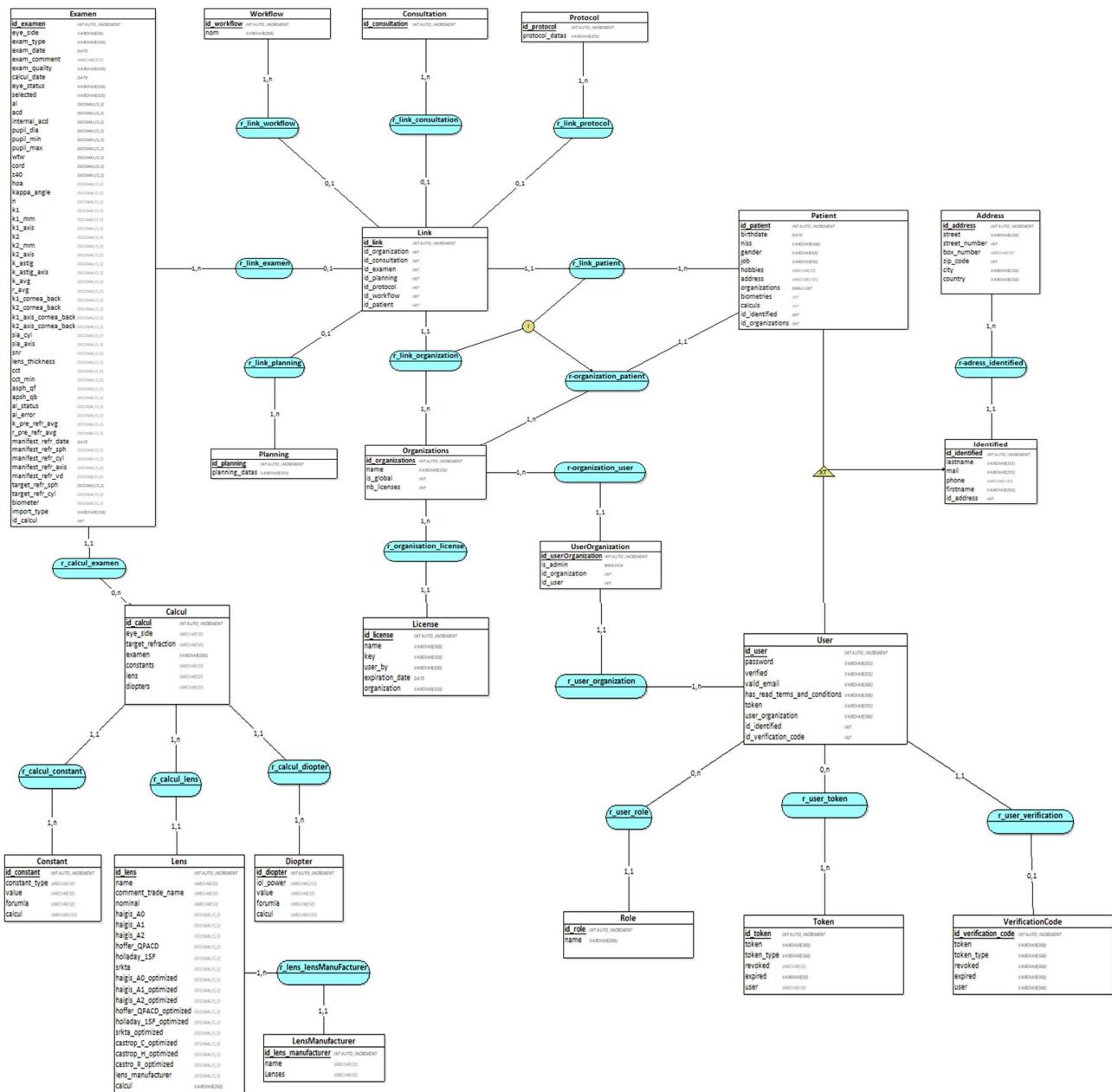
- CNIL. (n.d.). Commission Nationale de l’Informatique et des Libertés. Retrieved January 2025, from <https://www.cnil.fr>
- DigitalOcean. (n.d.). Community Tutorials. Retrieved March 2025, from <https://www.digitalocean.com/community/tutorials>
- Dev.to. (n.d.). Developer Community. Retrieved April 2025, from <https://dev.to>
- OWASP Foundation. (n.d.). OWASP Top 10 – Web Application Security Risks. Retrieved February 2025, from <https://owasp.org>

7.4 Bibliographie – Méthodologie Agile et SCRUM

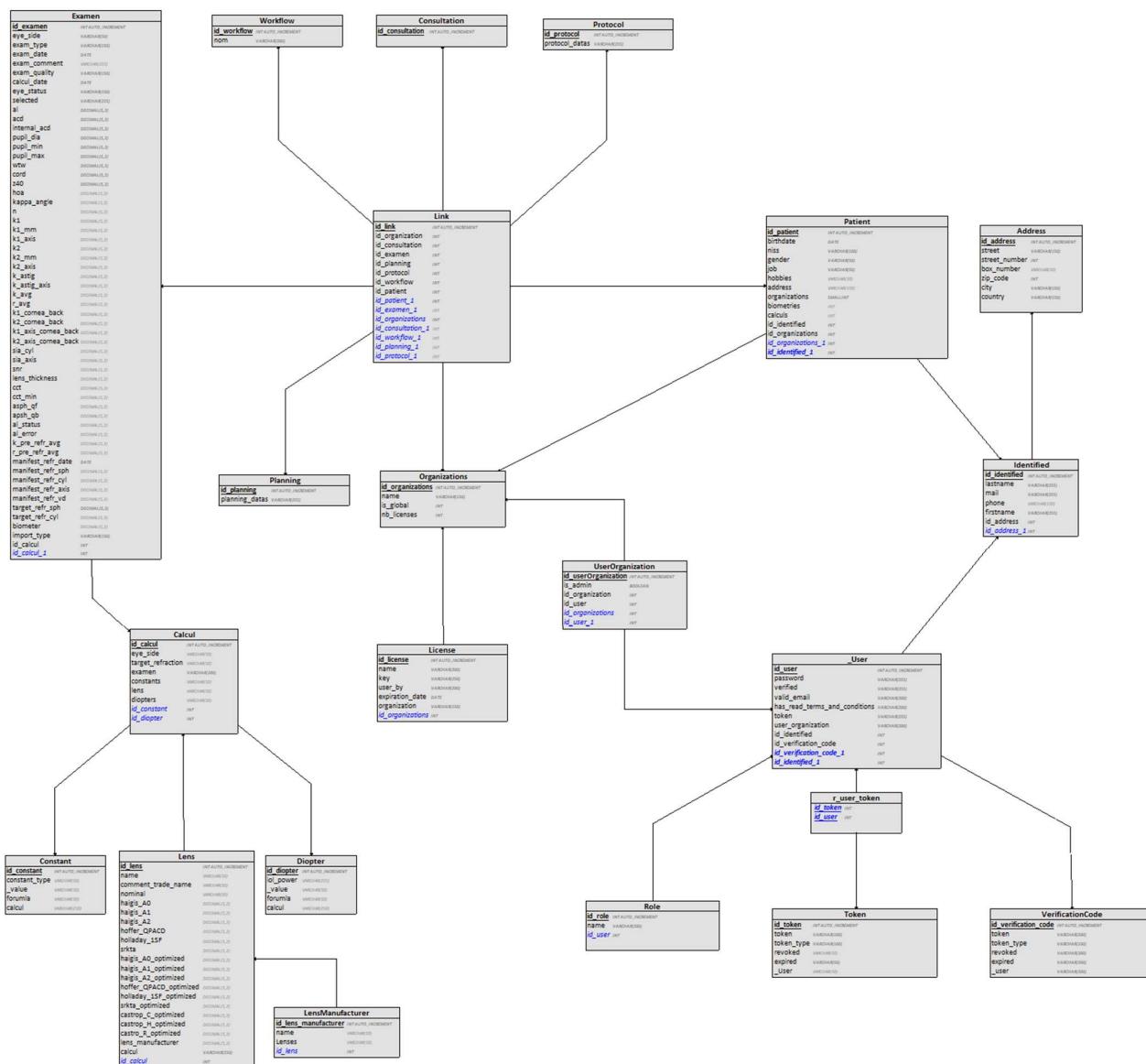
- Agile Alliance. (n.d.). What is Agile Software Development? Retrieved May 12, 2025, from <https://www.agilealliance.org/agile101>
- MIT OpenCourseWare. (n.d.). Software Engineering for Web Applications. Retrieved May 12, 2025, from <https://ocw.mit.edu>
- Schwaber, K., & Sutherland, J. (2020). The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game. Scrum.org. <https://www.scrum.org/resources/scrum-guide>

8. ANNEXES

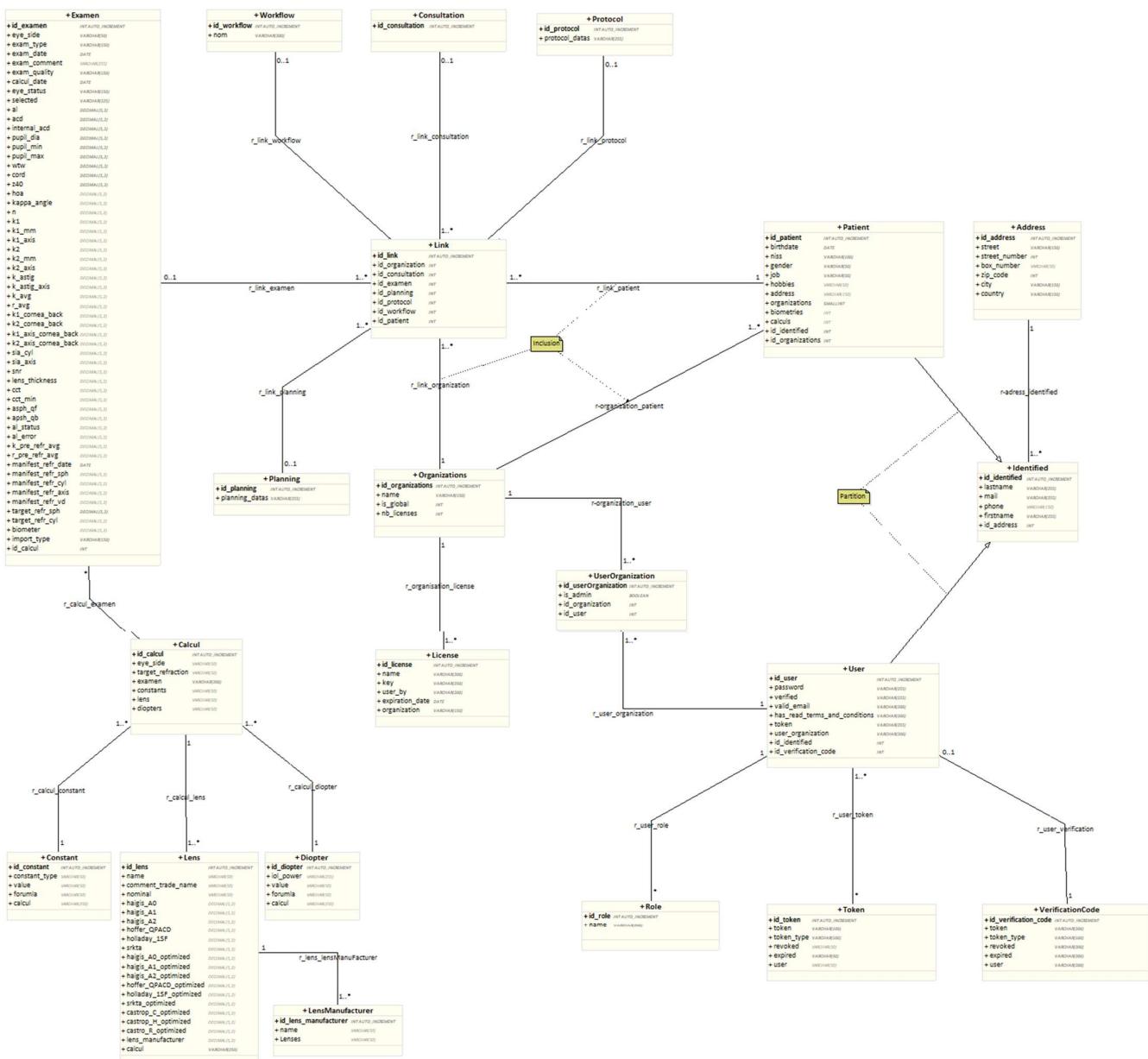
8.1 Annexe I – Modèle Conceptuel de Données (MCD)



8.2 Annexe II - Modèle Logique de Données (MLD)



8.3 Annexe III - Modèle LogiqUnified Modeling Language (UML)



8.4 Annexe III – Thésaurus - Dictionnaire des données

8.4.1 Entité : Token

8.4.1.1 Description

La table Token gère les jetons d'authentification des utilisateurs via JWT (JSON Web Token). Elle permet de gérer la session d'un utilisateur, la validation des accès et la révocation des tokens.

8.4.1.2 Relations et Contraintes

Relation	Type	Explication
1-N avec User	Relation	Un utilisateur peut avoir plusieurs tokens actifs en même temps.
id_user (clé étrangère)	Relation	Référence vers l'utilisateur détenteur du token.

8.4.1.3 Contraintes d'Intégrité

Un token doit être lié à un utilisateur (id_user non nul)

Contrainte SQL : ALTER TABLE Token ADD CONSTRAINT fk_token_user

FOREIGN KEY (id_user) REFERENCES User(id_user) ON DELETE CASCADE;

Explication : Lorsqu'un utilisateur est supprimé, tous ses tokens sont supprimés automatiquement.

Un token ne peut pas être révoqué et valide en même temps

Contrainte SQL : ALTER TABLE Token ADD CONSTRAINT chk_token_validity

CHECK (revoked = FALSE OR expired = FALSE);

Explication : Un token est soit actif, soit expiré/révoqué, mais jamais les deux en même temps.

8.4.1.4 Détails

Attributs	Types	Définition	Exemples
id_token	INT AUTO_INCREMENT	Identifiant unique du token	1

token	VARCHAR(500)	Valeur du token	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
tokenType	VARCHAR(50)	Type du token	BEARER
revoked	BOOLEAN	Indique si le token est révoqué	FALSE
expired	BOOLEAN	Indique si le token est expiré	FALSE
id_user	INT	Référence vers un utilisateur	2

8.4.2 Entité : Address

8.4.2.1 Description

La table Address stocke les informations d'adresse des utilisateurs et des patients.

Elle est référencée par Identified, permettant ainsi à chaque utilisateur et patient d'avoir une adresse unique.

8.4.2.2 Relations et Contraintes

Relation	Type	Explication
1-N avec Identified	Relation	Une adresse peut être associée à plusieurs identités (User, Patient).
id_address (clé étrangère)	Relation	Référencée par la table Identified.

8.4.2.3 Contraintes d'Intégrité

Une adresse doit être unique pour chaque entité Identified

Contrainte SQL : ALTER TABLE Identified ADD CONSTRAINT fk_identified_address FOREIGN KEY (id_address) REFERENCES Address(id_address) ON DELETE SET NULL;
 Explication : Si une adresse est supprimée, la référence dans Identified est mise à NULL.

Un code postal doit être numérique

Contrainte SQL : ALTER TABLE Address ADD CONSTRAINT chk_address_zipcode CHECK (zip_code ~ '^[0-9]+\\$');

Explication : Assure que le code postal ne contient que des chiffres.

8.4.2.4 Détails des Attributs

Attributs	Types	Définition	Exemples
id_address	INT AUTO_INCREMENT	Identifiant unique de l'adresse	1
street	VARCHAR(255)	Rue	rue Saint-Laurent
street_number	VARCHAR(10)	Numéro	42
box_number	VARCHAR(10)	Numéro de boîte	2
zip_code	VARCHAR(10)	Code postal	4000
city	VARCHAR(100)	Ville	Liège
country	VARCHAR(100)	Pays	Belgique

8.4.3 Entité : *Organizations*

8.4.3.1 Description

La table Organizations stocke les organisations médicales auxquelles peuvent appartenir les utilisateurs et les patients.

8.4.3.2 Relations

Relation	Type	Explication
1-N avec Patient	Relation	Une organisation peut regrouper plusieurs patients.
1-N avec UserOrganization	Relation	Une organisation peut avoir plusieurs utilisateurs affiliés.
id_organizations (clé étrangère)	Relation	Référencé par Patient, License, UserOrganization.

8.4.3.3 Contraintes d'Intégrité

Un patient doit toujours être rattaché à une organisation (id_organizations non nul)

Contrainte SQL :ALTER TABLE Patient ADD CONSTRAINT fk_patient_organization
FOREIGN KEY (id_organizations) REFERENCES Organizations(id_organizations) ON
DELETE CASCADE;

→ Explication : Lorsqu'une organisation est supprimée, tous ses patients sont supprimés.

Une organisation ne peut pas avoir un nombre négatif de licences

```
Contrainte      SQL      :ALTER      TABLE      Organizations      ADD      CONSTRAINT
chk_organization_licenses
CHECK (nb_licenses >= 0);
```

Explication : Une organisation ne peut pas avoir un nombre négatif de licences actives.

8.4.3.4 Détails des Attributs

Attributs	Types	Définition	Exemples
id_organizations	INT AUTO_INCREMENT	Identifiant unique de l'organisation	1
name	VARCHAR(255)	Nom de l'organisation	C.H.R
is_global	BOOLEAN	Indique si l'organisation est globale	TRUE
nb_licenses	nb_licenses	nb_licenses	10

8.4.4 Entité : Patient

8.4.4.1 Description

La table Patient stocke les informations médicales et administratives des patients, ainsi que leur affiliation à une organisation médicale.

Chaque patient est associé à une identité unique dans Identified, permettant une gestion cohérente des données personnelles et de contact.

Un patient peut être lié à plusieurs examens et calculs intraoculaires, ainsi qu'à une organisation médicale spécifique où il est suivi.

8.4.4.2 Relations et Contraintes

Relation	Type	Explication
1-1 avec Identified	Héritage	Un patient hérite de Identified pour stocker ses informations personnelles et de contact.
1-N avec Organizations	Relation	Un patient est affilié à une seule organisation médicale.
1-N avec Examen	Relation	Un patient peut avoir plusieurs examens médicaux.
1-N avec Calcul	Relation	Un patient peut avoir plusieurs calculs biométriques réalisés.
1-N avec Consultation	Relation	Un patient peut être suivi dans plusieurs consultations médicales.

8.4.4.3 Héritage et Contraintes d'Inclusion

8.4.4.3.1 Héritage

- Identified est une entité parent utilisée pour stocker des informations communes aux entités User et Patient.
- User et Patient héritent de Identified, garantissant que chaque enregistrement est associé à une identité unique.

8.4.4.3.2 Contrainte d'inclusion

- id_organizations IS NOT NULL pour assurer qu'un patient appartient toujours à une organisation.
- Patient doit appartenir à une organisation

```
ALTER TABLE Patient ADD CONSTRAINT chk_patient_organization  
CHECK (id_organizations IS NOT NULL);
```

- User doit avoir une identité référencée

```
ALTER TABLE User ADD CONSTRAINT chk_user_identified  
CHECK (id_identified IS NOT NULL);
```

8.4.4.3.3 Trigger SQL

```
CREATE TRIGGER prevent_patient_deletion  
BEFORE DELETE ON Patient  
FOR EACH ROW  
WHEN EXISTS (SELECT 1 FROM Examen WHERE Examen.id_patient = OLD.id_patient)  
OR EXISTS (SELECT 1 FROM Calcul WHERE Calcul.id_patient = OLD.id_patient)  
BEGIN  
    RAISE EXCEPTION 'Impossible de supprimer un patient avec des examens ou calculs  
actifs';  
END;
```

Explication : Cette contrainte empêche la suppression accidentelle d'un patient qui possède encore des données médicales liées.

8.4.4.4 Détails des Attributs

Attributs	Types	Définition	Exemples
id_patient	INT AUTO_INCREMENT	Identifiant unique du patient	1
birthdate	DATE	Date de naissance du patient	03-12-85
niss	VARCHAR(100)	Numéro national du patient	85-12-03.1
gender	VARCHAR(50)	Genre du patient	M
job	VARCHAR(50)	Profession du patient	Ingénieur
hobbies	VARCHAR(50)	Loisirs du patient	Randonnée
Id_identified	INT	Référence vers l'entité Identified	2
id_organizations	INT	Référence vers l'organisation	1

8.4.5 Entité : License

8.4.5.1 Description

La table License stocke les licences attribuées aux organisations.

Elle permet de suivre l'activation des licences, leur expiration et leur utilisateur responsable.

8.4.5.2 Relations et contraintes

Relation	Type	Explication
1-N avec Organizations	Relation	Une organisation peut posséder plusieurs licences actives.
id_organizations (clé étrangère)	Relation	Référence vers Organizations pour associer la licence à une organisation.

8.4.5.3 Contraintes d'Intégrité

Une license doit être toujours associée à une organisation (id_organizations non NULL)

→ Contrainte SQL : ALTER TABLE License ADD CONSTRAINT fk_license_organization

FOREIGN KEY (id_organizations) REFERENCES Organizations(id_organizations) ON DELETE CASCADE;

Explication : Lorsqu'une organisation est supprimée, toutes ses licences sont supprimées.

La clé de license doit être unique

→ Contrainte SQL :
ALTER TABLE License ADD CONSTRAINT unique_license_key
UNIQUE (key);

Explication : Une même clé de licence ne peut pas être attribuée à plusieurs organisations.

8.4.5.4 Détails

Attributs	Types	Définition	Exemples
id_license	INT AUTO_INCREMENT	Identifiant unique pour license	1
name	VARCHAR(200)	Nom de la licence	PREMIUM
key	VARCHAR(100)	Clé de la license.	L_0001
user_by	VARCHAR(255)	Utilisateur ayant activé la license.	ADMIN
expiration_date	VARCHAR(20)	Date d'expiration.	2025-12-31
id_organizations	INT	Organisation associée.	2

8.4.6 Entité : VerificationCode

8.4.6.1 Description

La table VerificationCode stocke les codes de validation utilisés pour l'authentification et la vérification des utilisateurs lors de leur inscription ou de la récupération de mot de passe.

8.4.6.2 Relations et Contraintes

Relation	Type	Explication
1-1 avec User	Relation	Chaque utilisateur peut avoir un seul code de vérification actif.
id_user (clé étrangère)	Relation	Référence vers User.

8.4.6.3 Contraintes d'Intégrité

Un utilisateur ne peut pas avoir plusieurs codes de vérifications actifs

Contrainte SQL :
ALTER TABLE VerificationCode ADD CONSTRAINT
unique_verification_code
UNIQUE (id_user);

Explication : Un utilisateur ne peut avoir qu'un seul code actif à la fois.

Un code ne peut pas être révoqué et valide en même temps

Contrainte SQL : ALTER TABLE VerificationCode ADD CONSTRAINT chk_verification_validity
CHECK (revoked = FALSE OR expired = FALSE);

Explication : Un code est soit valide, soit révoqué ou expiré, mais jamais les deux en même temps.

8.4.6.4 Détails des Attributs

Attributs	Types	Définition	Exemples
id_verification_code	INT AUTO_INCREMENT	Identifiant unique du code	1
token	VARCHAR(100)	Code de validation	A1B2C3
token_type	VARCHAR(50)	Type du token	EMAIL_CONFIRMATION
revoked	BOOLEAN	Indique si le code est révoqué	FALSE
expired	BOOLEAN	Indique si le code est expiré	FALSE
id_user	INT	Référence vers un utilisateur	3

8.4.7 Entité : Examen

8.4.7.1 Description

La table Examen stocke les examens biométriques réalisés sur les patients, comprenant des mesures ophtalmologiques essentielles.

8.4.7.2 Relations et Contraintes

Relation	Type	Explication
1-N avec Patient	Relation	Un patient peut avoir plusieurs examens enregistrés.
id_patient (clé étrangère)	Relation	Référence vers Patient pour identifier le patient examiné.
1-N avec Calcul	Relation	Un examen peut être utilisé pour plusieurs calculs intraoculaires (id_calcul).

8.4.7.3 Contraintes d'Intégrité

Chaque examen doit être associé à un patient (id_patient non nul)

→ Contrainte SQL : ALTER TABLE Examen ADD CONSTRAINT fk_examen_patient
FOREIGN KEY (id_patient) REFERENCES Patient(id_patient) ON DELETE
CASCADE;

Explication : Lorsqu'un patient est supprimé, tous ses examens sont également supprimés.

Le type d'examen doit être valide (oct, topographie, aberrométrie, etc.)

→ Contrainte SQL : ALTER TABLE Examen ADD CONSTRAINT chk_examen_type
CHECK (exam_type IN ('OCT', 'Topographie', 'Aberrométrie', 'Pachymétrie',
'Biométrie'));

Explication : Empêche les valeurs invalides dans exam_type.

La qualité de l'examen doit être bien renseignée (haute, moyenne, basse)

→ Contrainte SQL : ALTER TABLE Examen ADD CONSTRAINT chk_examen_quality
CHECK (exam_quality IN ('Haute', 'Moyenne', 'Basse'));

Explication : Garantit la standardisation des données de qualité.

8.4.7.4 Détails

Attributs	Types	Définition	Exemples
id_examen	INT AUTO_INCREMENT	Identifiant unique de l'examen	1
eye_side	VARCHAR(25)	Côté de l'œil concerné (OD = œil droit, OG = œil gauche)	OD
exam_type	VARCHAR(25)	Type d'examen effectué	OCT
exam_date	DATE	Date de l'examen	2025-01-05
exam_comment	VARCHAR(255)	Commentaire sur l'examen	Bien déroulé
exam_quality	VARCHAR(150)	Qualité de l'examen	Haute
calcul_date	DATE	Date du calcul effectué	2025-06-02
eye_status	VARCHAR(150)	Statut de l'œil examiné	Stable
selected	VARCHAR(225)	Indique si cet examen est sélectionné	Oui
acd	DECIMAL(5,2)	Profondeur de la chambre antérieure	3.45
internal_acd	DECIMAL(5,2)	Profondeur interne de la chambre antérieure	3.20
pupil_dia	DECIMAL(5,2)	Diamètre pupillaire	4.50

pupil_min	DECIMAL(5,2)	Diamètre pupillaire minimum	2.50
pupil_max	DECIMAL(5,2)	Diamètre pupillaire maximum	6.00
wtw	DECIMAL(5,2)	Largeur cornéenne blanche à blanche	11.80
cordk	DECIMAL(5,2)	Distance centrale cornéenne	0.50
z40	DECIMAL(5,2)	Aberration de 4e ordre Zernike	0.10
hoa	DECIMAL(5,2)	Aberration optique de haut niveau	0.12
kappa_angle	DECIMAL(5,2)	Angle kappa	5.6
n	DECIMAL(5,2)	Rayon de courbure K1	42.75
k1	DECIMAL(5,2)	Rayon de courbure K1	42.75
k1_mm	DECIMAL(5,2)	Rayon de courbure K1 en mm	7.80
k1_axis	DECIMAL(5,2)	Axe de l'astigmatisme K1	90
k2	DECIMAL(5,2)	Rayon de courbure K2	44.50
k2_mm	DECIMAL(5,2)	Rayon de courbure K2 en mm	7.60
k2_axis	DECIMAL(5,2)	Axe de l'astigmatisme K2	180
k_astig	DECIMAL(5,2)	Astigmatisme cornéen	1.75
k_astig_axis	DECIMAL(5,2)	Axe de l'astigmatisme	170
k_avg	DECIMAL(5,2)	Moyenne des rayons K	43.80
k1_cornea_back	DECIMAL(5,2)	Rayon de courbure K1 de la face postérieure de la cornée	42.50
k2_cornea_back	DECIMAL(5,2)	Rayon de courbure K2 de la face postérieure de la cornée	44.00
k1_axis_cornea_back	DECIMAL(5,2)	Axe K1 de la face postérieure de la cornée	85
k2_axis_cornea_back	DECIMAL(5,2)	Axe K2 de la face postérieure de la cornée	175
cct	DECIMAL(5,2)	Épaisseur centrale de la cornée	550
cct_min	DECIMAL(5,2)	Épaisseur minimale centrale de la cornée	530
sia_cyl	DECIMAL(5,2)	Astigmatisme chirurgical induit	0.50
sia_axis	DECIMAL(5,2)	Axe de l'astigmatisme chirurgical induit	95
snr	DECIMAL(5,2)	Rapport signal/bruit	25.00
lens_thickness	DECIMAL(5,2)	Épaisseur du cristallin	4.00
asph_qf	DECIMAL(5,2)	Asphericité frontale de la cornée	0.20
apsh_qb	DECIMAL(5,2)	Asphericité postérieure de la cornée	0.10
al_status	DECIMAL(5,2)	Statut de la longueur axiale	23.75
al_error	DECIMAL(5,2)	Erreur de mesure de la longueur axiale	0.02
k_pre_refr_avg	DECIMAL(5,2)	Moyenne de la réfraction préopératoire	43.25
r_pre_refr_avg	DECIMAL(5,2)	Rayon moyen de la réfraction préopératoire	7.80
manifest_refr_date	DATE	Date de la réfraction manifeste	2025-06-02
manifest_refr_sph	DECIMAL(5,2)	Sphère de la réfraction manifeste	-2.50
manifest_refr_cyl	DECIMAL(5,2)	Cylindre de la réfraction manifeste	0.75
manifest_refr_axis	DECIMAL(5,2)	Axe de la réfraction manifeste	90

manifest_refr_vd	DECIMAL(5,2)	Distance vertex en mm	12.00
target_refr_sph	DECIMAL(5,2)	Sphère de la réfraction cible	-2.00
target_refr_cyl	DECIMAL(5,2)	Cylindre de la réfraction cible	0.50
biometer	DECIMAL(5,2)	Longueur axiale mesurée	23.75
importType	VARCHAR(25)	Type d'importation des données	Manuel
id_calcul	INT	Identifiant du calcul associé	123

8.4.8 Entité : Calcul

8.4.8.1 Description

La table Calcul stocke les résultats des calculs biométriques nécessaires aux interventions ophtalmologiques. Ces calculs permettent de déterminer la réfraction cible et l'implant intraoculaire optimal pour le patient.

8.4.8.2 Relations et contraintes

Relation	Type	Explication
1-N avec Patient	Relation	Un patient peut avoir plusieurs calculs biométriques.
1-N avec Examen	Relation	Un calcul est basé sur un examen biométrique existant.
1-N avec Constant et Diopter	Relation	Un calcul peut utiliser plusieurs constantes et dioptres.
1-N avec Lens	Relation	Un calcul peut proposer plusieurs implants intraoculaires.

8.4.8.3 Contraintes d'Intégrité

Un calcul doit être toujours associé à un patient (id_patient non nul)

→ Contrainte SQL : ALTER TABLE Calcul ADD CONSTRAINT fk_calcul_patient FOREIGN KEY (id_patient) REFERENCES Patient(id_patient) ON DELETE CASCADE;

Un calcul doit être basé sur un examen existant (id_examen non nul)

→ Contrainte SQL : ALTER TABLE Calcul ADD CONSTRAINT fk_calcul_examen FOREIGN KEY (id_examen) REFERENCES Examen(id_examen);

un calcul doit avoir une constante biométrique (id_constant non nul)

→ Contrainte SQL : ALTER TABLE Calcul ADD CONSTRAINT fk_calcul_constant
FOREIGN KEY (id_constant) REFERENCES Constant(id_constant);

8.4.8.4 Détails des Attributs

Attributs	Types	Définition	Exemples
id_calcul	INT AUTO_INCREMENT	Identifiant unique du calcul	1
eye_side	VARCHAR(25)	Côté de l'œil concerné	OD
target_refraction	DECIMAL(5,2)	Réfraction cible	-2.50
examen	INT	Référence vers un examen	5
constants	INT	Référence vers des constantes	8
lens	INT	Référence vers une lentille	12
diopters	INT	Référence vers une dioptrie	15
id_constant	INT	Identifiant de la constante associée	18
id_diopter	INT	Identifiant de la dioptrie associée	3
id_patient	INT	Identifiant du patient associé	6

8.4.9 Entité : UserOrganization

8.4.9.1 Description

La table UserOrganization établit une relation entre les utilisateurs et les organisations. Elle définit les rôles des utilisateurs au sein des différentes organisations.

8.4.9.2 Relations et contraintes

Relation	Type	Explication
N-N entre User et Organizations	Relation	Un utilisateur peut être affilié à plusieurs organisations et vice-versa.
1-N avec Role	Relation	Chaque utilisateur dans une organisation a un rôle spécifique (Admin, Médecin, Secrétaire).

8.4.9.3 Contraintes d'Intégrité

Un utilisateur ne peut pas être lié à plusieurs organisations avec le même rôle

→ Contrainte SQL : ALTER TABLE UserOrganization ADD CONSTRAINT unique_user_org_role UNIQUE (id_user, id_organizations);

Un utilisateur doit avoir une organisation définie

→ Contrainte SQL : ALTER TABLE UserOrganization ADD CONSTRAINT fk_userOrganization_organization FOREIGN KEY (id_organizations) REFERENCES Organizations(id_organizations) ON DELETE CASCADE;

8.4.9.4 Détails des Attributs

Attributs	Types	Définition	Exemples
id_userOrganization	INT AUTO_INCREMENT	Identifiant unique de l'association	10
is_admin	BOOLEAN	Indique si l'utilisateur est admin	TRUE
id_user	INT	Référence vers une organisation	2
id_organizations	INT	Référence vers l'organisation concernée	5

8.4.10 Entité : Link

8.4.10.1 Description

La table Link est une table de faits qui établit les relations entre différentes entités médicales.

8.4.10.2 Relations et contraintes

Relation	Type	Explication
1-N avec Patient, Consultation, Workflow, Planning, Examen, Protocol	Relation	Permet de lier ces entités médicales entre elles.

8.4.10.3 Contraintes d'Intégrité

Une entité liée doit exister (id_patient, id_consultation, etc. non nuls)

→ Contrainte SQL : ALTER TABLE Link ADD CONSTRAINT fk_link_patient
FOREIGN KEY (id_patient) REFERENCES Patient(id_patient);

8.4.10.4 Détails des Attributs

Attributs	Types	Définition	Exemples
id_link	INT AUTO_INCREMENT	Identifiant unique du lien.	10
id_consultation	INT	Référence vers une consultation.	3
id_organization	INT	Référence vers l'organisation concernée.	5
id_examen	INT	Référence vers l'examen.	7
id_planning	INT	Référence vers le planning.	4
id_workflow	INT	Référence vers le workflow.	2
id_patient	INT	Référence vers le patient concerné.	9
id_protocol	INT	Référence vers le protocol concerné.	5

8.4.11 Entité : r_user_token

8.4.11.1 Description

Table d'association entre User et Token pour gérer l'authentification.

8.4.11.2 Relations et contraintes

Relation	Type	Explication
N-N entre _User et Token	Relation	Un utilisateur peut avoir plusieurs tokens, et un token peut être associé à plusieurs utilisateurs.

8.4.11.3 Détails

Attributs	Types	Définition	Exemples
id_token	INT AUTO_INCREMENT	Identifiant unique du fabricant	1
id_user	VARCHAR(100)	Nom du fabricant	Essilor

8.4.12 Entité : Identified

8.4.12.1 Description

Stocke les informations d'identification des utilisateurs et des patients. Cette table est une table de base utilisée par les entités User et Patient.

8.4.12.2 Relations et contraintes

Relation	Type	Explication
1-1 avec User et Patient	Héritage	Chaque utilisateur et patient a une identité unique.
1-1 avec Address	Relation	Chaque identité est associée à une adresse unique.

8.4.12.3 Contraintes d'Intégrité

Un patient et un utilisateur doivent avoir une identité définie

→ Contrainte SQL : ALTER TABLE Patient ADD CONSTRAINT fk_patient_identified
FOREIGN KEY (id_identified) REFERENCES Identified(id_identified);

8.4.12.4 Détails des Attributs

Attributs	Types	Définition	Exemples
id_identified	INT AUTO_INCREMENT	Identifiant unique	1
firstname	VARCHAR(100)	Prénom	Vincent
lastname	VARCHAR(100)	Nom	Vanhees
mail	VARCHAR(255)	Adresse e-mail	vincent.vanhees@hotmail.com
phone	VARCHAR(20)	Numéro de téléphone	0499887756
id_address	INT	Référence vers une adresse	3

8.4.13 Entité : User

8.4.13.1 Description

La table User stocke les informations des utilisateurs inscrits dans le système.

Cela inclut les médecins, administrateurs, secrétaires médicales et tout autre personnel ayant un accès à la plate-forme. Elle est fortement sécurisée car elle contient des données sensibles (mots

de passe, statuts d'authentification). L'authentification repose sur un système de tokens JWT et la validation d'email se fait via un code de vérification.

8.4.13.2 Relations

Relation	Type	Explication
1-1 avec Identified	Héritage	Chaque User possède une identité unique stockée dans Identified.
1-N avec Token	Relation	Un utilisateur peut avoir plusieurs sessions actives stockées dans Token.
1-N avec VerificationCode	Relation	Un utilisateur peut générer plusieurs codes de vérification stockés dans VerificationCode.
N-N avec Organizations via UserOrganization	Relation	Un utilisateur peut appartenir à plusieurs organisations via la table pivot UserOrganization.
1-N avec Role	Relation	Un utilisateur peut avoir plusieurs rôles stockés dans Role.

8.4.13.3 Héritage et Contraintes d'Inclusion

L'entité User hérite de Identified et permet donc de garantir que tout utilisateur possède une identité unique définie dans Identified.

8.4.13.4 Contraintes d'Inclusion

L'identifiant `id_identified` doit être unique pour chaque utilisateur afin d'assurer qu'un utilisateur est toujours rattaché à une identité

→ Contrainte SQL : `ALTER TABLE User ADD CONSTRAINT fk_user_identified FOREIGN KEY (id_identified) REFERENCES Identified(id_identified) ON DELETE CASCADE;`

Explication : Si l'entrée associée dans Identified est supprimée, l'utilisateur est supprimé aussi (ON DE-LETE CASCADE) et assure donc une relation 1-1 stricte entre User et Identified.

L'email de l'utilisateur doit être valide avant activation (`valid_email = TRUE`)

→ Contrainte SQL : `ALTER TABLE User ADD CONSTRAINT chk_valid_email CHECK (valid_email = TRUE OR verified = FALSE);`

Explication : Un utilisateur ne peut pas être activé (verified = TRUE) si son email n'a pas été validé et ne peut donc pas être supprimé s'il possède des tokens actifs

8.4.13.5 Trigger SQL :

```
CREATE TRIGGER prevent_user_deletion
BEFORE DELETE ON User
FOR EACH ROW
WHEN EXISTS (SELECT 1 FROM Token WHERE Token.id_user = OLD.id_user)
BEGIN
    RAISE EXCEPTION 'Impossible de supprimer un utilisateur avec des tokens actifs';
END;
```

Explication : Cela empêche la suppression accidentelle d'un utilisateur qui a encore des sessions ouvertes.

8.4.13.6 Détails des Attributs

Attributs	Types	Définition	Exemples
id_user	INT AUTO_INCREMENT	Identifiant unique de l'utilisateur	1
password	VARCHAR(255)	Mot de passe hashé	*****
verified	BOOLEAN	Indique si l'utilisateur est vérifié	TRUE
valid_email	BOOLEAN	Indique si l'adresse e-mail est valide	TRUE
has_read_terms_and_conditions	BOOLEAN	Indique si l'utilisateur a lu les CGU	TRUE
token	VARCHAR(255)	Jeton d'authentification de l'utilisateur	eyJhbghui0zerb
user_organization	VARCHAR(200)	Organisation de l'utilisateur	ADMIN
id_token	INT	Référence vers un token	4

id_verification_code	INT	Référence vers un code de vérification	5
id_identified	INT	Référence vers une identité associée	2

L'attribut role est une relation @ManyToOne vers l'entité Role, permettant une gestion dynamique des permissions.

8.4.14 Entité : Workflow

8.4.14.1 Description

La table Workflow représente les flux de travail médicaux liés aux examens, calculs et interventions chirurgicales.

8.4.14.2 Relations et Contraintes

Relation	Type	Explication
1-N avec Link	Relation	Un workflow peut être associé à plusieurs entités médicales.

8.4.14.3 Contraintes d'Intégrité

Un workflow doit être unique et avoir un nom distinct.

Contrainte SQL : ALTER TABLE Workflow ADD CONSTRAINT unique_workflow_name UNIQUE (nom);

8.4.14.4 Détails des Attributs

Attributs	Types	Définition	Exemples
d_workflow	INT AUTO_INCREMENT	Identifiant unique du workflow	1
nom	VARCHAR(200)	Nom du workflow	pré-chirurgie

8.4.15 Entité : Consultation

8.4.15.1 Description

La table Consultation stocke les consultations médicales effectuées par les praticiens.

8.4.15.2 Relations et contraintes

Attributs	Types	Définition	Exemples
d_workflow	INT AUTO_INCREMENT	Identifiant unique du workflow	1
nom	VARCHAR(200)	Nom du workflow	pré-chirurgie

8.4.15.3 Détails des Attributs

Attributs	Types	Définition	Exemples
id_consultation	INT AUTO_INCREMENT	Identifiant unique de la consultation	1

8.4.16 Entité : Protocol

8.4.16.1 Description

La table Protocol stocke les protocoles médicaux liés aux patients et aux interventions.

8.4.16.2 Relations et contraintes

Relation	Type	Explication
1-N avec Link	Relation	Un protocole peut être associé à plusieurs examens et traitements.

8.4.16.3 Détails des Attributs

Attributs	Types	Définition	Exemples
id_protocol	INT AUTO_INCREMENT	Identifiant unique du protocole	1
protocol_datas	VARCHAR(255)	Données du protocole	"Préparation chirurgie"

8.4.17 Entité : Planning

8.4.17.1 Description

La table Planning stocke les informations de planification médicale (rendez-vous, interventions, etc.).

8.4.17.2 Relations et contraintes

Relation	Type	Explication
1-N avec Link	Relation	Un planning peut être associé à plusieurs consultations et workflows.

8.4.17.3 Détails des Attributs

Attributs	Types	Définition	Exemples
id_planning	INT AUTO_INCREMENT	Identifiant unique du planning	1
planning_datas	VARCHAR(255)	Données du planning	"RDV 10h - Chirurgie"

8.4.18 Entité : Role

8.4.18.1 Description

La table Role définit les rôles des utilisateurs dans l'application (Admin, Médecin, Secrétaire, etc.).

8.4.18.2 Relations et contraintes

Relation	Type	Explication
1-N avec User	Relation	Un utilisateur peut avoir plusieurs rôles.

1.8.1.11.3 Détails des attributs

Attributs	Types	Définition	Exemples
id_role	INT AUTO_INCREMENT	Identifiant unique du rôle	1
name	VARCHAR(200)	Nom du rôle	ADMIN
id_user	INT	Référence vers un utilisateur	2

8.4.19 Entité : Lens

8.4.19.1 Description

La table Lens stocke les informations sur les lentilles intraoculaires utilisées en chirurgie.

8.4.19.2 Relations et contraintes

Relation	Type	Explication
1-N avec LensManufacturer	Relation	Une lentille est fabriquée par un seul fabricant.
1-N avec Calcul	Relation	Un calcul peut proposer plusieurs lentilles.

8.4.19.2.1 Détails

Attributs	Types	Définition	Exemples
id_lens	INT AUTO_INCREMENT	Identifiant unique de la lentille	1
name	VARCHAR(50)	Nom de la lentille	FineVision
comment_trade_name	VARCHAR(50)	Nom commercial de la lentille	FineVision Toric
nominal	VARCHAR(50)	Nominal	50
haigis_A0	DECIMAL(5,2)	Constante Haigis A0	1.45
haigis_A1	DECIMAL(5,2)	Constante Haigis A1	0.4
haigis_A2	DECIMAL(5,2)	Constante Haigis A2	0.1
hoffer_QPACD	DECIMAL(5,2)	Constante Hoffer QPACD	4.00
holladay_1SF	DECIMAL(5,2)	Constante Holladay 1 SF	118.00
srkta	DECIMAL(5,2)	Constante SRK/T	118.00
haigis_A0_optimized	DECIMAL(5,2)	Constante Haigis A0 optimisée	1.48
haigis_A1_optimized	DECIMAL(5,2)	Constante Haigis A1 optimisée	0.41
haigis_A2_optimized	DECIMAL(5,2)	Constante Haigis A2 optimisée	0.11
hoffer_QPACD_optimized	DECIMAL(5,2)	Constante Hoffer QPACD optimisée	4.05
holladay_1SF_optimized	DECIMAL(5,2)	Constante Holladay 1 SF optimisée	118.50
srkta_optimized	DECIMAL(5,2)	Constante SRK/T optimisée	118.50
castrop_C_optimized	DECIMAL(5,2)	Constante Castrop C optimisée	0.3
castrop_H_optimized	DECIMAL(5,2)	Constante Castrop H optimisée	0.2
castro_R_optimized	DECIMAL(5,2)	Constante Castrop R optimisée	0.1
lens_manufacturer	DECIMAL(5,2)	Référence vers le fabricant	2
calcul	VARCHAR(250)	Description du calcul appliquée	SRK/T
id_calcul	INT	Référence vers le calcul associé	3

8.4.20 Entité : Constant

8.4.20.1 Description

La table Constant stocke les constantes biométriques utilisées dans les calculs.

8.4.20.2 Relations et contraintes

Relation	Type		Explication
1-N avec Calcul	Relation		Un calcul peut utiliser plusieurs constantes.

8.4.20.3 Détails des Attributs

Attributs	Types	Définition	Exemples
id_constant	INT AUTO_INCREMENT	Identifiant unique de la constante	1
constant_type	VARCHAR(50)	Type de constante utilisée dans les calculs	Haigis
value	VARCHAR(50)	Valeur de la constante	1.55
formula	VARCHAR(50)	Formule associée	SRK/T
Calcul	VARCHAR(250)	Description du calcul appliqué	SRK/T Lens

8.4.21 Entité : Diopter

8.4.21.1 Description

La table Diopter stocke les dioptries utilisées dans les calculs intraoculaires.

8.4.21.2 Relations et contraintes

Relation	Type	Explication
1-N avec Calcul	Relation	Un calcul peut générer plusieurs dioptries.

8.4.21.3 Détails des Attributs

Attributs	Types	Définition	Exemples
id_diopter	INT AUTO_INCREMENT	Identifiant unique de la dioptrie	1
iol_power	DECIMAL(5,2)	Puissance de l'implant intraoculaire	21.50
value	DECIMAL(5,2)	Valeur de la dioptrie	2.00
formula	VARCHAR(50)	Formule de calcul de la dioptrie	Haigis
calcul	VARCHAR(250)	Description du calcul appliqué	SRK/T

8.4.22 Entité : LensManufacturer

8.4.22.1 Description

La table LensManufacturer stocke les fabricants de lentilles intraoculaires.

8.4.22.2 Relations et contraintes

Relation	Type	Explication
1-N avec Lens	Relation	Un fabricant peut produire plusieurs lentilles.

8.4.22.3 Détails des Attributs

Attributs	Types	Définition	Exemples
id_lens_manufacturer	INT AUTO_INCREMENT	Identifiant unique du fabricant	1
name	VARCHAR(50)	Nom du fabricant	"Essilor"
Lenses	VARCHAR(50)	Lentilles fabriquées	"FineVision"
id_lens	INT	Référence vers une lentille	3