DOSSIERE PROJET

MNS GARAGE

Aliyev Ogtay

Concepteur Développeur D’Application

CONTENU

1. LISTE DES COMPETENCES COUVERTES PAR LE PROJET

1. Installation et Configuration mon environnement de travail en fonction du projet

J'ai installé et configuré mon environnement de travail selon les besoins spécifiques du projet. Cela comprenait l'installation des outils de développement, des Framework nécessaires, et la configuration des paramètres pour garantir une efficacité optimale.

2. Développement des interfaces utilisateur

J'ai conçu et développé des interfaces utilisateur intuitives et ergonomiques en utilisant les technologies frontend appropriées. Mon objectif était de garantir une expérience utilisateur fluide et agréable.

3. Développement des composants métier

J'ai développé des composants métiers robustes, permettant de gérer la logique métier de l'application. Ces composants assurent le bon fonctionnement des fonctionnalités principales tout en respectant les exigences du projet.

4.Contribuer à la gestion d'un projet informatique

J'ai activement contribué à la gestion du projet en participant aux réunions de planification, en suivant les méthodologies agiles, et en assurant la communication entre les membres de l'équipe. J'ai également utilisé des outils de gestion de projet pour suivre l'avancement et les tâches à réaliser.

5. Analyser les besoins et maquetter mon application

J'ai analysé les besoins des utilisateurs et des parties prenantes pour comprendre les exigences du projet. Ensuite, j'ai créé des maquettes et des prototypes pour visualiser et valider les concepts avant le développement.

6. Définir l'architecture logicielle d'une application

J'ai défini l'architecture logicielle de l'application en choisissant les modèles de conception appropriés et en structurant les différentes couches du système. Cette architecture assure la scalabilité, la maintenabilité et la performance de l'application.

7. **Concevoir et mettre en place une base de données relationnelle**

J'ai conçu et mis en place une base de données relationnelle pour stocker et gérer les données de l'application. J'ai utilisé des techniques de modélisation de données pour créer un schéma efficace et optimisé.

8. **Développer des composants d'accès aux données SQL et NoSQL**

J'ai développé des composants permettant d'accéder et de manipuler les données dans des bases de données SQL. Ces composants assurent des opérations de lecture, écriture, mise à jour et suppression de manière sécurisée et efficace.

9. **Préparer et exécuter les plans de tests d'une application**

J'ai préparé et exécuté des plans de tests pour vérifier la qualité et la fiabilité de l'application. Cela incluait des tests unitaires.

10**. Préparer et documenter le déploiement d'une application**

J'ai préparé et documenté le processus de déploiement de l'application. Cette documentation détaillait les étapes nécessaires pour mettre en production l'application, assurant ainsi une transition fluide et sans erreur.

11**. Contribuer à la mise en production dans une démarche DevOps**

J'ai contribué à la mise en production de l'application en adoptant une démarche DevOps. J'ai utilisé des outils d'intégration continue et de déploiement continu pour automatiser et sécuriser le processus de mise en production, garantissant ainsi une livraison rapide et fiable des nouvelles fonctionnalités et corrections.

2. CAHIER DES CHARGES OU L'EXPRESSION DES BESOINS DU PROJET

Ce projet est conçu pour répondre aux besoins des garages automobiles, en particulier ceux souhaitant mettre en place un système collaboratif pour la gestion des réparations et l'optimisation de leurs agendas. L'objectif principal est de développer une application complète qui facilite la gestion quotidienne des tâches et améliore la coordination entre les différents acteurs du garage, qu'il s'agisse des mécaniciens, des administrateurs ou des clients.

La solution proposée doit non seulement répondre aux exigences techniques, mais aussi tenir compte des aspects organisationnels et humains. En tant que prestataires, nous devons proposer des ajustements et des options fonctionnelles en fonction de notre compréhension des besoins spécifiques du garage. Cela inclut une gestion efficace des profils, la prise de rendez-vous, la reconnaissance des véhicules, et la gestion des réservations pour les services solidaires.

Contexte

Le projet consiste à développer un système collaboratif pour les garages automobiles souhaitant optimiser leur gestion des réparations et des rendez-vous. L'objectif est de fournir une solution moderne qui facilite la gestion quotidienne des opérations tout en améliorant la satisfaction des clients.

Objectifs

- Mettre en place une solution numérique pour gérer les profils utilisateurs et les comptes de garages.
- Faciliter la planification et la gestion des rendez-vous de réparation.

- Optimiser l’utilisation des ressources et des créneaux horaires disponibles.
- Offrir un système de gestion du catalogue des véhicules d'occasion.
- Améliorer la communication entre les différents acteurs (mécaniciens, administrateurs, clients).

Besoins Fonctionnels

Gestion des Utilisateurs

- Création, modification et suppression de profils utilisateurs.
- Gestion des comptes utilisateurs pour le garage solidaire.

Organisation et Collaboration

- Création de la structure d’organisation du garage (services, agences, équipes).
- Affectation des collaborateurs aux différentes structures organisationnelles.

Catalogue de Véhicules d'Occasion

- Consultation et gestion du catalogue de véhicules d'occasion.
- Recherche et filtrage des véhicules selon divers critères.

Prise de Rendez-Vous

- Planification et gestion des rendez-vous de réparation.
- Système de confirmation et d’alerte pour les rendez-vous.

Reconnaissance des Véhicules (Optionnel)

- Identification des véhicules par numéro de plaque minéralogique.

Réservations pour le Garage Solidaire

- Réservation de créneaux horaires avec limitations de durée.

Notifications et Alertes

- Système d’alerte pour les nouveaux véhicules d'occasion.
- Envoi de rappels et de suivis pour les rendez-vous.

Le cahier des charges ou l’expression des besoins du projet fournit une vision claire et détaillée des attentes et des exigences pour le développement du système collaboratif de réparation automobile. En suivant ce document, nous nous assurons que toutes les parties prenantes sont alignées et que le projet répondra aux besoins fonctionnels et non fonctionnels identifiés.

3. PRESENTATION DE L'ENTREPRISE ET DU SERVICE

MNS Garage est un garage associatif dédié à fournir des services de réparation et d'entretien de véhicules tout en promouvant des valeurs de solidarité et de collaboration. MNS Garage s'engage à offrir des solutions automobiles accessibles et de haute qualité à notre communauté. En tant qu'association, notre mission va au-delà de la simple réparation de véhicules ; nous cherchons à créer un environnement où les compétences et les ressources sont partagées pour le bénéfice de tous.

MNS Garage est plus qu'un simple garage ; c'est un lieu où la solidarité et l'entraide sont au cœur de notre activité. Nous croyons que, par la collaboration et le partage des connaissances, nous pouvons non seulement améliorer les conditions de vie de nos clients, mais aussi renforcer les liens au sein de notre communauté. Notre engagement envers l'excellence et la responsabilité sociale fait de nous un partenaire de confiance pour tous vos besoins automobiles.

Vision et Mission

Vision : Devenir un modèle de garage associatif où la collaboration et l'innovation se combinent pour offrir des services de réparation automobile exceptionnels tout en renforçant la cohésion sociale.

Mission : Fournir des services de réparation et d'entretien de véhicules de manière collaborative, en optimisant nos processus internes grâce à des technologies avancées, et en rendant nos services accessibles à tous, notamment aux personnes en situation de précarité.

Services Offerts

Entretien et Réparation de Véhicules : MNS Garage propose une gamme complète de services incluant les vidanges, les changements de freins, les diagnostics électroniques, et bien plus encore. Nos techniciens qualifiés utilisent des outils modernes pour garantir des réparations rapides et fiables.

Vente de Véhicules d'Occasion : Nous offrons une sélection de véhicules d'occasion inspectés et certifiés. Chaque véhicule est soigneusement vérifié pour garantir qu'il répond à nos normes de qualité et de sécurité.

Service de Prise de Rendez-vous en Ligne : Pour simplifier la gestion des réparations, nos clients peuvent prendre rendez-vous en ligne via notre plateforme. Ce service permet de choisir facilement le type de réparation et la plage horaire souhaitée.

Gestion des Profils et Comptes Clients : Nous permettons à nos clients de créer et de gérer leurs comptes en ligne. Ils peuvent y suivre l'historique de leurs services, recevoir des rappels pour les entretiens à venir, et accéder à des offres spéciales.

Services Solidaires : En tant que garage associatif, nous offrons des services à tarifs réduits ou gratuits pour les personnes en situation de précarité. Nous croyons fermement que l'accès à des services de qualité ne devrait pas être limité par des considérations financières.

Engagement envers la Qualité

MNS Garage s'engage à offrir des services de haute qualité en respectant les normes de l'industrie automobile. Nous utilisons des pièces de rechange d'origine et des équipements modernes pour garantir des réparations durables et efficaces. Nos techniciens sont formés en continu pour rester à jour avec les dernières technologies et méthodes de réparation.

MNS Garage vise à intégrer des solutions technologiques avancées pour optimiser la gestion de ses services, améliorer l'efficacité opérationnelle et offrir une expérience client exceptionnelle. Notre engagement envers la qualité et l'innovation fait de nous le partenaire idéal pour répondre aux besoins automobiles de notre communauté tout en renforçant la solidarité et la coopération.

4. GESTION DE PROJET

Réunions Hebdomadaires

Chaque début et fin de semaine, nous avons organisé des petites réunions pour faire le point sur l'avancement du projet. Ces réunions étaient essentielles pour maintenir une communication fluide entre les membres de l'équipe et s'assurer que tout le monde était sur la même longueur d'onde. Nous commencions chaque réunion par un récapitulatif des tâches accomplies depuis la dernière rencontre, suivi d'une discussion sur les problèmes rencontrés et les solutions possibles.

Ces sessions de suivi étaient également l'occasion de réévaluer nos priorités et de réajuster notre planning en fonction des nouvelles informations et des imprévus. Elles permettaient d'identifier les éventuels risques et de trouver des stratégies pour les atténuer. Par exemple, si nous étions en retard sur une tâche critique, nous discutions des moyens de redistribuer le travail ou de modifier le scope du projet pour respecter les délais.

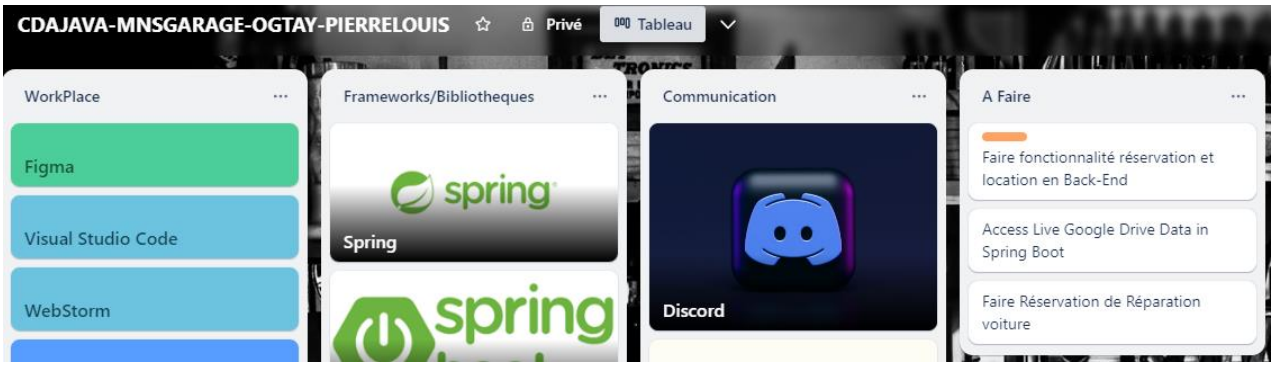
Méthode Agile Scrum

Nous avons adopté la méthode Agile Scrum pour structurer notre travail et optimiser notre efficacité. Chaque semaine, nous organisions des sprints, durant lesquels nous définissions des objectifs clairs et atteignables. Lors de la planification de sprint, nous utilisions le backlog pour prioriser les tâches en fonction de leur importance et de leur urgence.

Chaque jour, nous réalisions des réunions d'avancement de type "stand-up meeting" où chacun exposait ce qu'il avait fait la veille, ce qu'il comptait faire aujourd'hui, et les obstacles rencontrés. Cela nous a permis de rester concentrés et de réagir rapidement en cas de problème.

À la fin de chaque sprint, nous tenions une réunion de rétrospective pour évaluer ce qui avait bien fonctionné et ce qui pouvait être amélioré. Nous avons utilisé ces retours pour adapter notre manière de travailler et pour continuellement améliorer notre processus.

Utilisation de Trello

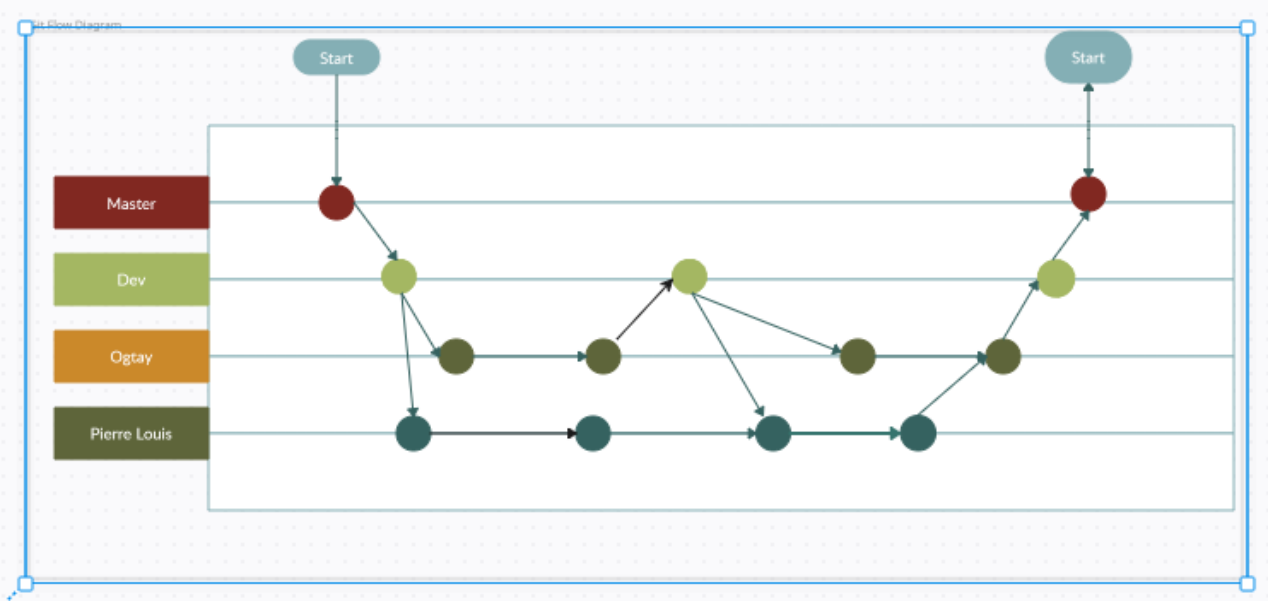


Pour une gestion efficace des tâches, nous avons utilisé Trello. Chaque carte sur Trello représentait une tâche ou un objectif spécifique. Nous avons structuré notre tableau Trello en plusieurs colonnes : "À faire", "En cours", "En révision" et "Terminé".

Cela nous a permis d'avoir une vue d'ensemble claire de l'état d'avancement du projet et d'identifier rapidement les tâches en retard. Chaque membre de l'équipe pouvait ajouter des commentaires, des pièces jointes, et des checklists sur les cartes, ce qui facilitait la collaboration et la communication.

De plus, nous avons régulièrement mis à jour le tableau pour refléter l'état réel du projet. Cette transparence a aidé à maintenir un haut niveau de motivation et d'engagement au sein de l'équipe.

Gestion du Code Source avec GitHub



Pour le développement de notre projet, nous avons utilisé GitHub comme système de gestion de versions. Le code source, tant pour le front-end que pour le back-end, était stocké sur GitHub. Chaque modification du code était clairement documentée dans les commentaires des commits, ce qui facilitait la collaboration et la compréhension des changements effectués.

Nous avons mis en place des branches comme master et dev pour différentes fonctionnalités afin de permettre le développement parallèle. Avant de fusionner une branche dans la branche principale, nous réalisions des revues de code pour nous assurer de la qualité et de la fonctionnalité du code. Cette pratique a non seulement amélioré la qualité du code, mais a aussi encouragé l'apprentissage

Discussions Techniques

Lors des réunions, nous avons également discuté des aspects techniques du projet. Nous avons abordé des sujets tels que les fonctionnalités à implémenter, les designs à adopter, les outils et langages de programmation à utiliser, ainsi que les environnements de développement (IDE). Nous avons pris des décisions collectives pour nous assurer que tout le monde était aligné et savait quoi faire.

Nous avons également discuté des stratégies de test et de déploiement, assurant que notre projet non seulement fonctionnait correctement en développement, mais était aussi prêt pour la production. Ces discussions nous ont permis de garantir que notre projet était techniquement solide et répondait aux exigences initiales.

Collaboration et Communication

Nous avons travaillé à deux sur ce projet, en maintenant une communication constante pour nous assurer de rester synchronisés. Chaque membre de l'équipe avait des responsabilités spécifiques, mais nous avons également veillé à collaborer étroitement pour surmonter les défis et respecter les délais.

Nous avons utilisé divers outils de communication tels que Discord pour des échanges rapides et Google Meet pour des réunions plus formelles. Cette utilisation combinée des outils nous a permis de rester connectés et efficaces, même lorsque nous travaillions à distance.

5. SPECIFICATIONS FONCTIONNELLES

Les spécifications fonctionnelles décrivent les fonctionnalités et les comportements attendus du système que nous développons. Elles servent de base pour le développement et la validation de l'application. Voici les spécifications fonctionnelles pour le projet de système collaboratif de réparation automobile et d'optimisation de l'agenda de réparation :

1. Gestion des Profils Utilisateurs

- Création de Profils : Les utilisateurs peuvent créer des profils en fournissant des informations telles que leur nom, adresse e-mail, numéro de téléphone, et rôle (mécanicien, client, administrateur).
- Modification de Profils : Les utilisateurs peuvent mettre à jour leurs informations personnelles et leurs préférences.

- Suppression de Profils : Les administrateurs peuvent supprimer les profils d'utilisateurs.

2. Gestion des Comptes Utilisateurs pour le Garage Solidaire

- Inscription : Les nouveaux utilisateurs peuvent s'inscrire en fournissant leurs informations personnelles.
- Connexion/Déconnexion : Les utilisateurs peuvent se connecter et se déconnecter de leur compte.
- Récupération de Mot de Passe : Les utilisateurs peuvent récupérer leur mot de passe via un e-mail de réinitialisation.

3. Organisation des Collaborateurs

- Création de la Structure d'Organisation : Les administrateurs peuvent définir la structure du garage, en créant des services, agences, ou équipes.
- Affectation des Collaborateurs : Les administrateurs peuvent assigner des collaborateurs à différents services, agences ou équipes.
- Modification de la Structure : Les administrateurs peuvent modifier la structure organisationnelle en ajoutant ou en supprimant des services, agences ou équipes.

4. Catalogue des Véhicules d'Occasion

- Affichage du Catalogue : Les utilisateurs peuvent consulter le catalogue des véhicules d'occasion disponibles.
- Recherche et Filtrage : Les utilisateurs peuvent rechercher des véhicules en fonction de divers critères (marque, modèle, année, prix).
- Ajout de Véhicules : Les administrateurs peuvent ajouter de nouveaux véhicules au catalogue.
- Modification de Véhicules : Les administrateurs peuvent mettre à jour les informations sur les véhicules.
- Suppression de Véhicules : Les administrateurs peuvent supprimer des véhicules du catalogue.

5. Prise de Rendez-Vous pour les Réparations

- Planification des Rendez-Vous : Les utilisateurs peuvent planifier des rendez-vous pour des réparations en sélectionnant une date et une plage horaire disponibles.
- Automatisation des Plages Horaires : Le système attribue automatiquement des plages horaires en fonction de la durée estimée des réparations (par exemple, une vidange sur une Mégane IV diesel).
- Confirmation des Rendez-Vous : Les utilisateurs reçoivent une confirmation par e-mail ou SMS une fois le rendez-vous planifié.
- Modification des Rendez-Vous : Les utilisateurs peuvent modifier la date et l'heure de leurs rendez-vous.
- Annulation des Rendez-Vous : Les utilisateurs peuvent annuler leurs rendez-vous.

6. Reconnaissance des Véhicules par Numéro de Plaque Minéralogique (Optionnel)

- Identification des Véhicules : Le système peut identifier les véhicules en scannant leur numéro de plaque minéralogique.
- Affichage des Informations : Les informations sur le véhicule sont automatiquement affichées une fois le numéro de plaque reconnu.

7. Système de Confirmation et d'Alerte des Rendez-Vous

- Envoi de Rappels : Le système envoie des rappels automatiques aux utilisateurs avant leurs rendez-vous (par e-mail ou SMS).
- Alertes de Suivi : Les utilisateurs reçoivent des alertes de suivi après les rendez-vous pour recueillir des avis ou fournir des informations supplémentaires.

8. Réservations pour le Garage Solidaire

- Réservation de Créneaux Horaires : Les utilisateurs peuvent réserver des créneaux horaires pour utiliser les services du garage solidaire.
- Limitation des Réservations : Les réservations sont limitées à des tranches d'une heure, avec un maximum de trois heures consécutives.
- Confirmation des Réservations : Les utilisateurs reçoivent une confirmation de leur réservation par e-mail ou SMS.

9. Système d'Alerte pour les Nouveaux Véhicules d'Occasion

- Notifications des Nouveautés : Les utilisateurs peuvent s'abonner pour recevoir des notifications lorsque de nouveaux véhicules d'occasion sont ajoutés au catalogue.
- Personnalisation des Alertes : Les utilisateurs peuvent personnaliser leurs préférences de notification en fonction de critères spécifiques (marque, modèle, prix, etc.).

5.1. CONTRAINTES DU PROJET ET LIVRABLES ATTENDUS

L'objectif est de mettre en place un système collaboratif de réparation et d'optimisation de l'agenda de réparation. Ce projet ne vise pas à fournir une application clé en main à l'entreprise, mais plutôt à démontrer les avantages de l'utilisation de technologies modernes. Les fonctionnalités minimums attendu pour la livraison de ce projet sont :

Back End

- Gestion des profils
- Création et gestion des comptes utilisateurs du garage solidaire
- Création de la structure organisationnelle (répartition des collaborateurs par services, agences ou équipes)
- Catalogue des véhicules
- Proposer un catalogue de l'ensemble des véhicules d'occasion
- Gestion des rendez-vous
- Prise de rendez-vous pour les réparations de véhicules avec une répartition automatique sur des plages horaires en fonction de la durée de la réparation demandée (par exemple, vidange sur une Mégane IV diesel)
- Reconnaissance des véhicules par numéro de plaque minéralogique (optionnel)
- Système de confirmation des rendez-vous

- Alertes sur les prochains rendez-vous
- Réservations pour le garage solidaire avec des créneaux d'une heure, trois heures maximums.
- Système d'alerte pour les nouveaux véhicules d'occasion

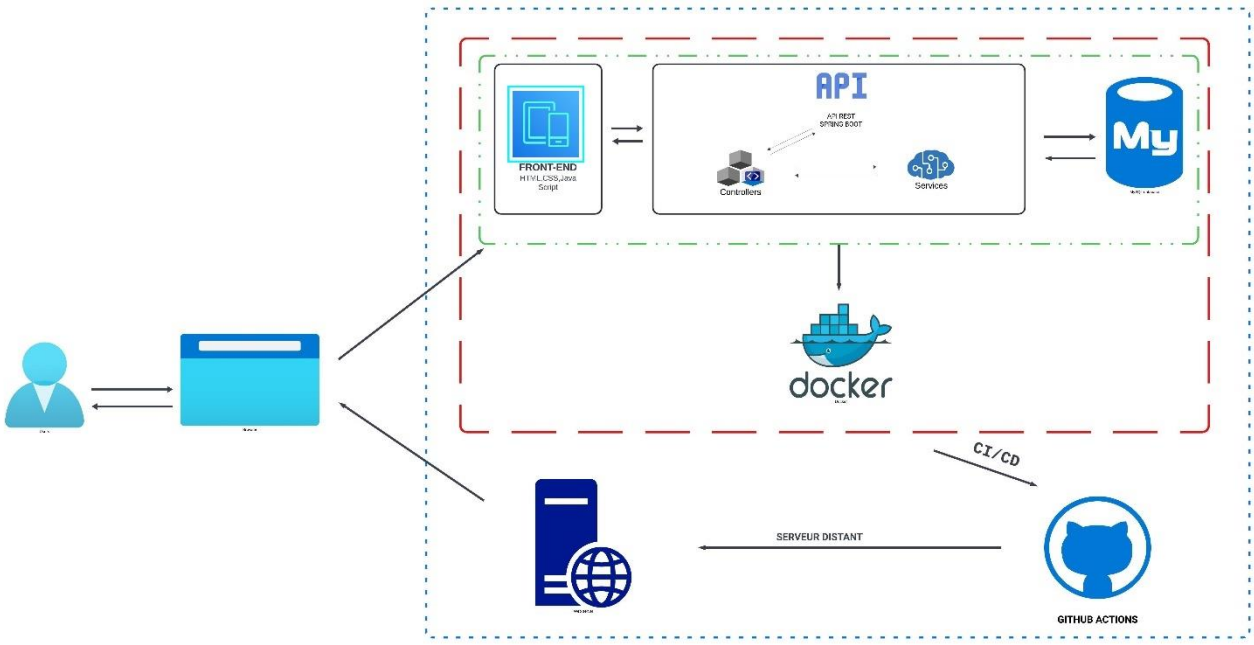
Front End

- Inscription et connexion
- Profil utilisateur
- Modification des information et ajout voiture de l'utilisateur
- Affichage historique des location et réservation pour chaque utilisateur
- Affichages voiture de l'utilisateur
- Location box de garage
- Réservation de l'entretien véhicule
- Affichage liste des voitures occasion proposes par garage

5.2. ARCHITECTURE LOGICIELLE DU PROJET

Dans ce chapitre, je vais présenter l'architecture logicielle du projet, qui repose sur une architecture microservices. Cette approche permet de diviser l'application en fonctionnalités encapsulées au sein de services autonomes, facilitant ainsi la gestion et l'évolution indépendantes de chaque service.

Vue d'Ensemble de l'Architecture



J'ai choisi une architecture microservices pour ce projet afin de garantir une meilleure modularité et flexibilité. Chaque microservice est responsable d'une fonctionnalité spécifique et peut être mis à jour, étendu et déployé indépendamment des autres services. Cela permet une mise à jour rapide et réduit le risque de compromettre l'ensemble de l'application.

Composants Principaux

Front-End

- Technologies Utilisées : J'ai opté pour React en raison de sa modularité et de sa performance, ainsi que Redux pour la gestion de l'état global de l'application.
- Structure du Code : Le code est organisé en composants réutilisables, divisés en :
- Composants de Présentation : Responsables de l'affichage des données.
- Composants Conteneurs : Gèrent la logique d'affaires et la communication avec le back-end.
- Communication avec le Back-End : Le front-end communique avec le back-end via des API REST, en utilisant Axios pour les appels HTTP.

Back-End

- Technologies Utilisées : J'ai choisi Java avec le Spring Boot pour sa simplicité, sécurité et son extensibilité.
- Architecture : L'architecture est basée sur des microservices, ce qui permet une meilleure scalabilité et une isolation des différentes parties du système.
- Gestion des Profils : Les profils utilisateurs sont gérés avec une base de données SQL, utilisant ORM pour faciliter les interactions avec la base de données.
- Gestion des Rendez-Vous : Un service dédié gère la planification et l'envoi des alertes pour les rendez-vous.
- Catalogue des Véhicules : Un autre service gère l'inventaire des véhicules, permettant des mises à jour dynamiques et des recherches efficaces.

Base de Données

- Choix de la Base de Données : J'ai utilisé une base de données MySQL pour sa fiabilité et ses capacités de gestion des transactions.
- Modèle de Données : Le modèle de données est conçu pour supporter la gestion des profils utilisateurs, des rendez-vous, et du catalogue de véhicules.
- Gestion des Transactions : Les transactions sont gérées pour assurer l'intégrité des données et éviter les incohérences.
- API et Services
- API REST : Les microservices communiquent entre eux sans état, à l’aide d’interfaces de programmation d’application (API) indépendantes de tout langage.

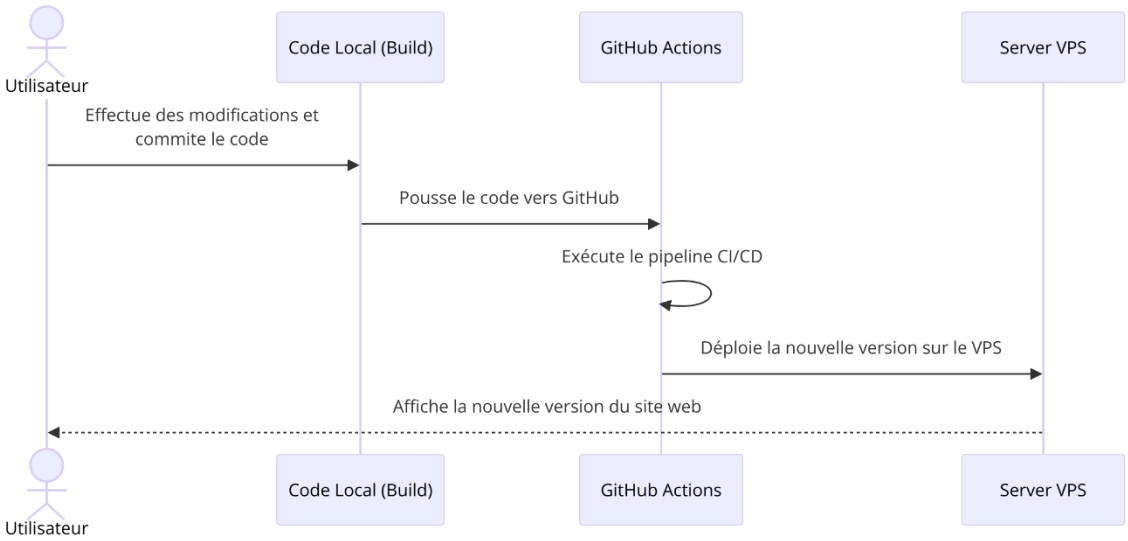
- Services Internes : Les services internes sont utilisés pour la communication entre les différentes parties du système, garantissant ainsi une interaction fluide et efficace.

Sécurité

- Authentification et Autorisation : J'ai mis en place des mécanismes robustes pour l'authentification et l'autorisation des utilisateurs.

- Protection des Données : Des mesures de sécurité sont en place pour protéger les données des utilisateurs et du système.

Déploiement



- Environnement de Développement et de Production : Le système est déployé en environnement de développement et en production, en utilisant des pratiques DevOps.

- CI/CD : J'ai mis en place des pipelines d'intégration et de déploiement continu pour assurer une livraison rapide et fiable des mises à jour.

En adoptant une architecture microservices, j'ai pu diviser l'application en services autonomes, chacun étant capable d'évoluer et d'être déployé indépendamment. Cela permet non seulement d'accélérer le développement et la mise à jour des fonctionnalités, mais aussi de réduire les risques de compromettre l'ensemble du système. Cette architecture, combinée à des pratiques DevOps, constitue la base des applications cloud-native, optimisant ainsi la gestion et l'automatisation des applications sur divers environnements cloud.

5.3. MAQUETTES ET ENCHAINEMENT DES MAQUETTES

J'ai utilisé l'application Figma pour créer les maquettes de l'application web. Tout d'abord, j'ai défini les couleurs, la taille des fenêtres pour Desktop Large :1440 x 1024 px, et le design général que je voulais utiliser. J'ai également importé les images qui seront utilisées dans le frontend.

J'ai commencé par créer des wireframes basse fidélité pour avoir une idée générale de la disposition des éléments sur chaque page. Ensuite, j'ai travaillé sur les prototypes, en particulier pour les boutons, afin de définir leur taille, couleur et design. Après cela, j'ai créé des maquettes en couleur pour donner une idée plus précise de l'apparence finale de l'application.

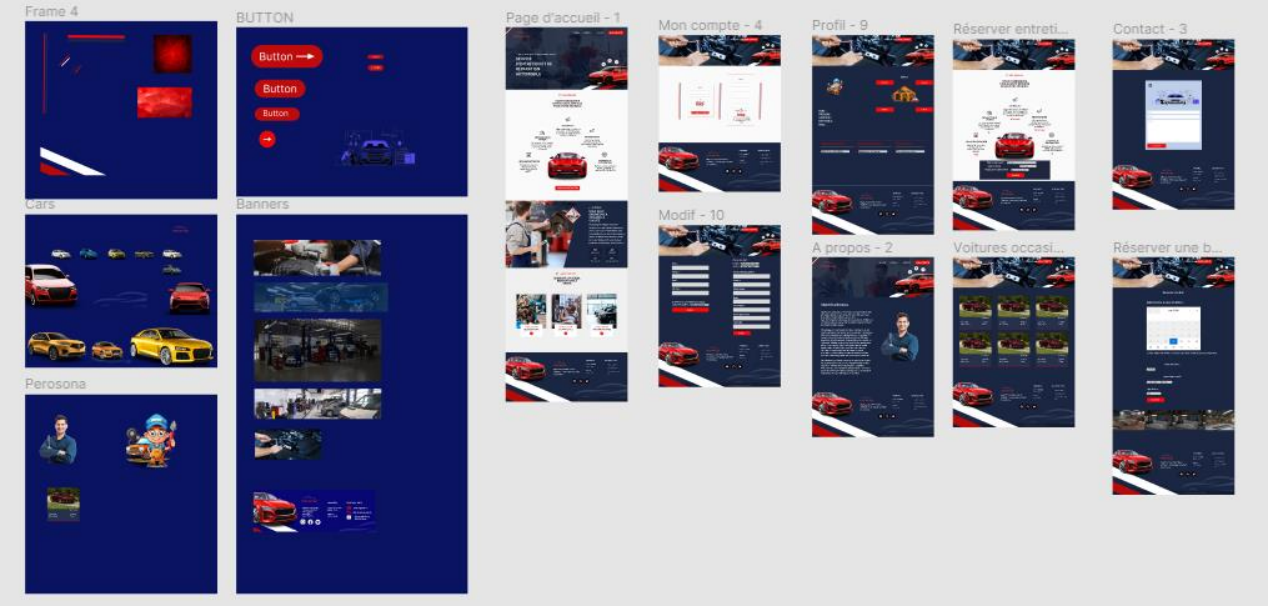
Voici les principales maquettes que j'ai réalisées :

Page d'accueil :

Description : La page d'accueil est la première page que les utilisateurs voient. Elle présente les services offerts par MNS Garage et permet d'accéder aux différentes sections du site.

Fonctionnalités principales : Barre de navigation, présentation des services, formulaire de contact, authentification utilisateur, voitures occasions à vendre, section <À propos >, section Footer avec les informations et cordonnées de l'entreprise.

Design : La page d'accueil utilise une palette de couleurs harmonieuse pour attirer l'attention des utilisateurs tout en assurant une bonne lisibilité. Les images des voitures et des services sont bien intégrées pour illustrer les offres de l'entreprise.



Après avoir travaillé sur les maquettes et les wireframes, je vais passer à la création de prototypes interactifs. L'objectif principal de cette étape est de donner vie à l'interface utilisateur et de simuler le comportement de l'application. En créant des prototypes, je pourrai mieux démontrer les flux de navigation, les interactions utilisateur et le fonctionnement général de l'application.

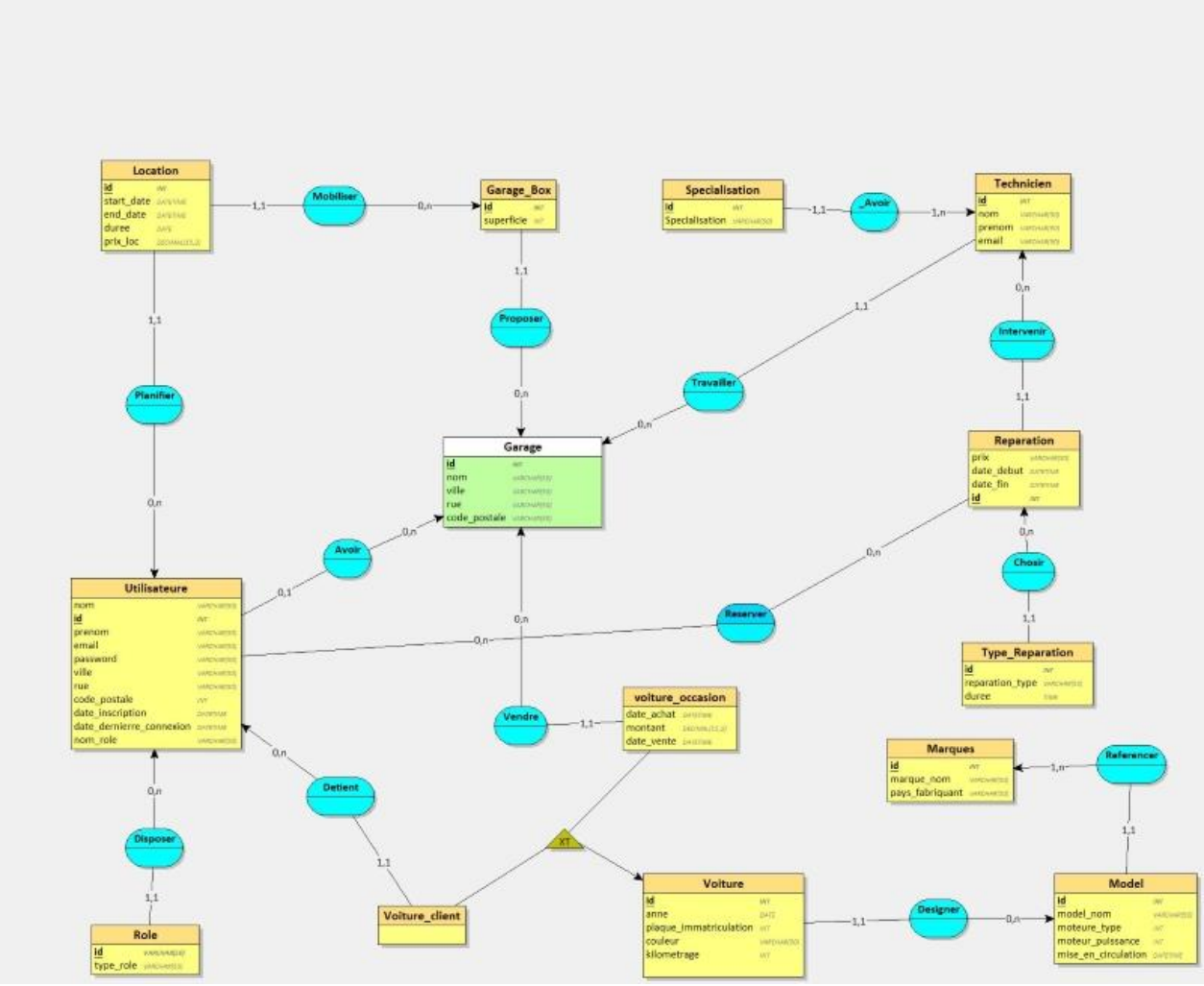
Les prototypes permettent aux parties prenantes, y compris les membres de l'équipe de développement et les utilisateurs finaux, de visualiser concrètement l'expérience utilisateur attendue.

Cela permet également de recueillir des commentaires précieux et d'identifier les éventuels problèmes ou améliorations dès les premières étapes du processus de développement

En résumé, la création des maquettes pour "MNS Garage" a suivi un processus méthodique, en commençant par des wireframes basse fidélité et en progressant vers des prototypes interactifs et des maquettes en couleur. L'enchaînement des maquettes a été soigneusement planifié pour offrir une expérience utilisateur cohérente et agréable.

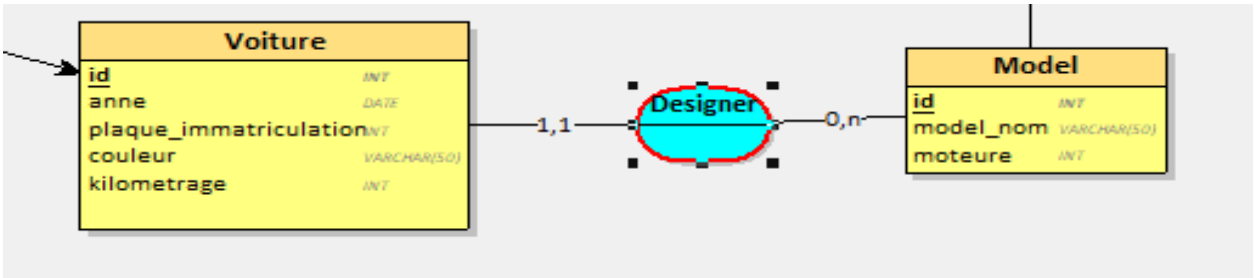
5.4. MODELE ENTITES-ASSOCIATIONS ET MODELE PHYSIQUE DE LA BASE DE DONNEES

J’ai utilisé l'application JMerise afin de créer les modèles conceptuels et physiques de la base de données (MCD et MLD).



Le Modèle Conceptuel de Données (MCD) représente les entités principales et leurs relations dans mon application. Les objectifs de cette modélisation sont de définir clairement les différentes entités (telles que Utilisateurs, Voitures, Réservations Locations, et Garage) et leurs attri-

buts, ainsi que les associations entre ces entités. Les cardinalités dans cette modélisation décrivent les relations entre les entités, indiquant combien d'instances d'une entité peuvent être liées à une instance d'une autre entité. Par exemple, dans une relation entre Utilisateurs et Réservations, un Utilisateur peut avoir plusieurs Réservations (cardinalité "1 à N").



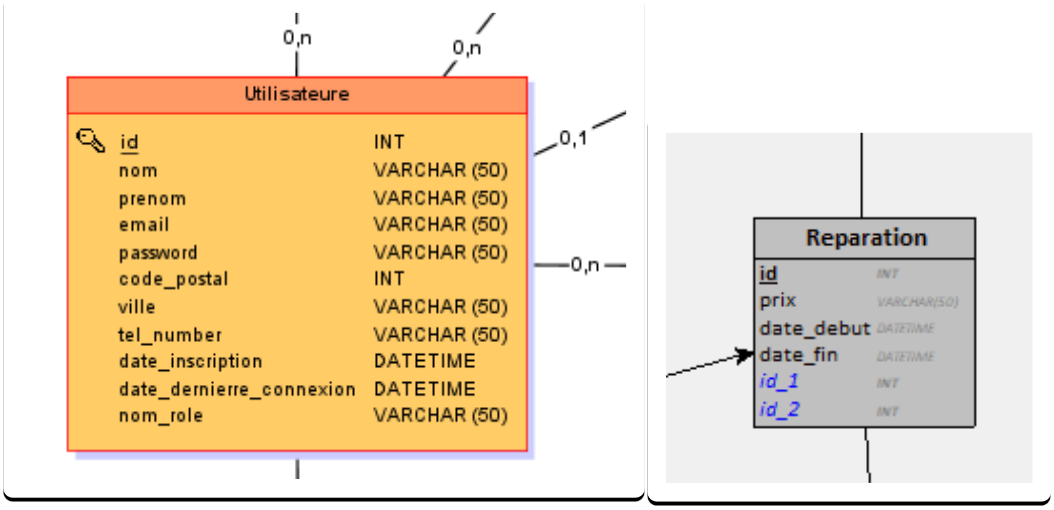
Définition des Clés :

Clé Primaire (Primary Key) :

Je définis une clé primaire comme un attribut ou un ensemble d'attributs qui permet d'identifier de manière unique chaque instance d'une table. Par exemple, dans la table "Utilisateurs", je peux utiliser l'ID utilisateur comme clé primaire, car il permet de distinguer chaque utilisateur de manière unique.

Clé Étrangère (Foreign Key) :

Je définis une clé étrangère comme un attribut dans une table qui établit une relation avec une clé primaire dans une autre table. Par exemple, dans la table "Réparation", j'utilise l'ID utilisateur comme clé étrangère. Cette clé fait référence à l'ID utilisateur dans la table "Utilisateurs", créant ainsi un lien logique entre les réservations et les utilisateurs.

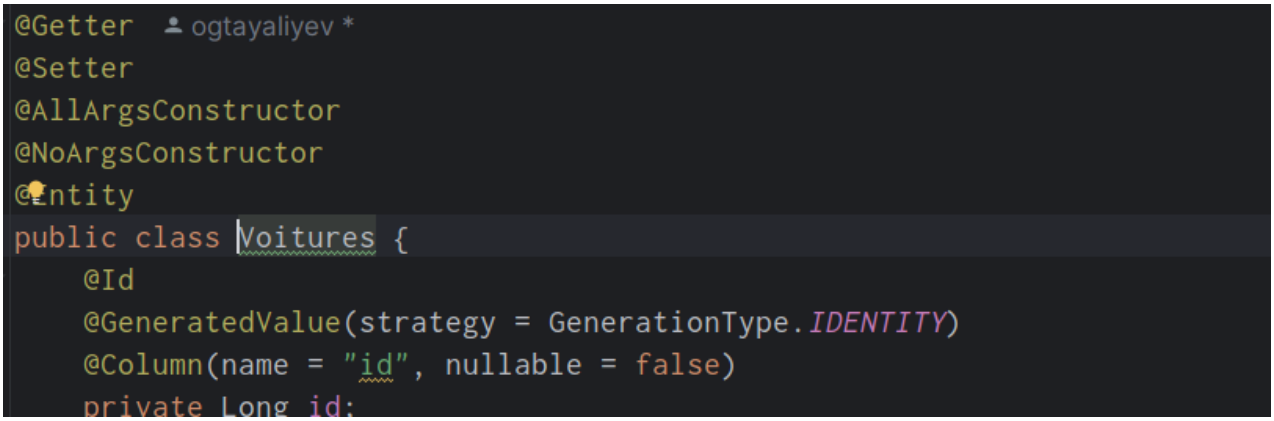


Après avoir validé le MCD, j'ai transformé ce modèle en Modèle Logique de Données (MLD), qui détaille la structure de la base de données relationnelle. Les cardinalités dans le MLD indiquent comment les tables sont liées les unes aux autres, ce qui est essentiel pour concevoir des schémas de base de données efficaces. Par exemple, une cardinalité "1 à N" entre deux tables signifie qu'une instance dans une table peut être associée à plusieurs instances dans l'autre table.

5.5. SCRIPT DE CREATION OU DE MODIFICATION DE LA BASE DE DONNÉES

Création des Tables

Pour la création des tables et les liens entre elles, j'utilise le modèle ORM de Spring Boot avec les entités et les repositories. Dans mes entités, j'ai bien défini les colonnes et les relations nécessaires.



Voitures : Cette table contient les informations sur les voitures, telles que la kilométrage, plaque immatriculation, couleur, année fabrication et lie avec les tables marque et le modèle. Elle inclut également une clé étrangère pointant vers la table `Utilisateur` pour établir une relation many-to-one

Pour tester les relations et les modifications des données des tables, j'ai créé un fichier data.sql où j'ai inclus des scripts SQL pour ajouter, modifier, et supprimer certaines données. Ce fichier me permet de valider les intégrités référentielles et les contraintes de mes tables.

Pour sélectionner des voitures en fonction de leur modèle et de leur marque en utilisant SQL, j'ai utilisé une requête SELECTE avec une clause JOIN pour joindre les tables Voitures, Voiture model et Voiture Marque. Voici un exemple de requête SQL :

```sql
SELECT
    V.id,
    V.anne_fabrication,
    V.couleur,
    V.boite,
    V.carroserie,
    V.carburant,
    V.kilometrage,
    V.plaqueImmatriculation,
    V.etatVente,
    VM.model_nom AS model_nom,
    U.nom AS utilisateur_nom
FROM
    Voitures V
    JOIN voiture_model VM ON V.model_id = VM.id
    JOIN utilisateur U ON V.utilisateur_id = U.id;
```

Pour modifier :

```sql
UPDATE Voitures
SET couleur = 'Bleu', kilometrage = 16000
WHERE id = 1;
```

Définition des Clés Étrangères

Dans mon modèle, chaque voiture est associée à un utilisateur. Cela se traduit par une relation many-to-one de `Voitures` à `Utilisateur` et une relation one-to-many de `Utilisateur` à `Voitures`.

Relations Many-to-One et One-to-Many

- Relation One-to-Many (Utilisateur vers Voitures) : En tant qu'utilisateur, je peux posséder plusieurs voitures. Cela se traduit par une liste de voitures associées à mon profil.

Par exemple : J'ai l'ID 1 et je peux posséder plusieurs voitures enregistrées dans la table `Voitures`.

- Relation Many-to-One (Voitures vers Utilisateur) : Chaque voiture est associée à un seul utilisateur. Cela se traduit par une référence à l'utilisateur propriétaire de la voiture.

Par exemple : Une voiture avec l'ID 101 peut m'appartenir, en tant qu'utilisateur avec l'ID 1.

Types de Fetch et Cascade

```java
@OneToOne(cascade = CascadeType.ALL)
private Role role;

public String getEmail() { return this.email; }

@OneToMany(mappedBy="utilisateur",fetch = FetchType.EAGER)
private List<Voitures> voitures;
```

Fetch Type

- `FetchType.EAGER` : Lorsque j'accède à mes données utilisateur, toutes les voitures associées à mon profil sont également chargées immédiatement. Ce type de Fetch est utilisé lorsque j'ai besoin d'accéder à toutes les entités associées en même temps pour éviter les problèmes de latence ou les requêtes supplémentaires.

Cascade Type

- `CascadeType.ALL` : Cela signifie que toutes les opérations sur mon profil utilisateur sont propagées à mes voitures associées. Par exemple, si je suis supprimé, toutes mes voitures seront également supprimées. Les types de cascade courants incluent `PERSIST`, `MERGE`, `REMOVE`, `REFRESH`, et `DETACH`.

Join Column

La clé étrangère dans la table `Voitures` est définie par l'annotation `@JoinColumn`. Cela permet de spécifier le nom de la colonne qui sera utilisée pour la jointure dans la base de données.

```java
@ManyToOne
@JoinColumn(name = "utilisateur_id")
private Utilisateur utilisateur;
```

Cette annotation spécifie que la colonne `utilisateur_id` dans la table `Voitures` est une clé étrangère pointant vers mon ID dans la table `Utilisateur`.

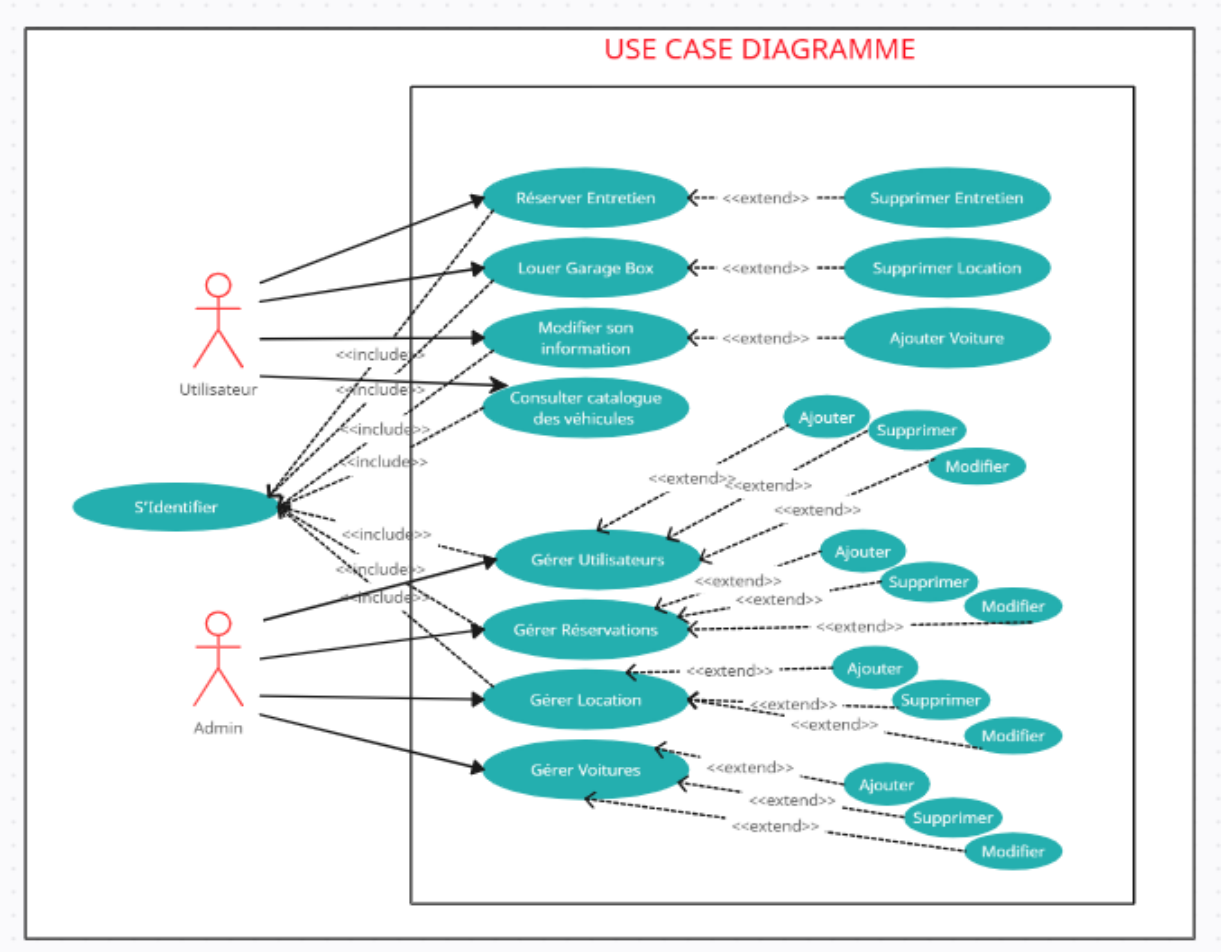
Dans mon application, l'utilisation de `@Temporal (TemporalType.TIMESTAMP) ` est essentielle pour la gestion précise des données temporelles. Je l'applique spécifiquement au champ `startDate` de type `LocalDateTime`, ce qui permet à mon application de mapper correctement ce champ dans la base de données avec une précision allant jusqu'à la fraction de seconde.

```java
    @Temporal(TemporalType.TIMESTAMP)
    private LocalDateTime startDate;
    @Temporal(TemporalType.TIMESTAMP)
    private LocalDateTime returnDate;
```

L'exactitude des informations de date et d'heure est cruciale dans mon application, notamment pour enregistrer les moments précis où une entité est créée ou modifiée. En utilisant `@Temporal(TemporalType.TIMESTAMP)`, chaque fois qu'une entité est persistée dans la base de données, le champ `startDate` est automatiquement rempli avec le timestamp actuel. Cette fonctionnalité garantit un suivi précis du moment où chaque enregistrement est créé ou modifié, ce qui est indispensable pour de nombreuses fonctionnalités critiques de mon application.

5.6. DIAGRAMME DU COMPORTEMENT DES FONCTIONNALITES DE TYPE CAS D'UTILISATIONS

Dans ce chapitre, je vais décrire les interactions entre les acteurs (Utilisateur et Administrateur) et les cas d'utilisation. Je vais expliquer également les relations `extends` et `include` pour représenter les fonctionnalités spécifiques.



1. Identification des Acteurs

Les acteurs principaux de notre système sont :

- Utilisateur : Ce sont les clients du garage qui utilisent les services en ligne.

- Administrateur : Ce sont les employés du garage qui gèrent les opérations et les utilisateurs.

2. Identification des Cas d'Utilisation

Les cas d'utilisation représentent les différentes fonctionnalités du système. Pour chaque acteur, voici les principaux cas d'utilisation identifiés :

Pour l'Utilisateur :

- S'identifier : Permet à l'utilisateur de se connecter au système.

- Réserver un entretien : Permet à l'utilisateur de réserver un créneau pour un entretien. Cette fonctionnalité peut être étendue par la possibilité de Supprimer un entretien.

- Louer un box : Permet à l'utilisateur de réserver un box pour une durée déterminée. Cette fonctionnalité peut être étendue par la possibilité de Supprimer une location de box.

- Consulter le catalogue de voitures d'occasion : Permet à l'utilisateur de voir les voitures d'occasion disponibles.

- Modifier ses données personnelles : Permet à l'utilisateur de mettre à jour ses informations personnelles. Cette fonctionnalité peut être étendue par la possibilité d'Ajouter sa propre voiture.

Pour l'Administrateur :

- S'identifier : Permet à l'administrateur de se connecter au système.

- Gérer les locations : Permet à l'administrateur de gérer les locations de box. Cette fonctionnalité inclut Ajouter, Supprimer et Modifier des locations.

- Gérer les réservations : Permet à l'administrateur de gérer les réservations d'entretien. Cette fonctionnalité inclut Ajouter, Supprimer et Modifier des réservations.

- Gérer les utilisateurs : Permet à l'administrateur de gérer les comptes utilisateurs. Cette fonctionnalité inclut Ajouter, Supprimer et Modifier des utilisateurs.

- Gérer les voitures : Permet à l'administrateur de gérer les voitures d'occasion. Cette fonctionnalité inclut Ajouter, Supprimer et Modifier des voitures.

3. Description des Cas d'Utilisation

Pour chaque cas d'utilisation, nous décrivons les interactions principales et les étapes :

Cas d'Utilisation : Réserver un Entretien

- Acteur Principal : Utilisateur

- Description : L'utilisateur réserve un créneau pour un entretien.

- Étapes :

1. L'utilisateur s'identifie.

2. L'utilisateur accède à la section "Réservation d'entretien".

3. L'utilisateur sélectionne la date et l'heure souhaitées.

4. Le système vérifie la disponibilité et confirme la réservation.

5. L'utilisateur peut supprimer une réservation si nécessaire (extends).

- Préconditions : L'utilisateur doit être authentifié.

- Postconditions : Une réservation d'entretien est créée et stockée dans le système.

Cas d'Utilisation : Louer un Box

- Acteur Principal : Utilisateur

- Description : L'utilisateur réserve un box pour une durée déterminée.

- Étapes :

1. L'utilisateur s'identifie.

2. L'utilisateur accède à la section "Location de box".

3. L'utilisateur sélectionne la durée de location.

4. Le système vérifie la disponibilité et confirme la location.

5. L'utilisateur peut supprimer une location si nécessaire (extends).

- Préconditions : L'utilisateur doit être authentifié.

- Postconditions : Une location de box est créée et stockée dans le système.

Cas d'Utilisation : Gérer les Locations

- Acteur Principal : Administrateur

- Description : L'administrateur gère les locations de box.

- Étapes :

1. L'administrateur s'identifie.

2. L'administrateur accède à la section "Gestion des locations".

3. L'administrateur peut ajouter, supprimer ou modifier une location.

- Préconditions : L'administrateur doit être authentifié.

- Postconditions : Les modifications apportées aux locations sont enregistrées dans le système.

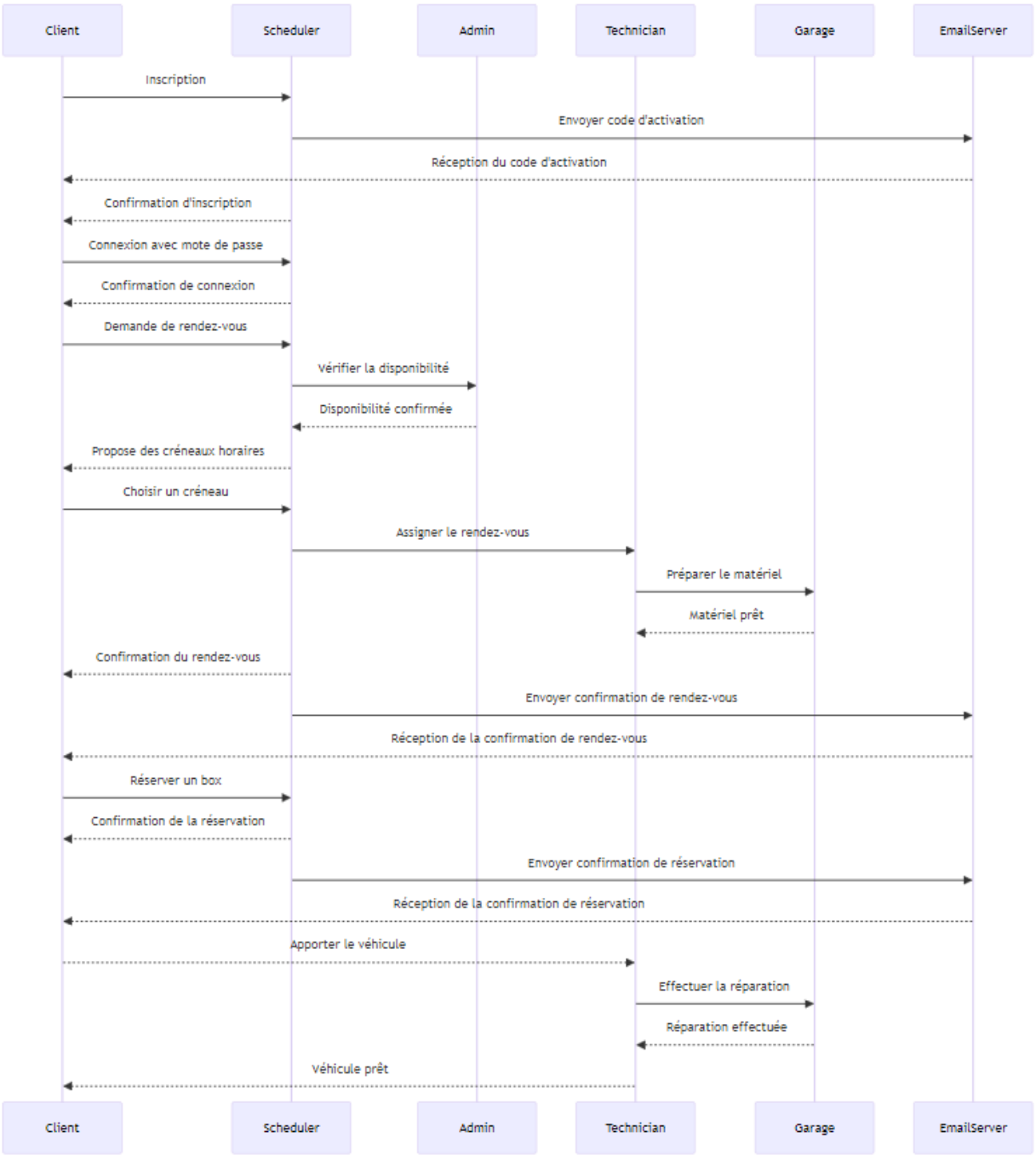
4. Relations `Include` et `Extends`

Les relations `Include` et `Extends` sont utilisées pour représenter les dépendances et les extensions des cas d'utilisation.

- Relation Include : Indique qu'un cas d'utilisation inclut systématiquement le comportement d'un autre cas d'utilisation. Par exemple, S'identifier est inclus dans tous les cas d'utilisation car l'authentification est nécessaire pour accéder aux fonctionnalités.

- Relation Extends : Indique qu'un cas d'utilisation peut être étendu par un autre cas d'utilisation facultatif. Par exemple, Réserver un entretien peut être étendu par Supprimer un entretien car cette action est une extension de la réservation initiale.

5.7. DIAGRAMME DU DETAIL DES CAS D'UTILISATIONS LES PLUS SIGNIFICATIFS DE TYPE DIAGRAMME DE SEQUENCE.



Description Détaillée du Diagramme de Séquence

Le diagramme de séquence pour l'application de gestion du temps de travail des techniciens et des activités du garage automobile illustre les interactions entre les principaux acteurs (Client, Schedule, Admin, Technicien, Garage, et Email Server) et les étapes clés du processus de gestion des rendez-vous, des inscriptions, et des réservations.

Acteurs et Interactions

1. Inscription et Connexion :

- Client → Schedule : Le client commence par s'inscrire sur la plateforme en fournissant ses informations personnelles. Cette étape est cruciale pour l'identification et la gestion des utilisateurs.

- Schedule → Email Server : Une fois les informations du client enregistrées, le système envoie un code d'activation à l'adresse e-mail fournie par le client.

- Email Server → Client : Le client reçoit un e-mail contenant le code d'activation, nécessaire pour valider son compte.

- Client → Schedule : Le client utilise le code d'activation pour se connecter à la plateforme. Cette étape assure que les comptes sont vérifiés et sécurisés.

- Schedule → Client : Le système confirme la connexion du client, lui donnant accès aux fonctionnalités de l'application.

2. Demande de Rendez-vous :

- Client → Schedule : Le client peut ensuite demander un rendez-vous pour une réparation de son véhicule. Cette demande est transmise au planificateur.

- Schedule → Admin : Le planificateur consulte l'administrateur pour vérifier la disponibilité des techniciens et des ressources nécessaires pour la réparation.

- Admin → Schedule : L'administrateur confirme la disponibilité et renvoie cette information au planificateur.

- Schedule → Client : Le système propose des créneaux horaires disponibles au client, en fonction des informations reçues de l'administrateur.

- Client → Schedule : Le client sélectionne un créneau horaire parmi ceux proposés.

- Schedule → Technicien : Le planificateur assigne le rendez-vous au technicien disponible pour la réparation.

- Technicien → Garage : Le technicien prépare le matériel et les équipements nécessaires pour la réparation.

- Garage → Technicien : Le garage confirme que tout le matériel est prêt pour l'intervention.

- Schedule → Client : Le planificateur envoie une confirmation de rendez-vous au client, complétant ainsi le processus de prise de rendez-vous.

- Schedule → Email Server : Un e-mail de confirmation de rendez-vous est envoyé au client pour garder une trace écrite et rappeler l'heure du rendez-vous.

- Email Server → Client : Le client reçoit l'e-mail de confirmation.

3. Réservation d'un Box :

- Client → Schedule : Le client a également la possibilité de réserver un box pour effectuer lui-même certaines réparations. Cette demande est transmise au planificateur.

- Schedule → Client : Le planificateur confirme la réservation du box et fournit les détails nécessaires.

- Schedule → Email Server : Un e-mail de confirmation de réservation est envoyé au client.

- Email Server → Client : Le client reçoit l'e-mail de confirmation de réservation.

4. Apport du Véhicule et Réparation :

- Client → Technicien : À l'heure prévue du rendez-vous, le client apporte son véhicule au technicien assigné.

- Technicien → Garage : Le technicien commence la réparation du véhicule en utilisant le matériel préparé.

- Garage → Technicien : Le garage confirme que la réparation est effectuée avec succès.

- Technicien → Client : Le technicien informe le client que la réparation est terminée et que le véhicule est prêt à être récupéré.

Importance des Notifications par E-mail

Les interactions avec l'Email Server montrent l'importance de maintenir les clients informés tout au long du processus. Les e-mails de confirmation après l'inscription, les rendez-vous, et les réservations de box assurent que les clients sont toujours au courant des étapes suivantes et réduisent les risques de non-présentation ou de confusion.

6. SPECIFICATIONS TECHNIQUES DU PROJET

Cette section présente les spécifications techniques du projet de gestion de garage automobile. Ces spécifications couvrent les aspects matériels et logiciels utilisés, l'architecture du système, les Framework et bibliothèques intégrées, ainsi que les technologies et outils employés pour le développement et le déploiement de l'application.

1. Backend :

- Java 21 : Langage de programmation utilisé pour le développement backend.
- Spring Boot : Framework utilisé pour créer des applications Spring autonomes et prêtes pour la production.
- Spring Data JPA : Utilisé pour l'accès et la manipulation des données en base.
- MySQL : Système de gestion de base de données relationnelle utilisé pour stocker les données de l'application.

2. Frontend :

- React.JS : Bibliothèque JavaScript pour la construction d'interfaces utilisateur réactives.
- Redux : Utilisé pour la gestion de l'état de l'application.
- React Date Time : Utilisé pour la sélection et la gestion des dates et heures dans l'interface utilisateur.
- React Hook : Utilisé pour simplifier la gestion de l'état et des effets dans les composants React.

- Material Icons : Utilisé pour l'affichage d'icônes dans l'interface utilisateur.

3. Outils de Développement :

- WebStorm : IDE utilisé respectivement pour le frontend et le backend.
- PHPMyAdmin : Interface web pour la gestion de la base de données MySQL.
- -Postman : Utilisé pour tester les API et les requêtes HTTP.
- Figma : Outil de conception d'interface utilisateur utilisé pour créer des maquettes et des prototypes.
- JMerise : Outil de modélisation de données utilisé pour concevoir et visualiser la structure de la base de données.

4. Outils de Gestion de Projet :

- Trello : Utilisé pour la gestion des tâches et des sprints.
- Discord : Utilisé pour la communication en équipe.
- GitHub : Plateforme utilisée pour héberger le code source et faciliter la collaboration.

Spécifications Matérielles

Le développement et le test de l'application ont été réalisés sur des machines ayant les spécifications minimales suivantes :

- Processeur : Intel Core i7 11800H
- Mémoire Vive : 16 Go de RAM
- Stockage : 256 Go SSD
- Système d'exploitation : Windows 11

Déploiement et Environnement

L'application peut être déployée sur un serveur VPS en ligne. L'utilisation de Docker pour la containerisation permet une gestion simplifiée des déploiements et des mises à jour. L'environnement de développement est configuré pour être le plus proche possible de l'environnement de production afin de minimiser les différences et les erreurs de déploiement.

- Docker : Utilisé pour containeriser l'application.
- CI/CD : GitHub Actions utilisé pour automatiser les tests et les déploiements.
- Serveur Web : Nginx utilisé pour servir l'application frontend.

7. LES REALISATIONS DU CANDIDAT

Dans cette section, je vais détailler les différentes réalisations que j'ai accomplies au cours du projet de gestion de garage automobile. Ces réalisations couvrent plusieurs aspects du développement, de la conception de l'architecture du système à l'implémentation de fonctionnalités spécifiques, en passant par la gestion de la sécurité et l'amélioration de l'expérience utilisateur.

1. Définition de l'Architecture du Système : J'ai conçu une architecture robuste en utilisant Java Spring Boot pour le backend et React.js pour le frontend. Cette architecture repose sur une séparation claire des préoccupations, facilitant la maintenance et l'évolution du système.
2. Modélisation des Données : J'ai créé un modèle de données relationnel en utilisant JPA (Java Persistence API), définissant les entités, les relations entre elles et les contraintes nécessaires pour assurer l'intégrité des données.
3. J'ai développé des fonctionnalités permettant l'inscription des utilisateurs, leur authentification via JWT, et la gestion de leurs profils.
4. J'ai mis en place un système de gestion des rôles pour différencier les permissions des techniciens, des clients et des responsables du garage.
5. J'ai implémenté un système de prise de rendez-vous en ligne, permettant aux clients de sélectionner des créneaux horaires disponibles en fonction de la durée estimée des réparations.
6. J'ai intégré des notifications par email pour confirmer les rendez-vous et rappeler aux clients leurs rendez-vous à venir.
7. J'ai développé des fonctionnalités pour l'ajout, la modification et la suppression de véhicules utilisateur.
8. J'ai créé un catalogue en ligne permettant aux clients de consulter les véhicules disponibles avec leurs caractéristiques et prix.
9. J'ai implémenté l'authentification et l'autorisation en utilisant JWT pour sécuriser les endpoints de l'API.
10. J'ai ajouté des fonctionnalités de réinitialisation et de validation des mots de passe pour assurer la sécurité des comptes utilisateurs.
11. J'ai rédigé des tests unitaires et d'intégration pour vérifier la fiabilité des composants critiques de l'application.
12. J'ai mis en place des scénarios de tests automatisés pour garantir que les nouvelles fonctionnalités n'introduisent pas de régressions.

13. Interface Utilisateur Intuitive : J'ai conçu et développé une interface utilisateur réactive et intuitive avec React.js, permettant aux utilisateurs de naviguer facilement dans l'application.
14. Notifications et Alertes : J'ai intégré un système de notifications par email pour informer les utilisateurs des événements importants, comme la confirmation d'inscription, la prise de rendez-vous, et les rappels de rendez-vous.
15. Documentation : J'ai rédigé une documentation détaillée pour les développeurs et les utilisateurs finaux, couvrant l'installation, l'utilisation et le développement de l'application.

7.1. INTERFACES UTILISATEUR ET LE CODE CORRESPONDANT

1.Inscription et connexion à l'API

Pour l'inscription et la connexion à l'API, j'ai développé deux formulaires distincts : l'un dédié à l'inscription et l'autre à la connexion. Afin de collecter les informations des utilisateurs et de les transmettre au back-end de mon API REST de manière sécurisée, j'ai opté pour l'utilisation de React Hook Form en conjonction avec des requêtes Axios en méthode POST.

Pour garantir l'intégrité des données et la sécurité de l'application, j'ai implémenté des patterns de validation dans les formulaires. Ces patterns assurent que les utilisateurs saisissent des informations correctes, en limitant par exemple l'entrée à des caractères alphabétiques ou numériques, tout en évitant l'injection de code HTML (XSS).

Pour la gestion de l'authentification et de l'autorisation dans mon application, j'ai choisi d'implémenter JSON Web Tokens (JWT). Les JWT sont des jetons sécurisés qui permettent de vérifier l'identité d'un utilisateur de manière fiable et efficace.

Lorsqu'un utilisateur s'inscrit ou se connecte à mon API, après avoir vérifié et validé les données via React Hook Form et les patterns de validation, le serveur génère un JWT qui est ensuite renvoyé au client. Ce JWT contient des informations spécifiques à l'utilisateur, telles que son ID ou son rôle, et est signé numériquement pour garantir son intégrité.

Une fois que l'utilisateur possède ce JWT, il peut l'inclure dans les en-têtes de ses requêtes vers l'API. Le serveur peut alors vérifier la validité du JWT pour chaque requête reçue, en s'assurant qu'il n'a pas été altéré et qu'il n'a pas expiré. Cela permet à mon application de contrôler l'accès aux ressources et de garantir que seuls les utilisateurs authentifiés et autorisés peuvent effectuer certaines actions.

De plus, du côté de Front-End j'ai mis en place un système d'affichage d'erreurs pour signaler à l'utilisateur les champs incorrectement remplis ou les erreurs rencontrées lors de la soumission du formulaire. Cela permet à l'utilisateur de comprendre facilement où se situent les problèmes et comment les résoudre.

Code correspondent :

```
<label htmlFor="signup-adresse">Adresse*</label>
<input type="text" id="signup-adresse" required
       autoComplete="off" {...register( name: "adresse",  options: {required: true, minLength: 5})}/>
{errors.adresse && <p className="error-message">L'adresse est requis.</p>}

<label htmlFor="signup-phone">Numéro de téléphone*</label>
<input type="text" id="signup-phone" required autoComplete="off" {...register( name: "phone_number
    required: true,
    pattern: /^(\+[0-9]{1,3}[-\s]?)?([0-9]{10})$/

})}/>
{errors.phone_number && <p className="error-message">Le numéro de téléphone est requis.</p>}

<label htmlFor="signup-email">Email*</label>
<input type="email" id="signup-email" required autoComplete="off" {...register( name: "email",  opt
```

2. Validation par email

Pour renforcer la sécurité, j'ai utilisé la double authentification. Lors de l'inscription, l'utilisateur reçoit un code de validation dans sa boîte mail. Il doit ensuite entrer ce code dans le champ d'activation pour que son compte soit activé.

J'ai aussi implémenté une fonctionnalité de mot de passe oublié. Si un utilisateur oublie son mot de passe, il peut le réinitialiser en cliquant sur le bouton "Mot de passe oublié". Il doit ensuite entrer son adresse mail. Dès qu'il entre son adresse mail, il recevra un code de réinitialisation. Ce code est actif pendant 10 minutes. Une fois qu'il a le code, il peut entrer ce code et son nouveau mot de passe. L'utilisateur doit choisir un mot de passe qui contient des minuscules, des majuscules, des chiffres, des caractères spéciaux, et qui a une longueur minimale de 8 caractères.

Code correspondent :

```
{emailSent && (
    <form onSubmit={handleSubmit(resetPassword)}>
        <h3>Veuillez insérer votre code d'activation et votre nouveau mot de passe.</h3>
        <label>Code reçu</label>
        <input type="text" {...register( name: "code",  options: { required: true })} />
        {errors.code && <span>Ce champ est requis</span>}
        <label>Nouveau mot de passe</label>
        <p>Le mot de passe doit contenir au moins 8 caractères, <span style={{color:'red',fontSize:'2.5rem'}}>Incluant</span>
            <li>une majuscule</li>
            <li>chiffre(0,1,2,3,...)</li>
            <li>caractère spécial(.,/*-+@.....)</li>
        <input type="password" {...register( name: "password",  options: { required: true })} />
        {errors.password && <span>Ce champ est requis</span>}
        <button type="submit">Réinitialiser le mot de passe</button>
    </form>
)}
```

3. Page de profil utilisateur

La page de profil utilisateur constitue un espace central où les utilisateurs peuvent gérer et visualiser leurs informations personnelles ainsi que leur historique d'activités sur l'application. Voici une description détaillée des fonctionnalités de cette page :

1. Affichage des informations personnelles : L'utilisateur peut voir ses informations personnelles telles que son nom, son adresse e-mail, son numéro de téléphone, etc.

2. Historique des réservations et des locations : Une section de l'interface affiche l'historique complet des réservations et des locations effectuées par l'utilisateur. Ces informations sont généralement présentées sous forme de liste ou de tableau, avec des détails tels que la date, l'heure, la durée de la réservation ou de la location, et les informations sur le véhicule loué.

3. Modification des données personnelles : L'utilisateur a la possibilité de modifier ses informations personnelles directement à partir de cette page. Un formulaire de mise à jour peut être inclus, permettant à l'utilisateur de modifier son nom, son adresse, son numéro de téléphone, etc.

4. Ajout de voitures : Une fonctionnalité permet à l'utilisateur d'ajouter de nouveaux véhicules à sa liste. Cela peut être utile s'il souhaite louer ou vendre un nouveau véhicule via l'application.

5. Faire des réservations et des locations : L'utilisateur peut accéder à des liens ou des boutons qui le dirigent vers des pages spécifiques où il peut effectuer de nouvelles réservations ou locations de véhicules.

6. Affichage des détails des véhicules dans une fenêtre modale : Lorsque l'utilisateur consulte son historique de réservations ou de locations, il peut cliquer sur un élément pour afficher plus de détails sur le véhicule concerné. Ces détails s'affichent généralement dans une fenêtre modale, offrant ainsi une expérience utilisateur fluide et contextuelle.

7. Navigation intuitive à travers le menu : L'utilisateur peut accéder aux différentes fonctionnalités de la page en cliquant sur des images ou des icônes de menu, qui ouvrent des liens vers d'autres pages ou déclenchent des actions spécifiques.

7.2. EXTRAITS DE CODE DE COMPOSANTS METIER

Dans ce chapitre, je vais présenter les composants métier de mon projet de gestion de garage automobile. Ces composants encapsulent la logique métier essentielle pour gérer les plannings, les rendez-vous, les utilisateurs et les véhicules. En utilisant Java Spring Boot et JWT pour le backend, j'ai pu développer une application robuste et sécurisée. L'accent sera mis sur la logique métier implémentée dans les services, les contrôleurs et les entités de l'application.

Description des Composants Métier

Les composants métier de mon projet sont structurés de manière à assurer une gestion efficace et intuitive des différentes fonctionnalités du garage. Voici une description de certains de ces composants :

Gestion des Utilisateurs :

- Inscription et Authentification : La gestion des utilisateurs inclut l'inscription de nouveaux utilisateurs et leur authentification via JWT.
- Gestion des Profils : Permet la mise à jour et la gestion des informations de profil des utilisateurs.

- Gestion des Rôles : Les utilisateurs peuvent avoir différents rôles (technicien, client, responsable) avec des permissions spécifiques.

```
public void inscription(Utilisateur utilisateur) {  1 usage   ogtayaliyev
    PasswordValidator passwordValidator = new PasswordValidator();
    String email = utilisateur.getEmail();


    if (!passwordValidator.isPasswordValid(utilisateur.getPassword())) {
        throw new RuntimeException("Le mot de passe ne répond pas aux critères de complexité.");
    }

    if(!utilisateur.getEmail().contains("@")) {
        throw  new RuntimeException("Votre mail invalide");
    }
    if(!utilisateur.getEmail().contains(".")) {
        throw  new RuntimeException("Votre mail invalide");
    }
      utilisateur.setActif(Boolean.TRUE);
    Optional<Utilisateur> utilisateurOptional = this.utilisateurRepository.findByEmail(utilisateur.getEmail());
    if(utilisateurOptional.isPresent()) {
        throw  new RuntimeException("Votre mail est déjà utilisé");
    }
```

Gestion des Rendez-vous :

- Prise de Rendez-vous : Les clients peuvent prendre des rendez-vous pour des réparations en ligne, en choisissant des créneaux horaires disponibles.
- Confirmation des Rendez-vous : Les rendez-vous sont confirmés via des notifications par email.
- Gestion des Plannings : Les plannings des techniciens sont optimisés pour éviter les conflits et maximiser l'efficacité.

```
@PostMapping   ogtayaliyev
public ResponseEntity<String> makeReservation(@RequestBody Reparation reparation) throws Exception {
    // Extraire l'utilisateur actuellement authentifié
    Utilisateur utilisateur = (Utilisateur) SecurityContextHolder.getContext().getAuthentication().getPrincipal();
    String userEmail = utilisateur.getEmail();
    // Récupérer l'ID de l'utilisateur
    int utilisateurId = utilisateur.getId();

    // Récupérer la plaque d'immatriculation de la réparation
    String plaqueImmatriculation = reparation.getPlaqueImmatriculation();

    // Trouver la voiture en fonction de la plaque d'immatriculation
    Voitures voiture = voitureRepository.findByPlaqueImmatriculation(plaqueImmatriculation);
    if (voiture == null) {
        return ResponseEntity.badRequest().body("Aucune voiture trouvée pour la plaque d'immatriculation fournie."
    }
```

Gestion des Véhicules :

- Ajout et Modification de Véhicules : Permet l'ajout, la modification et la suppression des véhicules d'occasion disponibles dans le garage.
- Catalogue de Véhicules : Les clients peuvent consulter le catalogue des véhicules disponibles.

```java
public void addVoiture(int idUtilisateur, Voitures voitures) {  1 usage  ogtayaliyev

    Utilisateur utilisateur = utilisateurRepository.findById(idUtilisateur)
            .orElseThrow(() -> new RuntimeException("Utilisateur introuvable avec l'ID : " + idUtilisateur));

    // Associer la voiture à l'utilisateur
    voitures.setUtilisateur(utilisateur);

    // Sauvegarder la voiture dans la base de données
    voitureRepository.save(voitures);
}
```

Gestion de la Sécurité :

- Réinitialisation de Mot de Passe : Fonctionnalité permettant aux utilisateurs de réinitialiser leur mot de passe en cas d'oubli, avec envoi d'un lien sécurisé par email.
- Validation du Mot de Passe : Les mots de passe sont validés pour assurer qu'ils respectent les critères de sécurité (longueur, complexité).

```java
public void modifierMdp(Map<String, String> parametres) {  1 usage  ogtayaliyev
    Utilisateur utilisateur = this.loadUserByUsername(parametres.get("email"));
    this.validationService.enregistrer(utilisateur);
}

public void nouveauMdp(Map<String, String> parametres) {  1 usage  ogtayaliyev
    Utilisateur utilisateur = this.loadUserByUsername(parametres.get("email"));
   final Validation validation = validationService.lireEnFonctionDuCode(parametres.get("code"));
   if(validation.getUtilisateur().getEmail().equals(utilisateur.getEmail())){
    String mdpCrypte = this.passwordEncoder.encode(parametres.get("password"));
    utilisateur.setPassword(mdpCrypte);}
   this.utilisateurRepository.save(utilisateur);
}
```

Notifications :

- Notification d'Inscription : Les utilisateurs reçoivent une notification par email lors de leur inscription.
- Notification de Prise de Rendez-vous : Confirmation par email des rendez-vous pris par les clients.
- Notification de Réservation de Garage Solidaire : Envoi d'un email aux clients adhérents lorsqu'ils réservent des créneaux pour utiliser le garage solidaire.

```java
public class NotificationService {
    JavaMailSender javaMailSender;
    public void envoyer(Validation validation) {  1 usage   ogtayaliyev
        SimpleMailMessage message = new SimpleMailMessage();
        message.setFrom("ogtayaliyev9@gmail.com");
        message.setTo(validation.getUtilisateur().getEmail());
        message.setSubject("Votre code d'activation");

        String texte = String.format(
                "Bonjour %s, " +
                        " Votre code d'activation est %s,veuiller saisir votre code sur le page dans la champ activation
                validation.getUtilisateur().getNom(),
                validation.getCode()
        );
        message.setText(texte);

        javaMailSender.send(message);
    }
```

7.3. EXTRAITS DE CODE DE COMPOSANTS D'ACCES AUX DONNEES

Dans cette section, je vais présenter des extraits de code qui illustrent les composants d'accès aux données utilisés dans notre projet de gestion de garage automobile. En utilisant Java Spring Boot et JWT pour le backend et React.js pour le frontend, j'ai mis en place une architecture robuste et sécurisée pour gérer les données du garage, y compris les utilisateurs, les techniciens, les rendez-vous, et les véhicules.

Structure des Composants d'Accès aux Données

J'ai structuré les composants d'accès aux données en utilisant les repositories Spring Data JPA, les services pour la logique métier, et les contrôleurs REST pour exposer les endpoints de l'API. Voici un aperçu de cette structure :

1. Entité Classes : Représentent les tables de la base de données.

2. Repositories : Interfaces qui permettent les opérations CRUD sur les entités.

3. Services : Contiennent la logique métier pour manipuler les données.

Extraits de Code

Entité Classes

Les classes entités mappent les tables de la base de données. Voici un exemple avec la classe `Utilisateur` **:**

```java
@Entity
@Table(name = "utilisateur")
public class Utilisateur implements UserDetails {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    private String nom;
    private String prenom;
    private String password;
    private String email;
    private String phone_number;
    private String adresse;
```

Explication : La classe `Utilisateur` représente un utilisateur dans une base de données. Chaque utilisateur a des informations comme un identifiant, un nom, un prénom, un mot de passe, un email, un numéro de téléphone et une adresse. Il y a aussi des dates automatiques de création et de modification, et un statut pour savoir si l'utilisateur est actif ou non.

L'utilisateur a un rôle spécifique, des voitures, des boxes de location et des réparations qui lui sont associés. La classe inclut aussi des méthodes pour intégrer la sécurité, comme obtenir le mot de passe, le nom d'utilisateur et vérifier si le compte est actif et non expiré.

Repositories

Les repositories permettent d'accéder aux données et de les manipuler facilement. Voici un exemple avec le `UserRepository` :

```java
public interface UtilisateurRepository extends CrudRepository<Utilisateur, Integer> {  17 usages  ogtayaliyev

    // Method to find a user by email.
    Optional<Utilisateur> findByEmail(String email);  5 usages  ogtayaliyev

    // Method to find a user by ID.
    Optional<Utilisateur> findById(Long id);  1 usage  ogtayaliyev

    // Method to delete a user by ID.
    void deleteById(Long id);  no usages  ogtayaliyev
    boolean existsByEmail(String email);  1 usage  ogtayaliyev
}
```

Explication : L'interface `UtilisateurRepository` est une interface Java qui étend `CrudRepository`. Cela permet d'utiliser les méthodes CRUD (Create, Read, Update, Delete) pour interagir avec les données de la table `Utilisateur` dans une base de données.

1. `Optional<Utilisateur> findByEmail (String email) ; `

- Cette méthode permet de trouver un utilisateur par son adresse e-mail.

- Elle prend une chaîne de caractères représentant l'e-mail comme paramètre.

- Elle renvoie un `Optional<Utilisateur>`, ce qui signifie qu'elle peut renvoyer un utilisateur ou un objet vide si aucun utilisateur n'est trouvé avec cet e-mail

2. `Optional<Utilisateur> findById (Long id) ; `

- Cette méthode permet de trouver un utilisateur par son identifiant (ID).

- Elle prend un `Long` représentant l'ID comme paramètre.

- Elle renvoie un `Optional<Utilisateur>`, ce qui signifie qu'elle peut renvoyer un utilisateur ou un objet vide si aucun utilisateur n'est trouvé avec cet ID.

3. `void deleteById (Long id) ; `

- Cette méthode permet de supprimer un utilisateur par son identifiant (ID).

- Elle prend un `Long` représentant l'ID comme paramètre.

- Elle ne renvoie rien (`void`).

Services

Les services contiennent la logique métier. Voici un exemple de service pour gérer les utilisateurs :

```
    throw new RuntimeException("Le mot de passe ne répond pas aux critères de complexité.");}

    if(!utilisateur.getEmail().contains("@")) {
        throw  new RuntimeException("Votre mail invalide");}
    if(!utilisateur.getEmail().contains(".")) {
        throw  new RuntimeException("Votre mail invalide");}
      utilisateur.setActif(Boolean.TRUE);
    Optional<Utilisateur> utilisateurOptional = this.utilisateurRepository.findByEmail(utilisateur.getEmail());
    if(utilisateurOptional.isPresent()) {
        throw  new RuntimeException("Votre mail est déjà utilisé");}
    if (utilisateurRepository.existsByEmail(email)) {
        throw new RuntimeException("Cette mail deja existe");}
    String mdpCrypte = this.passwordEncoder.encode(utilisateur.getPassword());
    utilisateur.setPassword(mdpCrypte);

    Role roleUtilisateur = new Role();
    roleUtilisateur.setLibelle(TypeDeRole.UTILISATEUR);
    utilisateur.setRole(roleUtilisateur);
```

Explication : Mon Service`UtilisateurService` est un composant central de mon application qui s'occupe de diverses opérations liées aux utilisateurs. Voici une vue d'ensemble de ses responsabilités :

1. Inscription des utilisateurs : Le service gère l'inscription des nouveaux utilisateurs en validant leurs informations, en cryptant leurs mots de passe et en les enregistrant dans la base de données. Il s'assure également que les e-mails fournis sont valides et uniques.

2. Activation des comptes : Après l'inscription, les utilisateurs doivent activer leurs comptes. Le service vérifie les codes d'activation pour s'assurer qu'ils ne sont pas expirés et active les comptes en conséquence.

3. Mise à jour des informations utilisateur : Le service permet la mise à jour des informations personnelles des utilisateurs, telles que le nom, le prénom, l'e-mail et le numéro de téléphone.

4. Modification des mots de passe : Il gère les modifications de mots de passe, que ce soit pour des raisons de sécurité ou de réinitialisation, en validant les demandes et en enregistrant les nouveaux mots de passe cryptés

5. Gestion des validations : Le service interagit avec `ValidationService` pour enregistrer et vérifier les validations nécessaires pour des opérations comme l'inscription et la modification des mots de passe.

6. Chargement des utilisateurs pour l'authentification : En implémentant `UserDetailsService` de Spring Security, le service permet de charger les détails des utilisateurs par leur e-mail pour les besoins de l'authentification.

7.4. EXTRAITS DE CODE D'AUTRES COMPOSANTS (CONTROLEURS, UTILITAIRES...).

Dans cette section, je vais présenter le contrôleur, la gestion de la sécurité avec JWT, et le validateur de mot de passe.

Controller :

Les contrôleurs REST exposent les services via des endpoints API. Voici un exemple avec le ` AuthentificationController ` :

```
@PostMapping(path = "inscription",consumes = MediaType.APPLICATION_JSON_VALUE)  ogtayaliyev
public ResponseEntity<?> inscription(@RequestBody Utilisateur utilisateur) {
    log.info("Inscription");
    utilisateurService.inscription(utilisateur);
    return ResponseEntity.ok().build();
}
@PostMapping(path = "activation")  ogtayaliyev
public void activation(@RequestBody Map<String, String> activation) {
    this.utilisateurService.activation(activation);
}
@PostMapping(path = "modifier-mdp")  ogtayaliyev
public void modifierMdp(@RequestBody Map<String, String> activation){
    this.utilisateurService.modifierMdp(activation);
}
@PostMapping(path = "nouveau-mdp")  ogtayaliyev
public void nouveauMdp(@RequestBody Map<String, String> activation){
    this.utilisateurService.nouveauMdp(activation);
}
@PostMapping(path = "connexion")  ogtayaliyev
public ResponseEntity connexion(@RequestBody AuthentificationDTO authentificationDTO) {
    final Authentication authenticate = authenticationManager.authenticate(
            new UsernamePasswordAuthenticationToken(authentificationDTO.getEmail(), authentificationDTO.getPassword())
    );
    if(authenticate.isAuthenticated()) {
        Map<String,String> token = jwtService.generate(authentificationDTO.getEmail());
        return ResponseEntity.ok(token);
```

Explication : Le contrôleur `AuthentificationController` gère les requêtes liées à l'authentification et à la gestion des utilisateurs dans votre application. Il offre plusieurs points d'entrée pour des opérations telles que l'inscription, l'activation de compte, la modification et la réinitialisation du mot de passe, ainsi que la connexion.

Lorsqu'un utilisateur souhaite s'inscrire, il envoie une requête POST à l'endpoint `/inscription`, fournissant les détails de l'utilisateur. Le contrôleur appelle ensuite le service `inscription` de `UtilisateurService` pour enregistrer l'utilisateur.

Pour activer un compte, l'utilisateur envoie une requête POST à l'endpoint `/activation` avec un code d'activation. Le contrôleur utilise le service `activation` de `UtilisateurService` pour activer le compte associé au code fourni.

Les demandes de modification et de réinitialisation du mot de passe sont gérées par les endpoints `/modifier-mot de passe` et `/nouveau-mot de passe`. Ces requêtes contiennent les informations nécessaires pour effectuer les opérations correspondantes, telles que le changement de mot de passe et la validation des codes.

Enfin, le processus de connexion est géré par l'endpoint `/connexion`. L'utilisateur envoie ses informations d'identification via un objet `AuthentificationDTO`. Le contrôleur utilise l'`AuthenticationManager` de Spring Security pour authentifier les informations fournies. Si l'authentification réussit, un jeton JWT est généré à l'aide de `JwtService` et renvoyé dans la réponse.

Dans l'ensemble, `AuthentificationController` assure la gestion fluide des opérations liées aux utilisateurs, en garantissant la sécurité et la validité des données à chaque étape du processus.

Gestion de la Sécurité avec JWT

Pour sécuriser les accès, j'ai mis en place un système de tokens JWT. Voici un exemple de génération de token avec `JwtTokenProvider` :

```
public Map<String, String> generate(String username) {  1 usage   ogtayaliyev
    Utilisateur utilisateur = this.utilisateurService.loadUserByUsername(username);
    Map<String, String> jwtMap = this.generateJwt(utilisateur);

    final Jwt jwt = Jwt
            .builder()
            .valeur(jwtMap.get(BEARER))
            .desactive(false)
            .expire(false)
            .utilisateur(utilisateur)
            .build();
    this.jwtRepository.save(jwt);

    return jwtMap;
}
```

Explication : Le service `JwtService` gère la création, la validation et la manipulation des jetons JWT (JSON Web Tokens) pour l'authentification des utilisateurs. Il comprend les fonctionnalités suivantes :

- Génération de Jeton JWT : La méthode `generate` crée un jeton JWT pour un utilisateur donné, avec une durée de validité de 30 minutes. Le jeton est signé à l'aide d'une clé secrète.

- Extraction du Nom d'Utilisateur : La méthode `extractUsername` extrait le nom d'utilisateur à partir d'un jeton JWT.

- Validation de la Date d'Expiration du Jeton : La méthode `isTokenExpired` vérifie si la date d'expiration du jeton JWT est dépassée.

- Remplacement du Jeton : La méthode `replaceToken` génère un nouveau jeton JWT pour un utilisateur, utilisée par exemple lorsqu'un utilisateur demande un remplacement de jeton.

- Accès aux Réclamations du Jeton : Des méthodes privées permettent d'accéder aux différentes réclamations (claims) du jeton JWT, telles que la date d'expiration et le sujet

- Clé de Cryptage : La méthode privée `getKey` retourne la clé utilisée pour signer et vérifier les jetons JWT. La clé est décodée à partir d'une chaîne hexadécimale et utilisée pour générer une clé HMAC.

8. ELEMENTS DE SECURITE DE L'APPLICATION

La sécurité est un aspect crucial dans le développement de mon application. J'ai mis en place plusieurs mesures pour garantir la protection des données des utilisateurs et la sécurité générale de l'application. Voici les principales mesures de sécurité que nous avons intégrées :

1. Authentification et autorisation

- Authentification multi-facteurs (MFA) : Pour renforcer la sécurité des comptes utilisateurs, nous avons implémenté une authentification multi-facteurs. En plus du mot de passe, un code de vérification est envoyé par SMS ou e-mail.

- Gestion des rôles et des permissions : J'ai mis en place un système de gestion des rôles pour contrôler l'accès aux différentes fonctionnalités de l'application. Chaque utilisateur dispose de permissions spécifiques en fonction de son rôle (administrateur, utilisateur.).

2. Authentification et autorisation avec JWT

- J'ai utilisé JSON Web Tokens (JWT) pour l'authentification des utilisateurs dans notre application. JWT nous permet de générer des jetons d'accès sécurisés après l'authentification réussie d'un utilisateur.

- Ces jetons sont signés cryptographiquement et contiennent des informations sur l'utilisateur, comme son identifiant et ses autorisations. Ils sont ensuite envoyés au client et utilisés pour authentifier les requêtes subséquentes vers notre API.

- De plus, j'ai utilisé une fonction de hachage de mot de passe (passwordencoder) pour sécuriser les mots de passe des utilisateurs avant de les stocker dans notre base de données.

```
public ConfigurationSecuriteApplication(BCryptPasswordEncoder bCryptPasswordEncoder,
                                        JwtFilter jwtFilter,
                                        UserDetailsService userDetailsService) {
    this.bCryptPasswordEncoder = bCryptPasswordEncoder;
    this.jwtFilter = jwtFilter;
    this.userDetailsService = userDetailsService;
}
```

3. Protection des ressources avec Spring Boot

- Mon back-end est développé en utilisant Spring Boot, un framework Java pour le développement d'applications web. Spring Boot offre des fonctionnalités de sécurité robustes, notamment la protection contre les attaques par injection SQL et les failles de sécurité courantes.

- J'ai utilisé les fonctionnalités de sécurité fournies par Spring Boot pour configurer l'authentification basée sur les jetons JWT et pour gérer les autorisations des utilisateurs.

4. Communications sécurisées avec Axios

- Dans mon front-end, développé avec React.js, J'ai utilisé Axios comme client HTTP pour interagir avec notre API back-end. Axios nous permet d'envoyer des requêtes HTTP de manière sécurisée en incluant automatiquement les jetons JWT nécessaires dans les en-têtes des requêtes.

- J'ai utilisé également Axios pour gérer les réponses HTTP et traiter les erreurs de manière sécurisée, en mettant en œuvre des mécanismes de gestion des erreurs appropriés pour protéger les données sensibles.

5. Protection contre les attaques courantes

- Prévention des injections SQL : J'ai utilisé des requêtes préparées et des ORM (Object-Relationnel Mapping) pour empêcher les attaques par injection SQL.

```
public interface ReparationRepository extends JpaRepository<Reparation,Long> {  3 usages  ogtayaliyev
    @Query("SELECT r FROM Reparation r WHERE r.reparationType = :reparationType AND r.endDate > CURRENT_TIMESTAMP")
    List<Reparation> findActiveReservationsByReparationType(@Param("reparationType") ReparationType reparationType);
```

A l'aide de @Param, ce qui permet à Spring Data JPA de lier dynamiquement les valeurs des paramètres et d'éviter les injections SQL. Ainsi, en utilisant @Query avec des paramètres nommés, je peux sécuriser mon application Spring Boot contre les injections SQL dans mes requêtes personnalisées.

- Protection contre les attaques XSS (Cross-Site Scripting) : J'ai validé systématiquement toutes les entrées utilisateur pour prévenir l'injection de scripts malveillants.

```
<label htmlFor="signup-adresse">Adresse*</label>
<input type="text" id="signup-adresse" required
       autoComplete="off" {...register( name: "adresse",  options: {required: true, minLength: 5})}/>
{errors.adresse && <p className="error-message">L'adresse est requis.</p>}

<label htmlFor="signup-phone">Numéro de téléphone*</label>
<input type="text" id="signup-phone" required autoComplete="off" {...register( name: "phone_number
    required: true,
    pattern: /^(\+[0-9]{1,3}[-\s]?)?([0-9]{10})$/

})}/>
{errors.phone_number && <p className="error-message">Le numéro de téléphone est requis.</p>}

<label htmlFor="signup-email">Email*</label>
<input type="email" id="signup-email" required autoComplete="off" {...register( name: "email",  opt
```

Dans le JSX, le contenu texte est affiché en utilisant la propriété textContent des éléments. Cela assure que le texte est traité comme du texte brut et non comme du code HTML potentiellement dangereux.

Les attributs des champs de formulaire, tels que required et autoComplete, sont utilisés pour améliorer la sécurité du formulaire en s'assurant que les champs obligatoires sont remplis et que les informations sensibles ne sont pas automatiquement complétées par le navigateur.

6. Mises à jour et gestion des correctifs

- Mises à jour régulières : J'ai assuré que tous les composants logiciels, y compris les bibliothèques et Framework utilisés, sont régulièrement mis à jour pour corriger les vulnérabilités connues.

En mettant en œuvre ces mesures de sécurité, nous nous efforçons de protéger les données des utilisateurs et d'assurer la fiabilité et la sécurité de notre application.

9. PLAN DE TESTS

Dans cette section, je vais décrire le plan de tests que j'ai élaboré pour l'application web que j'ai développée. L'application est construite en utilisant Spring Boot pour le backend et React JS pour le frontend, avec une API REST sécurisée par JWT (JSON Web Tokens). Pour tester l'API REST, j'ai utilisé Postman.

Objectif des tests

L'objectif principal des tests est de garantir que toutes les fonctionnalités de l'application fonctionnent comme prévu et de détecter tout bug ou comportement inattendu. Cela inclut la vérification de l'exactitude des données renvoyées par l'API, la validation des fonctionnalités du frontend, ainsi que la vérification de la sécurité et de la stabilité de l'application.

Méthodologie de test

1. Tests unitaires (Backend) : Je vais créer des tests unitaires pour chaque composant logiciel du backend de l'application en utilisant JUnit et Mockito pour simuler les dépendances externes. Ces tests seront axés sur l'isolation des fonctionnalités individuelles du backend pour assurer leur bon fonctionnement.

2. Tests unitaires (Frontend) : Je vais également créer des tests unitaires pour les composants React du frontend en utilisant Jest. Ces tests se concentreront sur la vérification du comportement des différents composants frontend de manière isolée.

3. Tests d'intégration : Je vais effectuer des tests d'intégration pour vérifier le bon fonctionnement des différentes parties de l'application lorsqu'elles sont combinées. Cela comprendra des tests pour s'assurer que les composants frontend et backend communiquent correctement via l'API REST.

4. Tests avec Postman : Pour tester l'API REST, j'ai utilisé Postman. Je vais créer des suites de tests dans Postman pour vérifier chaque endpoint de l'API, en incluant des scénarios de test pour les cas normaux et les cas d'erreur.

5. Tests de performance : Je vais évaluer les performances de l'application en effectuant des tests de charge pour déterminer sa capacité à gérer un grand nombre d'utilisateurs simultanés. Je vais également surveiller les temps de réponse de l'API pour m'assurer qu'ils restent dans des limites acceptables.

6. Tests de sécurité : Je vais tester la sécurité de l'application en essayant d'exploiter les vulnérabilités potentielles telles que les injections SQL ou les attaques par déni de service. Je vais également vérifier que le système d'authentification basé sur JWT fonctionne correctement et protège les ressources de manière adéquate.

Environnement de test

Les tests seront effectués dans un environnement similaire à celui de production, en utilisant des bases de données et des serveurs similaires à ceux qui seront utilisés en production.

Calendrier des tests

Les tests seront effectués tout au long du cycle de développement de l'application, avec des tests unitaires et d'intégration exécutés à chaque itération de développement. Les tests de régression, de performance et de sécurité seront effectués avant le déploiement de chaque nouvelle version de l'application.

10. JEU D'ESSAI

Scénario	Valeurs saisies	Résultat et réponse
Test de mise à jour des informations utilisateur	Nouvelles données	Informations utilisateur mises à jour avec succès.
Ajout d'un utilisateur sans paramètres.	Aucune valeur.	Le test est validé si le statut de la réponse est égal à 400. succès : false message : Validation errors, username : Le nom d'utilisateur est obligatoire email : Veuillez entrer une adresse mail valide, l'adresse mail est obligatoire. password : Le mot de passe est obligatoire.
Ajout d'un utilisateur sans nom d'utilisateur.	Nom d'utilisateur : aucune valeur. Email : email unique généré automatiquement. Mot de passe : 123456789. Confirmation : 123456789.	Le test est validé si le statut de la réponse est égal à 400. succès : false message : Validation errors, username : Le nom d'utilisateur est obligatoire
Ajout d'un utilisateur sans mot de passe	Nom d'utilisateur : usertest. Email : email unique généré automatiquement. Mot de passe : aucune valeur. Confirmation : aucune valeur.	Le test est validé si le statut de la réponse est égal à 400. succès : false message : Validation errors, password : Le mot de passe est obligatoire.
Ajout d'un utilisateur avec un mot de passe trop court.	Nom d'utilisateur : usertest. Email : email unique généré automatiquement. Mot de passe : 123. Confirmation : 123.	Le test est validé si le statut de la réponse est égal à 400. succès : false message : Validation errors, password : Le mot de passe doit comporter au moins 8 caractères.
Ajout d'un utilisateur sans mot de passe de confirmation.	Nom d'utilisateur : usertest. Email : email unique généré automatiquement. Mot de passe : 123456789. Confirmation : aucune valeur.	Le test est validé si le statut de la réponse est égal à 400. succès : false message : Validation errors, password : La confirmation du mot de passe ne correspond pas.
Ajout d'un utilisateur avec un mot de passe de confirmation différent.	Nom d'utilisateur : usertest. Email : email unique généré automatiquement. Mot de passe : 123456789. Confirmation : 987654321.	Le test est validé si le statut de la réponse est égal à 400. succès : false message : Validation errors, password : La confirmation du mot de passe ne correspond pas.
Ajout d'un utilisateur avec un nom d'utilisateur déjà existant.	Nom d'utilisateur : admin. Email : email unique généré automatiquement. Mot de passe : 123456789 Confirmation : 123456789	Le test est validé si le statut de la réponse est égal à 400. succès : false Message : Validation errons, username : Le nom d'utilisateur « admin »

		existe déjà.
Ajout d'un utilisateur avec une adresse e-mail déjà existante.	Nom d'utilisateur : usertest. Email : admin@admin.fr. Mot de passe : 123456789 Confirmation : 123456789	Le test est validé si le statut de la réponse est égal à 400. succès : false Message : Validation errons, Email : L'adresse e-mail « admin@admin.f » existe déjà.
Test de récupération des données utilisateur avec un identifiant invalide	ID utilisateur invalide	Erreur de récupération d'informations. Identifiant invalide.
Test de récupération des données utilisateur avec un jeton JWT non valide	Jeton JWT non valide	Erreur d'authentification. Jeton non valide.

11. VEILLE TECHNOLOGIQUE

Pour rester à la pointe des avancées dans mon domaine d'études, je mets en place une stratégie de veille technologique structurée et régulière. Voici les principales étapes et outils que j'utilise pour effectuer cette veille :

1. Identification des sources d'information pertinentes :

- Sites web spécialisés : Je consulte régulièrement des sites web spécialisés dans mon domaine, tels que des blogs, des magazines en ligne, et des portails d'information technologique.

- Publications scientifiques et techniques : J'accède à des bases de données académiques pour lire des articles scientifiques récents et des rapports techniques.

- Réseaux sociaux professionnels : LinkedIn et Twitter sont des sources importantes pour suivre les experts du secteur et les entreprises innovantes. Je suis particulièrement abonné aux groupes et pages où je peux surveiller les nouveautés des langages de programmation qui m'intéressent, comme Java, JavaScript, et les Framework React et Vue.js

-Chaînes YouTube : Je m'abonne à des chaînes YouTube éducatives comme Chillo Tech et Diego Dev pour des tutoriels vidéo et des cours en ligne.

Exemple : Une série de vidéos sur Java Spring Security sur la chaîne Chillo Tech m'a aidé à mieux comprendre cette bibliothèque Java et à l'utiliser efficacement dans mes projets.

2. Utilisation d'outils de veille :

- Alertes Google : Je configure des alertes Google sur des mots-clés spécifiques afin de recevoir des notifications par e-mail dès qu'un nouvel article correspondant est publié.

- Plateformes de veille technologique : Des outils comme Scoop.it et Paper.li m'aident à organiser et partager les informations pertinentes que je trouve.

3. Participation à des événements et communautés :

- Conférences et webinaires : Je participe à des conférences, webinaires et ateliers pour découvrir les dernières innovations et tendances technologiques.

- Forums et groupes de discussion : Je suis membre de forums et groupes de discussion en ligne où je peux échanger avec d'autres professionnels et experts du domaine.

4. Synthèse et analyse des informations :

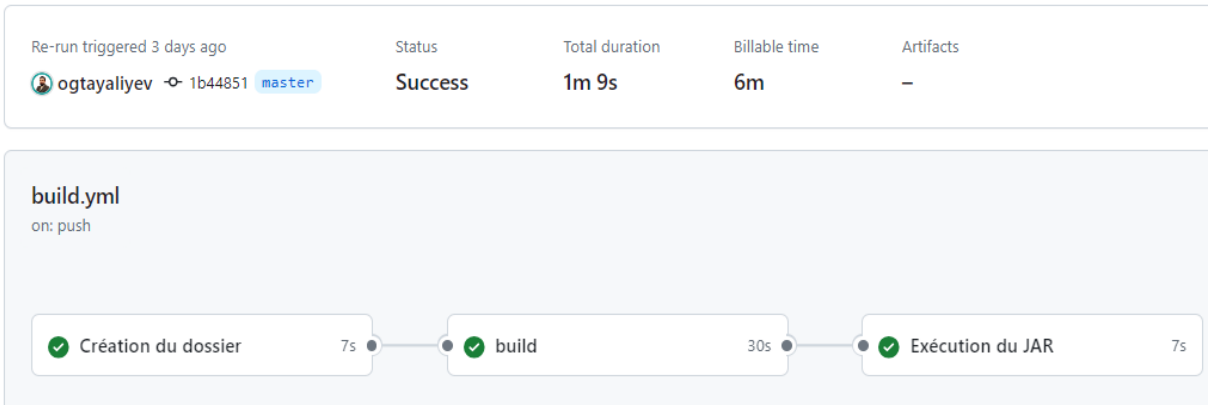
- Rédaction de rapports de veille : Régulièrement, je compile les informations collectées dans des rapports de veille que je partage avec mon équipe ou mes collègues.

- Mise à jour des connaissances : Je mets à jour mes connaissances en intégrant les nouvelles informations et en ajustant mes compétences et pratiques en conséquence.

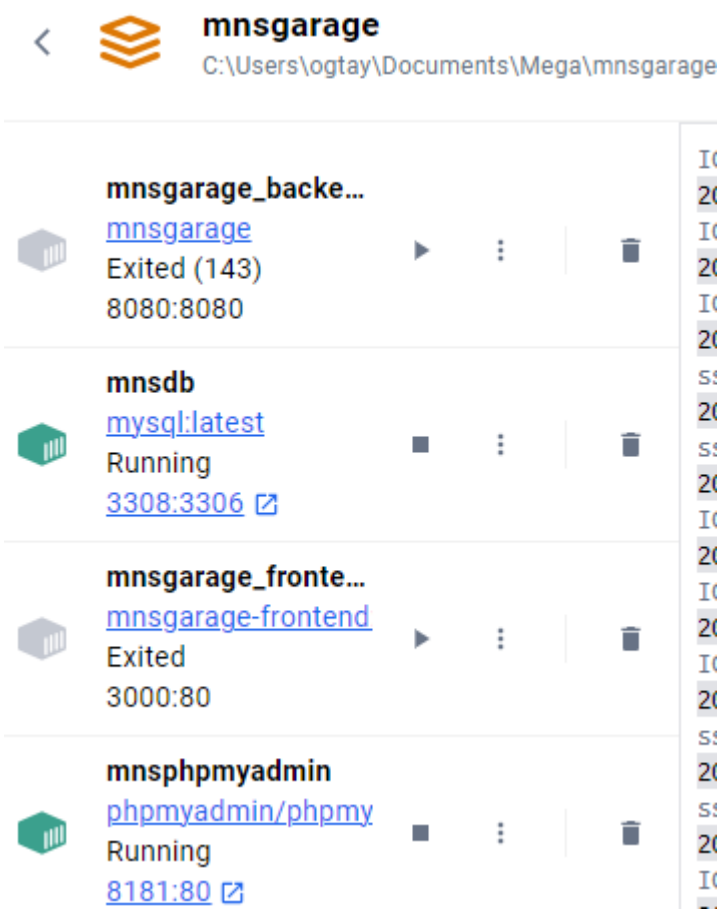
12.ANNEXES

Model MCD :

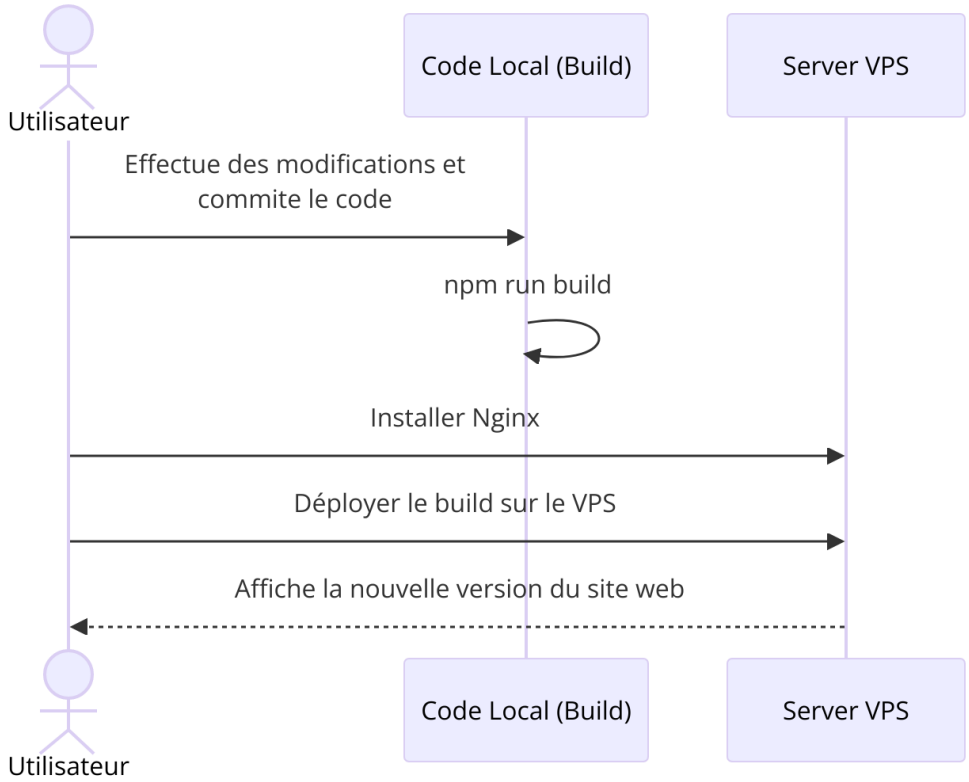
GitHub Actions :



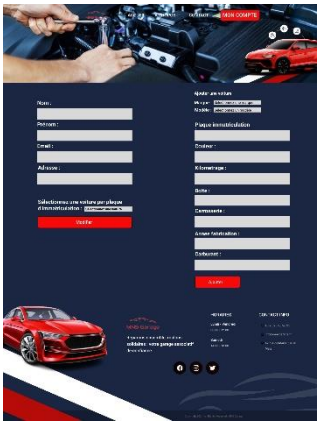
Docker containeur :

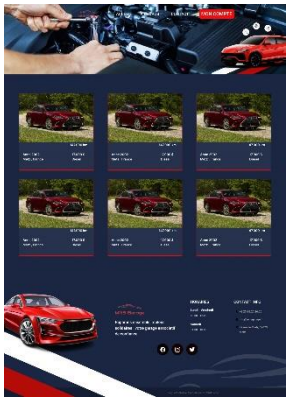


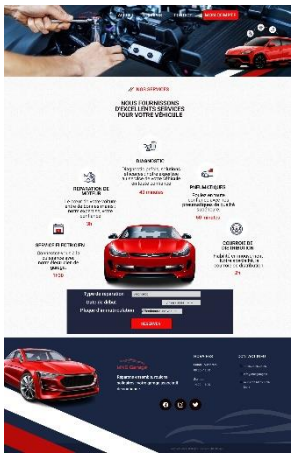
Déploiement Front-End vers serveur v VPS :



12.1. MAQUETTES DES INTERFACES UTILISATEUR







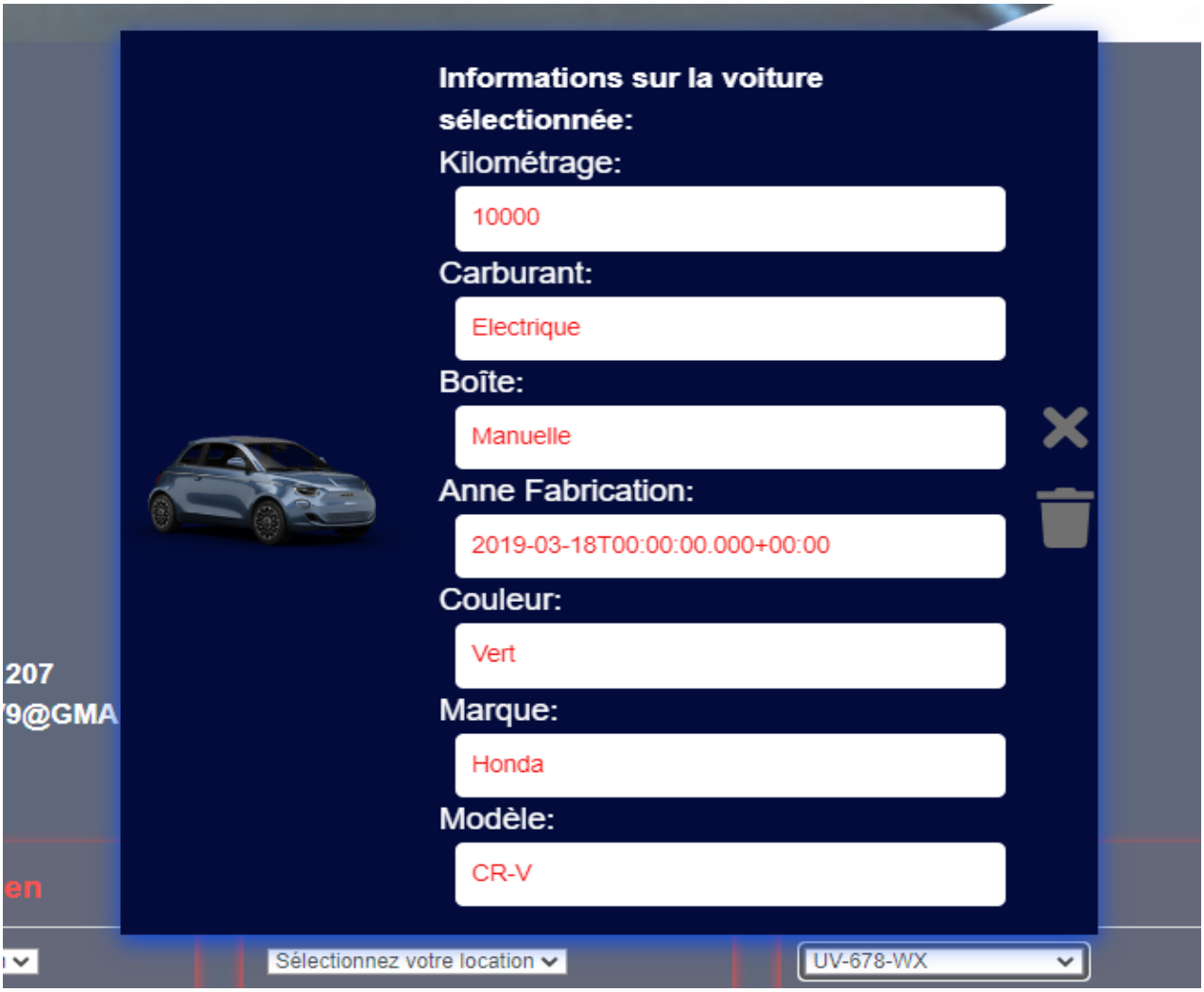
12.2. CAPTURES D'ECRANS D'INTERFACES UTILISATEURS ET LE CODE CORRESPONDANT

Button Menu :

Code correspondant :

```
return (
    <div className="floating-button-container">
        <h2>Menu</h2>
        {/* Utilisation photo a la place de button */}
        <img
            src={icon}
            alt="Floating Button"
            className="floating-button-image"
            onClick={isExpanded ? closeExpansion : toggleExpansion}
            title="Cliquez pour ouvrir le menu"
        />
        {isExpanded && (
            <div className="expanded-buttons">
                <Link to="/reservationentretien" className="sub-button">Entretien</Link>
                <Link to="/garagebox" className="sub-button">Location</Link>
                <Link to="/modif" className="sub-button">Modifier</Link>
                <Link to="/contact" className="sub-button">Contact</Link>
            </div>
        )}
    </div>
```

Modal :

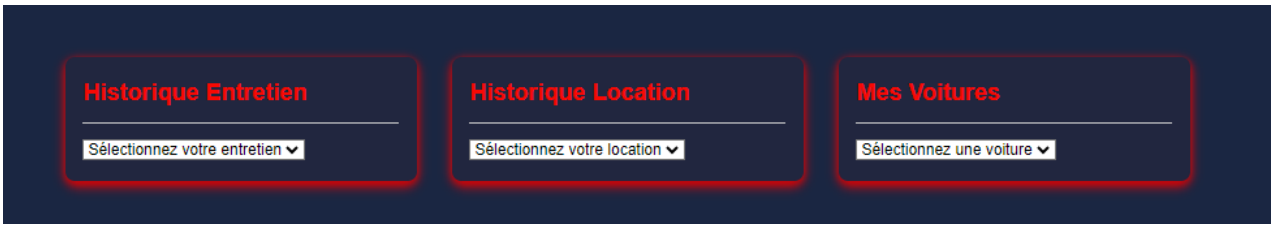


Code correspondant :

```
<div className="modal" style={{display: carModalVisible ? 'flex' : 'none'}}>
    <div className="modal-content">

        <img src={profilcar} alt=""/>
        {selectedCarData && (
            <div className='container'>
                <h3>Informations sur la voiture sélectionnée:</h3>
                <label>
                    Kilométrage:
                    <p>{selectedCarData.kilometrage}</p>
                </label>
```

Choix historique des réservation ou location :



Code correspondant :

```
<div className="inf-card">
    <div className="inf-card-header">
        <h2>Historique Location</h2>
    </div>
    <div className="inf-card-content">
        <select value={selectedLocationBox} onChange={handleLocationBoxChange}>
            <option value="">Sélectionnez votre location</option>
            {user?.locationBoxes.map((box, index) => (
                <option key={index} value={box.startDate}>
                    {box.startDate}
                </option>
            ))}
        </select>
    </div>
</div>
```

12.3. CODE DE COMPOSANTS METIER LES PLUS SIGNIFICATIFS

Docker File :

```
# First stage: Build the application
FROM maven:eclipse-temurin:21-jdk-alpine as build
WORKDIR /app
COPY pom.xml .
COPY src ./src
RUN mvn clean package -Dmaven.test.skip=true

# Second stage: Dockerize
FROM eclipse-temurin:21.0.2_13-jdk-jammy
WORKDIR /app
COPY --from=build /app/target/mnsgarage-0.0.1-SNAPSHOT.jar ./mnsgarage.jar
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "mnsgarage.jar"]
```

Contenu compose.yaml pour créer les images base des données et mettre dans le containeur isolé . Le nom de l'image utilisée (récupérée par défaut sur hub.docker.com).

```yaml
version: '3'
services:
  db:
    image: mysql:latest
    container_name: mnsdb
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: apirest

    ports:
      - "3308:3306"
    volumes:
      - dbdata:/var/lib/mysql

  phpmyadmin:
    image: phpmyadmin/phpmyadmin
    container_name: mnsphpmyadmin
    environment:
      PMA_HOST: db
      PMA_PORT: 3306
    restart: always
    ports:
      - "8181:80"
    depends_on:
      - db
```

Les images back-end et front-end dans le containeur :

```yaml
  app:
    image: mnsgarage
    container_name: mnsgarage_backend
    build:
      context: .
      dockerfile: Dockerfile
    depends_on:
      - db
    environment:
      SPRING_DATASOURCE_URL: jdbc:mysql://db:3306/apirest
      SPRING_DATASOURCE_USERNAME: root
      SPRING_DATASOURCE_PASSWORD: root
    ports:
      - "8080:8080"

  frontend:
    image: mnsgarage-frontend:tag
    container_name: mnsgarage_frontend
    build:
      context: ./path/to/frontend
      dockerfile: Dockerfile
    depends_on:
      - app
    ports:
      - "3000:80"
volumes:
  dbdata:
```

12.4. CODE DE COMPOSANTS D'ACCES AUX DONNEES LES PLUSSIGNIFICATIFS

Dans le cadre du projet de garage associatif, j'ai intégré les dépendances MySQL Connector dans Spring Boot afin de rendre possible la connexion avec la base de données. Voici les étapes que j'ai suivies :

1. Ajouter les dépendances MySQL dans le fichier `pom.xml`

```xml
        <dependency>
            <groupId>com.mysql</groupId>
            <artifactId>mysql-connector-j</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
```

Pour commencer, j'ai ajouté les dépendances nécessaires dans le fichier `pom.xml` de mon projet Spring Boot. Cela inclut la dépendance MySQL Connector et le starter JPA de Spring Boot pour l'ORM (Object-Relationnel Mapping).

2. Configurer le fichier `application. properties`

```
spring.datasource.url=jdbc:mysql://localhost:3306/apirest?serverTimezone=UTC&createDatabaseIfNotExist=true
spring.datasource.username=root
spring.datasource.password=root

server.servlet.context-path=/api

spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
```

Ensuite, j'ai configuré les paramètres de connexion à la base de données dans le fichier `application. properties`, qui se trouve dans le répertoire `src/main/resources`. J'ai spécifié l'URL de la base de données, le nom d'utilisateur, le mot de passe, ainsi que d'autres propriétés liées à Hibernate et à JPA.

3. Créer un modèle (Entity)

```java
@Getter  10 usages  ogtayaliyev
@Setter
@AllArgsConstructor
@NoArgsConstructor
@Entity
public class Voiture_model {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id", nullable = false)
    private Long id;
    private String model_nom;
    private Date mise_circulation;
    private String moteure_type;
    private int moteure_puissance;
    private String carburant;
    @ManyToOne
    @JoinColumn(name = "marque_id", nullable = false)
    private Voiture_Marque marque;
}
```

Après avoir configuré la connexion à la base de données, j'ai créé un modèle pour gérer les données. Une entité est une classe Java annotée qui est mappée à une table de la base de données. Cette classe définit les champs représentant les colonnes de la table, ainsi que les annotations JPA pour spécifier les relations et les comportements de la base de données.

4. Créer un Repository

```
public interface UtilisateurRepository extends CrudRepository<Utilisateur, Integer> {  15 usa

    // Method to find a user by email.
    Optional<Utilisateur> findByEmail(String email);  3 usages  ogtayaliyev

    // Method to find a user by ID.
    Optional<Utilisateur> findById(Long id);  1 usage  ogtayaliyev

    // Method to delete a user by ID.
    void deleteById(Long id);  no usages  ogtayaliyev
    boolean existsByEmail(String email);  1 usage  ogtayaliyev
}
```

Pour interagir avec la base de données, j'ai également créé un repository. Un repository est une interface qui étend `JpaRepository` ou `CrudRepository`, et qui fournit des méthodes CRUD (Create, Read, Update, Delete) prêtes à l'emploi.

12.5. CODE D'AUTRES COMPOSANTS (CONTROLEURS, UTILITAIRES...).

Email contrôleur :

```
@CrossOrigin  ogtayaliyev
@RestController
public class EmailController {

    @Autowired
    private JavaMailSender emailSender;

    @PostMapping("send-email")  ogtayaliyev
    public void sendEmail(@RequestBody EmailRequest request) {
        SimpleMailMessage message = new SimpleMailMessage();
        message.setFrom(request.getEmail());
        message.setTo("ogtayaliyev9@gmail.com");
        message.setSubject(request.getSubject());
        message.setText(request.getMessage());

        emailSender.send(message);
    }
}
```

Contrôleur pour chercher voiture par son plaque immatriculation :

```
@RestController  ogtayaliyev *
public class LicensePlateController {
    private final LicensePlateRecognitionService licensePlateRecognitionService;  2 usages

    @Autowired  ogtayaliyev
    public LicensePlateController(LicensePlateRecognitionService licensePlateRecognitionService) {
        this.licensePlateRecognitionService = licensePlateRecognitionService;
    }

    @GetMapping("recognize/{plateNumber}")  ogtayaliyev *
    public ResponseEntity<Voitures> recognizeVehicleByLicensePlate(@PathVariable("plateNumber")
                                                                   String plaqueImmatriculation) {
        Voitures recognizedVehicle = licensePlateRecognitionService.
                recognizeVehicleByLicensePlate(plaqueImmatriculation);
        if (recognizedVehicle != null) {
            return ResponseEntity.ok(recognizedVehicle);
        } else {
            return ResponseEntity.notFound().build();
        }
    }
```