

Concepteur développeur d'applications

Rapport de compétences

Mathieu AUDIBERT

Etudiant en B3 Développeur web & applications

Concepteur développeur en alternance à la CPAM du Val-de-Marne (94)



CONTACT

 06 52 10 33 02

 mathieu.audibert27@gmail.com

 92380 Garches | France

 Mathieu Audibert

FORMATIONS

Bachelor dev & data | 3ème année

Efrei Paris
2022 - 2025

NSI - Physique Chimie - Maths C.

Lycée Alexandre Dumas
2019 - 2022
Section euro-espagnol

COMPETENCES

Python 

AWS 

Java/JS 

SQL 

Bash 

PHP/Symfony 

LANGUES

Anglais (C1)

Espagnol (B2)

Portugais (notions)

AUDIBERT Mathieu

Junior dev - Backend/Cloud

Etudiant en **3ème année Bachelor Développement & Data**, je suis à la recherche d'un contrat en alternance à partir de septembre 2025 en vue d'un mastère data & ia.

Durée du contrat d'apprentissage - de 6 à 12 mois

Rythme - 2 semaines en entreprise / 1 semaine à l'école

Titulaire du permis B - Possède un véhicule

EXPERIENCES

Alternance : Concepteur-dev Septembre – Juin 2025

CPAM du Val-de-Marne | Service informatique | Créteil 94000

- Formation sur le phishing/sécurité
- Refonte **front & back** d'un outil local de gestion du matériel informatique utilisant PHP, JS & mysql. Utilisé par les secrétariats & les techniciens, l'outil a pour **but de simplifier les processus administratifs chronophages**
- Conception & développement d'un site ayant pour objectif d'aider le suivi et l'accompagnement des bénéficiaires dans le cadre des campagnes d'aides de la Mission Accompagnement Santé (MISAS)

Adjoint administratif Juin – Juillet 2024

APHP | Accueil/Urgences | Boulogne-Billancourt 92100

- Accueil/Administration des patients administrativement
- Urgences

Stage 2nde année : Dev Ops Cloud Avril – Juin 2024

Société Générale | Equipe Network | Val de fontenay 94120

- Découverte du cloud hybride SG/network
- Réalisation d'un **script Python** d'interrogation et rapport sur les **Transit Gateway** du cloud public
- Apprentissage des **reflexes DevOps** (utilisation de SonarQube, github actions...)
- Découverte des **principes agiles** & d'architectures (Terraform...)

Stage 1ère année : Data engineer Avril – Juin 2024

LCL | Entrepôt de données | Villejuif 94800

- Découverte de **l'entrepôt de données** du secteur bancaire (plus particulièrement LCL)
- Découverte des concepts clés : **DataLake, BigData, TerraData, Géo-marketing**
- Réalisation d'un **script bash** de traitement des données sur la TVA & rapport par mail
- Réalisation de la documentation sur ces contrôles
- Découverte de Control-M
- Amélioration de mes **compétences en bash & python**

PROJETS & CERTIFICATIONS LINKEDIN LEARNING

ProSend

La première solution d'UnityWeb a été la création d'une **application web : ProSend**.

ProSend offre aux enseignants la possibilité de proposer des projets par matière à leurs élèves, leur permettant de soumettre leurs travaux directement au **format zip**.

Certificats

AWS : Juin 2024

Terraform : Juin 2024

R : Juin 2024

| | |
|---|--|
|  <p>efrei PARIS PANTHÉON-ASSAS UNIVERSITÉ</p> |  <p>L'Assurance Maladie Agir ensemble, protéger chacun</p> |
| Efrei Paris | CPAM du Val-de-Marne |
| Adresse : 30 Av de la République Ville : 94800 Villejuif Tél : 01 88 28 90 00 | Adresse : 93-95 Av. du Général de Gaulle Ville : 94000 Créteil Tél : 0 811 70 36 46 |
| Tuteur : Caroline Richshoffer | Tuteur : Frédéric Schneider |

II. Remerciements

Je voudrais, à travers ces quelques lignes, exprimer ma gratitude envers l'ensemble des personnes qui m'ont accompagné tout au long de cette année et durant la réalisation de ce projet.

Je remercie tout particulièrement **M. Frédéric SCHNEIDER & M. Régis GRANDMAGNAC** pour leur pédagogie bienveillante, le temps accordé et l'occasion offerte d'avoir pu réaliser cette alternance dans de très bonnes conditions.

Une reconnaissance très particulière envers la principale actrice de mon projet d'alternance **Mme. Caroline RICHSHOFFER** qui m'a accompagné durant mon alternance au sein de la société.

Je tiens également à exprimer mes remerciements à tout le service informatique de la **CPAM du Val-de-Marne**, en particulier l'équipe de développement pour leur coopération et l'intérêt qu'ils m'ont accordé afin de mener à bien mes missions.

Ma gratitude également à notre encadrante de formation **Mme. Lea DELACROIX** ainsi qu'à tous nos intervenants du Bachelor 3 Développement Web & Applications pour la qualité de l'enseignement et le suivi dont ils m'ont fait bénéficier, ainsi qu'à tous ceux et celles qui ont contribué à ma formation et mon éducation tout au long de mon séjour à l'école.

Je souhaite également adresser mes sincères remerciements aux membres du jury pour le temps qu'ils ont consacré à l'évaluation de mon travail, ainsi que pour leurs remarques constructives et enrichissantes.

III. Avant-propos

Etudiant en 3ème Année de Bachelor Développement Web & Applications au sein d'**EFREI Paris**, j'ai effectué mon alternance au sein de la Caisse Primaire d'Assurance Maladie du Val-de-Marne qui m'a accueilli dans ses locaux et m'a fourni le matériel nécessaire à la réalisation de mes missions.

Cette formation en alternance nous offre la possibilité de mettre en pratique les connaissances acquises lors de la formation et ainsi renforcer notre orientation professionnelle.

Ce document est le rapport de compétences suivant mon année de formation en alternance à EFREI Paris et à la CPAM du Val-de-Marne et visant la validation de mon diplôme.

Au cours de ma formation à l'**EFREI Paris**, j'ai eu l'opportunité de développer les compétences techniques et méthodologiques nécessaires à la réalisation de ce projet.

Tables des matières

| | | |
|------|--|----|
| I. | Curriculum Vitae..... | 2 |
| II. | Remerciements..... | 3 |
| III. | Avant-propos..... | 4 |
| IV. | Table des figures..... | 7 |
| V. | Présentation de l'entreprise..... | 8 |
| | A. Histoire de la CPAM..... | 8 |
| | B. Organigramme..... | 11 |
| | C. Les objectifs de la CPAM..... | 12 |
| | D. Mes missions..... | 14 |
| | 1. AAVA..... | 14 |
| | 2. Mouv'Invent..... | 16 |
| VI. | Projet Cooloc..... | 18 |
| | A. Contexte..... | 18 |
| | B. Problématique..... | 19 |
| | C. Objectifs..... | 19 |
| | D. Public Cible..... | 20 |
| | E. Contribuer à la gestion d'un projet informatique..... | 20 |
| | 1. Méthode d'organisation..... | 20 |
| | 2. Gestion & planning des tâches..... | 21 |
| | 3. Outils..... | 22 |
| | F. Démarche de conception..... | 24 |
| | 1. Diagramme de cas d'utilisation..... | 24 |
| | 2. Diagramme de Séquence..... | 29 |
| | 3. Diagramme d'activité..... | 30 |
| | 4. Diagramme d'architecture..... | 33 |
| | 5. MCD..... | 33 |
| | 6. MLD..... | 35 |
| | 7. MPD..... | 36 |
| | G. Analyser et maquetter les besoins..... | 37 |
| | 1. Charte graphique..... | 37 |
| | 2. Zoning/Wireframes/Maquettes..... | 37 |
| | H. Développement de l'interface utilisateur..... | 44 |
| | 1. Point d'entrée de l'application et navigation conditionnelle..... | 44 |
| | 2. Navigation par onglets et séparation des rôles..... | 44 |
| | 3. Fonctionnalités disponibles pour un colocataire..... | 44 |
| | 4. Fonctionnalités disponibles pour un responsable..... | 45 |
| | 5. Interface dédiée à l'administrateur..... | 45 |
| | 6. Bilan technique..... | 46 |
| | I. Développement Backend..... | 46 |
| | 1. Architecture & modules python..... | 46 |
| | 2. Authentification sécurisée..... | 47 |
| | 3. Gestion des rôles..... | 48 |

| | |
|--|----|
| 4. API RESTful..... | 48 |
| 5. Base de données conforme..... | 49 |
| J. Sécurité..... | 49 |
| 1. Gestion des rôles..... | 49 |
| 2. Mots de passe sécurisés..... | 50 |
| 3. Injections XSS..... | 50 |
| 4. Injections SQL..... | 51 |
| 5. Configuration..... | 52 |
| 6. Authentification..... | 52 |
| 7. CSRF..... | 53 |
| 8. RGPD..... | 53 |
| K. DevOps..... | 54 |
| 1. Validation manuelle..... | 54 |
| 2. Vérification des endpoints avec Postman..... | 55 |
| 3. Analyse statique avec SonarQube..... | 57 |
| 4. GitHub Actions & analyse continue du code..... | 57 |
| 5. Conteneurisation Docker..... | 58 |
| 6. Perspectives..... | 59 |
| VII. Bilan & conclusion..... | 59 |
| A. Compétences validées..... | 59 |
| B. Axes d'améliorations sur les contrôles de champs..... | 60 |
| C. Axes d'améliorations sur les tokens..... | 61 |
| D. Evolutions & futur de Cooloc..... | 61 |
| E. Projection professionnelle..... | 62 |
| VIII. Acronymes..... | 63 |

IV. Table des figures

| | |
|--|----|
| Figure 1 : Logo EFREI Paris..... | 1 |
| Figure 2 : Logo CPAM du Val-de-Marne..... | 1 |
| Figure 3 : Mon CV..... | 2 |
| Figure 4 : Notre Histoire L'assurance Maladie..... | 10 |
| Figure 5 : Organigramme Général..... | 11 |
| Figure 6 : Organigramme du Service Informatique..... | 11 |
| Figure 7 : Que financent 1000€ d'impôts ? | 13 |
| Figure 8 : Acteur visiteur de mon Use Case..... | 24 |
| Figure 9 : Acteur colocataire de mon Use Case..... | 24 |
| Figure 10 : Acteur responsable connecté de mon Use Case..... | 24 |
| Figure 11 : Acteur admin connecté..... | 24 |
| Figure 12 : Exemple de cas d'utilisation de mon Use Case..... | 25 |
| Figure 13 : Association “voir les logs” - “Admin connecté” de mon Use Case..... | 25 |
| Figure 14 : Exemple d’include de mon Use Case..... | 25 |
| Figure 15 : Exemple d’extends de mon Use Case..... | 26 |
| Figure 16 : Exemple de généralisation de mon Use Case..... | 26 |
| Figure 17 : Mon Use Case en entier..... | 28 |
| Figure 18 : Mon Diagramme de Séquence..... | 30 |
| Figure 19 : Mon Diagramme d’activité pour la création d’une colocation..... | 31 |
| Figure 20 : Mon Diagramme d’activité Inscription & Connexion..... | 32 |
| Figure 21 : Mon Diagramme d’architecture..... | 33 |
| Figure 22 : Modèle Conceptuel de Données..... | 34 |
| Figure 23 : Modèle Logique de Données..... | 35 |
| Figure 24 : Modèle Physique de Données..... | 36 |
| Figure 25 : Ma charte graphique..... | 37 |
| Figure 26 : Mon Zoning PC | 38 |
| Figure 27 : Mon Zoning mobile..... | 39 |
| Figure 28 : Mon wireframe PC..... | 40 |
| Figure 29 : Mon wireframe mobile..... | 41 |
| Figure 30 : Mes maquettes PC..... | 42 |
| Figure 31 : Mes maquettes mobiles..... | 43 |
| Figure 32 : Extrait de code, hachage de mots de passes..... | 47 |
| Figure 33 : Extrait de code, création de token..... | 47 |
| Figure 34 : Extrait de code, vérification du token..... | 47 |
| Figure 35 : Extrait de code, vérification du csrf..... | 47 |
| Figure 36 : Ma collection Postman..... | 55 |

V. Présentation de l'entreprise

A. Histoire de la CPAM

L'histoire de la Caisse Primaire d'Assurance Maladie (CPAM) s'enracine dans le vaste mouvement de refondation sociale amorcé à la Libération, avec la volonté politique forte de bâtir une société plus juste et solidaire. En 1945, les ordonnances du Gouvernement provisoire de la République française fondent le système de Sécurité sociale sur un principe clé : la solidarité nationale. Chaque citoyen, quels que soient ses revenus ou son statut, doit pouvoir accéder aux soins et bénéficier d'une protection contre les aléas de la vie. La CPAM, en tant qu'organe déconcentré de ce dispositif, est chargée de faire vivre ce principe sur l'ensemble du territoire.

Au fil des décennies, les CPAM ont vu leurs missions s'intensifier et se complexifier pour s'adapter à une société en constante mutation. De simples guichets administratifs à leurs débuts, elles sont devenues de véritables plateformes de services, jouant un rôle central dans la régulation du système de soins et dans la mise en œuvre des politiques de santé publique. Ce changement s'est opéré notamment avec l'essor des maladies chroniques, l'allongement de l'espérance de vie, et l'accroissement des inégalités d'accès aux soins.

Le développement des outils numériques a constitué un tournant majeur dans la transformation de la CPAM. À partir des années 2000, la mise en place de la carte Vitale, du dossier médical partagé (DMP), puis des services en ligne comme Ameli.fr a permis d'optimiser la gestion des dossiers, de fluidifier les échanges avec les professionnels de santé, et de rapprocher les usagers des services auxquels ils ont droit. Ces innovations ont également renforcé les capacités de contrôle et de pilotage des dépenses, un enjeu devenu crucial dans un contexte de maîtrise budgétaire.

Parallèlement, la CPAM s'est engagée dans des actions de plus en plus ciblées pour répondre aux besoins spécifiques de certaines populations : personnes en situation de précarité, travailleurs indépendants, personnes âgées, étudiants, ou encore assurés atteints de pathologies lourdes. Des dispositifs tels que la Complémentaire santé solidaire (CSS), l'Assurance Maladie maternité/paternité, ou encore le programme « Mission accompagnement santé » illustrent cette volonté d'offrir une couverture sociale personnalisée, adaptée aux parcours de vie.

La crise sanitaire liée à la pandémie de COVID-19 en 2020 a mis en lumière, de manière spectaculaire, le rôle essentiel de la CPAM dans la protection de la population. Elle a été en première ligne pour organiser les campagnes de dépistage, assurer le suivi des cas contacts, faciliter la vaccination, verser les indemnités d'isolement, et accompagner les employeurs et salariés dans la gestion des arrêts de travail. Cette mobilisation exceptionnelle a démontré la capacité de l'institution à faire face à l'urgence tout en maintenant ses missions traditionnelles.

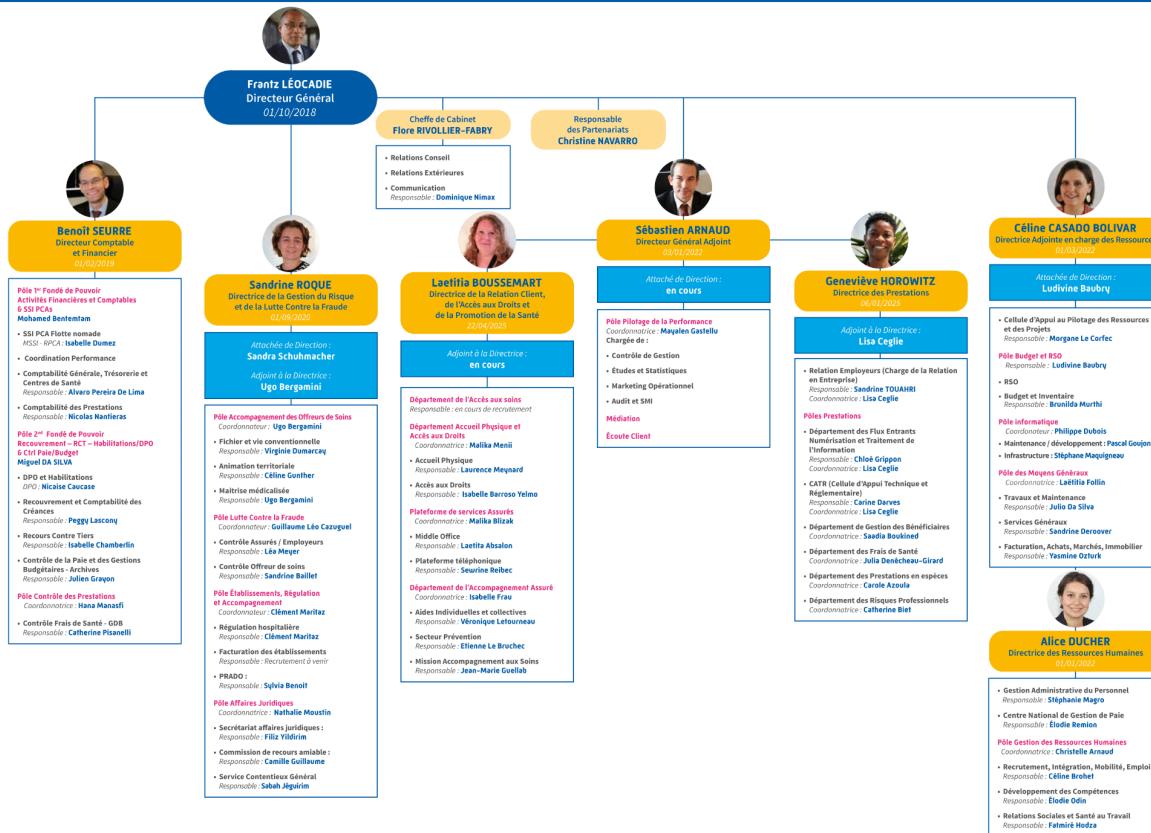
Aujourd’hui encore, la CPAM continue de se réinventer pour répondre aux défis contemporains : transformation du système de santé vers plus de prévention, développement de la e-santé, renforcement de la proximité territoriale, amélioration de l’expérience usager, et intégration des enjeux de santé mentale et environnementale. Elle agit en lien étroit avec les autres acteurs de la santé – hôpitaux, médecins, collectivités locales, associations – dans une logique de coopération et de coordination renforcée.

Ainsi, plus de 75 ans après sa création, la Caisse Primaire d’Assurance Maladie demeure un pilier fondamental du pacte social français. Elle incarne au quotidien l’un des principes fondateurs de la République : garantir à chacun, sans distinction, le droit à la santé. À travers son action, la CPAM contribue non seulement à soigner, mais aussi à prévenir, à informer, à accompagner et à protéger. Elle est l’un des visages les plus visibles et les plus accessibles de l’État au service des citoyens.

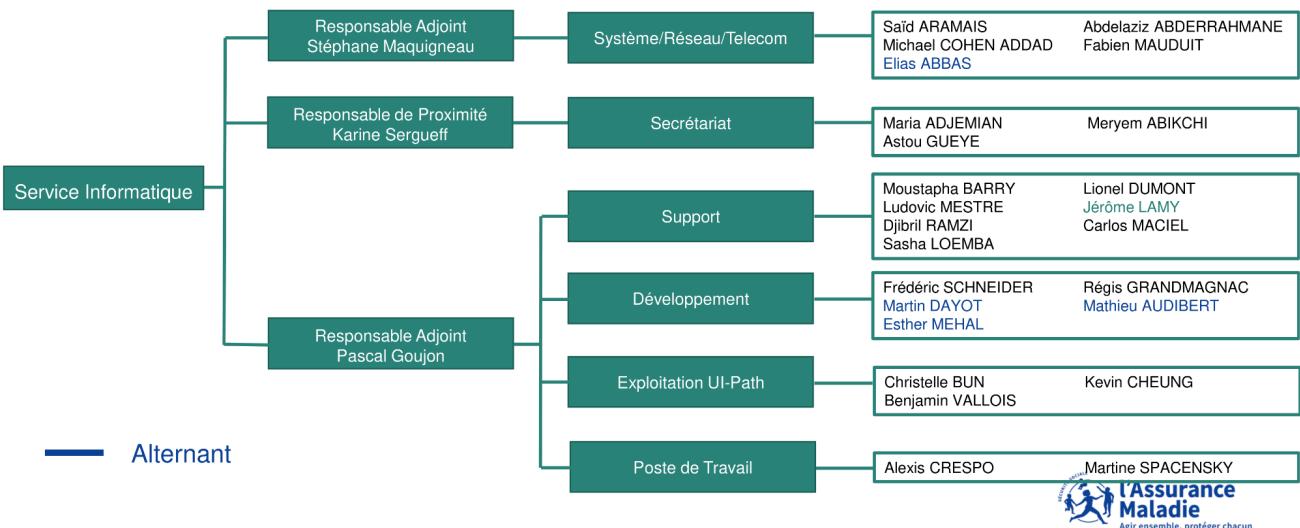


B. Organigramme

ORGANIGRAMME DE DIRECTION DE LA CPAM DU VAL-DE-MARNE - Avril 2025



ANNEXE ETAT DES LIEUX – ORGANIGRAMME



Dans l'organigramme fourni, j'appartiens au *Service Informatique*, je fais plus particulièrement partie de l'équipe de **Développement**, une composante relativement importante du service. Cette équipe est placée sous la responsabilité de M. Pascal GOUJON, responsable adjoint. Mon rôle au sein de cette équipe consiste principalement à participer à la conception, à la programmation et au développement de logiciels ou d'applications.

L'équipe de développement joue un rôle central dans la création et la maintenance des solutions logicielles utilisées par l'organisation. Nous collaborons régulièrement avec d'autres équipes du service informatique, telles que le *Support*, *l'Exploitation UI-Path*, ou encore le *Secrétariat*, afin de garantir le bon fonctionnement et l'efficacité des systèmes informatiques.

C. Les objectifs de la CPAM

La CPAM du Val-de-Marne constitue un des **102 maillons** du système de protection sociale français. Acteur **privé dépendant de fonds publics** majeur dans la gestion de l'assurance maladie au niveau local, elle remplit des missions fondamentales auprès des assurés, des professionnels de santé et des partenaires institutionnels. Située dans un département particulièrement dynamique de la région d'Île-de-France, la CPAM du Val-de-Marne assure la couverture de santé **d'1 439 302 bénéficiaires**, tout en jouant un rôle de proximité en matière d'accompagnement, de prévention et de gestion des droits.

Le système de sécurité sociale français repose sur plusieurs branches, dont la branche maladie, gérée principalement par l'Assurance Maladie. Ce régime national couvre la majorité de la population française et **garantit l'accès aux soins, le remboursement des dépenses de santé, et le versement d'indemnités en cas d'arrêt de travail par accidents, maladies ou maternité**.

L'Assurance Maladie est structurée autour de la Caisse Nationale de l'Assurance Maladie (CNAM), qui délègue la mise en œuvre opérationnelle de ses missions aux CPAM, réparties sur l'ensemble du territoire. La CNAM finance les CPAMs grâce aux fonds issus principalement des impôts et des cotisations sociales. Depuis la mise en place de la contribution sociale généralisée (CSG), une grande partie du financement de la Sécurité sociale provient de l'impôt. En effet, **environ 54 %** des recettes publiques affectées à la protection sociale sont destinées à la couverture santé, ce qui en fait **le poste de dépense le plus important**. Ces fonds sont ensuite redistribués aux différentes CPAM, comme celle du Val-de-Marne.



La CPAM du Val-de-Marne remplit plusieurs missions essentielles au service des assurés :

- ❖ **La gestion des droits des assurés.** Elle assure l'ouverture des droits à l'assurance maladie, gère les affiliations, les mutations, et veille à la mise à jour régulière de la situation des bénéficiaires (notamment via la carte Vitale).
- ❖ **Le remboursement des soins.** La CPAM traite chaque année des millions de feuilles de soins, électroniques ou papiers. Elle veille à rembourser les frais médicaux engagés par les assurés (consultations, médicaments, examens, hospitalisations, etc.).
- ❖ **Le versement des prestations en espèces.** Elle prend en charge les indemnités journalières versées aux assurés en arrêt de travail, que ce soit pour maladie, maternité, paternité, ou accident du travail.
- ❖ **La gestion du risque et la prévention.** La CPAM du Val-de-Marne joue un rôle important dans la gestion du risque, en promouvant des actions de prévention et d'éducation à la santé, et en menant des campagnes de dépistage (cancer, diabète, etc.) et de vaccination. Elle travaille aussi à la lutte contre les comportements à risque et les fraudes à l'Assurance Maladie.
- ❖ **L'accompagnement des publics fragiles.** Elle propose des dispositifs spécifiques pour les populations vulnérables : couverture maladie universelle complémentaire (CMU-C), aide médicale d'État (AME), complémentaire santé solidaire (CSS), ou encore accompagnement personnalisé des patients atteints de pathologies chroniques.
- ❖ **Le lien avec les professionnels de santé.** La CPAM du Val-de-Marne entretient un dialogue constant avec les professionnels libéraux (médecins,

pharmaciens, dentistes, infirmiers, etc.), assure la mise en œuvre des conventions nationales, et participe à l'organisation de l'offre de soins sur le territoire.

Comme toutes les caisses primaires, la CPAM du Val-de-Marne fait face à de nombreux défis dans un contexte de transformation du système de santé :

- ❖ **La dématérialisation des démarches** et l'amélioration de la relation numérique avec les usagers (via le compte ameli, l'espace santé, etc.)
- ❖ **L'accompagnement de la réforme du système de santé**, notamment avec le déploiement des dispositifs comme le parcours de soins coordonné, le service d'accès aux soins (SAS), ou le réseau de santé territorialisé
- ❖ **La lutte contre les déserts médicaux**, en collaboration avec les acteurs du territoire
- ❖ **La modernisation du service public et l'amélioration de la qualité de service**
- ❖ **Le soutien aux assurés** face aux crises sanitaires (ex : COVID-19).

Son action quotidienne s'inscrit dans une démarche de **solidarité, d'équité** et **d'innovation**, en constante adaptation aux évolutions de la société et aux besoins de ses assurés.

D. Mes missions

1. AAVA

Dans le cadre de mon alternance au sein de la CPAM, j'ai eu l'opportunité de participer activement à la conception, au développement et au déploiement d'un **site web métier** destiné à améliorer le **suivi et l'accompagnement de près de 800 000 bénéficiaires**, dans le cadre des campagnes menées par la **Mission Accompagnement Santé (MISAS)**. Cette mission, au cœur des priorités de l'Assurance Maladie, vise à contacter et accompagner les publics les plus vulnérables – personnes âgées, femmes enceintes, patients atteints d'affections de longue durée (ALD), bénéficiaires de la Complémentaire santé solidaire (CSS), etc. – afin de renforcer l'accès aux droits et aux soins.

Le projet, baptisé **AAVA** (Accompagnement Automatisé des Visites et Appels), a été lancé dans un objectif clair de **modernisation des outils internes**. Le fonctionnement antérieur reposait sur une gestion chronophage et peu fiable des données via **de volumineux fichiers Excel**, modifiés manuellement par les agents de la MISAS. Cette méthode, bien que fonctionnelle, engendrait des erreurs fréquentes, un manque de visibilité en temps réel sur les opérations en cours, et une perte de temps considérable.

Face à ces constats, nous avons imaginé une **plateforme centralisée**, ergonomique et sécurisée, offrant une interface simple d'utilisation adaptée aux besoins des utilisateurs métiers. Grâce à AAVA, les agents peuvent désormais :

- ❖ **Saisir, modifier et consulter les données des bénéficiaires** en quelques clics,
- ❖ **Automatiser l'importation des fichiers** transmis régulièrement par la CNAM,
- ❖ **Suivre en temps réel l'état d'avancement des campagnes** (contacts effectués, motifs d'appel, suivis nécessaires),
- ❖ Générer des **statistiques consolidées** pour le pilotage de l'activité,
- ❖ Et **gérer les utilisateurs** de manière structurée selon les rôles et permissions définis.

Ce projet a permis un **gain de temps significatif**, une réduction des risques d'erreurs humaines et une meilleure traçabilité des interventions réalisées. Il a aussi facilité la communication entre les différents acteurs de la chaîne, grâce à une base de données unique, fiable et partagée.

Nous étions trois alternants mobilisés sur ce projet :

- ❖ **Martin Dayot**, en charge du **développement front-end** et de la conception du design responsive de l'interface utilisateur, avec une attention particulière à l'accessibilité (conformité RGAA) et à l'ergonomie,
- ❖ **Esther**, responsable de la **modélisation et de la gestion de la base de données**, ainsi que de la rédaction des requêtes optimisées pour garantir la performance,
- ❖ **Et moi-même**, intervenant de manière transversale sur le **développement full stack** (back-end Symfony et front-end JavaScript/HTML/CSS), tout en assurant également les aspects **DevOps**.

Sur le plan technique, le projet repose sur une stack composée de :

- ❖ **Symfony** (version 5) pour le back-end, garantissant robustesse et maintenabilité du code,
- ❖ **JavaScript**, avec une structure modulaire pour le front-end interactif,

- ❖ **MySQL** pour la gestion des données,
- ❖ Un **versioning Git** collaboratif via GitLab,
- ❖ Et une chaîne **CI/CD (intégration et déploiement continu)** mise en place avec GitLab CI, permettant une livraison fluide et contrôlée des mises à jour en environnement de test puis de production.

Le projet a été lancé en **décembre**, à partir d'un **cahier des charges co-construit** avec les équipes métiers, incluant plusieurs ateliers de recueil de besoins et des phases de tests utilisateurs en conditions réelles. Depuis, nous travaillons selon une approche **agile**, avec des livraisons itératives, des démonstrations régulières, et des phases de validation fonctionnelle menées en étroite collaboration avec les référents de la MISAS.

Cette expérience m'a permis de développer de nombreuses compétences techniques, mais aussi **organisationnelles et relationnelles** : compréhension des enjeux métiers, gestion de projet, communication avec les utilisateurs finaux, adaptation aux contraintes de sécurité et de confidentialité des données de santé.

Aujourd'hui, AAVA est en cours de déploiement progressif auprès de plusieurs équipes départementales, et des évolutions sont déjà prévues, notamment l'interfaçage avec d'autres outils internes de la CPAM, la gestion multi-campagnes, et l'ajout de modules statistiques avancés.

2. Mouv'Invent

J'ai également participé à la **refonte complète, à la fois front-end et back-end, d'un outil local de gestion du matériel informatique**, utilisé quotidiennement par les **secrétariats administratifs** et les **techniciens** de la CPAM. Cet outil interne, historiquement développé en **PHP** avec une interface en **JavaScript** et une base de données **MySQL**, avait pour but initial de suivre les mouvements de matériels informatiques (ordinateurs, écrans, imprimantes, téléphones, etc.), ainsi que les opérations d'installation, de restitution ou de transfert entre agents ou sites.

Avec le temps, l'outil avait montré ses **limites en termes d'ergonomie, de lisibilité et de performance**. Les interfaces vieillissantes, peu intuitives, rendaient la saisie d'informations laborieuse, augmentaient le risque d'erreurs, et nécessitaient souvent des manipulations manuelles en base de données. De nombreux utilisateurs exprimaient une certaine **frustration quant à la lenteur de l'application** et à son **manque de clarté fonctionnelle**.

Face à ce constat, j'ai été chargé de **repenser l'ensemble de l'outil**, aussi bien sur le plan technique que fonctionnel. Cette refonte a poursuivi plusieurs objectifs :

- ❖ **Améliorer l'expérience utilisateur**, en proposant une interface plus moderne, plus fluide, et pensée pour répondre aux usages concrets des techniciens sur le terrain ;
- ❖ **Optimiser les performances**, en allégeant les appels serveur, en réduisant les temps de chargement et en améliorant la gestion des requêtes SQL ;
- ❖ **Renforcer la cohérence métier**, en restructurant la logique applicative et en assurant une meilleure séparation des responsabilités dans le code (MVC, gestion des erreurs, validations, etc.) ;
- ❖ **Sécuriser l'application**, en ajoutant une gestion fine des droits utilisateurs, une meilleure protection des données, et une traçabilité des opérations.

Concrètement, j'ai :

- ❖ **Moderniser l'interface utilisateur** avec une refonte complète du front-end : nouveaux composants interactifs, hiérarchisation visuelle de l'information, navigation plus intuitive, responsive design, etc. ;
- ❖ **Refondu le code PHP**, en rationalisant les contrôleurs, les vues et les accès à la base de données, avec une meilleure utilisation des requêtes préparées pour renforcer la sécurité ;
- ❖ **Retravaillé le modèle de données**, en normalisant certaines tables, en ajoutant des relations manquantes, et en créant de nouvelles vues SQL pour faciliter les rapports et exports ;
- ❖ **Ajouté de nouvelles fonctionnalités**, comme la recherche avancée de matériel, l'export des inventaires, la génération automatique de fiches de mouvement, ou encore un tableau de bord récapitulatif pour les administrateurs.

Cette refonte a été menée **en étroite collaboration avec les utilisateurs finaux**, à travers plusieurs phases d'écoute, de démonstrations intermédiaires et de tests. Grâce à leurs retours, nous avons pu affiner les priorités, corriger les éventuels points de blocage, et déployer une solution réellement adaptée aux besoins du terrain.

Le nouvel outil a été **mis en production progressivement**, avec un accompagnement des équipes pour la prise en main, et a rapidement permis une **réduction significative du temps passé sur chaque opération**, ainsi qu'une meilleure visibilité sur l'état du parc informatique. Ce projet a été une excellente

opportunité pour moi de consolider mes compétences en **développement full stack**, en **optimisation de base de données**, mais aussi en **gestion de projet** et en **conduite du changement**.

VI. Projet Cooloc

A. Contexte

Ces dernières années, la précarité étudiante s'est imposée comme un enjeu majeur dans le paysage social et économique. Le coût de la vie n'a cessé d'augmenter, tandis que les aides attribuées aux étudiants, souvent jugées insuffisantes, peinent à compenser les dépenses nécessaires à une scolarité stable et à des conditions de vie dignes. Parmi ces dépenses, **le logement représente une part importante du budget des étudiants**, et constitue l'un des principaux obstacles à leur autonomie.

Dans les grandes villes universitaires, la tension sur le marché locatif est particulièrement forte : l'offre de logements accessibles est largement inférieure à la demande, et les prix sont souvent prohibitifs pour des jeunes sans revenus fixes. Trouver un logement devient alors un parcours du combattant, entre la sélection drastique des propriétaires, les garanties exigées, et la rapidité avec laquelle les biens disponibles sont loués. Cette situation pousse de nombreux étudiants à accepter des compromis parfois risqués : logements insalubres, excentrés, ou encore colocation imposée avec des inconnus, dans des conditions peu encadrées.

Face à cette réalité, **la colocation apparaît comme une alternative de plus en plus plébiscitée**, notamment pour des raisons économiques. Elle permet de mutualiser les coûts (loyer, charges, abonnements...), de rompre l'isolement, et parfois de nouer des liens sociaux forts, ce qui est essentiel dans un contexte où de nombreux étudiants souffrent également de solitude ou de détresse psychologique. Toutefois, vivre en colocation ne va pas sans poser de nouvelles contraintes : l'organisation du quotidien, la répartition des responsabilités, la gestion financière collective ou encore la communication entre colocataires peuvent vite devenir sources de tensions, voire de conflits.

En parallèle, le mode de vie étudiant évolue. Les jeunes générations sont nées avec le numérique, utilisent quotidiennement des outils collaboratifs et attendent des solutions pratiques, flexibles et accessibles depuis leur smartphone ou leur ordinateur. Pourtant, peu d'outils sont spécifiquement conçus pour répondre aux problématiques de la colocation étudiante. La gestion repose souvent sur des feuilles Excel, des groupes de messagerie, ou des applications généralistes, qui ne prennent pas en compte les spécificités du mode de vie en colocation.

Dans ce contexte social et technologique en pleine mutation, il devient pertinent d'interroger les besoins spécifiques des étudiants vivant en colocation, les difficultés qu'ils rencontrent au quotidien, ainsi que les leviers qui pourraient améliorer leur organisation, leur communication et, plus globalement, leur qualité de vie. La colocation n'est pas seulement une réponse économique à la crise du logement étudiant : elle est aussi un **modèle de vie collective** qui mérite d'être pensé, accompagné et facilité.

B. Problématique

Dans un contexte où la précarité étudiante s'aggrave et où l'accès au logement devient de plus en plus complexe, la colocation constitue une solution privilégiée par de nombreux étudiants. Toutefois, si elle permet de réduire les coûts et de créer un cadre de vie plus convivial, la colocation entraîne également son lot de défis quotidiens. L'absence d'organisation claire peut engendrer des tensions : oubli ou inégalité dans la répartition des tâches ménagères, mauvaise gestion des dépenses communes, retards dans les remboursements, manque de communication entre colocataires, etc.

La question centrale se pose alors : **comment faciliter la gestion quotidienne d'une colocation étudiante afin de prévenir les conflits, améliorer l'organisation collective et alléger la charge mentale des colocataires ?**

Dans un monde de plus en plus connecté, existe-t-il une solution numérique capable de structurer la vie en colocation sans la complexifier ?

C. Objectifs

L'objectif principal de ce projet est de concevoir et de développer une **application web dédiée à la gestion de la colocation étudiante**, pensée comme un outil pratique, intuitif et collaboratif.

Cooloc vise à répondre concrètement aux problématiques évoquées en proposant plusieurs fonctionnalités clés :

- ❖ **Gestion des tâches ménagères** : création, attribution et suivi des tâches pour éviter les oubli et garantir une répartition équitable.
- ❖ **Suivi des dépenses communes** : enregistrement des achats effectués par chaque colocataire, calcul automatique des parts de chacun, et suivi des remboursements à effectuer.
- ❖ **Gestion des membres de la colocation** : possibilité d'ajouter, modifier ou supprimer des colocataires selon les changements de situation.

- ❖ **Interface simple et accessible** : une ergonomie pensée pour être utilisée facilement depuis un smartphone ou un ordinateur, sans compétences techniques particulières.

Ce projet a également pour objectif de **favoriser la communication** entre colocataires, de **réduire les sources de conflit**, et d'**améliorer la qualité de vie en communauté**. En somme, il s'agit de proposer une solution numérique complète, au service d'une meilleure organisation et d'une vie collective plus harmonieuse.

D. Public Cible

Le public visé par Cooloc est principalement composé :

- ❖ **Des étudiants vivant en colocation**, qu'ils soient dans une résidence universitaire, dans un logement privé partagé, ou dans un appartement familial temporairement occupé à plusieurs.
- ❖ **Des jeunes adultes en formation ou en début de vie active**, qui vivent en colocation pour des raisons économiques, pratiques ou sociales, et qui partagent les mêmes besoins organisationnels que les étudiants.
- ❖ **Des futurs colocataires**, en recherche d'un outil pour structurer leur vie commune dès leur emménagement.

Ce public est caractérisé par une familiarité avec les outils numériques, une volonté de partager équitablement les responsabilités, et une recherche constante de solutions pratiques pour alléger la charge mentale liée à la vie collective. L'application vise donc à répondre à des besoins concrets, dans un langage accessible, avec une interface pensée pour une **utilisation rapide, fonctionnelle et collaborative**.

E. Contribuer à la gestion d'un projet informatique

1. Méthode d'organisation

J'ai développé Cooloc tout seul et donc, j'ai opté pour une **méthodologie de développement en cycle en V**, adaptée aux projets nécessitant une rigueur dans les phases de conception, de développement et de tests. Le cycle en V permet une organisation linéaire et structurée du processus de création, tout en intégrant des phases de validation à chaque étape clé.

Le cycle se compose de deux grandes branches :

- ❖ **La phase descendante**, qui comprend l'expression des besoins, l'analyse fonctionnelle, la conception générale et détaillée ;
- ❖ **La phase montante**, qui inclut le développement, les tests unitaires, les tests d'intégration, la validation finale et la mise en production.

Chaque étape de conception trouve sa phase de vérification correspondante en fin de cycle, assurant une cohérence entre les spécifications initiales et le produit livré. Ce modèle s'est révélé pertinent pour un projet individuel car il permet de **mieux anticiper les tâches à réaliser, de limiter les retours en arrière**, et de garantir une **traçabilité claire** du développement.

2. Gestion & planning des tâches

Le planning du projet a été réalisé à l'aide d'un **diagramme de Gantt**, structurant les grandes phases de travail sur une période d'environ 300 jours. Ce diagramme permet de visualiser la répartition temporelle des tâches principales du projet, leur durée approximative, ainsi que les chevauchements éventuels.

Voici une synthèse des différentes étapes :

- ❖ **Conception** : amorcée dès le début du projet (jours 0 à 100), elle a posé les bases du cahier des charges, des maquettes et de l'architecture technique.
- ❖ **Configuration** : réalisée dans les premiers 100 jours, cette phase a permis d'installer les environnements de développement et les outils nécessaires.
- ❖ **Back-end** : développement du serveur, des API et des modèles de données entre les jours 50 et 200.
- ❖ **Sécurité** : réflexion et intégration des mécanismes de protection (authentification, JWT, rôles, etc.) entre les jours 100 et 200.
- ❖ **Front-end** : développement de l'interface utilisateur avec React/Vite, entre les jours 150 et 250.
- ❖ **Tests** : menés parallèlement aux développements entre les jours 200 et 300.
- ❖ **Rédaction du rapport** : lancée dès le jour 200 jusqu'au jour 350, en parallèle des tests et des ajustements.
- ❖ **Préparation à la soutenance** : prévue entre les jours 300 et 350.

Cette planification m'a permis de **garder un rythme de travail cohérent**, tout en respectant les priorités du projet.

Pour assurer un **suivi clair et dynamique des tâches**, j'ai utilisé l'outil **Trello** sous forme de tableau Kanban. Le tableau se compose de colonnes correspondant à l'état d'avancement des tâches :

- ❖ **À faire** : tâches planifiées mais non commencées (pages fonctionnelles etc.).
- ❖ **En cours** : éléments en développement ou en rédaction (documentation, développement).
- ❖ **À corriger / bugs** : liste des éléments identifiés comme problématiques, nécessitant une correction (ex. GitHub Actions, SonarQube).
- ❖ **Terminé** : tâches finalisées, telles que le MCD, MPD, JWT, login, rôles, etc.
- ❖ **Idées** : espace de réflexion pour de potentielles évolutions futures (ex. documentation complémentaire).

Ce tableau m'a permis de **structurer ma charge de travail, de suivre l'avancement des modules**, et **d'identifier rapidement les points bloquants** à corriger.

3. Outils

Bien que le cœur applicatif repose sur des technologies web modernes, **Python 3** a été utilisé ponctuellement pour des besoins de **serveur local léger**, notamment via le module intégré http.server. Ce choix permet de **servir rapidement des fichiers statiques**, de tester l'affichage ou des flux sans mettre en place un serveur complexe. L'outil est simple, rapide à configurer, et ne nécessite aucune installation supplémentaire.

Pour le **suivi de version**, j'ai utilisé **Git** en local, avec **GitHub** comme plateforme distante. Ce choix me permet de :

- ❖ Sauvegarder l'historique du projet ;
- ❖ Travailler de manière organisée avec des commits clairs ;
- ❖ Gérer les versions et les branches ;
- ❖ Documenter les changements importants dans un cadre professionnel.

Même en solo, Git offre une **structure de travail rigoureuse** et permet une **tracabilité complète** du développement.

Pour le front-end, j'ai choisi **ReactJS** combiné à **Vite**. React permet de créer une **interface utilisateur réactive, modulaire et performante**, tandis que Vite offre :

- ❖ Une **expérience de développement rapide** grâce à un hot reload ultra-efficace ;
- ❖ Une **configuration simplifiée** ;
- ❖ Des **temps de build courts**, ce qui est appréciable lors des phases de test ou de mise en ligne.

Ce duo s'est imposé comme une solution moderne et productive pour développer l'interface de l'application.

Docker a été utilisé pour **standardiser l'environnement de développement**, notamment en isolant la base de données, le back-end et d'autres services. Grâce à Docker :

- ❖ Je m'assure que l'environnement est reproductible sur n'importe quelle machine ;
- ❖ Je limite les problèmes de compatibilité entre dépendances ;
- ❖ Je peux lancer l'ensemble de l'application via un seul fichier docker-compose.yml.

Cela facilite grandement la phase de configuration, tout en garantissant une **cohérence entre les environnements de développement, de test et de production**.

Enfin, pour la base de données, j'ai opté pour **PostgreSQL**, un système **robuste, open-source et orienté production**. Il permet :

- ❖ Une **gestion avancée des données** ;
- ❖ Une compatibilité parfaite avec les ORM modernes ;
- ❖ Un bon support des types complexes et des contraintes d'intégrité.

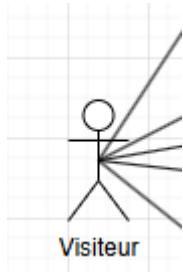
Ce choix s'est imposé face à d'autres SCBD pour sa **performance, sa stabilité**, et la richesse de sa documentation.

F. Démarches de conception

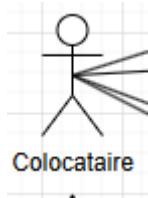
1. Diagramme de cas d'utilisation

Dans un diagramme de cas d'utilisation, un **acteur** représente une entité externe (généralement une personne ou un système) qui interagit avec le système étudié. Les acteurs déclenchent des cas d'utilisation pour réaliser un objectif précis. Dans le diagramme de l'application **Cooloc**, on distingue quatre types d'acteurs :

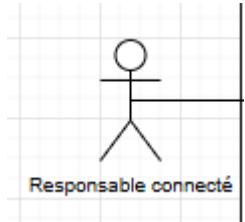
Visiteur : utilisateur non connecté ayant un accès très limité.



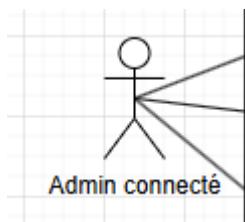
Colocataire : utilisateur connecté avec des droits standards liés à la vie en colocation.



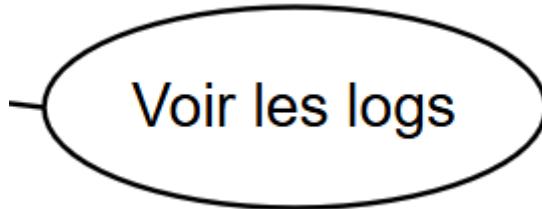
Responsable connecté : un colocataire ayant des droits supplémentaires pour gérer la colocation et ses membres.



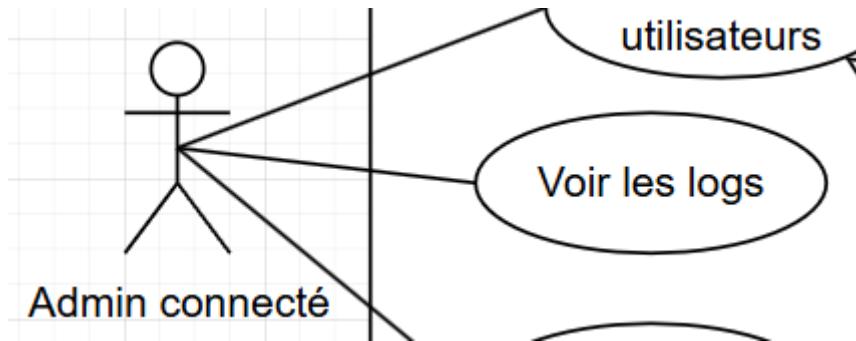
Admin connecté : administrateur ayant un contrôle total sur les utilisateurs et les colocations.



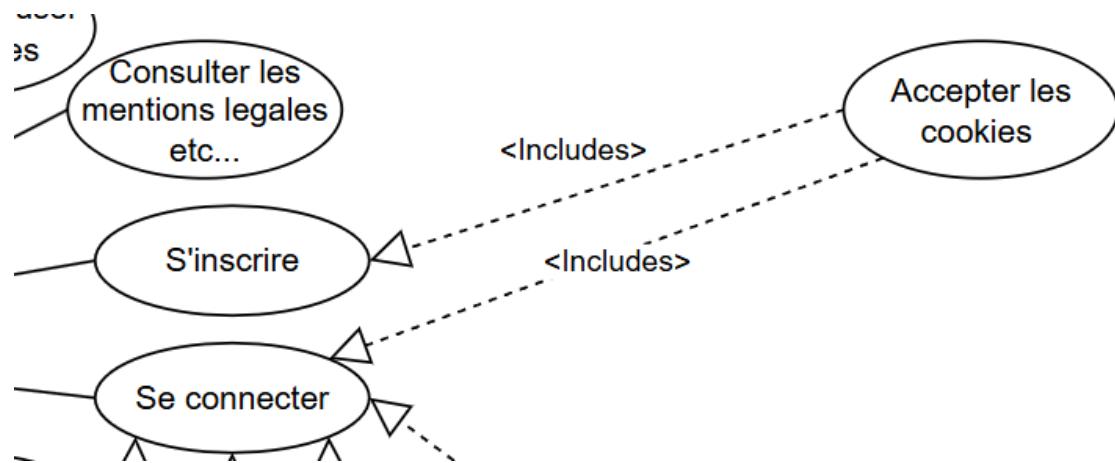
Un **cas d'utilisation** représente une fonctionnalité ou un service offert par le système à un ou plusieurs acteurs. Il décrit une interaction entre un acteur et le système pour atteindre un but concret. Par exemple, des cas comme "Se connecter", "Créer une tâche", ou "Partager une dépense" sont des cas d'utilisation dans Cooloc.



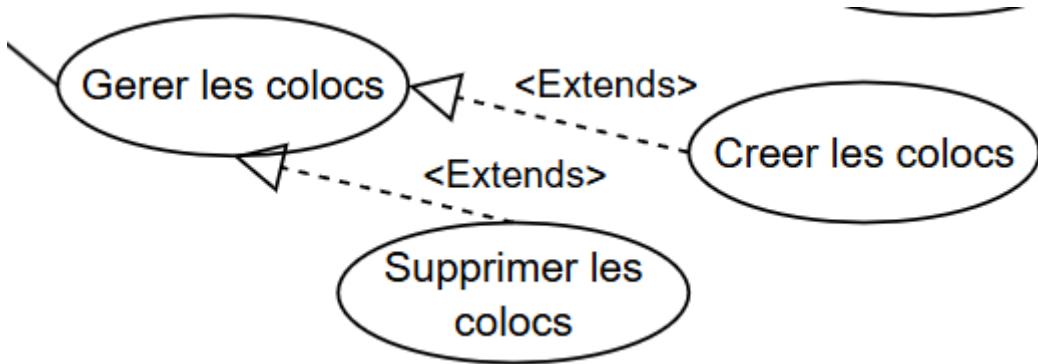
Une **association** est une ligne pleine qui relie un acteur à un cas d'utilisation. Elle indique que l'acteur peut initier ou participer à ce cas d'utilisation. Par exemple, le lien entre le *Colocataire* et *Consulter les dépenses* signifie que ce type d'utilisateur a la possibilité d'effectuer cette action dans le système.



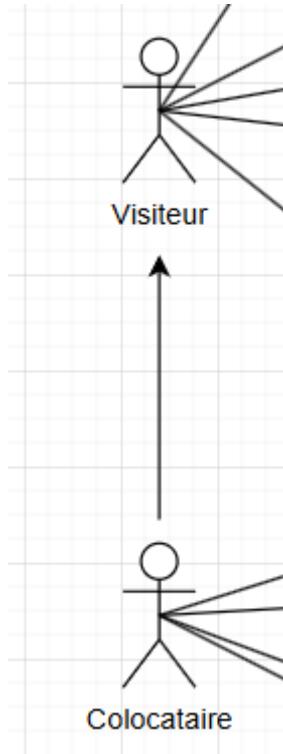
La relation « **include** » (<<include>>) est une relation entre deux cas d'utilisation, signifiant que le second cas est **toujours exécuté** dans le cadre du premier. Elle permet de factoriser un comportement commun. Par exemple, "Se connecter" inclut "Vérifier le mot de passe". Dans Cooloc, des cas comme "S'inscrire", "Se connecter", ou "Consulter qui sommes-nous" incluent des fonctionnalités partagées comme "Consulter les mentions légales" ou "Accepter les cookies".



La relation « **extend** » (<<extend>>) signifie que le comportement du cas d'utilisation étendu est **optionnel**, exécuté **sous certaines conditions**. C'est une extension de fonctionnalité. Par exemple, dans Cooloc, "Consulter les tâches" peut être étendu par "Créer une tâche", "S'attribuer une tâche", "Modifier la tâche", etc. Ces actions ne sont pas toujours exécutées, mais peuvent l'être selon le contexte.



La **généralisation** est une relation hiérarchique entre acteurs (ou entre cas d'utilisation), où un acteur hérite des rôles et droits d'un autre acteur. Par exemple, dans le diagramme de Cooloc, le **Responsable connecté** hérite des droits du **Colocataire**, ce qui signifie qu'il peut effectuer toutes les actions d'un colocataire en plus de ses priviléges propres. De même, l'**Admin connecté** hérite des responsabilités du Responsable.



Le diagramme de cas d'utilisation de **Cooloc** représente l'ensemble des fonctionnalités offertes par l'application de gestion de colocation, en les

répartissant selon les types d'utilisateurs. Un **visiteur**, qui n'est pas encore connecté, peut s'inscrire, se connecter, consulter des informations publiques (mentions légales, qui sommes-nous), et accepter ou refuser les cookies. Une fois connecté, un utilisateur devient **colocataire** et accède à des fonctionnalités liées à son profil, à la gestion des absences, des tâches ménagères et des dépenses. Il peut consulter, créer, modifier ou supprimer des éléments selon les cas. Le **responsable de colocation**, en tant que colocataire avec des droits supplémentaires, peut modifier les informations de la colocation, gérer les membres (ajouter, supprimer, exclure) et superviser l'organisation générale. Enfin, **l'administrateur** dispose de priviléges élargis sur toute la plateforme : gestion globale des utilisateurs, des colocations et accès aux logs du système. Le diagramme montre clairement comment les fonctionnalités sont partagées, héritées et étendues, offrant ainsi une vue complète de l'architecture fonctionnelle de l'application selon les profils utilisateurs.



2. Diagramme de Séquence

Ce diagramme de séquence décrit le processus de création d'une tâche dans l'application, initié par un **acteur** (utilisateur connecté). L'utilisateur commence par **remplir un formulaire de création de tâche** via l'interface du **Frontend**, en saisissant les informations suivantes : le nom de la tâche, la date de début, la date de fin, une priorité, éventuellement un utilisateur assigné, ainsi que les jetons de sécurité **JWT** et **CSRF**.

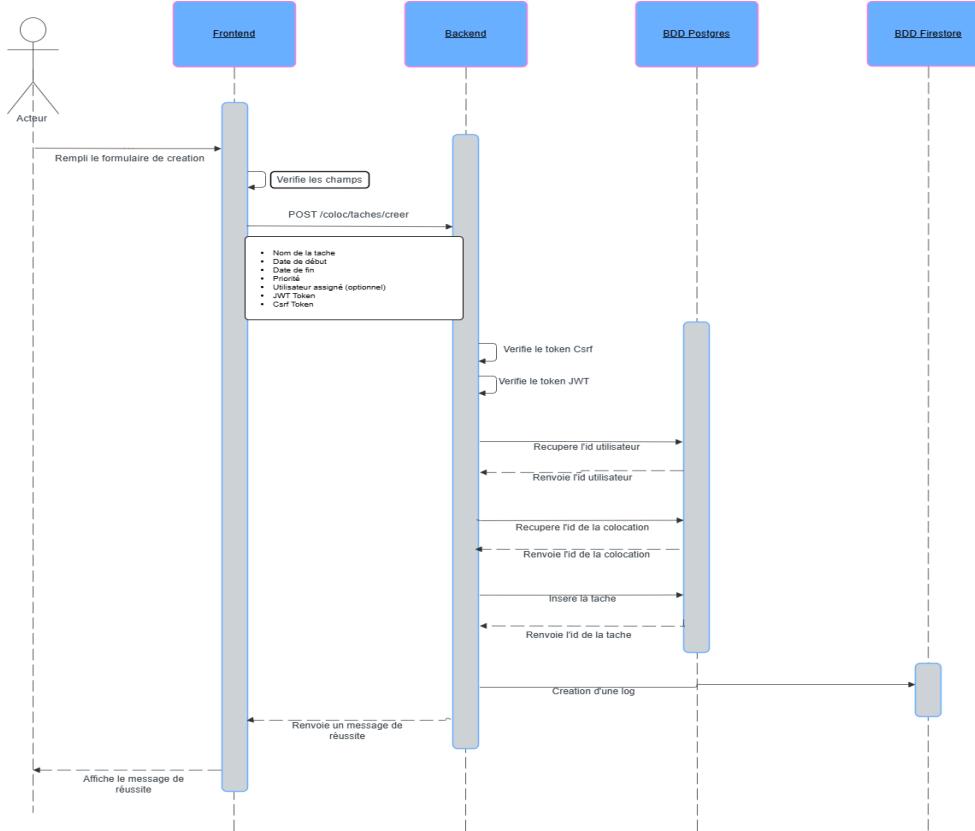
Une fois le formulaire soumis, le Frontend effectue une vérification des champs, puis envoie une requête **POST** vers le **Backend** à l'URL /coloctaches/creer, en incluant toutes les données saisies et les jetons de sécurité.

Le Backend reçoit la requête et commence par **vérifier la validité du token JWT**, qui permet d'authentifier l'utilisateur, puis il vérifie le **token CSRF**, utilisé pour se prémunir contre les attaques par falsification de requête inter-site.

Une fois l'utilisateur authentifié, le Backend interroge la **base de données PostgreSQL** pour **récupérer l'identifiant de l'utilisateur**, qui lui est retourné. Ensuite, il récupère **l'identifiant de la colocation** à laquelle l'utilisateur appartient.

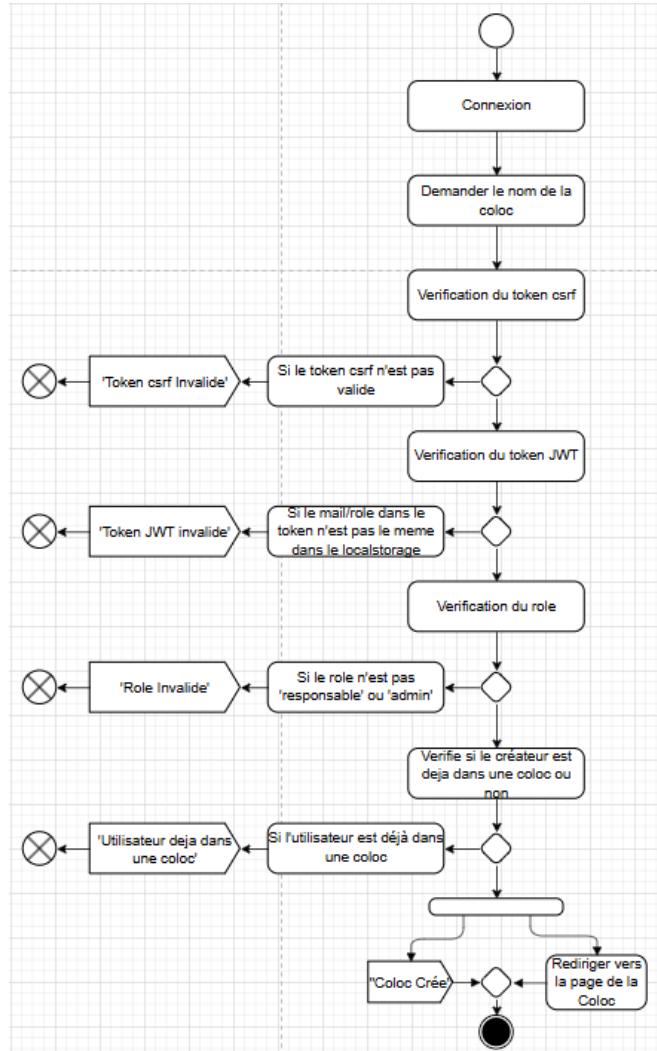
Avec toutes ces informations, le Backend procède à **l'insertion de la tâche dans la base de données**, puis récupère **l'identifiant de la tâche nouvellement créée**. En parallèle, un **log de l'action est généré** dans une base de données séparée, ici **Firestore**, afin de conserver une trace de l'opération.

Enfin, le Backend envoie un **message de réussite** au Frontend, qui l'affiche à l'utilisateur, confirmant que la tâche a bien été créée avec succès.



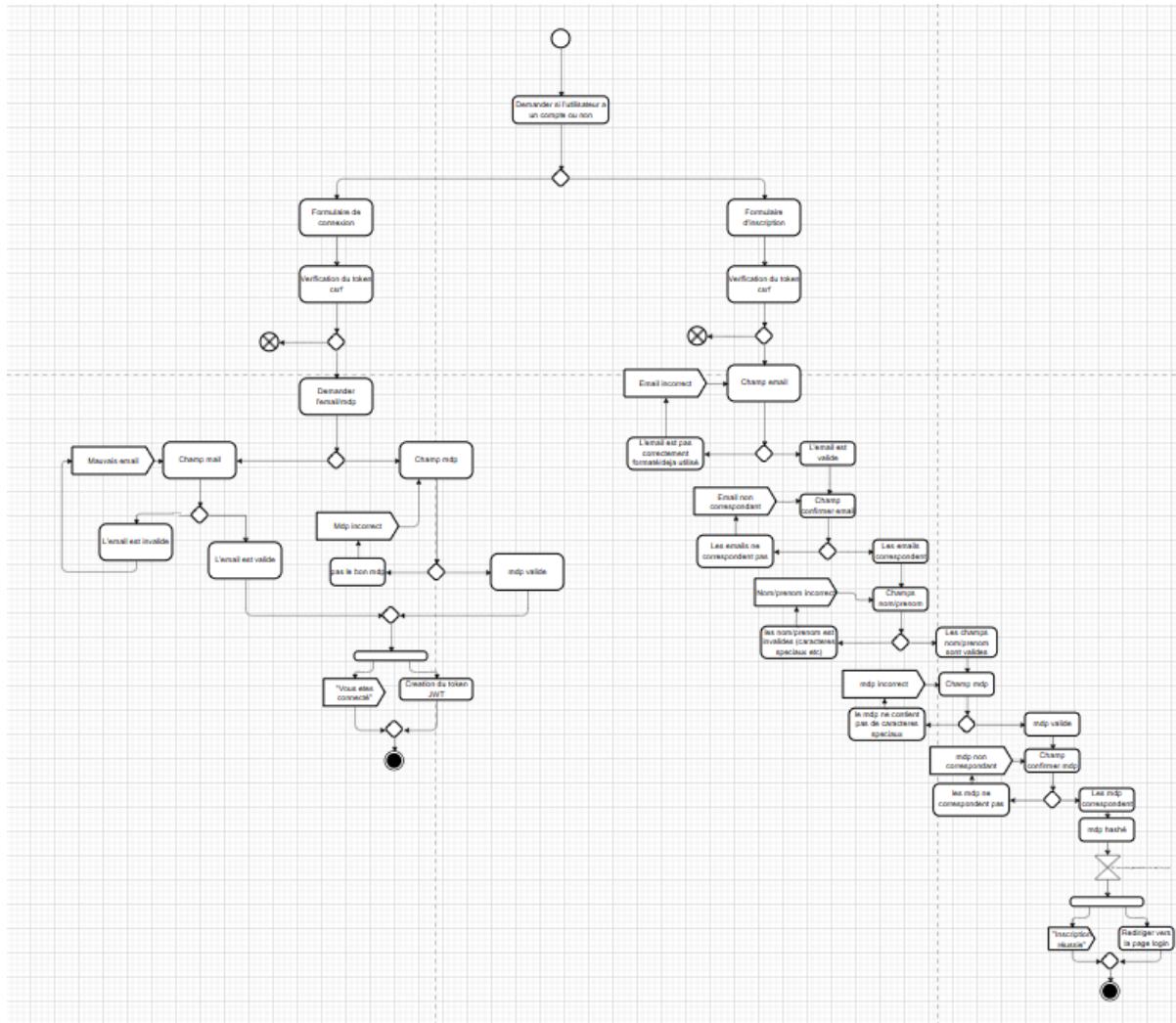
3. Diagramme d'activité

Ce diagramme décrit le processus de création d'une colocation par un utilisateur. Le processus débute après la connexion, lorsque le système demande à l'utilisateur le nom de la colocation à créer. Ensuite, le système vérifie la validité du token CSRF : si celui-ci est invalide, le processus s'arrête avec un message "Token csrf invalide". Si le token CSRF est valide, le système procède à la vérification du token JWT. Si le token JWT est invalide ou si les informations du token ne correspondent pas à celles stockées localement, le processus s'arrête avec un message "Token JWT invalide". Si le token JWT est valide, le système vérifie le rôle de l'utilisateur. Si le rôle n'est pas "responsable" ou "admin", le processus s'arrête avec un message "Rôle invalide". Si le rôle est correct, le système vérifie si l'utilisateur est déjà membre d'une colocation. Si c'est le cas, le processus s'arrête avec un message indiquant que l'utilisateur est déjà dans une colocation. Sinon, la colocation est créée et l'utilisateur est redirigé vers la page de la colocation.



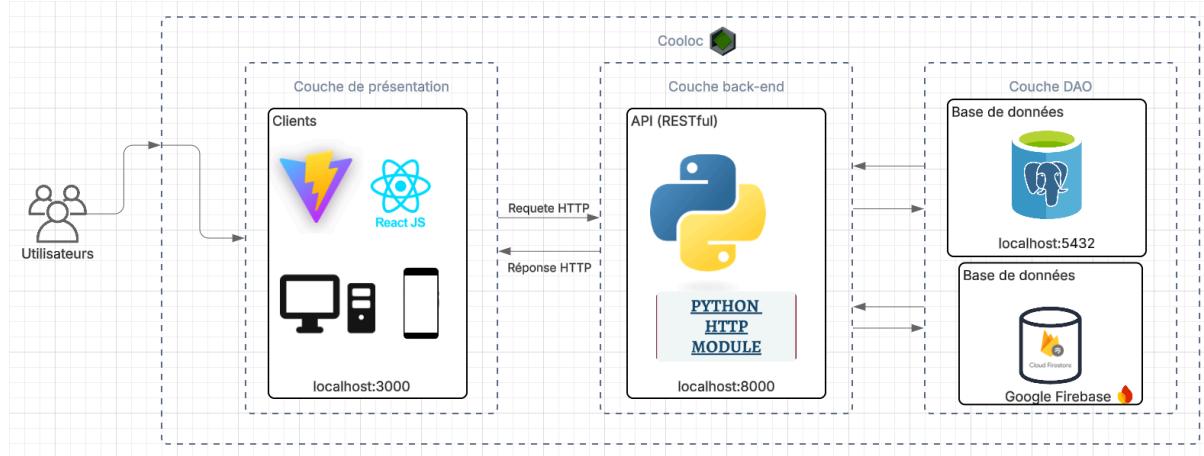
Ce diagramme illustre le processus d'inscription de l'utilisateur et de gestion de profil. Le processus commence lorsque l'utilisateur soumet un formulaire d'inscription. Le système vérifie le token associé au formulaire d'inscription. Si le token est invalide, le processus se termine avec un message d'erreur. Si le token est valide, le système valide le formulaire d'inscription. Si le formulaire est invalide, le processus se termine avec un message d'erreur. Si le formulaire est valide, le système sauvegarde les informations du nouvel utilisateur. Le système envoie un email à l'utilisateur. Si l'envoi de l'email échoue, le processus se termine avec un message d'erreur. Le système met à jour le mot de passe de l'utilisateur. Si la mise à jour du mot de passe échoue, le processus se termine avec un message d'erreur. Le système met à jour le profil de l'utilisateur. Si la mise à jour du profil échoue, le processus se termine avec un message d'erreur. Le système envoie un email confirmant la mise à jour du profil. Si l'envoi de l'email échoue, le processus se termine avec un message d'erreur. Le système met à jour le mot de passe de l'utilisateur pour le profil. Si la mise à jour du mot de passe échoue, le processus se termine avec un message d'erreur. Le système effectue une mise à jour finale sur le profil de l'utilisateur. Si la mise à jour finale échoue, le processus se termine avec

un message d'erreur. Si toutes les étapes sont réussies, l'utilisateur est redirigé vers la page appropriée.



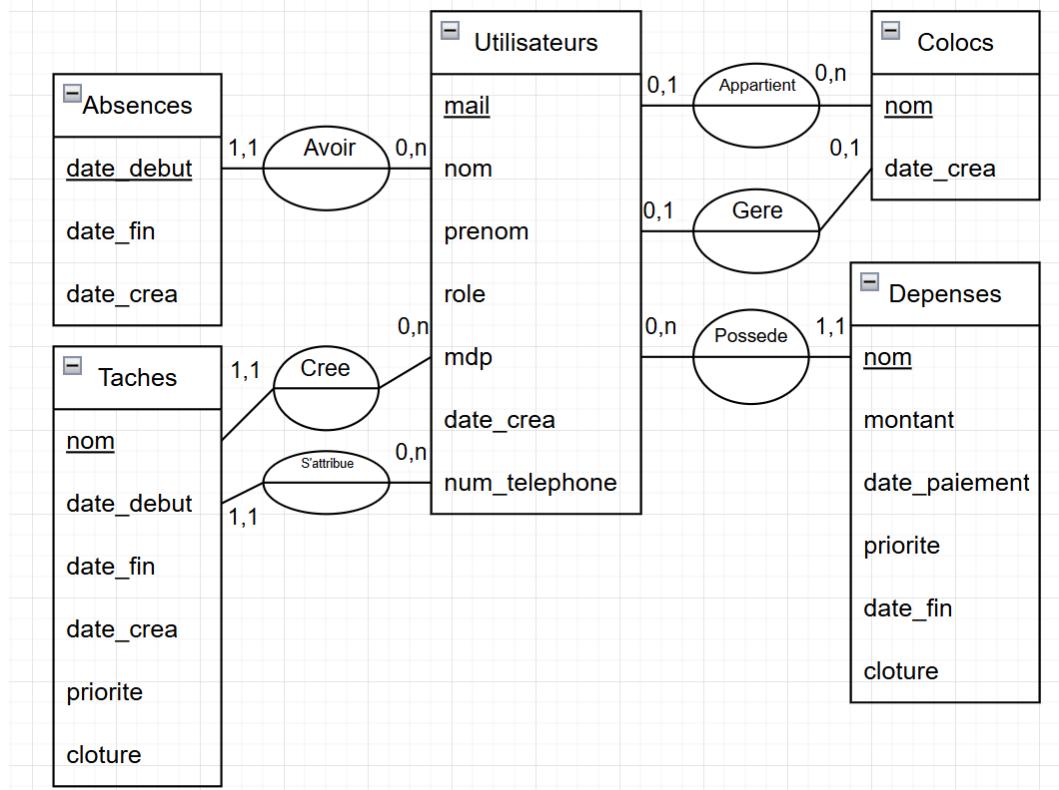
Ces diagrammes servent à visualiser les flux de travail pour l'authentification, l'autorisation, l'inscription et la gestion de profil des utilisateurs, en s'assurant que chaque étape est clairement définie et que les problèmes potentiels sont abordés.

4. Diagramme d'architecture



Le diagramme d'architecture présenté illustre l'organisation globale de l'application Cooloc, structurée en trois couches principales : la couche de présentation, la couche back-end et la couche DAO (Data Access Object). Les utilisateurs interagissent d'abord avec la couche de présentation, qui regroupe différents clients, notamment des applications web développées avec Vite.js et React.js, accessibles depuis divers appareils (ordinateurs, smartphones, etc.) via l'URL localhost:3000. Ces clients envoient des requêtes HTTP à la couche back-end, qui héberge une API RESTful développée en Python, fonctionnant sur localhost:8000. Cette API agit comme un intermédiaire, recevant les requêtes des clients, traitant les données et renvoyant les réponses appropriées. Pour accéder et manipuler les données, la couche back-end communique avec la couche DAO, qui comprend deux types de bases de données : une base PostgreSQL, accessible sur localhost:5432, et une base de données cloud, Google Firebase (Cloud Firestore). Ainsi, l'architecture met en avant une séparation claire des responsabilités : la présentation pour l'interface utilisateur, le back-end pour la logique métier et l'orchestration, et la DAO pour la gestion et la persistance des données, garantissant ainsi modularité, évolutivité et maintenabilité de l'application.

5. MCD



Le Modèle Conceptuel de Données (MCD) est un outil fondamental en conception de bases de données. Il permet de représenter de manière graphique et abstraite l'organisation des données, les entités principales du système, leurs attributs, ainsi que les relations qui existent entre elles. Le MCD facilite la compréhension du fonctionnement global d'une application, en mettant en avant les interactions entre les différents objets manipulés, sans se soucier des détails techniques de l'implémentation.

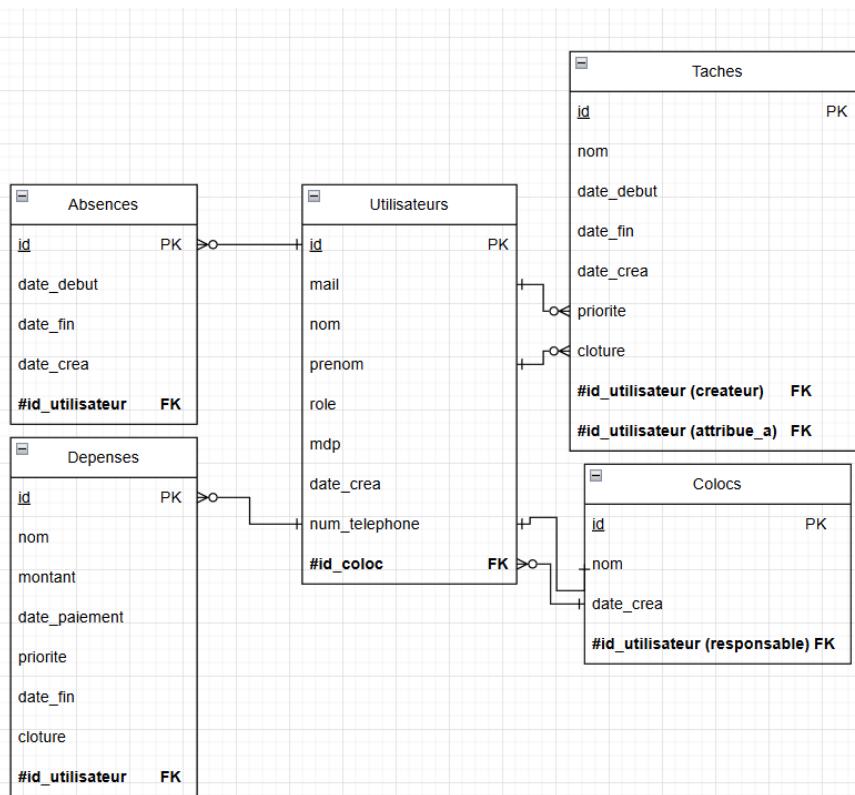
Dans un MCD, **les entités** (représentées par des rectangles) correspondent aux objets ou concepts majeurs du domaine étudié, comme ici « Utilisateurs », « Colocs », « Absences », « Taches » et « Dépenses ». **Les attributs** (listés à l'intérieur des entités) décrivent les propriétés de chaque entité, par exemple, pour « Utilisateurs » : mail, nom, prénom, rôle, mot de passe, date de création, numéro de téléphone, etc. **Les associations** (représentées par des ovales reliés aux entités) traduisent les liens ou interactions entre les entités, comme « Avoir », « Crée », « S'attribue », « Appartient », « Gère » et « Possède ». **Les cardinalités** (indiquées par des chiffres près des liens) précisent le nombre minimum et maximum d'occurrences impliquées dans chaque relation.

Dans ce MCD, l'entité centrale est « Utilisateurs », qui regroupe toutes les informations personnelles et d'identification des membres de l'application. Un utilisateur peut avoir plusieurs absences (« Avoir »), créer plusieurs tâches (« Crée »), et se voir attribuer des tâches (« S'attribue »). Les utilisateurs appartiennent à des colocations (« Appartient »), peuvent en gérer une (« Gère »), et possèdent des dépenses (« Possède »). L'entité « Colocs » représente les différentes colocations,

chacune identifiée par un nom et une date de création, et peut être gérée par un utilisateur. L'entité « Tâches » recense les tâches à effectuer, avec leurs dates, priorités et état de clôture, tandis que « Absences » permet de suivre les périodes d'indisponibilité des utilisateurs. Enfin, l'entité « Dépenses » regroupe les dépenses liées à la colocation, avec leur montant, date de paiement, priorité, date de fin et état de clôture.

Ainsi, ce MCD offre une vision claire et structurée de l'ensemble des données manipulées par l'application, ainsi que des interactions possibles entre les utilisateurs, les tâches, les absences, les colocations et les dépenses, facilitant ainsi la conception et la gestion de la base de données sous-jacente.

6. MLD



Le Modèle Logique de Données (MLD) est une étape clé dans la conception d'une base de données relationnelle. Il traduit le Modèle Conceptuel de Données (MCD) en une structure plus technique, adaptée à l'implémentation dans un Système de Gestion de Base de Données (SGBD) comme PostgreSQL. Le MLD précise la structure des tables, les clés primaires (PK), les clés étrangères (FK), ainsi que les types de relations entre les différentes entités.

Dans un MLD, **chaque entité du MCD devient une table** (représentée par un rectangle), les **attributs** deviennent des colonnes de ces tables, et **les relations** sont matérialisées par des clés étrangères, qui assurent l'intégrité référentielle entre les tables. Les clés primaires (PK) identifient de façon unique chaque enregistrement d'une table, tandis que les clés étrangères (FK) font le lien entre les tables, en pointant vers la clé primaire d'une autre table.

Dans ce MLD, on retrouve cinq tables principales : **Utilisateurs**, **Colocs**, **Taches**, **Absences** et **Dépenses**.

La table **Utilisateurs** contient les informations personnelles de chaque utilisateur (mail, nom, prénom, rôle, mot de passe, date de création, numéro de téléphone) et possède une clé étrangère `id_coloc` qui relie chaque utilisateur à une colocation.

La table **Colocs** recense les différentes colocations, identifiées par un nom et une date de création, et possède une clé étrangère `id_utilisateur (responsable)` qui désigne l'utilisateur responsable de la colocation.

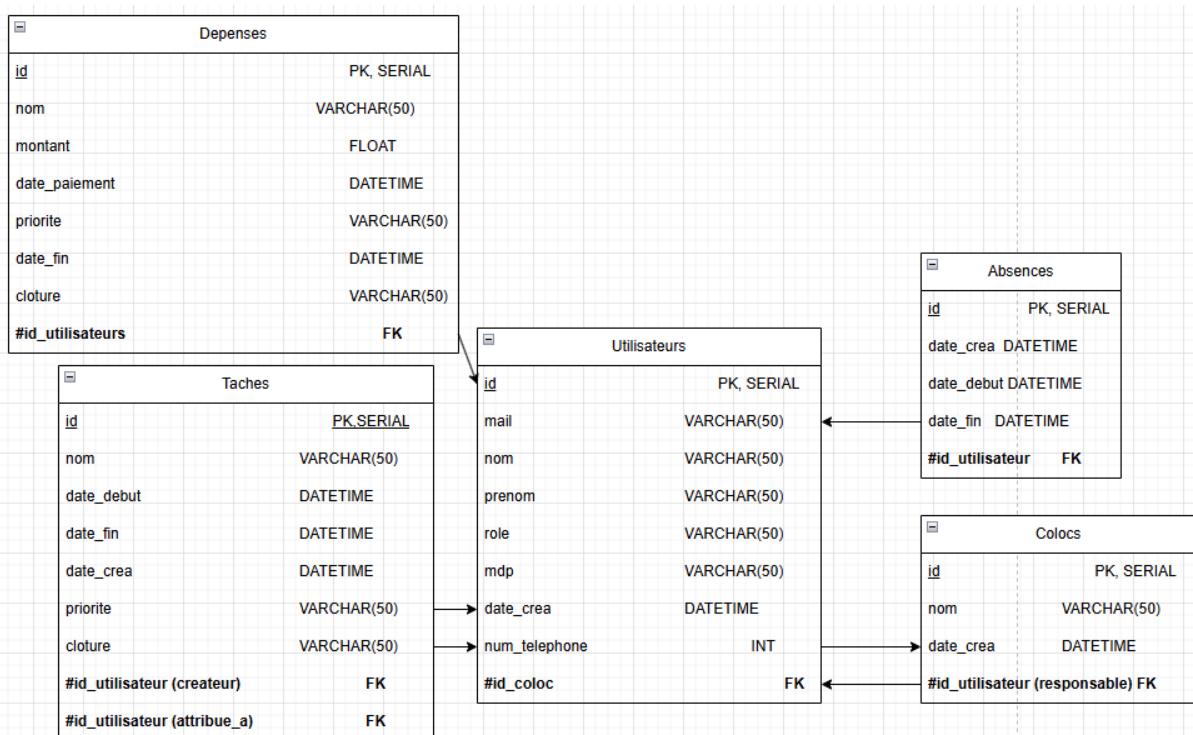
La table **Taches** regroupe les tâches à effectuer, avec leurs caractéristiques (nom, dates, priorité, clôture) et deux clés étrangères : `id_utilisateur (createur)` pour l'utilisateur ayant créé la tâche, et `id_utilisateur (attribue_a)` pour l'utilisateur à qui la tâche est attribuée.

La table **Absences** permet de suivre les périodes d'indisponibilité des utilisateurs, chaque absence étant liée à un utilisateur via la clé étrangère `id_utilisateur`.

Enfin, la table **Dépenses** recense les dépenses liées à la colocation, avec leurs attributs (nom, montant, date de paiement, priorité, date de fin, clôture) et une clé étrangère `id_utilisateur` pour indiquer à quel utilisateur la dépense est associée.

Ainsi, ce MLD offre une représentation détaillée et normalisée de la base de données, en explicitant les liens entre les différentes tables grâce aux clés étrangères, ce qui garantit la cohérence et l'intégrité des données lors de leur manipulation dans le système.

7. MPD



G. Analyser et maquetter les besoins

1. Charte graphique

Couleurs



Logo & Polices



Cooloc utilise Montserrat, Lorem ipsum 123456789

Cooloc utilise Montserrat, Lorem ipsum 123456789

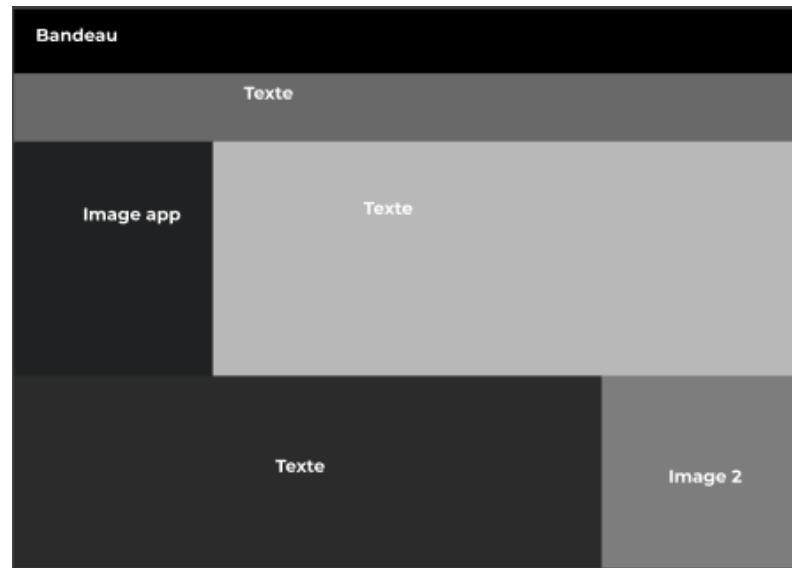
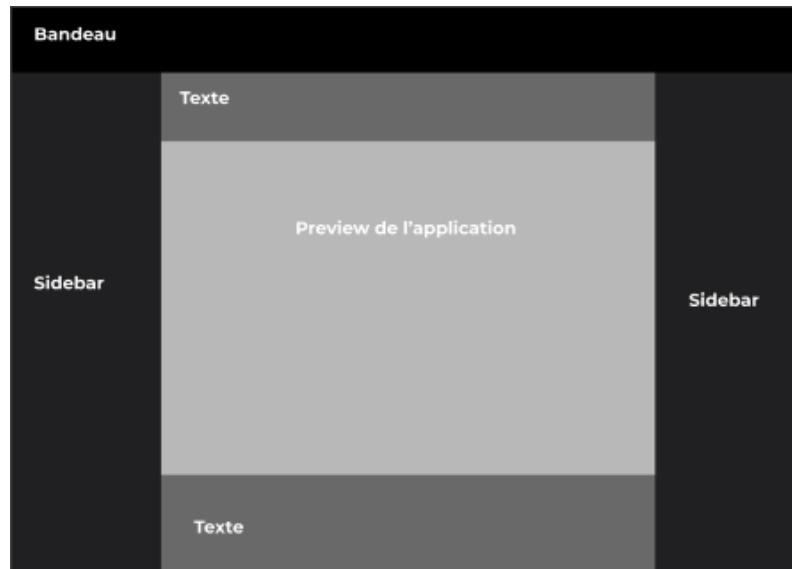
Cooloc utilise Montserrat, Lorem ipsum 123456789

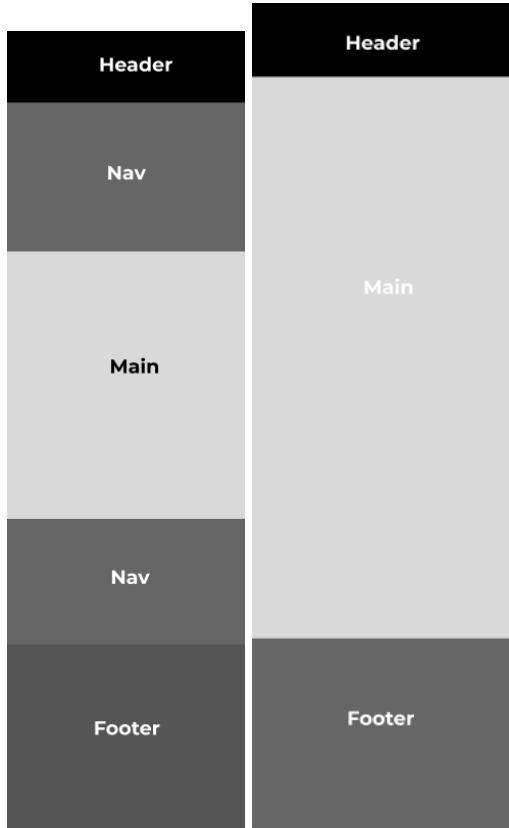
Cooloc utilise Montserrat, Lorem ipsum 123456789

La charte graphique de Cooloc se compose d'une palette de couleurs harmonieuse, principalement basée sur la teinte #FEFAE0, déclinée en plusieurs nuances allant du beige clair à des tons plus soutenus comme le vert olive, le marron, le noir et un rouge vif (#FE4E0). Ce choix de couleurs vise à instaurer une ambiance chaleureuse, naturelle et conviviale, en accord avec l'esprit de la colocation et la simplicité d'utilisation recherchée pour l'application. Le vert, couleur dominante, évoque la sérénité, la confiance et l'aspect communautaire, tandis que les touches de rouge permettent de mettre en avant les éléments importants ou d'alerte.

Côté identité visuelle, le logo de Cooloc est mis en avant, accompagné d'une typographie unique : la police Montserrat. Cette police moderne, élégante et très lisible est utilisée dans différentes graisses et tailles pour structurer l'information, du titre principal aux textes secondaires et aux éléments en italique. Ce choix typographique renforce la clarté, la modernité et l'accessibilité de l'interface, tout en assurant une cohérence graphique sur l'ensemble des supports de communication de Cooloc.

2. Zoning/Wireframes/Maquettes





Le zoning est une étape essentielle dans la conception d'une interface utilisateur. Il consiste à organiser visuellement les différentes zones fonctionnelles d'une page ou d'une application, sans entrer dans le détail graphique. L'objectif est de définir la structure générale, la hiérarchie de l'information et la disposition des éléments principaux (bandeau, navigation, contenu, images, etc.), afin d'optimiser l'ergonomie et l'expérience utilisateur avant de passer à la phase de maquettage détaillé.

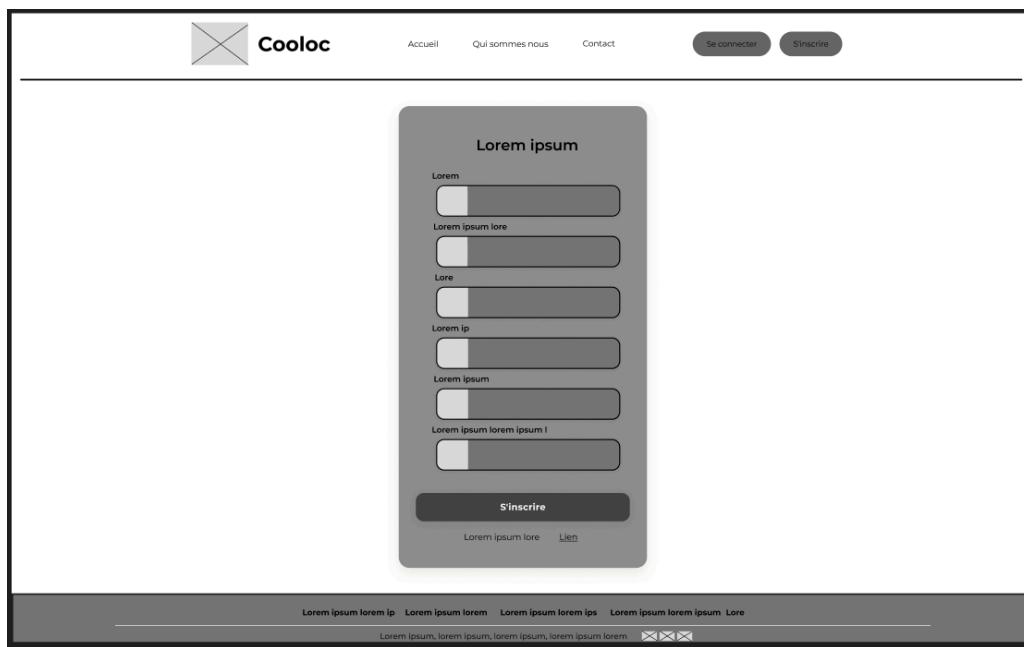
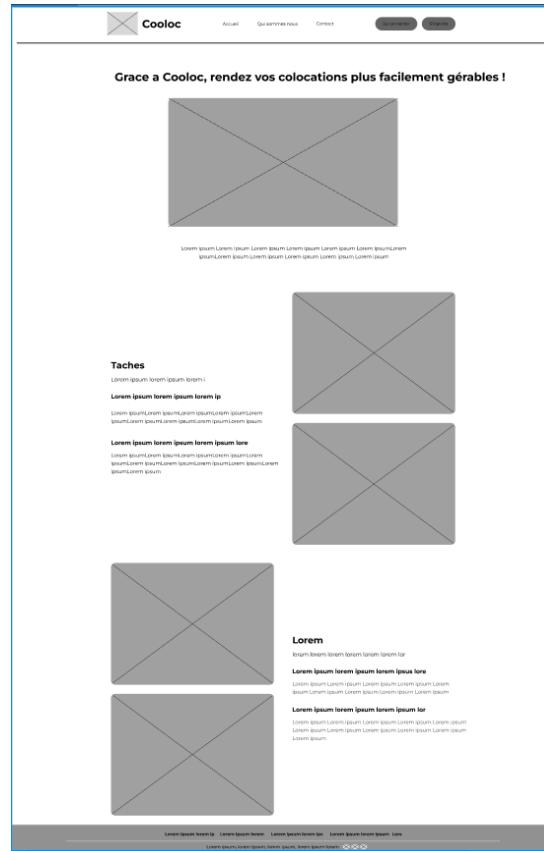
Dans mes zoning, on distingue clairement une adaptation de la structure selon le support utilisé, que ce soit sur ordinateur ou sur mobile.

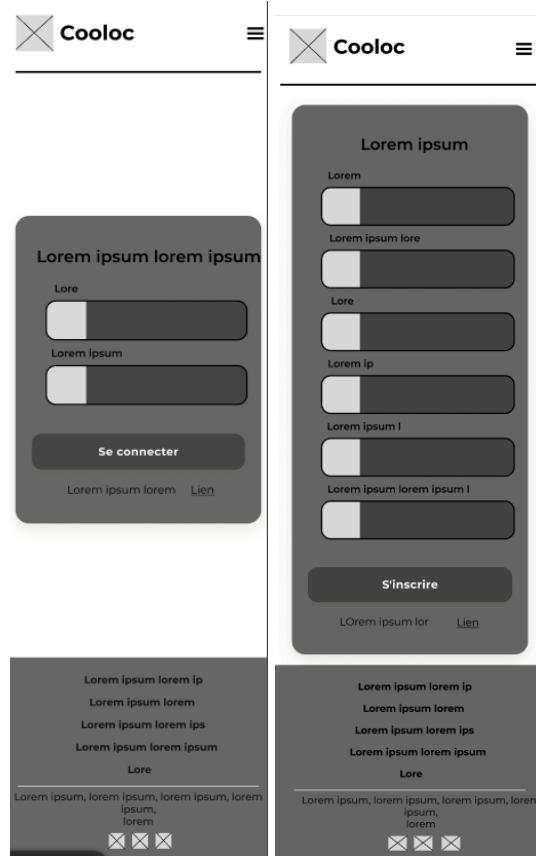
Pour la version PC, les schémas montrent des interfaces riches, avec un bandeau horizontal en haut de page pour le titre ou la navigation principale, des sidebars latéraux pour accéder rapidement à des fonctionnalités ou à des menus secondaires, et une large zone centrale dédiée au contenu principal, comme la prévisualisation de l'application ou des textes explicatifs. Certains écrans intègrent également des espaces réservés à des images ou à des blocs de texte complémentaires, permettant de bien séparer les informations et d'aérer la présentation.

Pour la version mobile, la structure est repensée pour s'adapter à la verticalité de l'écran. On retrouve un header en haut, suivi d'une navigation (Nav) qui peut être placée avant ou après la zone principale (Main), puis un footer en bas de page. L'organisation privilégie la simplicité et la lisibilité, avec un enchaînement vertical

des sections pour faciliter la navigation tactile et garantir un accès rapide aux informations essentielles, même sur un petit écran.

Ainsi, ces zoning illustrent une réflexion sur l'ergonomie et l'accessibilité, en proposant des interfaces adaptées à chaque type de support, tout en assurant une cohérence visuelle et fonctionnelle sur l'ensemble de l'application.





Le wireframe présenté illustre la structure et l'organisation visuelle des principales pages de l'application Cooloc, en se concentrant sur l'agencement des éléments sans entrer dans les détails graphiques finaux.

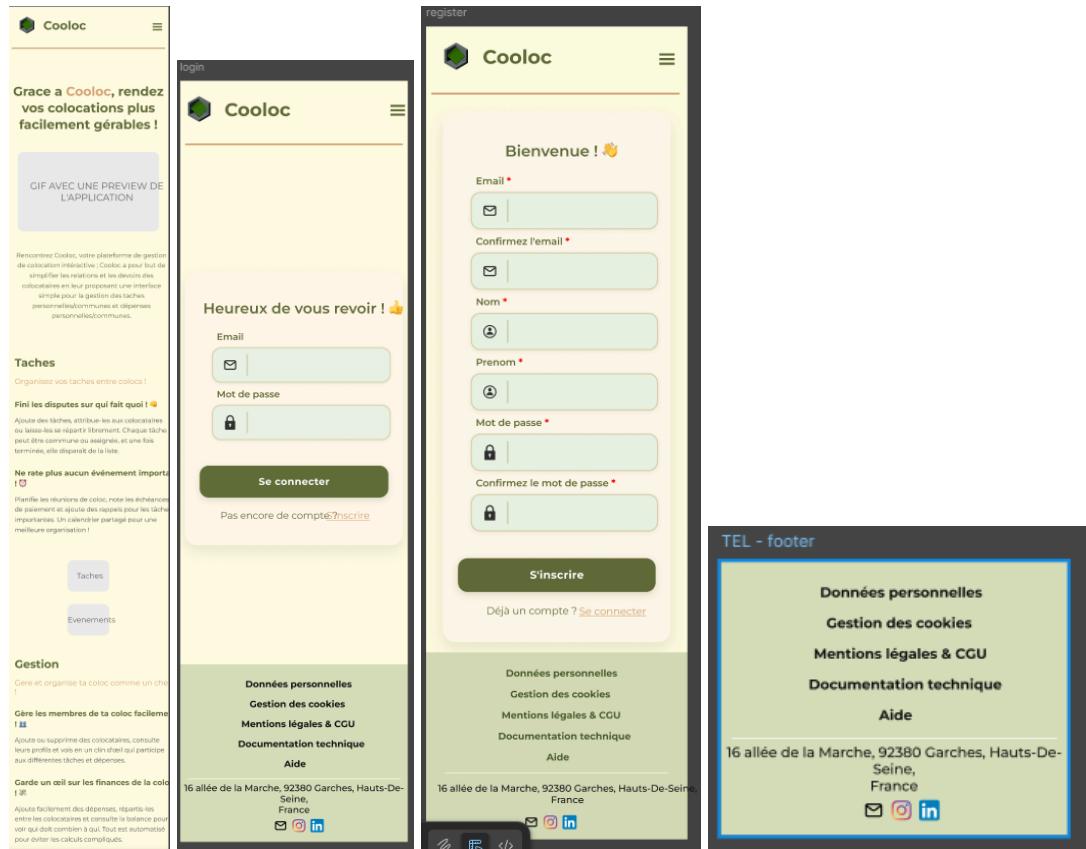
Sur la première maquette (à gauche), on retrouve la page d'accueil, structurée de façon verticale avec un en-tête comprenant le logo Cooloc et un menu de navigation simplifié. Juste en dessous, un message d'accroche met en avant la promesse de l'application. La page est ensuite organisée en plusieurs sections, chacune composée d'un visuel (représenté par un rectangle gris) accompagné d'un titre et d'un court texte descriptif, permettant de présenter les fonctionnalités principales comme la gestion des tâches ou d'autres aspects de la colocation. Enfin, un pied de page regroupe les liens secondaires et les informations légales, assurant une navigation claire et complète.

La seconde maquette (à droite) correspond à une page de type formulaire, probablement la page d'inscription ou de connexion. On y retrouve un en-tête similaire, avec le logo, les liens de navigation principaux (Accueil, Qui sommes-nous, Contact) et des boutons d'accès rapide pour se connecter ou s'inscrire. Le formulaire est centré sur la page, présenté dans un encadré pour le mettre en valeur et faciliter la saisie des informations par l'utilisateur. Chaque champ du formulaire est bien espacé, avec un bouton d'action principal pour valider l'inscription. Le pied de page, en bas de la page, reprend la structure de l'accueil, avec des liens et informations complémentaires.

Ces wireframes témoignent d'une volonté de clarté, de simplicité et d'accessibilité, en mettant en avant les éléments essentiels pour l'utilisateur tout en assurant une navigation fluide et intuitive sur l'ensemble de l'application.

The image displays three wireframe prototypes for the Cooloc application, arranged vertically:

- PC - Homepage:** This wireframe shows the main landing page with a large central area for previewing the application. Below it, there are sections for "Tâches" (Tasks) and "Gestion" (Management), each containing several cards with text and icons.
- Gestion des tâches:** This wireframe shows a detailed view of the task management section. It includes a "Créer une tâche" (Create Task) form with fields for date, priority, and assignee, and a "Liste des tâches" (List of Tasks) table showing task details like title, date, and status.
- Mon Profil:** This wireframe shows the user profile management screen. It features a "Mon Profil" header, a "Informations personnelles" (Personal Information) form with fields for email, first name, last name, and a "Modifier" (Edit) button, and a "Supprimer le compte" (Delete Account) button at the bottom.



Les maquettes présentées illustrent l'apparence finale et l'ergonomie de l'application Cooloc, aussi bien sur mobile que sur ordinateur, en mettant en avant la cohérence graphique, la lisibilité et la simplicité d'utilisation.

Pour la version mobile, chaque écran est pensé pour offrir une expérience fluide et intuitive. Sur la page d'accueil, on retrouve le logo Cooloc et un menu hamburger en haut, suivis d'un message d'accroche et d'un aperçu animé de l'application. Les fonctionnalités principales, comme la gestion des tâches ou la gestion des membres, sont mises en avant à travers des sections distinctes, chacune accompagnée d'un bouton d'accès rapide. Les pages de connexion et d'inscription sont épurées, avec des champs bien espacés, des icônes pour renforcer la compréhension, et un bouton d'action clair. Le footer, présent sur chaque écran, regroupe les liens essentiels (données personnelles, gestion des cookies, mentions légales, documentation technique, aide) ainsi que les coordonnées et les réseaux sociaux, assurant ainsi un accès rapide à l'information et au support.

Côté desktop, la maquette de la page d'accueil reprend la même logique de clarté et de hiérarchisation de l'information, avec un en-tête fixe comprenant le logo, la navigation principale et les boutons d'accès utilisateur. Le contenu est organisé en sections horizontales, alternant textes explicatifs et visuels pour présenter les atouts de l'application. Les pages de gestion de profil et de gestion des tâches sont centrées, avec des formulaires bien structurés, des boutons d'action visibles

et des informations personnelles facilement modifiables. Le footer, identique à la version mobile, garantit la cohérence et l'accessibilité des informations légales et de contact sur toutes les pages.

Ces maquettes traduisent une volonté de proposer une interface moderne, accessible et rassurante, en s'appuyant sur une palette de couleurs douce, une typographie lisible et des éléments graphiques sobres. L'ensemble vise à faciliter la prise en main de l'application, à valoriser les fonctionnalités clés et à offrir une expérience utilisateur agréable, quel que soit le support utilisé.

H. Développement de l'interface utilisateur

1. Point d'entrée de l'application et navigation conditionnelle

L'application Cooloc propose une expérience utilisateur fluide dès le point d'entrée, que ce soit sur mobile ou sur ordinateur. L'accès initial se fait via une page d'accueil accueillante, mettant en avant la promesse de l'application et ses principales fonctionnalités. Dès la première connexion, l'utilisateur est invité à s'authentifier ou à créer un compte.

La navigation conditionnelle est un élément central de l'ergonomie de Cooloc : selon le rôle de l'utilisateur (colocataire, responsable ou administrateur), l'interface et les options de navigation s'adaptent dynamiquement. Après authentification, l'utilisateur est redirigé vers un tableau de bord personnalisé, où seules les fonctionnalités correspondant à son rôle sont accessibles. Cette gestion conditionnelle de la navigation garantit à la fois la sécurité, la simplicité d'utilisation et la clarté de l'interface, en évitant de surcharger l'utilisateur avec des options qui ne le concernent pas.

2. Navigation par onglets et séparation des rôles

La navigation principale de Cooloc repose sur un système d'onglets ou de menus, clairement identifiés dans l'interface, permettant à l'utilisateur d'accéder rapidement aux différentes sections de l'application : gestion des tâches, gestion des membres, dépenses, profil, etc.

La séparation des rôles est strictement respectée grâce à une logique de rendu conditionnel côté front-end : chaque onglet ou section n'est visible et accessible que si le rôle de l'utilisateur le permet. Par exemple, un colocataire n'aura pas accès aux fonctionnalités de gestion avancée réservées au responsable, ni à l'interface d'administration. Cette séparation garantit la confidentialité des données, la simplicité de navigation et la sécurité des opérations sensibles.

3. Fonctionnalités disponibles pour un colocataire

Le rôle de colocataire dans Cooloc donne accès à un ensemble de fonctionnalités essentielles pour la vie en colocation. Un colocataire peut :

- ❖ Consulter la liste des tâches à effectuer, voir celles qui lui sont attribuées et marquer une tâche comme réalisée.
- ❖ Visualiser les absences des autres membres pour mieux s'organiser.
- ❖ Accéder à la liste des dépenses, consulter le détail des paiements et suivre l'état des remboursements.
- ❖ Modifier ses informations personnelles via la page de profil.
- ❖ Recevoir des notifications concernant les nouvelles tâches, les dépenses à régler ou les messages importants du responsable.

L'interface est conçue pour être intuitive, avec des boutons d'action clairs et des informations hiérarchisées, afin de faciliter la gestion quotidienne de la colocation.

4. Fonctionnalités disponibles pour un responsable

Le responsable de la colocation dispose de droits supplémentaires, lui permettant de gérer l'ensemble du groupe. Il peut :

- ❖ Créer, modifier ou supprimer des tâches et les attribuer aux membres de la colocation.
- ❖ Gérer la liste des membres : inviter de nouveaux colocataires, modifier leurs rôles ou supprimer un membre.
- ❖ Ajouter, modifier ou supprimer des dépenses communes, et suivre l'état des paiements.
- ❖ Gérer les absences des membres pour une meilleure organisation.
- ❖ Accéder à des statistiques ou des bilans sur la gestion de la colocation (tâches accomplies, dépenses, etc.).
- ❖Modifier les informations générales de la colocation (nom, description, etc.).

L'interface dédiée au responsable met l'accent sur la gestion et l'organisation, avec des outils de filtrage, de recherche et de visualisation synthétique des informations.

5. Interface dédiée à l'administrateur

L'administrateur dispose d'une interface spécifique, distincte de celle des utilisateurs classiques. Cette interface lui permet :

- ❖ De superviser l'ensemble des colocations enregistrées sur la plateforme.
- ❖ De gérer les utilisateurs à l'échelle globale : création, modification, suppression, gestion des droits.
- ❖ D'accéder à des outils de monitoring et de statistiques sur l'utilisation de l'application.
- ❖ De gérer les incidents ou les demandes de support des utilisateurs.
- ❖ D'effectuer des opérations de maintenance ou de sauvegarde de la base de données.

L'interface administrateur est conçue pour offrir une vue d'ensemble rapide et des accès directs aux fonctions critiques, tout en assurant la sécurité et la traçabilité des actions.

6. Bilan technique

Sur le plan technique, Cooloc s'appuie sur une architecture moderne et modulaire. Le front-end, développé en React (et potentiellement compatible avec d'autres frameworks comme Vue.js), assure une expérience utilisateur réactive et responsive, adaptée à tous les supports. Le back-end, construit en Python, expose une API RESTful robuste, sécurisée et facilement extensible.

La gestion des rôles et des droits d'accès est centralisée, garantissant la confidentialité et la sécurité des données. La base de données, structurée selon des modèles conceptuels, logiques et physiques rigoureux, permet une gestion efficace des informations et une évolutivité de la plateforme.

L'application a été pensée pour être maintenable, évolutive et facilement déployable grâce à l'utilisation de conteneurs Docker et d'outils de gestion de dépendances modernes. Les choix techniques réalisés assurent à la fois la performance, la fiabilité et la sécurité de l'ensemble du système, tout en offrant une expérience utilisateur optimale.

I. Développement Backend

1. Architecture & modules Python

L'application Cooloc repose sur une architecture modulaire Python bien structurée, organisée autour d'un serveur HTTP personnalisé utilisant la bibliothèque standard ThreadingHTTPServer. L'architecture sépare les responsabilités.

Le serveur principal (server.py) agit comme point d'entrée unique, gérant les requêtes HTTP via les méthodes do_GET, do_POST et do_PUT. Il implémente également la gestion CORS pour permettre les requêtes cross-origin depuis le frontend React. La structure modulaire se manifeste par l'organisation des API en dossiers spécialisés : login/, register/, colocations/, adm/, profil/, roles/, tel/ et forms/. Chaque module Python est conçu pour une fonctionnalité spécifique. Par exemple, le module login gère l'authentification avec hachage bcrypt et génération de tokens JWT, tandis que le module colocations se subdivise en sous-modules pour la création, suppression, gestion des tâches et des utilisateurs. Le module adm regroupe les fonctionnalités d'administration avec gestion des logs et des utilisateurs.

L'architecture utilise également des utilitaires partagés comme utils/generer_csrf.py pour la sécurité, et une couche d'abstraction de base de données via bdd/connexion.py qui gère les connexions PostgreSQL et Firebase.

Cette organisation modulaire facilite la maintenance, les tests unitaires et l'évolution de l'application.

2. Authentification sécurisée

L'authentification de Cooloc implémente plusieurs couches de sécurité pour garantir la protection des données utilisateur. Le système utilise une combinaison de hachage bcrypt, de tokens JWT et de protection CSRF.

La fonction mdp_hash() dans le module login utilise bcrypt avec un facteur de coût de 12, ce qui ralentit significativement les attaques par force brute. Les mots de passe sont stockés sous forme hexadécimale dans la base de données, nécessitant une conversion avant vérification avec bcrypt.checkpw().

```

9  def mdp_hash(mdp):
10     mdp_propre = mdp.encode('utf-8')
11     salt = bcrypt.gensalt(12) # 12 --> eviter bruteforce ralentir l'algo
12     mdp_hashe = bcrypt.hashpw(mdp_propre, salt)
13     return mdp_hashe

```

L'authentification JWT est gérée par la fonction create_token() qui génère des tokens contenant l'email et le rôle de l'utilisateur, signés avec une clé secrète stockée dans les variables d'environnement. Ces tokens sont utilisés pour authentifier toutes les requêtes API suivantes.

```

13  def create_token(data):
14      return jwt.encode({'mail': data['mail'], 'role': data['role']}, jwt_secret, algorithm=jwt_algo)
15
16  def verifier_token(data, token):
17      token_decode = jwt.decode(token, jwt_secret, algorithms=[jwt_algo])
18
19      if token_decode['mail'] != data['mail']:
20          return {'status': 403, 'message': 'Token KO'}
21
22      return {'status': 200, 'message': 'Token OK'}

```

La protection CSRF est implémentée via la fonction verifier_csrf() qui vérifie la présence et la validité du token CSRF dans chaque requête POST/PUT. Cette vérification empêche les attaques cross-site request forgery en s'assurant que les requêtes proviennent bien de l'interface utilisateur légitime.

```

16  def verifier_csrf(data):
17      csrf = data['csrf']
18
19      if not csrf:
20          return {'status': 403, 'message': 'CSRF KO'}
21
22      return {'status': 200, 'message': 'CSRF OK'}

```

L'ensemble du processus d'authentification est centralisé et cohérent, avec des vérifications systématiques de tokens et de CSRF sur toutes les routes sensibles, garantissant ainsi la sécurité de l'application.

3. Gestion des rôles

La gestion des rôles dans Cooloc est basée sur un système de permissions granulaires, permettant de contrôler l'accès aux fonctionnalités selon le rôle de l'utilisateur (colocataire, responsable, administrateur).

Le module `roles/route.py` implémente la logique de changement de rôles avec la fonction `changer_role()`. Cette fonction vérifie d'abord l'authentification via `verifier_token()` en comparant les informations du token JWT avec les données de la requête. Elle vérifie également la protection CSRF avant d'autoriser la modification.

La vérification des rôles est intégrée dans chaque endpoint API. Par exemple, les routes d'administration (`/adm/*`) vérifient que l'utilisateur a le rôle administrateur, tandis que les routes de gestion de colocation vérifient les droits du responsable. Cette vérification se fait via la décodification du token JWT qui contient le rôle de l'utilisateur.

Le système de logs intégré enregistre automatiquement les changements de rôles dans Firebase Firestore, permettant un audit trail complet des modifications de permissions. Cette traçabilité est essentielle pour la sécurité et la conformité. La gestion des rôles est également reflétée dans l'interface utilisateur, où seules les fonctionnalités appropriées au rôle de l'utilisateur sont affichées, créant une expérience utilisateur adaptée et sécurisée.

4. API RESTful

L'API de Cooloc suit les principes RESTful avec une structure de routes cohérente et des méthodes HTTP appropriées. Le serveur implémente les méthodes GET, POST, PUT et OPTIONS pour gérer les différentes opérations CRUD.

Les routes GET sont utilisées pour la récupération de données : `/coloc/voir` pour les informations de colocation, `/coloc/taches/voir-disponibles` pour les tâches, `/profil/voir` pour les profils utilisateur, et `/adm/*` pour les données d'administration. Ces routes acceptent des paramètres de requête et des tokens d'authentification.

Les routes POST gèrent la création de ressources : `/login` et `/register` pour l'authentification, `/coloc/creer` pour les colocations, `/coloc/taches/creer` pour les tâches, et `/adm/*/supprimer` pour les suppressions administratives.

Les routes PUT sont dédiées aux modifications : `/coloc/maj/nom` pour renommer une colocation, `/coloc/utilisateurs/ajouter` et `/coloc/utilisateurs/supprimer` pour la gestion des membres, `/coloc/taches/attribuer` et `/coloc/taches/cloturer` pour la gestion des tâches, et `/profil/maj` pour les mises à jour de profil.

L'API retourne des réponses JSON standardisées avec des codes de statut HTTP appropriés (200 pour succès, 400 pour erreur client, 403 pour accès refusé, 404 pour ressource non trouvée). La documentation de l'API est accessible via Swagger UI sur /docs, facilitant l'intégration et les tests.

5. Base de donnée conforme

La base de données de Cooloc respecte parfaitement les modèles conceptuel, logique et physique définis dans la conception. L'implémentation utilise PostgreSQL comme SGBD principal et Firebase Firestore pour les logs, offrant ainsi une solution hybride optimisée.

La classe Bdd dans bdd/connexion.py gère la connexion PostgreSQL avec les paramètres de configuration stockés dans des variables d'environnement (BDD_USER, BDD_MDP, BDD_NOM, BDD_HOST, BDD_PORT). Cette approche sécurise les informations sensibles et facilite le déploiement.

La structure des tables respecte exactement le MPD défini : la table Utilisateurs avec ses clés primaires et étrangères vers Colocs, la table Colocs avec sa relation vers le responsable, les tables Taches, Absences et Dépenses avec leurs relations appropriées. Les requêtes SQL utilisent des paramètres préparés pour éviter les injections SQL.

La classe Logs utilise Firebase Admin SDK pour connecter à Firestore, permettant un stockage cloud des logs d'activité. Cette séparation entre données métier (PostgreSQL) et logs (Firebase) optimise les performances et la scalabilité.

L'utilisation de transactions SQL via con.conn.commit() garantit l'intégrité des données lors des opérations complexes. La gestion des erreurs avec try/catch et les logs d'erreur permettent un monitoring efficace de la base de données.

Cette implémentation respecte les contraintes d'intégrité référentielle définies dans le MLD, garantissant la cohérence des données et la conformité avec les modèles de conception établis.

J. Sécurité

1. Gestion des rôles

La gestion des rôles dans Cooloc est basée sur un système de permissions granulaires et sécurisées. Trois niveaux de rôles sont définis : colocataire, responsable et administrateur, chacun disposant de droits d'accès spécifiques et contrôlés.

Le module roles/route.py implémente la logique de changement de rôles avec la fonction changer_role(), qui effectue plusieurs vérifications de sécurité avant d'autoriser la modification. La fonction vérifie d'abord l'authentification via verifier_token() en comparant les informations du token JWT avec les données de la requête, puis valide la protection CSRF.

Chaque endpoint API intègre une vérification des rôles appropriée. Les routes d'administration (/adm/*) vérifient que l'utilisateur a le rôle administrateur, les routes de gestion de colocation vérifient les droits du responsable, et les routes de consultation sont accessibles aux colocataires selon leur appartenance à la colocation.

Le système de logs intégré enregistre automatiquement tous les changements de rôles dans Firebase Firestore, créant un audit trail complet des modifications de permissions. Cette traçabilité est essentielle pour la sécurité et la conformité réglementaire.

La gestion des rôles se reflète également dans l'interface utilisateur, où seules les fonctionnalités appropriées au rôle de l'utilisateur sont affichées, créant une expérience utilisateur adaptée et sécurisée.

2. Mots de passe sécurisés

La sécurité des mots de passe dans Cooloc est assurée par l'utilisation de la bibliothèque bcrypt, reconnue pour sa robustesse et sa résistance aux attaques par force brute. L'implémentation utilise un facteur de coût de 12, ce qui ralentit significativement les tentatives de craquage tout en maintenant des performances acceptables.

La fonction mdp_hash() dans le module register génère un salt cryptographiquement sécurisé pour chaque mot de passe, garantissant que même des mots de passe identiques produisent des hachages différents. Les mots de passe sont stockés sous forme hexadécimale dans la base de données PostgreSQL, nécessitant une conversion appropriée lors de la vérification.

La vérification des mots de passe se fait via bcrypt.checkpw(), qui compare de manière sécurisée le mot de passe fourni par l'utilisateur avec le hachage stocké en base. Cette méthode évite les attaques par timing et garantit une comparaison constante.

L'application impose une politique de complexité stricte des mots de passe, et recommande l'utilisation de mots de passe forts. Les tentatives de connexion échouées sont gérées de manière appropriée, sans révéler d'informations sensibles sur l'existence des comptes.

La sécurité des mots de passe est renforcée par l'utilisation de HTTPS en production et par la protection CSRF qui empêche les attaques par force brute automatisées via des requêtes cross-site.

3. Injections XSS

La protection contre les injections XSS (Cross-Site Scripting) dans Cooloc est implémentée à plusieurs niveaux pour garantir la sécurité des données affichées à l'utilisateur.

Le module forms/sanitize.py est dédié à la prévention des injections XSS et SQL, bien que son implémentation soit actuellement minimale. Cette approche nécessite un renforcement pour une protection complète contre les attaques XSS. L'application utilise des paramètres préparés dans toutes les requêtes SQL via la bibliothèque psycopg2, ce qui empêche les injections SQL mais ne protège pas directement contre les XSS. Les données utilisateur sont échappées lors de leur insertion en base de données, mais une validation côté serveur plus stricte serait nécessaire.

Le frontend React utilise par défaut l'échappement automatique des variables dans les templates JSX, ce qui constitue une première ligne de défense contre les XSS. Cependant, l'utilisation de dangerouslySetInnerHTML ou d'autres méthodes d'injection de HTML non sécurisées doit être évitée.

Pour renforcer la protection XSS, il serait recommandé d'implémenter une validation stricte des entrées utilisateur, d'utiliser des bibliothèques de sanitisation comme DOMPurify, et d'ajouter des en-têtes de sécurité HTTP comme Content-Security-Policy.

La configuration actuelle offre une protection de base, mais nécessite des améliorations pour une sécurité optimale contre les attaques XSS sophistiquées.

4. Injections SQL

La protection contre les injections SQL dans Cooloc est robuste et implémentée de manière systématique à travers toute l'application. L'utilisation de paramètres préparés via la bibliothèque psycopg2 garantit une protection efficace contre ce type d'attaque.

Toutes les requêtes SQL dans l'application utilisent des paramètres préparés avec la syntaxe %s, évitant ainsi la concaténation directe de variables utilisateur dans les requêtes. Par exemple, dans le module login : con.cursor.execute(requete, (data['mail'],)) utilise un paramètre préparé pour l'email.

La classe Bdd dans bdd/connexion.py gère les connexions PostgreSQL de manière sécurisée, avec une gestion appropriée des erreurs et des connexions. Les requêtes sont exécutées via le curseur de psycopg2, qui applique automatiquement l'échappement des caractères spéciaux.

Les modules d'API comme roles/route.py, login/login.py et tous les autres modules utilisent systématiquement cette approche sécurisée. Les requêtes UPDATE, INSERT et DELETE sont également protégées par des paramètres préparés.

La validation des entrées utilisateur est effectuée avant l'exécution des requêtes, avec des vérifications de champs obligatoires et de format des données. Cette double protection (validation + paramètres préparés) garantit une sécurité maximale.

L'application n'utilise jamais de requêtes SQL dynamiques construites par concaténation de chaînes, éliminant ainsi le risque d'injection SQL même en cas d'erreur de validation des entrées.

5. Configuration

La configuration de l'application Cooloc est gérée de manière sécurisée et flexible, utilisant des variables d'environnement pour les paramètres sensibles et une structure modulaire pour les différents composants.

Le fichier .env est utilisé pour stocker les informations sensibles comme les clés JWT, les paramètres de base de données (BDD_USER, BDD_MDP, BDD_NOM, BDD_HOST, BDD_PORT) et les clés Firebase. Ce fichier est exclu du contrôle de version via .gitignore pour éviter l'exposition des secrets.

La bibliothèque python-dotenv est utilisée pour charger automatiquement les variables d'environnement au démarrage de l'application. Cette approche centralise la configuration et facilite le déploiement dans différents environnements.

La configuration du serveur est définie dans server.py avec les constantes HOST et PORT, permettant une personnalisation facile selon l'environnement de déploiement. Les en-têtes CORS sont configurés pour permettre les requêtes depuis le frontend React sur localhost:3000.

La configuration de la base de données est encapsulée dans la classe Bdd, avec une gestion d'erreur appropriée et des logs informatifs. La configuration Firebase est gérée via un fichier de clé JSON séparé, également exclu du contrôle de version.

Le fichier sonar-project.properties configure l'analyse statique du code avec SonarQube, définissant les chemins des sources, des tests et des rapports de couverture. Cette configuration facilite l'intégration continue et le monitoring de la qualité du code.

6. Authentification

L'authentification dans Cooloc est basée sur un système robuste combinant hachage sécurisé des mots de passe, tokens JWT et protection CSRF. Cette approche multi-couches garantit la sécurité des sessions utilisateur.

Le processus d'authentification commence par la validation des champs obligatoires (email et mot de passe) dans la fonction login(). L'application vérifie ensuite l'existence de l'utilisateur en base de données et valide le token CSRF pour prévenir les attaques cross-site.

La vérification du mot de passe utilise bcrypt avec bcrypt.checkpw(), comparant de manière sécurisée le mot de passe fourni avec le hachage stocké en base. Les mots de passe sont stockés sous forme hexadécimale, nécessitant une conversion appropriée avant vérification.

En cas d'authentification réussie, un token JWT est généré via create_token() contenant l'email et le rôle de l'utilisateur. Ce token est signé avec une clé secrète stockée dans les variables d'environnement et utilise l'algorithme HS256.

Le token JWT est utilisé pour authentifier toutes les requêtes API suivantes. La fonction verifier_token() dans le module roles décode et valide le token en

vérifiant la signature et en comparant les informations avec les données de la requête.

L'authentification est vérifiée systématiquement sur tous les endpoints sensibles, avec des codes de retour appropriés (200 pour succès, 400 pour erreur client, 403 pour accès refusé). Cette approche centralisée garantit la cohérence de la sécurité à travers l'application.

7. CSRF

La protection CSRF (Cross-Site Request Forgery) dans Cooloc est implémentée de manière systématique pour empêcher les attaques par requêtes forgées cross-site. Cette protection est essentielle pour la sécurité des opérations sensibles.

Le module `utils/generer_csrf.py` utilise la bibliothèque `secrets` pour générer des tokens CSRF cryptographiquement sécurisés via `secrets.token_urlsafe(32)`. Cette méthode produit des tokens de 32 octets encodés en base64, offrant une entropie suffisante pour résister aux attaques par force brute.

La fonction `verifier_csrf()` est implémentée dans plusieurs modules (`login`, `roles`) pour valider la présence et l'authenticité du token CSRF dans chaque requête POST et PUT. Cette vérification est obligatoire avant l'exécution de toute opération modifiant les données.

Le processus de vérification CSRF est intégré dans le flux d'authentification et dans toutes les opérations sensibles. Par exemple, dans le module `login`, la vérification CSRF est effectuée après la validation de l'existence de l'utilisateur mais avant la vérification du mot de passe.

Les tokens CSRF sont transmis dans le corps des requêtes JSON, permettant une intégration transparente avec l'API RESTful. Cette approche est compatible avec les applications frontend modernes utilisant des requêtes AJAX.

La protection CSRF est particulièrement importante pour les opérations d'administration, de modification de profil et de gestion des colocations, où des actions non autorisées pourraient avoir des conséquences graves. Cette implémentation garantit que seules les requêtes légitimes provenant de l'interface utilisateur sont traitées.

8. RGPD

La conformité RGPD (Règlement Général sur la Protection des Données) dans Cooloc est prise en compte à plusieurs niveaux, bien que l'implémentation actuelle nécessite des améliorations pour une conformité complète.

L'application collecte et traite des données personnelles des utilisateurs (nom, prénom, email, numéro de téléphone) pour les fonctionnalités de colocation. Ces données sont stockées dans PostgreSQL avec des mesures de sécurité appropriées (hachage des mots de passe, authentification sécurisée).

Le système de logs dans Firebase Firestore enregistre les actions des utilisateurs, créant un audit trail nécessaire pour la conformité RGPD. Cependant, la politique de rétention des logs et la gestion des droits d'accès aux données personnelles nécessitent une documentation plus détaillée.

L'application implémente le droit à la suppression des données via l'endpoint /profil/supprimer, permettant aux utilisateurs de supprimer leur compte et leurs données personnelles. Cette fonctionnalité respecte le droit à l'effacement (droit à l'oubli) du RGPD.

La sécurité des données est assurée par l'utilisation de variables d'environnement pour les secrets, l'exclusion des fichiers sensibles du contrôle de version, et l'utilisation de connexions sécurisées à la base de données.

Pour une conformité RGPD complète, il serait nécessaire d'ajouter : une politique de confidentialité détaillée, un système de consentement explicite, des mécanismes d'export des données personnelles, une documentation des traitements, et une formation des utilisateurs sur leurs droits.

L'architecture actuelle fournit une base solide pour la conformité RGPD, mais nécessite des améliorations pour répondre à tous les aspects du règlement.

K. DevOps

1. Validation manuelle

La validation manuelle de l'application Cooloc a été effectuée de manière systématique pour vérifier le bon fonctionnement de toutes les fonctionnalités et la conformité aux spécifications.

Les tests manuels ont couvert l'ensemble du cycle de vie utilisateur : inscription, connexion, gestion des profils, création et gestion des colocations, attribution et suivi des tâches, et administration du système. Chaque fonctionnalité a été testée avec différents rôles utilisateur pour vérifier la gestion des permissions.

La validation de l'interface utilisateur a confirmé la responsivité sur différents appareils (ordinateur, tablette, smartphone) et la compatibilité avec les navigateurs modernes. L'ergonomie et l'accessibilité ont été évaluées pour garantir une expérience utilisateur optimale.

Les tests de sécurité manuels ont vérifié la protection contre les injections SQL, la validation des tokens JWT, la protection CSRF et la gestion sécurisée des mots de passe. Les tentatives d'accès non autorisé ont été testées pour confirmer l'efficacité des contrôles d'accès.

La validation de la base de données a confirmé la conformité avec les modèles conceptuel, logique et physique définis. Les contraintes d'intégrité référentielle ont été testées, ainsi que la gestion des erreurs et la récupération après panne.

Les tests de performance ont évalué les temps de réponse de l'API, la gestion de la concurrence et la scalabilité de l'application. Ces tests ont permis d'identifier les goulots d'étranglement potentiels et d'optimiser les performances.

2. Vérification des endpoints avec Postman

La vérification des endpoints API avec Postman a permis de valider le bon fonctionnement de l'API RESTful de Cooloc et de documenter les interactions client-serveur.

Une collection Postman complète a été créée pour tester tous les endpoints de l'application : authentification (/login, /register), gestion des colocations (/coloc/*), gestion des tâches (/coloc/taches/*), administration (/adm/*) et gestion des profils (/profil/*).

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'My Workspace' containing collections like 'Coloc', environments, flows, and history. The main area displays a collection named 'Coloc / Login/Register'. Under this collection, there is a single POST request for the endpoint `http://localhost:8000/login`. The request body is set to 'raw' and contains the following JSON payload:

```

1 {
2     "email": "test@test.com",
3     "mdp": "TEST1234568",
4     "csrf": "g0S_QLR1lg89Pf-hT6ozC-kFnPxx-mQk0UEsJLunQ"
5 }

```

Below the request, there's a 'Response' section with a note 'Click Send to get a response'.

>Login/Register

- POST http://localhost:8000/login
- POST http://localhost:8000/register

Coloc

- GET SEL - Voir les utilisateurs d'une c...
- GET SEL - Voir les taches dispos
- GET SEL - Voir les taches completes
- GET SEL - Voir mes taches
- GET SEL - Voir les infos coloc
- POST INS - Creation d'une coloc
- POST INS - Creation d'une tachge
- POST DEL - Supprimer Coloc
- POST DEL - Supprimer Taches
- PUT MAJ - Nom coloc
- PUT MAJ - Nom tache
- PUT MAJ - Cloturer tache
- PUT MAJ - S'attribuer une tache
- PUT MAJ - Ajouter coloc
- PUT MAJ - Supprimer un coloc

Utils

- GET SEL - Profil
- PUT MAJ - Role
- PUT MAJ - Tel
- PUT MAJ - Profil

Admin

- > Utilisateurs
 - GET SEL - Voir toutes les logs
 - POST DEL - Supprimer une logs

Chaque endpoint a été testé avec différents scénarios : requêtes valides, données manquantes, tokens invalides, permissions insuffisantes et données malformées. Ces tests ont permis de vérifier la robustesse de l'API et la gestion appropriée des erreurs.

Les tests d'authentification ont validé le processus de connexion, la génération et validation des tokens JWT, et la protection CSRF. Les tests de gestion des rôles ont confirmé que seuls les utilisateurs autorisés peuvent accéder aux fonctionnalités sensibles.

La documentation Swagger de l'API a été utilisée en parallèle avec Postman pour vérifier la cohérence entre la documentation et l'implémentation. Les exemples de requêtes et réponses ont été validés pour assurer la précision de la documentation.

Les tests de performance avec Postman ont permis d'évaluer les temps de réponse des endpoints et d'identifier les optimisations nécessaires. Les tests de charge ont été effectués pour valider le comportement de l'application sous stress.

3. Analyse statique avec SonarQube

L'analyse statique du code avec SonarQube a été configurée pour maintenir la qualité du code et détecter les problèmes potentiels de sécurité et de performance dans l'application Cooloc.

Le fichier sonar-project.properties configure l'analyse pour le projet Python, définissant les chemins des sources (backend), des tests (backend/test) et des rapports de couverture (backend/coverage.xml). Cette configuration permet une analyse complète du code backend.

L'analyse SonarQube couvre plusieurs dimensions de la qualité du code : la fiabilité (bugs potentiels), la sécurité (vulnérabilités), la maintenabilité (dette technique) et la couverture de tests. Les règles d'analyse sont adaptées aux bonnes pratiques Python et aux standards de sécurité.

Les métriques de qualité incluent la couverture de code, la duplication de code, la complexité cyclomatique et la dette technique. Ces métriques permettent de suivre l'évolution de la qualité du code au fil du temps et d'identifier les zones nécessitant des améliorations.

L'intégration avec les tests unitaires via pytest et coverage permet de générer des rapports de couverture automatiques. Ces rapports sont analysés par SonarQube pour identifier les parties du code non testées et évaluer la qualité des tests.

L'analyse de sécurité détecte les vulnérabilités potentielles comme les injections SQL, les problèmes d'authentification et les failles de sécurité courantes. Les recommandations de SonarQube permettent d'améliorer la sécurité de l'application de manière proactive.

4. Github Actions & analyse continue du code

L'intégration continue avec GitHub Actions est configurée pour automatiser les tests, l'analyse statique et le déploiement de l'application Cooloc, garantissant la qualité du code à chaque modification.

Le workflow GitHub Actions s'exécute automatiquement à chaque push et pull request, effectuant une série de vérifications : tests unitaires avec pytest, analyse statique avec SonarQube, vérification de la couverture de code et tests de linting avec ESLint pour le frontend.

L'intégration avec SonarQube permet une analyse continue de la qualité du code, avec des rapports automatiques générés à chaque build. Les métriques de qualité sont suivies dans le temps, permettant d'identifier les tendances et les zones d'amélioration.

Les tests unitaires sont exécutés dans un environnement isolé avec des mocks appropriés pour les dépendances externes (base de données, Firebase). Cette approche garantit la fiabilité des tests et évite les faux positifs dus aux services externes.

La configuration des secrets GitHub permet de stocker de manière sécurisée les clés d'API, les tokens d'accès et les paramètres de configuration sensibles. Ces secrets sont utilisés par les workflows GitHub Actions sans être exposés dans le code source.

L'automatisation du déploiement via GitHub Actions permet de déployer automatiquement l'application après des tests réussis, réduisant le temps de mise en production et minimisant les erreurs humaines. Cette approche DevOps garantit la cohérence entre l'environnement de développement et de production.

5. Conteneurisation Docker

La conteneurisation Docker de Cooloc permet un déploiement cohérent et reproductible de l'application, facilitant le développement, les tests et la mise en production.

Le backend utilise un Dockerfile basé sur Python 3.9-slim-bookworm, optimisé pour la taille et la sécurité. L'utilisation de uv pour la gestion des dépendances améliore les performances d'installation et garantit la reproductibilité des builds. Le Dockerfile copie les requirements.txt, installe les dépendances, puis copie le code source.

Le frontend utilise un Dockerfile basé sur Node.js 20-alpine, optimisé pour les applications web. Le processus de build inclut l'installation des dépendances, la construction de l'application avec Vite, et l'exposition du port 3000 pour le développement.

Docker Compose orchestre les services backend et frontend, définissant les ports, volumes et dépendances entre les conteneurs. La configuration inclut des volumes pour le développement local, permettant la modification du code sans reconstruction des images.

Les fichiers .dockerignore excluent les fichiers non nécessaires (node_modules, pycache, .git) des contextes de build, optimisant la taille des images et la vitesse de construction. Cette approche améliore l'efficacité du processus de conteneurisation.

La configuration des variables d'environnement dans Docker Compose permet de personnaliser le comportement de l'application selon l'environnement (développement, test, production). Cette flexibilité facilite le déploiement dans différents environnements.

La conteneurisation garantit la cohérence entre les environnements de développement et de production, évitant les problèmes de "ça marche sur ma machine". Cette approche facilite également la scalabilité et la maintenance de l'application.

6. Perspectives

Les perspectives d'évolution de Cooloc s'articulent autour de plusieurs axes d'amélioration et d'extension, visant à enrichir les fonctionnalités, améliorer la sécurité et optimiser les performances.

L'extension des fonctionnalités pourrait inclure : un système de messagerie interne entre colocataires, un calendrier partagé pour les événements et absences, un système de gestion des factures et paiements, des notifications push en temps réel, et une application mobile native pour iOS et Android.

L'amélioration de la sécurité pourrait passer par : l'implémentation d'une authentification à deux facteurs (2FA), l'ajout d'en-têtes de sécurité HTTP (CSP, HSTS), l'amélioration de la protection XSS avec des bibliothèques de sanitisation, et l'audit de sécurité régulier avec des outils automatisés.

L'optimisation des performances pourrait inclure : la mise en cache Redis pour les données fréquemment consultées, l'optimisation des requêtes SQL avec des index appropriés, la pagination des résultats pour les grandes listes, et l'utilisation de CDN pour les ressources statiques.

L'amélioration de l'expérience utilisateur pourrait passer par : une interface plus intuitive avec des animations et transitions, l'accessibilité pour les utilisateurs en situation de handicap, la personnalisation des thèmes et couleurs, et l'internationalisation pour supporter plusieurs langues.

L'évolution technique pourrait inclure : la migration vers une architecture microservices, l'utilisation de GraphQL pour une API plus flexible, l'implémentation de WebSockets pour les communications en temps réel, et l'intégration avec des services tiers (calendriers, banques, assurances).

Ces perspectives permettront à Cooloc d'évoluer vers une plateforme complète et moderne de gestion de colocation, répondant aux besoins croissants des utilisateurs et aux évolutions technologiques.

VII. Bilan & conclusion

A. Compétences validées

Le développement de l'application Cooloc a permis de valider de nombreuses compétences du titre RNCP37873 "Développeur d'applications", démontrant une maîtrise complète du cycle de développement d'applications web modernes.

Développement d'applications (Bloc 1) : La création de Cooloc valide la compétence "Développer une application informatique" avec la conception et l'implémentation d'une application web complète utilisant React.js pour le frontend et Python pour le backend. L'architecture modulaire et la séparation des responsabilités démontrent une maîtrise des bonnes pratiques de développement.

Gestion de bases de données (Bloc 2) : La compétence "Gérer des données de l'information" est validée par la conception et l'implémentation des modèles conceptuel, logique et physique de la base de données PostgreSQL, ainsi que par l'intégration de Firebase Firestore pour les logs. La gestion des relations entre entités et l'optimisation des requêtes SQL démontrent une expertise en gestion de données.

Sécurité informatique (Bloc 3) : La compétence "Sécuriser une application informatique" est pleinement validée par l'implémentation de l'authentification JWT, du hachage bcrypt des mots de passe, de la protection CSRF, et de la gestion des rôles et permissions. Ces mesures de sécurité garantissent la protection des données utilisateur et l'intégrité du système.

Tests et déploiement (Bloc 4) : La compétence "Tester et déployer une application informatique" est validée par l'implémentation de tests unitaires avec pytest, l'analyse statique avec SonarQube, la conteneurisation Docker, et l'intégration continue. Ces pratiques garantissent la qualité du code et facilitent le déploiement.

Communication et travail en équipe : Bien que développé individuellement, le projet démontre la capacité à documenter le code, créer une documentation technique complète, et présenter les résultats de manière professionnelle, validant ainsi les compétences transversales du titre.

B. Axes d'améliorations sur les contrôles de champs

L'analyse du code de Cooloc révèle plusieurs axes d'amélioration pour renforcer les contrôles de champs et la validation des données, essentiels pour la sécurité et l'expérience utilisateur.

Validation côté client : Le frontend React implémente une validation basique (champs requis, confirmation email/mot de passe, longueur minimale du mot de passe), mais pourrait être enrichie. Il serait pertinent d'ajouter : la validation en temps réel avec feedback visuel, la vérification de la complexité des mots de passe (majuscules, chiffres), la validation du format des numéros de téléphone, et la détection de caractères spéciaux non autorisés.

Validation côté serveur : Le backend Python nécessite un renforcement de la validation des entrées. Actuellement, seule une vérification de présence des champs obligatoires est effectuée. Il serait essentiel d'ajouter : la validation du

format des emails avec des expressions régulières, la vérification de la complexité des mots de passe, la sanitisation des chaînes de caractères pour éviter les injections XSS, et la validation des types de données (entiers, dates, etc.).

Gestion des erreurs : L'application pourrait améliorer la gestion des erreurs de validation en fournissant des messages d'erreur plus précis et localisés. La création d'un système centralisé de validation avec des codes d'erreur standardisés faciliterait la maintenance et l'expérience utilisateur.

Validation des permissions : Bien que la gestion des rôles soit implémentée, la validation des permissions sur les champs spécifiques pourrait être renforcée. Par exemple, vérifier qu'un utilisateur ne peut modifier que ses propres données ou les données de sa colocation.

C. Axes d'améliorations sur les tokens

L'implémentation actuelle des tokens dans Cooloc présente plusieurs axes d'amélioration pour renforcer la sécurité et la robustesse du système d'authentification.

Gestion des tokens JWT : Le système actuel utilise des tokens JWT avec une durée de vie illimitée, ce qui présente un risque de sécurité. Il serait essentiel d'implémenter : une expiration automatique des tokens (par exemple 24h), un système de refresh tokens pour renouveler l'authentification, et une liste noire des tokens révoqués pour gérer les déconnexions.

Sécurisation des tokens : L'amélioration de la sécurité pourrait passer par : l'utilisation d'algorithmes de signature plus robustes (RS256 au lieu de HS256), la rotation des clés secrètes, l'ajout d'informations supplémentaires dans le payload (IP de connexion, user agent), et la validation de ces informations à chaque requête.

Gestion des sessions : L'implémentation d'un système de gestion des sessions permettrait de : suivre les connexions actives d'un utilisateur, permettre la déconnexion sur tous les appareils, détecter les connexions suspectes, et limiter le nombre de sessions simultanées.

Protection contre les attaques : Le renforcement de la sécurité pourrait inclure : la limitation du taux de tentatives de connexion (rate limiting), la détection de tentatives de brute force, l'ajout de captchas pour les tentatives suspectes, et la notification des connexions depuis de nouveaux appareils.

Audit et monitoring : L'amélioration du système de logs permettrait de : tracer toutes les utilisations de tokens, détecter les anomalies d'utilisation, générer des alertes en cas de comportement suspect, et faciliter l'investigation en cas d'incident de sécurité.

D. Evolutions & futur de Cooloc

L'application Cooloc présente un potentiel d'évolution significatif pour devenir une plateforme complète et moderne de gestion de colocation, répondant aux besoins croissants du marché immobilier partagé.

Fonctionnalités avancées : L'évolution pourrait inclure : un système de messagerie interne avec notifications push, un calendrier partagé pour les événements et absences, un module de gestion des factures et paiements avec intégration bancaire, un système de maintenance et réparations, et un espace de stockage partagé pour les documents.

Intelligence artificielle : L'intégration de l'IA pourrait apporter : des recommandations de tâches basées sur l'historique, la détection automatique de conflits entre colocataires, l'optimisation des dépenses communes, la prédiction des besoins de maintenance, et un chatbot d'assistance utilisateur.

Expérience utilisateur : L'amélioration de l'UX pourrait passer par : une interface plus intuitive avec des animations, l'accessibilité pour les utilisateurs en situation de handicap, la personnalisation des thèmes et couleurs, l'internationalisation multi-langues, et une application mobile native.

Intégrations externes : L'ouverture vers des services tiers pourrait inclure : l'intégration avec des calendriers (Google, Outlook), des services bancaires pour les paiements, des assurances habitation, des services de maintenance, et des plateformes immobilières.

Architecture technique : L'évolution technique pourrait passer par : la migration vers une architecture microservices, l'utilisation de GraphQL pour une API plus flexible, l'implémentation de WebSockets pour les communications temps réel, et l'adoption de technologies cloud natives.

Marché et business model : Le développement commercial pourrait inclure : des abonnements premium avec fonctionnalités avancées, des partenariats avec des agences immobilières, l'expansion vers d'autres marchés (colocations étudiantes, seniors), et l'adaptation pour d'autres types de logements partagés.

E. Projections professionnelle

Mon parcours professionnel s'oriente vers le domaine du Data Engineering, avec une alternance au Crédit Agricole Assurances et un Mastère en Data Engineering & IA, représentant une évolution naturelle de mes compétences développées lors du projet Cooloc.

Transition vers le Data Engineering : L'expérience acquise avec Cooloc, notamment dans la gestion de bases de données PostgreSQL et Firebase, la conception de modèles de données, et l'optimisation des requêtes SQL, constitue une base solide pour ma transition vers le Data Engineering. Les compétences en Python, en gestion de données et en architecture applicative sont directement transférables.

Alternance au Crédit Agricole Assurances : Cette opportunité permettra d'appliquer mes compétences dans un contexte professionnel exigeant, en travaillant sur des projets de data engineering à grande échelle. L'expérience dans

un secteur réglementé comme l'assurance renforcera mes compétences en sécurité des données et en conformité, aspects déjà abordés dans Cooloc.

Mastère Data Engineering & IA : La formation académique complémentaire permettra d'approfondir les aspects techniques avancés : le traitement de données massives (Big Data), les pipelines de données (ETL/ELT), le machine learning, et l'architecture de données distribuées. Ces compétences enrichiront ma palette technique et m'ouvriront de nouvelles perspectives professionnelles.

Évolution des compétences : Le passage du développement d'applications web vers le data engineering représente une évolution logique, où mes compétences en programmation, gestion de données et architecture seront valorisées dans un contexte de traitement et d'analyse de données à grande échelle.

Perspectives à long terme : Cette orientation vers le data engineering et l'IA ouvre des perspectives dans des domaines en forte croissance : l'analyse prédictive, l'automatisation des processus, l'intelligence artificielle appliquée, et la transformation numérique des entreprises. Ces compétences sont particulièrement recherchées dans le secteur financier et assurantiel.

L'expérience acquise avec Cooloc, combinée à la formation en data engineering et à l'expérience professionnelle au Crédit Agricole Assurances, constitue un parcours cohérent vers une expertise technique pointue dans le domaine des données et de l'intelligence artificielle.

VIII. Acronymes

CPAM : Caisse Primaire d'Assurance Maladie

CNAM : Caisse Nationale de l'Assurance Maladie

CSS : Complémentaire Santé Solidaire

DMP : Dossier médical Partagé

ALD : Affections de Longue Durée

MISAS : Mission Accompagnement Santé

AAVA : Action Aller vers les Assurés

CSG : Contribution Sociale Généralisée

CMU-C : Couverture Maladie Universelle Complémentaire

AME : Aide Médical d'Etat

UI-Path : Plateforme d'automatisation des processus robotiques

RGAA : Référentiel Général d'Amélioration de l'Accessibilité

CI/CD : Intégration Continue / Déploiement Continu

MCD : Modèle Conceptuel de Données

MLD : Modèle Logique de Données

MPD : Modèle Physique de Données

JWT : JSON Web Token

CSRF : Cross-Site Request Forgery

RGPD : Règlement Général sur la Protection des Données

XSS : Cross-Site Scripting

SQL : Structured Query Language

CORS : Cross-Origin Resource Sharing

API : Application Programming Interface

HTTP : HyperText Transfer Protocol

HTTPS : HyperText Transfer Protocol Secure

CRUD : Create Read Update Delete

ETL : Extract Load Transform

UI : User Interface

UX : User Experience

SAS : Service d'Accès aux Soins

RNCP : Répertoire National des Certifications Professionnelles