

Dossier technique

Aperture COntrOl Management Engineering

Fournisseur de solutions innovantes & innatendues

SOMMAIRE

I. Mise en contexte	3
Introduction	3
Expression des besoins.....	3
II. Spécification Fonctionnelle	4
Scénario	Erreur ! Signet non défini.
Fonctionnalité.....	5
Fonctionnalités principales	5
Fonctionnalités supplémentaires	6
Maquettes	7
Diagrammes de conception	8
Modèle Conceptuel des Données (MCD)	8
Modèle Logique des Données (MLD).....	9
Modèle Physique des Données (MPD).....	10
Diagramme d'activités.....	11
Diagramme de cas d'utilisation	12
III. Réalisation technique.....	14
Mise en place projet	14
Outils utilisés	14
Architecture monolithe.....	14
Visualisation de la mise en œuvre du CRM	15
Coté développement	16
Méthode afficherPasserCommandeClient dans le CommandeController	16
Vue " passer_commande "	17
Méthode passerCommandeClient dans le CommandeController	18
Vue " detail_commande "	21
Vue des entrées en base de données.....	21
IV. Gestion de projet	22
Organisation générale	22
Trello.....	22
Bilan du projet	24
Annexes.....	25
Bibliographie.....	25

I. Mise en contexte

Introduction

Ce dossier technique a pour objectif de détailler la réalisation de l'application CRM (Customer Relationship Management) demandé par l'entreprise ACME. Le dossier technique sera conclu par des spécifications liés à la gestion du projet non seulement de création du CRM mais également à la demande de l'entreprise concernant l'évolution de son réseau détaillé au sein du cahier des charges livré avec le dossier technique.

Expression des besoins

La réalisation du CRM répond à un besoin de gestion commercial de l'entreprise. Cette solution sera utilisée par l'ensemble des employés via leur poste de travail depuis le réseau local interne de l'entreprise mais également par les commerciaux, pouvant y accéder en mobilité. Les données de l'application seront centralisées sur une unique base de données, hébergée sur un serveur présent physiquement dans les locaux.

L'application CRM à développer gèrera les données clients, les produits, l'historique des commandes et les statistiques commerciales. Les fonctionnalités incluront la gestion des clients et des produits (liste, ajout, modification, suppression), la gestion des commandes (liste, ajout) et l'affichage des statistiques. Les spécifications techniques incluent l'utilisation de Java pour le langage de programmation, MySQL pour la base de données et une interface graphique web. L'application exigera une authentification avec un login et un mot de passe unique pour chaque employé de l'entreprise.

II. Spécification Fonctionnelle

Scénario : Utilisation du CRM par les différents utilisateurs

a) Connexion à l'application :

- L'utilisateur accède à l'application CRM en ouvrant son application ou son navigateur web et en saisissant l'URL fournie par l'entreprise ACME.
- À la page d'accueil, l'utilisateur est invité à saisir son identifiant (login) et son mot de passe.
- Après vérification des informations d'identification, l'utilisateur est redirigé vers menu principal de l'application.

b) Navigation dans l'application :

- Une fois connecté, l'utilisateur peut naviguer à travers les différentes fonctionnalités de l'application en fonction de son rôle attribué dans l'entreprise.

c) Actions disponibles pour l'administrateur :

- L'administrateur a un accès complet à toutes les fonctionnalités de l'application.
- Il peut visualiser, ajouter, modifier, gérer l'archivage des produits, des clients, des commandes et des employés.
- Il peut également accéder aux statistiques commerciales pour analyser les performances de l'entreprise.

d) Actions disponibles pour le responsable :

- Le responsable dispose de privilèges similaires à l'administrateur mais avec quelques restrictions sur la partie employés : il ne peut que voir la liste des employés.

e) Actions disponibles pour l'employé :

- L'employé dispose d'un accès limité par rapport à l'administrateur et au responsable.
- Il peut visualiser, ajouter, modifier des produits et des clients.
- Cependant, l'employé ne peut pas effectuer d'actions sur le désarchivage des clients et produits sauf pour les visualiser.
- De plus, l'employé n'a pas accès aux statistiques commerciales.

f) Archivage des données :

- Un élément est archivé au lieu d'être définitivement supprimé de la base de données.
- Cela permet une récupération ultérieure des données archivées si nécessaire.
- Seuls les utilisateurs autorisés peuvent accéder aux données archivées et effectuer des actions sur celles-ci.

Ce scénario met en lumière les différentes actions que peuvent effectuer les utilisateurs de l'application CRM en fonction de leurs rôles attribués dans l'entreprise ACME. Il détaille également le processus d'archivage des données pour assurer la sécurité et la récupération des données supprimées si nécessaire.

Fonctionnalités

Fonctionnalités principales

Le CRM conçu par nos équipes a été réalisé selon un cahier des charges avec des fonctionnalités précises demandées par l'entreprise. Ces fonctionnalités détaillées précédemment dans les besoins sont ici explicitées sous forme de tableau rappelant non seulement les différentes actions possibles offertes par notre application mais également la distinction entre les rôles que nous avons créée.

Administrateur	Voir	Ajouter	Modifier	Gestion Archivage
Produit	✓	✓	✓	✓
Client	✓	✓	✓	✓
Commande	✓	✓	✓	
Employé	✓	✓	✓	✓
Produit archivé	✓	✓	✓	✓
Client archivé	✓	✓	✓	✓
Employé archivé	✓	✓	✓	✓
Statistiques	✓			

Responsable	Voir	Ajouter	Modifier	Gestion Archivage
Produit	✓	✓	✓	✓
Client	✓	✓	✓	✓
Commande	✓	✓	✓	
Employé	✓	X	X	X
Produit archivé	✓	✓	✓	✓
Client archivé	✓	✓	✓	✓
Employé archivé	X	X	X	X
Statistiques	✓			

Employé	Voir	Ajouter	Modifier	Gestion Archivage
Produit	✓	✓	✓	✓
Client	✓	✓	✓	X
Commande	✓	✓	X	
Employé	X	X	X	X
Produit archivé	✓	X	X	X
Client archivé	✓	X	X	X
Employé archivé	X	X	X	X
Statistiques	X			

Figure 1 : tableaux des fonctionnalités des salariés d'ACME

Comme exigé dans le cahier des charges, nous avons également réalisé une authentification avec un login et un mot de passe unique pour chaque employé.

La fonctionnalité de suppression a été gérée de manière à supprimer visuellement pour l'utilisateur un produit, une commande, un employé ou un client. Dans les faits cela se passe par l'archivage qui, pour des raisons de sécurité, nous paraissait plus conforme avec la commande. De plus, une possibilité de suppression a été implémentée directement dans la base de données via un accès en direct par un compte root permettant selon les demandes des clients d'être supprimé définitivement de la base de données.

Fonctionnalités supplémentaires

Lors de la conception en amont de la réalisation il nous a semblé pertinent de déployer des fonctionnalités supplémentaires par rapport à celles faisant partie du strict minimum demandé :

Tout d'abord, nous avons souhaité pouvoir ajouter des images aux produits permettant ainsi aux équipes de ventes de mieux visualiser le produit mis en vente mais également servant aux équipes de ventes permettant de montrer le produit qu'ils allaient vendre dans le catalogue aux clients. Il est donc possible désormais non seulement de voir une image mais également d'en ajouter et de la modifier.

Il nous a semblé dommageable de ne pas pouvoir gérer les employés directement au sein du CRM. En effet, non seulement nous facilitait la tâche pour la visualisation des statistiques et l'attribution des rôles mais cela fait également sens dans une logique métier ou un responsable peut voir l'ensemble des salariés dans le CRM.

Ainsi, avec cette gestion des employés nous avons également pu déployer un accès selon les rôles des salariés en les distinguant en trois catégories comme mentionné plus haut : Administrateur applicatif, Responsable, Employé. Ces différents rôles en plus d'avoir des logins et des mots de passes différents seront directement limités selon leur rôle dans l'application par la présence ou non de certains boutons permettant d'accéder à des fonctionnalités critiques.

Dans les statistiques, nous avons mis en place une visualisation des commandes par statuts pour que les responsables puissent notamment voir en temps réel l'avancée des commandes qu'elles soient en cours de préparation, en cours d'acheminement ou livrées.

Un numéro de commande sous la forme 2024XXXXXX s'incrémentant de manière automatique nous a également semblé utile afin que les employés d'ACME puissent s'y retrouver au niveau des commandes.

Maquettes

Les maquettes suivantes ont été réalisées en amont du projet. Elles nous ont permis de cadrer le développement en particulier de la partie Front. Des modifications ont été apportées afin de se rapprocher au mieux de nos attentes tant au niveau des fonctionnalités mais également de la forme que nous souhaitons lui donner.

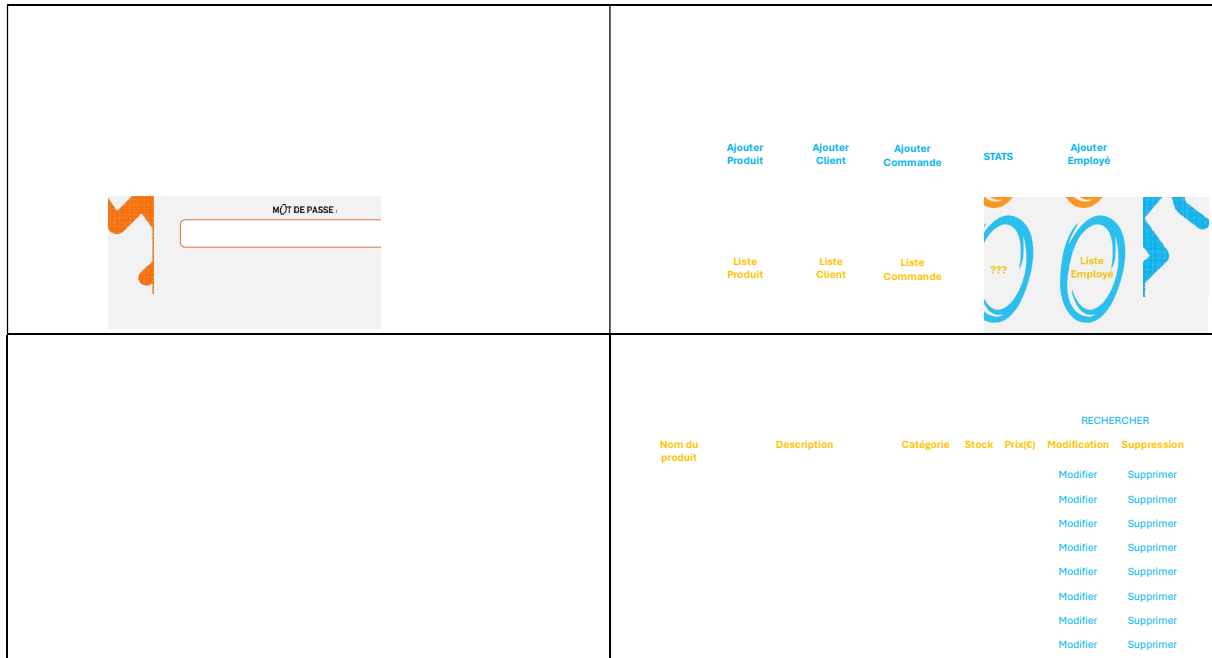


Figure 4 : maquettes du CRM

Diagrammes de conception

Les diagrammes de conception créés dès le début du projet ont permis de comprendre la direction que nous souhaitons donner au projet ainsi que les différents liens qu'il existait entre les fonctionnalités d'un CRM et la base de données.

Les diagrammes présentés ci-dessous sont réalisés en Merise et en UML et ont dû évoluer en marge de la conception pour correspondre à nos besoins.

Modèle Conceptuel des Données (MCD)

Le MCD vient représenter la structure des données en décrivant les entités, leurs attributs et les relations qui les lient. Cela permet ainsi de mieux comprendre les besoins métier et de concevoir une base de données adaptée.

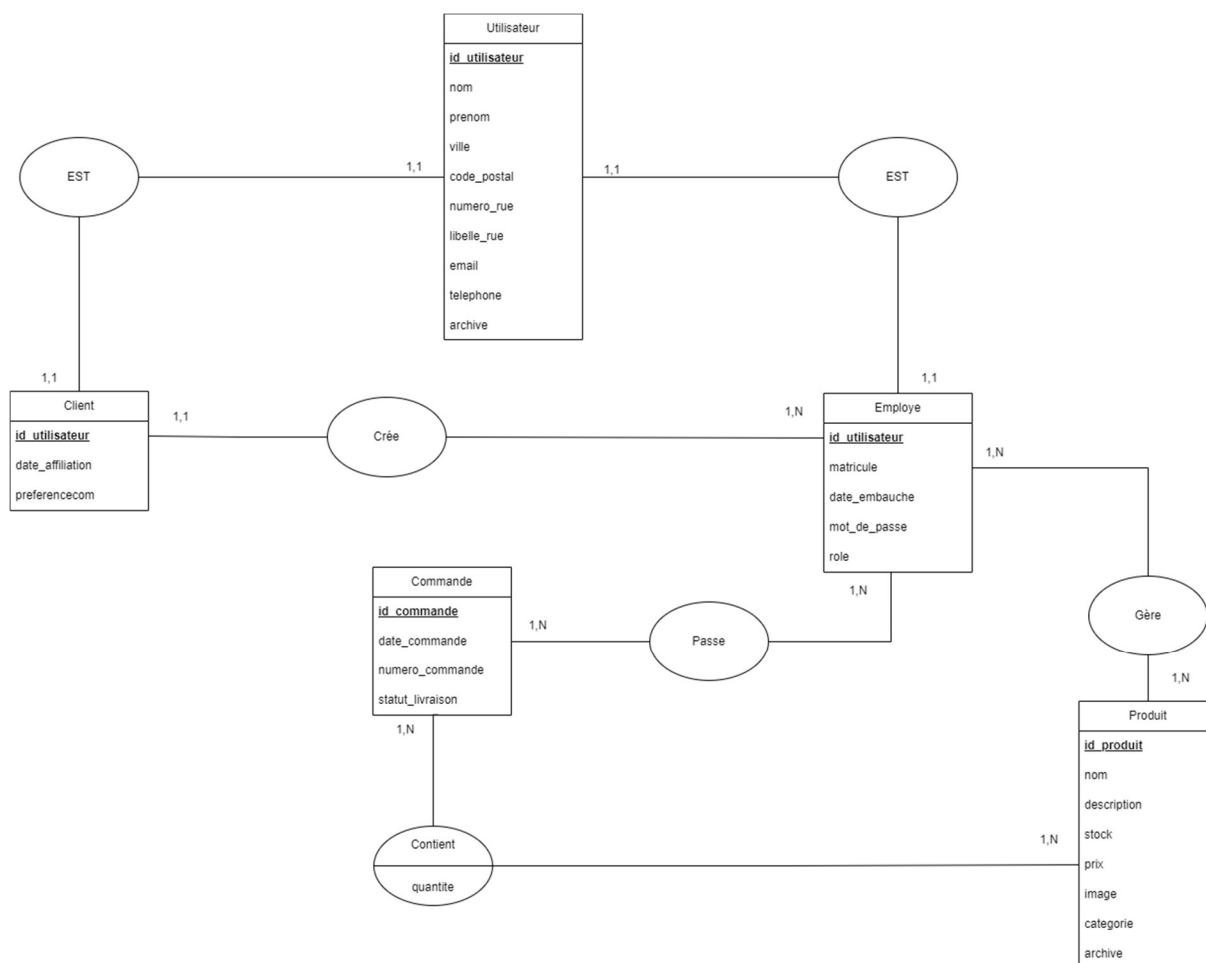


Figure 5 : modèle conceptuel des données (Merise)

Modèle Logique des Données (MLD)

Le MLD vient représenter de manière plus détaillée et structurée la base de données. Les concepts abstraits du MCD sont transformés ici en structures de données concrètes telles que des tables, des colonnes et des clés, fournissant ainsi une vue logique des données.

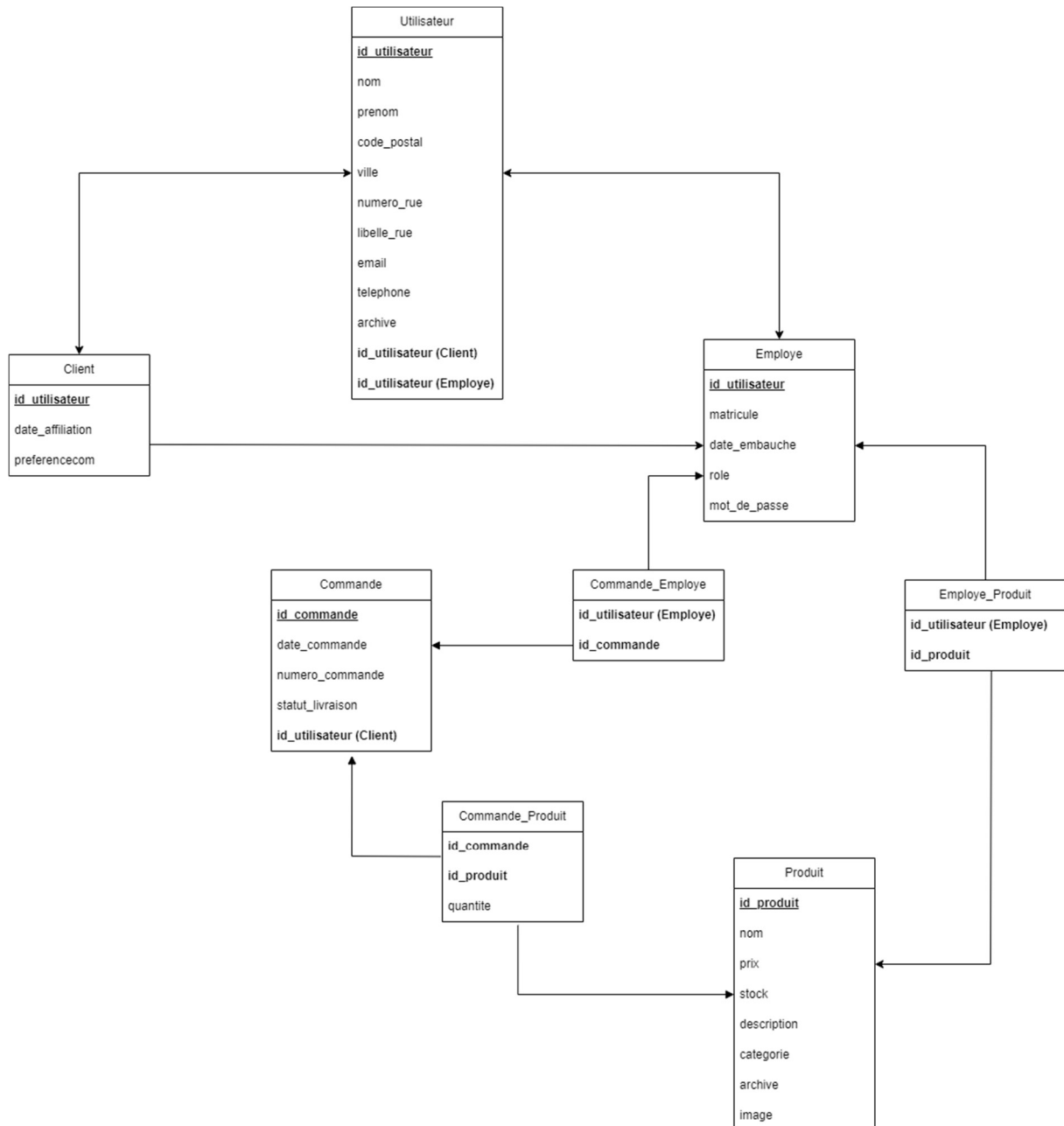


Figure 6 : modèle logique des données (Merise)

Modèle Physique des Données (MPD)

Le MPD vient finaliser ce triptyque de schéma en Merise en définissant les détails techniques spécifiques tels que les types de données, les contraintes d'intégrité et d'autres caractéristiques de mise en œuvre. Il représente ainsi une version concrète et opérationnelle de la base de données.

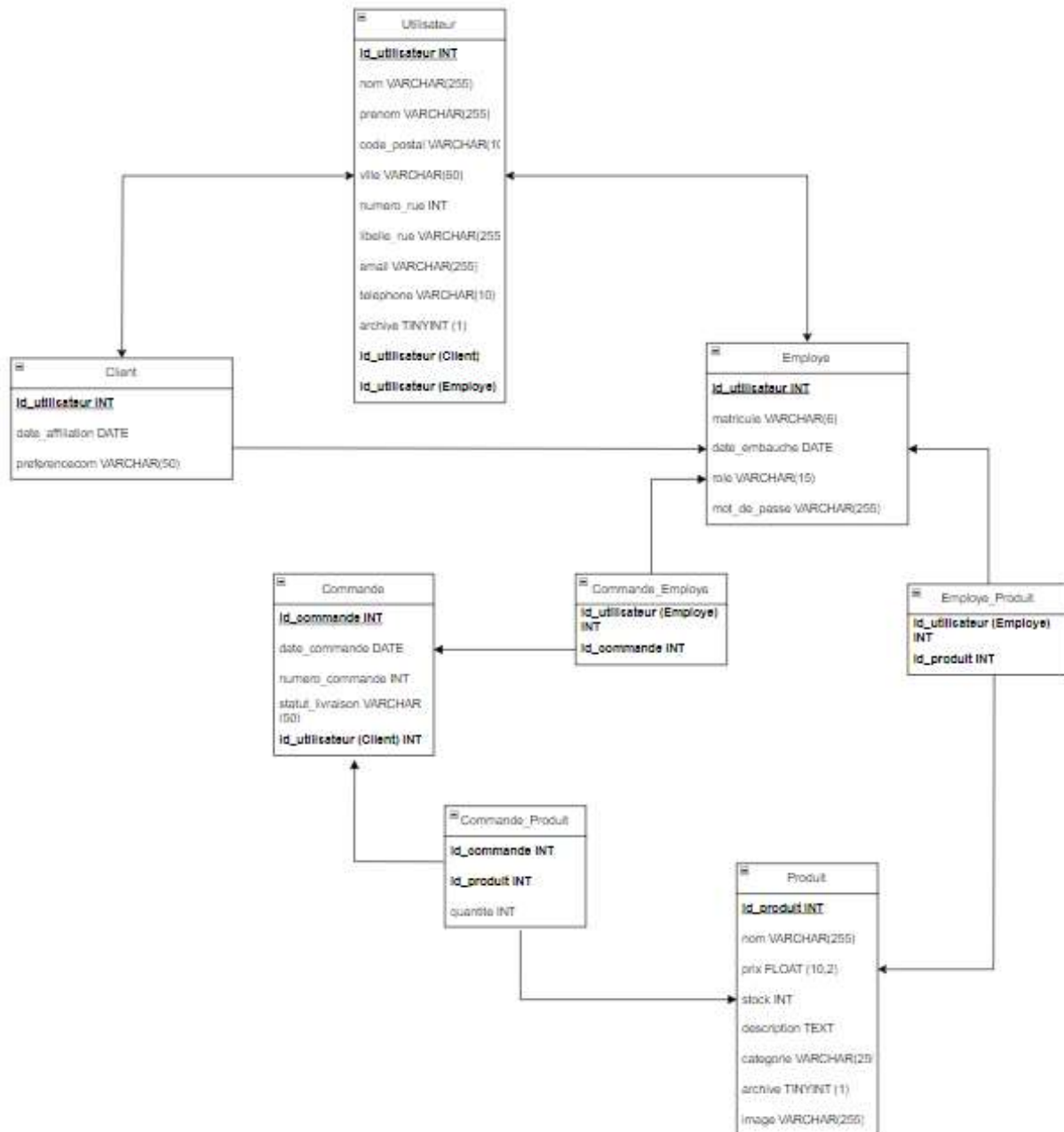


Figure 7 : modèle physique des données (Merise)

Diagramme d'activités

Ce diagramme d'activités est utilisé pour modéliser le flux de comportement d'un système, en mettant en évidence les différentes activités et les transitions entre elles. Il sert à décrire les processus métier vis-à-vis d'une de nos fonctionnalités (ici « Passer commande »).

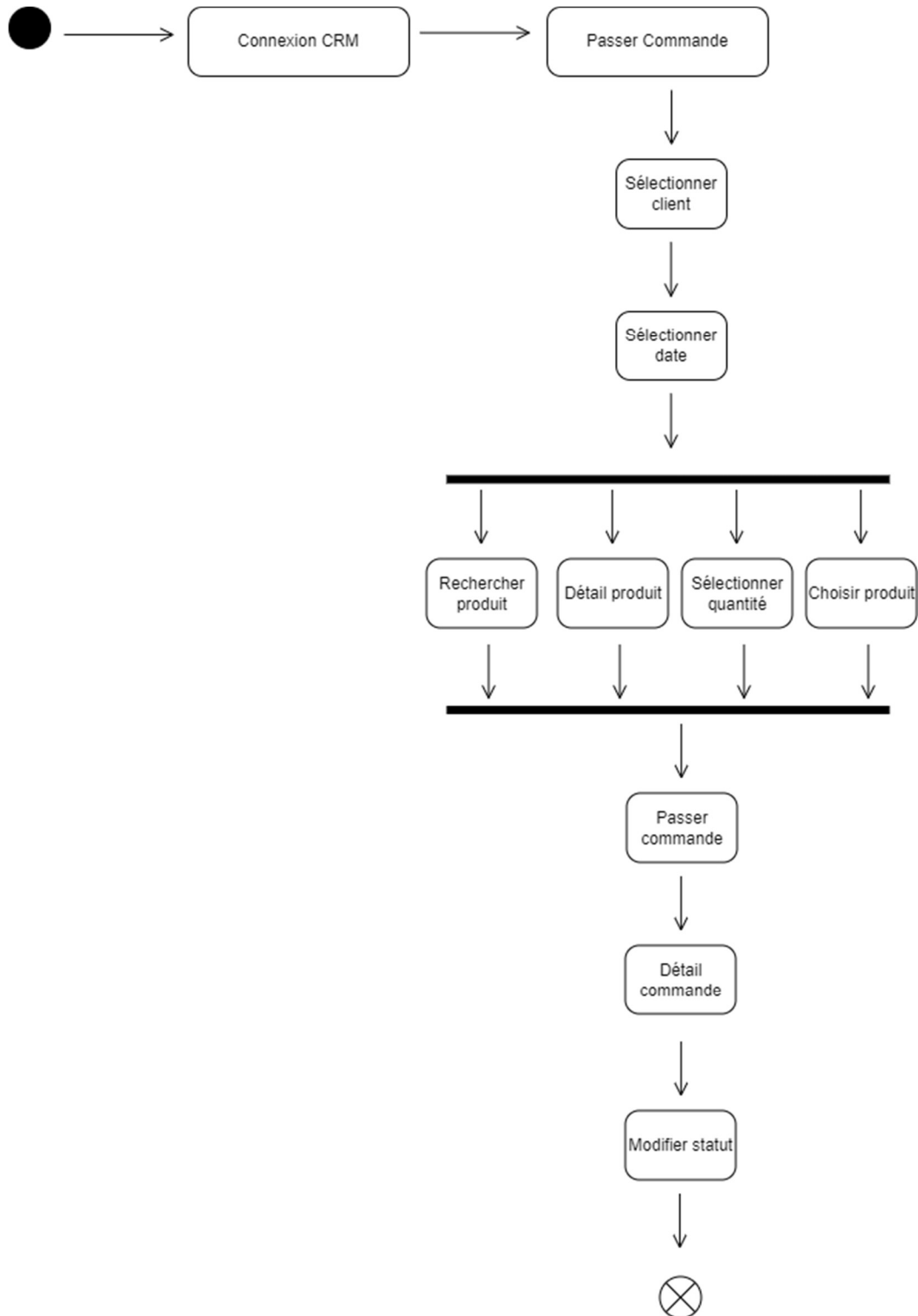


Figure 8 : diagramme d'activités (UML)

Diagramme de cas d'utilisation

Le diagramme de cas d'utilisation en UML (Unified Modeling Language) est ici utilisé pour représenter les interactions entre les acteurs de notre CRM (Administrateurs, Responsables, Employés) et notre logiciel, en mettant en évidence les différentes fonctionnalités offertes par l'application.

Ce premier schéma présente les fonctionnalités de bases.

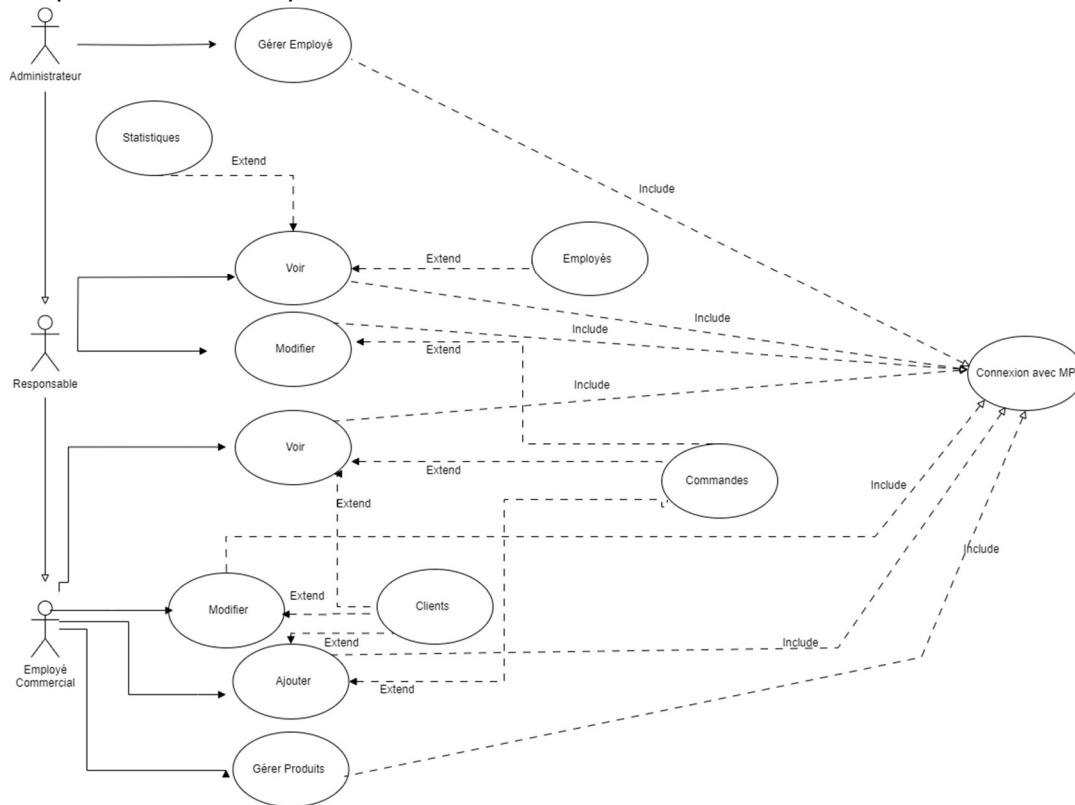


Figure 9 : diagramme de cas d'utilisations – fonctionnalités principales (UML)

Ce second schéma présente les fonctionnalités liées à l'archivage.

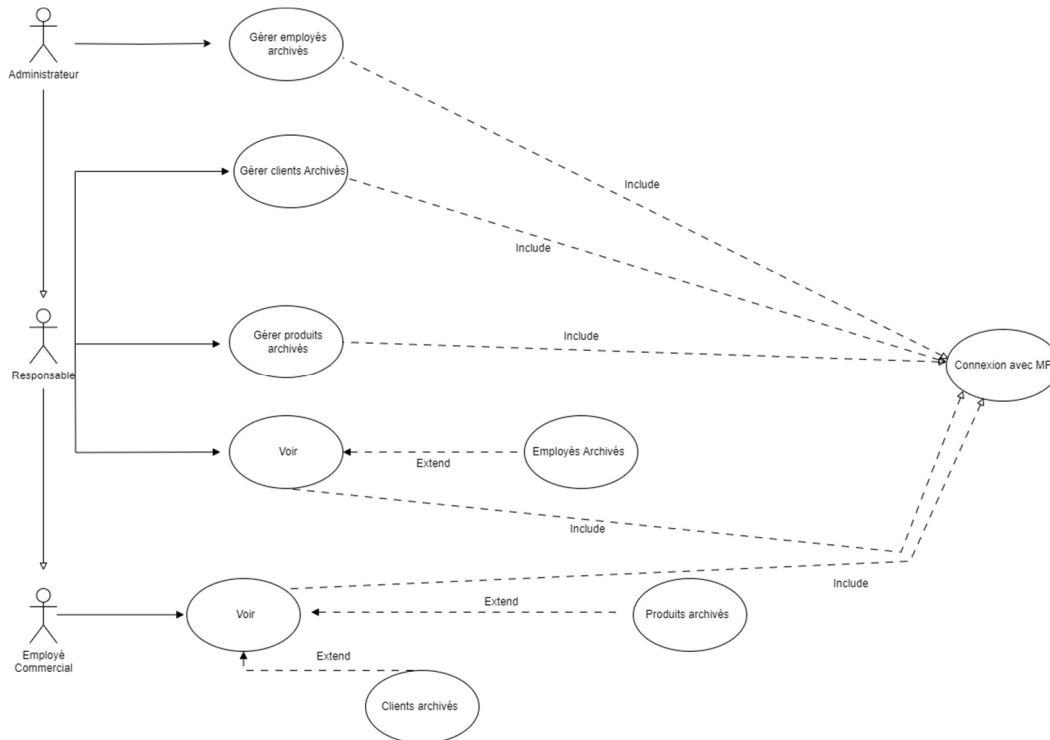


Figure 9 : diagramme de cas d'utilisations – fonctionnalités d'archivage (UML)

III. Réalisation technique

Mise en place projet

Outils utilisés

La création du projet s'appuie sur trois blocs distincts. Une partie Base de Données, une partie Back et une partie Front. Ces différentes parties ont été réalisées avec des technologies et des outils différents.

Tout d'abord, la base de données a été réalisée sous une version de MySQL 8.0.36.

Le côté Back a utilisé de nombreux outils tous inhérents au Framework Java « Spring ». En effet, bien qu'ayant utilisé Java 21 il a également été convenu lors de nos différents enseignements que l'utilisation d'un Framework comme Spring était plus que recommandé afin de pouvoir utiliser son pattern MVC. L'outil Spring Boot a permis de tout initialiser afin de paramétrer et de mettre en place le reste du développement du CRM. Spring Data JPA a notamment permis de simplifier la relation avec la base de données en facilitant le travail des requêtes SQL. Spring Security est un outil, qui comme son nom l'indique, gère les questions de sécurité au niveau applicatif et a donc été utilisé pour les questions d'authentification et d'invisibilisation des fonctionnalités non nécessaires pour certains employés.

Spring Boot DevTools est un outil ayant facilité le travail de création de l'application en elle-même.

Enfin, le Front a été réalisé via trois langages. L'HTML qui est un langage de balisage utilisé par le biais notamment de Thymeleaf permettant d'inclure des éléments de langage java et de simplifier le développement. Pour le style nous avons utilisé le Framework CSS (Tailwind) qui a permis la mise en forme de l'application web. Le JavaScript a quant à lui permis de réaliser les statistiques, la recherche dynamique et la confirmation du mot de passe dans la partie Front.

Architecture monolithe

L'architecture de notre CRM est monolithique, c'est-à-dire que toutes les composantes et les fonctionnalités de l'application sont regroupées au sein d'une seule et même unité de déploiement. Dans ce type d'architecture, le code source, la logique métier, l'interface utilisateur et la gestion des données sont généralement intégrés dans un seul programme ou dans une seule application.

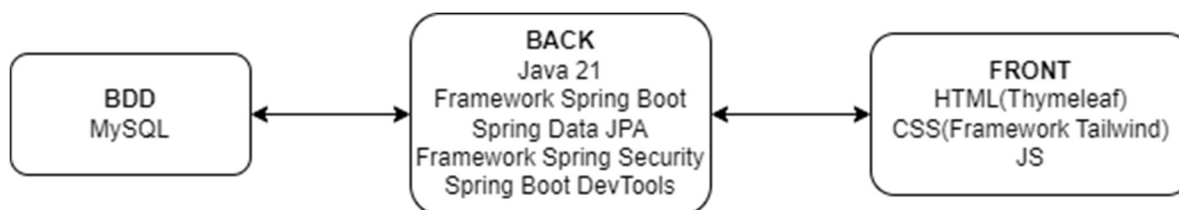


Figure 10 : schéma de l'architecture du CRM

Visualisation de la mise en œuvre du CRM

Il nous a semblé important d'avoir une visualisation de la mise en place du CRM avec une vue qui explique comment s'organise le lien entre la base de données, le côté applicatif et les utilisateurs.

Pour rappel, seuls trois rôles ont été créés. L'accès en direct à la Base de Données ne sera fait qu'en direct via un compte root.

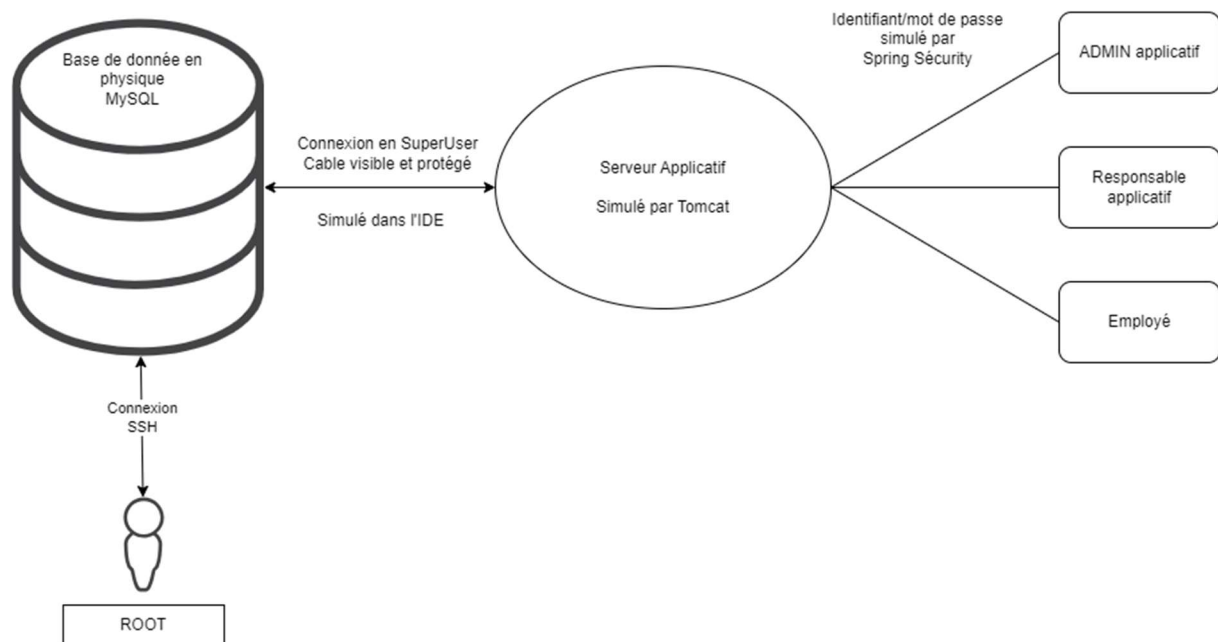


Figure 11 : schéma de la mise en place du CRM

Coté développement

Nous allons désormais aborder la partie développement du CRM. Celui-ci se distingue par plusieurs éléments qui vont être abordés en explication à ce code. Les classes **Controller** interagissent directement avec la partie Frontend, gérant les requêtes HTTP et les réponses. Elles peuvent interagir avec les classes **Services**, qui contiennent la logique métier de l'application, décrivant les actions que les objets peuvent réaliser. Les Services sont ainsi en relation directe avec les classes de **Domaine**, représentant les entités de l'application telles que les produits ou les employés. Enfin, les **Repository** s'occupent de la relation avec la Base de Données, fournissant une interface pour interagir avec la base de données.

Méthode afficherPasserCommandeClient dans le CommandeController

Cette méthode de type GET, nommée afficherPasserCommandeClient, est conçue pour fournir les données nécessaires à la vue "passer-commande", accessible via l'URL (/passer-commande).

Fonctionnalités :

- Récupération de la liste des clients et des produits non archivés.
- Récupération de la liste des statuts de livraison.
- Création d'une nouvelle instance de l'entité Commande.
- Récupération du matricule de l'employé connecté.

Objectif : Ces données sont ensuite transmises à la vue "passer_commande".

```
@GetMapping("/passer-commande")
public String afficherPasserCommandeClient(Model model){

    List<Client> clients = clientService.getClientsParArchive( actif: false);
    List<Produit> listProduits = produitService.getProduitsParArchive( actif: false);
    List<StatutLivraison> statutLivraisons = Arrays.asList(StatutLivraison.values());
    Commande commande = new Commande();
    Authentication authentication = SecurityContextHolder.getContext().getAuthentication();

    UserDetails userDetails = (UserDetails) authentication.getPrincipal();
    String username = userDetails.getUsername();
    Employee employeConnecte = employeService.findByMatricule(username);

    model.addAttribute( attributeName: "clients", clients);
    model.addAttribute( attributeName: "statutLivraisons", statutLivraisons);
    model.addAttribute( attributeName: "commande", commande);
    model.addAttribute( attributeName: "listProduits", listProduits);
    model.addAttribute( attributeName: "employeConnecte", employeConnecte);
    model.addAttribute( attributeName: "pageTitle", attributeValue: "Passer commande");
    return "passer_commande";
}
```

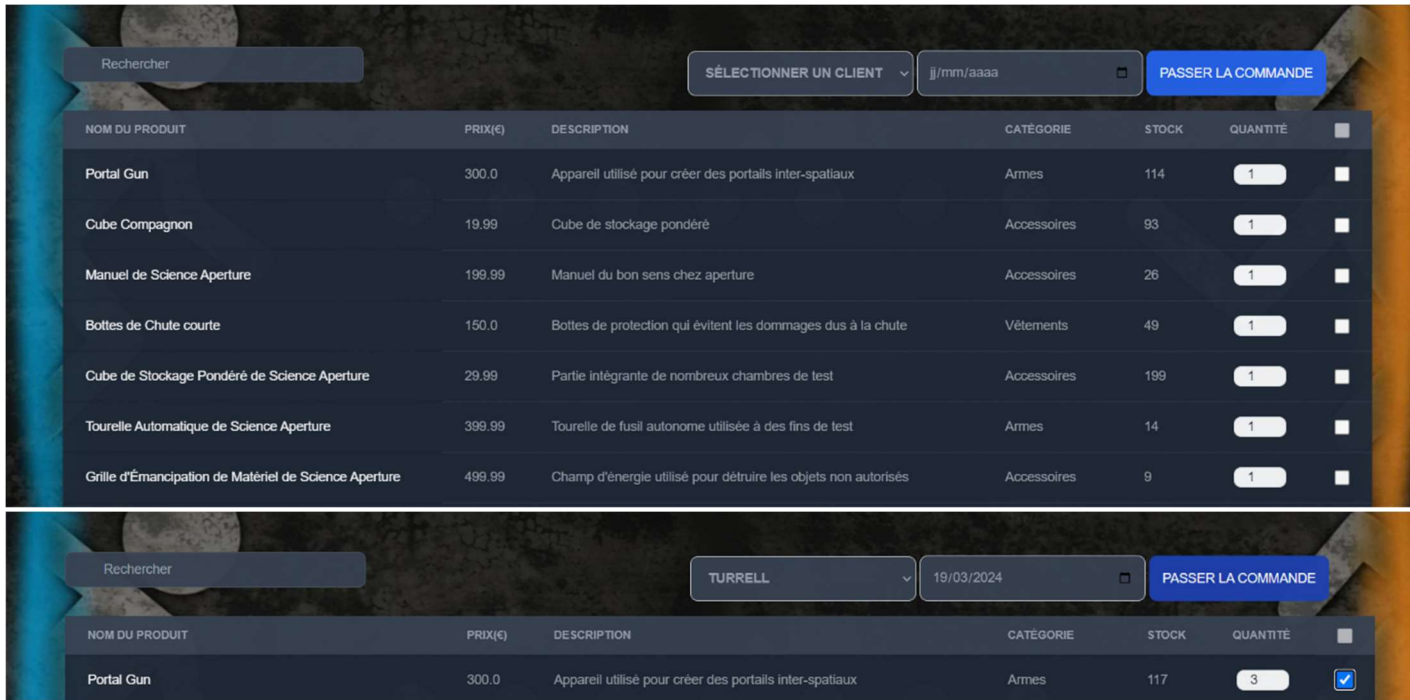



Vue " passer commande "

Formulaire de création d'une commande, afin de pouvoir choisir un client, une date, des produits dans une liste et choisir leur quantité. Une fois que les choix ont été fait, nous pouvons valider la commande.

Fonctionnalités :

- Choisir dans la liste des clients et des produits non archivés.
- Choisir une quantité de produit.
- Valider la commande.



The screenshot shows a web application interface for creating an order. It features a search bar, a client selection dropdown, a date picker, and a 'PASSER LA COMMANDE' button. Below these is a table of products with columns for name, price, description, category, stock, and quantity. The quantity is adjustable via a slider and a checkbox.

NOM DU PRODUIT	PRIX(€)	DESCRIPTION	CATÉGORIE	STOCK	QUANTITÉ	
Portal Gun	300.0	Appareil utilisé pour créer des portails inter-spatiaux	Armes	114	1	<input type="checkbox"/>
Cube Compagnon	19.99	Cube de stockage pondéré	Accessoires	93	1	<input type="checkbox"/>
Manuel de Science Aperture	199.99	Manuel du bon sens chez aperture	Accessoires	26	1	<input type="checkbox"/>
Bottes de Chute courte	150.0	Bottes de protection qui évitent les dommages dus à la chute	Vêtements	49	1	<input type="checkbox"/>
Cube de Stockage Pondéré de Science Aperture	29.99	Partie intégrante de nombreux chambres de test	Accessoires	199	1	<input type="checkbox"/>
Tourelle Automatique de Science Aperture	399.99	Tourelle de fusil autonome utilisée à des fins de test	Armes	14	1	<input type="checkbox"/>
Grille d'Émancipation de Matériel de Science Aperture	499.99	Champ d'énergie utilisé pour détruire les objets non autorisés	Accessoires	9	1	<input type="checkbox"/>

NOM DU PRODUIT	PRIX(€)	DESCRIPTION	CATÉGORIE	STOCK	QUANTITÉ	
Portal Gun	300.0	Appareil utilisé pour créer des portails inter-spatiaux	Armes	117	3	<input checked="" type="checkbox"/>

Objectif : Ces données sont ensuite transmises à la méthode de type POST `passerCommandeClient` dans le `CommandeController`.



Méthode passerCommandeClient dans le CommandeController

La méthode passerCommandeClient, de type POST, dans le CommandeController, est dédiée à la récupération des données soumises par le formulaire après validation. En utilisant l'annotation @RequestParam(), nous sélectionnons spécifiquement les données à extraire pour les traiter.

Fonctionnalités :

- Extraction des données pertinentes du formulaire.
- Utilisation de conditions if() pour vérifier la présence et la validité de toutes les informations nécessaires.
- Appel de la méthode creerCommande() provenant de notre classe CommandeService pour traiter les données et créer une nouvelle commande.

Objectif : L'objectif principal de cette méthode est de garantir que toutes les informations requises pour créer une commande sont correctes et complètes. En utilisant des conditions if(), nous nous assurons que le processus de commande se déroule sans accroc et que seules les données valides sont traitées.

```
@PostMapping("/passer-commande")
public String passerCommandeClient(@RequestParam("idUtilisateur") int idUtilisateur,
                                   @RequestParam("dateCommande") Date dateCommande,
                                   @RequestParam("produits") List<Integer> produitsIds,
                                   @RequestParam("idEmploye") int idEmploye,
                                   @RequestParam Map<String, String> formData,
                                   RedirectAttributes redirectAttributes) {

    1
    if (!commandeService.validerDateCommande(dateCommande, redirectAttributes)) {
        return "redirect:/passer-commande?erreur=date";
    }

    2
    if (!commandeService.validerIdClient(idUtilisateur, redirectAttributes)) {
        return "redirect:/passer-commande?erreur=clientEr";
    }

    3
    if (!commandeService.validerPanier(produitsIds, formData, redirectAttributes)) {
        return "redirect:/passer-commande?erreur=panier_vide";
    }

    4
    int nouvelleCommandeId = commandeService.creerCommande(idUtilisateur,
                                                            dateCommande, produitsIds, idEmploye, formData);

    return "redirect:/detail_commande/" + nouvelleCommandeId;
}
```

Cette méthode illustre l'intégration harmonieuse entre le contrôleur et le service, suivant les principes de conception MVC (Modèle-Vue-Contrôleur) pour une architecture logicielle robuste et modulaire.



Voici les méthodes de notre classe `CommandeService` :

1. **Validation de la date de la commande** : Cette méthode vérifie si une date a été saisie pour la commande. Si aucune date n'a été saisie, elle renvoie un message d'erreur indiquant que la date est obligatoire.
2. **Validation de l'ajout d'un client à la commande** : Cette méthode vérifie si un client a été sélectionné pour la commande. Si aucun client n'a été choisi, elle renvoie une erreur indiquant qu'un client doit être sélectionné pour pouvoir passer la commande.
3. **Validation du panier** : Cette méthode vérifie le contenu du panier pour s'assurer que la quantité de chaque article sélectionné n'est pas égale à zéro. Si la quantité d'un article est zéro, la méthode renvoie une erreur.

```
public boolean validerDateCommande(Date dateCommande, RedirectAttributes redirectAttributes) {  
    if (dateCommande == null) { 1  
        redirectAttributes.addFlashAttribute(attributeName: "erreurDate", attributeValue: "date");  
        return false;  
    }  
    return true;  
}  
  
public boolean validerIdClient(int idUtilisateur, RedirectAttributes redirectAttributes) {  
    if (idUtilisateur <= 0) { 2  
        redirectAttributes.addFlashAttribute(attributeName: "erreurClient", attributeValue: "clientEr");  
        return false;  
    }  
    return true;  
}  
  
public boolean validerPanier(List<Integer> produitsIds, Map<String, String> formData, RedirectAttributes redirectAttributes) {  
    boolean produitsNonVides = produitsIds.stream()  
        .anyMatch(produitId -> {  
            String quantiteKey = "quantite-" + produitId;  
            int quantite = Integer.parseInt(formData.get(quantiteKey));  
            return quantite > 0;  
        });  
  
    if (!produitsNonVides) {  
        redirectAttributes.addFlashAttribute(attributeName: "erreurPanier", attributeValue: "panier_vide");  
        return false;  
    }  
    return true;  
}
```

Ces méthodes de validation sont essentielles pour garantir l'intégrité des données de commande et assurer une expérience utilisateur cohérente et sans erreur lors du processus de commande. Elles contribuent à maintenir la qualité des données et à éviter les erreurs de saisie qui pourraient compromettre le bon déroulement des transactions.



4. Méthode `creerCommande` : Cette méthode est responsable de la création complète d'une commande en plusieurs étapes :

- Création d'une nouvelle commande en récupérant la date de commande et en lui attribuant un statut de livraison par défaut.
- Récupération du client et de l'employé associés à la commande.
- Sauvegarde de la commande dans la base de données.
- Association de la commande à l'employé et sauvegarde de cette association.
- Traitement des produits de la commande en ajoutant les produits sélectionnés avec leur quantité, tout en vérifiant le stock disponible.
- Création de l'association entre les produits et la commande (`CommandeProduit`).
- Mise à jour du stock des produits en fonction de la quantité choisie.
- Sauvegarde des associations entre les produits et la commande.
- Récupération de l'identifiant de la commande pour l'utiliser lors de la redirection à partir de la méthode `passerCommandeClient`.

Objectif : Ces données sont ensuite transmises à la vue "detail_commande".

```
public int creerCommande(int idUtilisateur, Date dateCommande, List<Integer> produitsIds, int idEmploye, Map<String, String> formData) {  
    Commande commande = new Commande();  
    commande.setDateCommande(dateCommande);  
    commande.setStatutLivraison(StatutLivraison.EN_PREPARATION);  
    Client client = clientService.getClientById(idUtilisateur);  
    commande.setClient(client);  
    Employee employe = employeeService.getEmployeById(idEmploye);  
    Commande savedCommande = saveCommande(commande);  
    CommandeEmploye commandeEmploye = new CommandeEmploye();  
    commandeEmploye.setCommande(savedCommande);  
    commandeEmploye.setEmploye(employe);  
    commandeEmployeService.saveCommandeEmploye(commandeEmploye);  
    int nouvelleCommandeId;  
    for (Integer produitId : produitsIds) {  
        String quantiteKey = "quantite-" + produitId;  
        int quantite = Integer.parseInt(formData.get(quantiteKey));  
        Produit produit = produitService.getProduitById(produitId);  
        if (quantite > produit.getStock()) {  
            quantite = produit.getStock();  
        }  
        CommandeProduit commandeProduit = new CommandeProduit();  
        commandeProduit.setProduit(produit);  
        commandeProduit.setCommande(savedCommande);  
        commandeProduit.setQuantite(quantite);  
        produit.setStock(produit.getStock() - quantite);  
        produitService.saveProduit(produit);  
        commandeProduitRepository.save(commandeProduit);  
    }  
    nouvelleCommandeId = savedCommande.getIdCommande();  
    return nouvelleCommandeId;  
}
```


Cette méthode assure une création précise et complète de la commande, tout en maintenant l'intégrité des données et en mettant à jour les stocks de produits de manière appropriée. Elle garantit également une transition fluide vers la visualisation des détails de la commande une fois celle-ci validée.



Vue "detail_commande "

Une fois la commande validée, l'utilisateur est redirigé vers le détail de la commande, où toutes les informations pertinentes de la commande sont affichées.

Statut de livraison :

EN_PREPARATION 

VALIDER

Détail de la commande











DATE	NUMÉRO DE COMMANDE	PASSÉE PAR	CLIENT	PRIX TOTAL(€)
2024-03-19	2024000045	test	Turrell	900.0

NOM DU PRODUIT	CATÉGORIE	QUANTITÉ	PRIX(€)
Portal Gun	Armes	3	300.0









Vue des entrées en base de données

Les id sont autogénérés par notre programme.

Le numéro de commande est une méthode du service commande qui incrémentera de 1 si non existant le numéro de commande sous le format 2024XXXXXX.

	 id_commande		 date_commande		 numero_commande		 statut_livraison		 id_utilisateur	
1	45		2024-03-19		2024000045		EN_PREPARATION			7

	 id_commande_employe		 id_commande		 id_utilisateur	
1			45		45	1

	 id_commande_produit		 id_produit		 id_commande		 quantite	
1			45		1		45	3

IV. Gestion de projet

Organisation générale

Le projet a débuté par une première réunion globale permettant de mettre à plat les attentes de chacun, les compétences et les connaissances sur la consigne (compréhension d'un CRM, ...).

Nous avons ainsi pu débiter à travailler sur les quelques aspects vus précédemment en cours, c'est-à-dire la modélisation et la base de données. Tout au long du projet nous avons pu réaliser des réunions hebdomadaires pour rappeler les points d'avancement de chacun. Des réunions mensuelles sont venues compléter ces réunions afin de faire un point d'étape et de compréhension des nouveaux enjeux selon les différents cours que l'on avait au fur et à mesure.

Nous nous sommes servis des outils mis à notre disposition afin de mettre en place une méthode de projet au plus près de l'agilité.

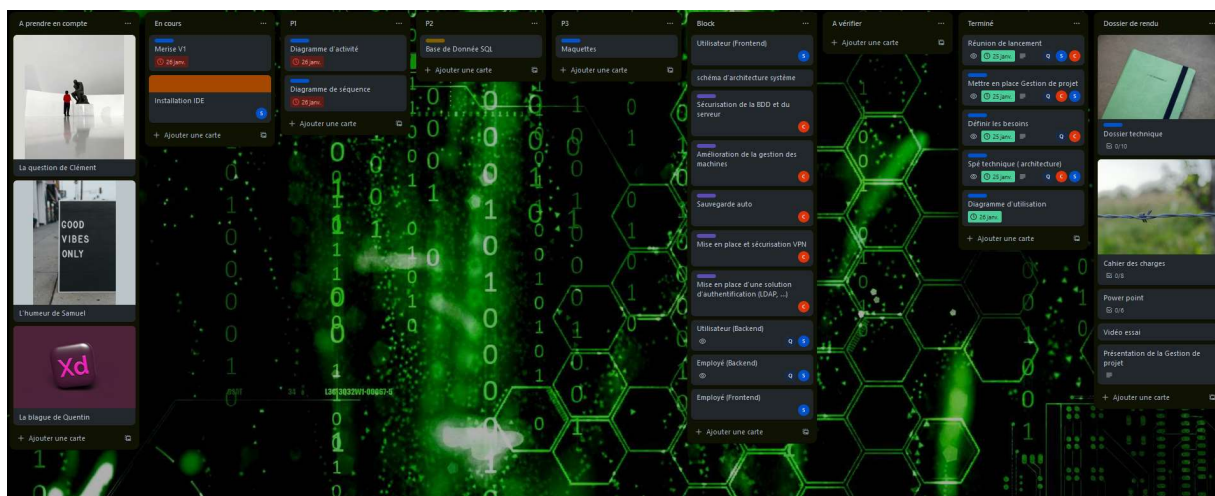
La réalisation d'un Kanban via l'outil Trello nous a permis de garder un œil sur les différentes étapes et la progression du projet.

L'utilisation de GitHub s'est avérée essentielle dans le partage de nos différentes réalisations et avancée tant sur la partie du développement que sur la rédaction des différents livrables.

Des outils tels que Discord ont également servi pour certaines réunions faites à distance.

Nous avons pu lors de ce projet, travailler notre cohésion de groupe en répartissant les tâches à réaliser tout en conservant un lien et une communication efficaces entre les différentes parties du projet.

Trello



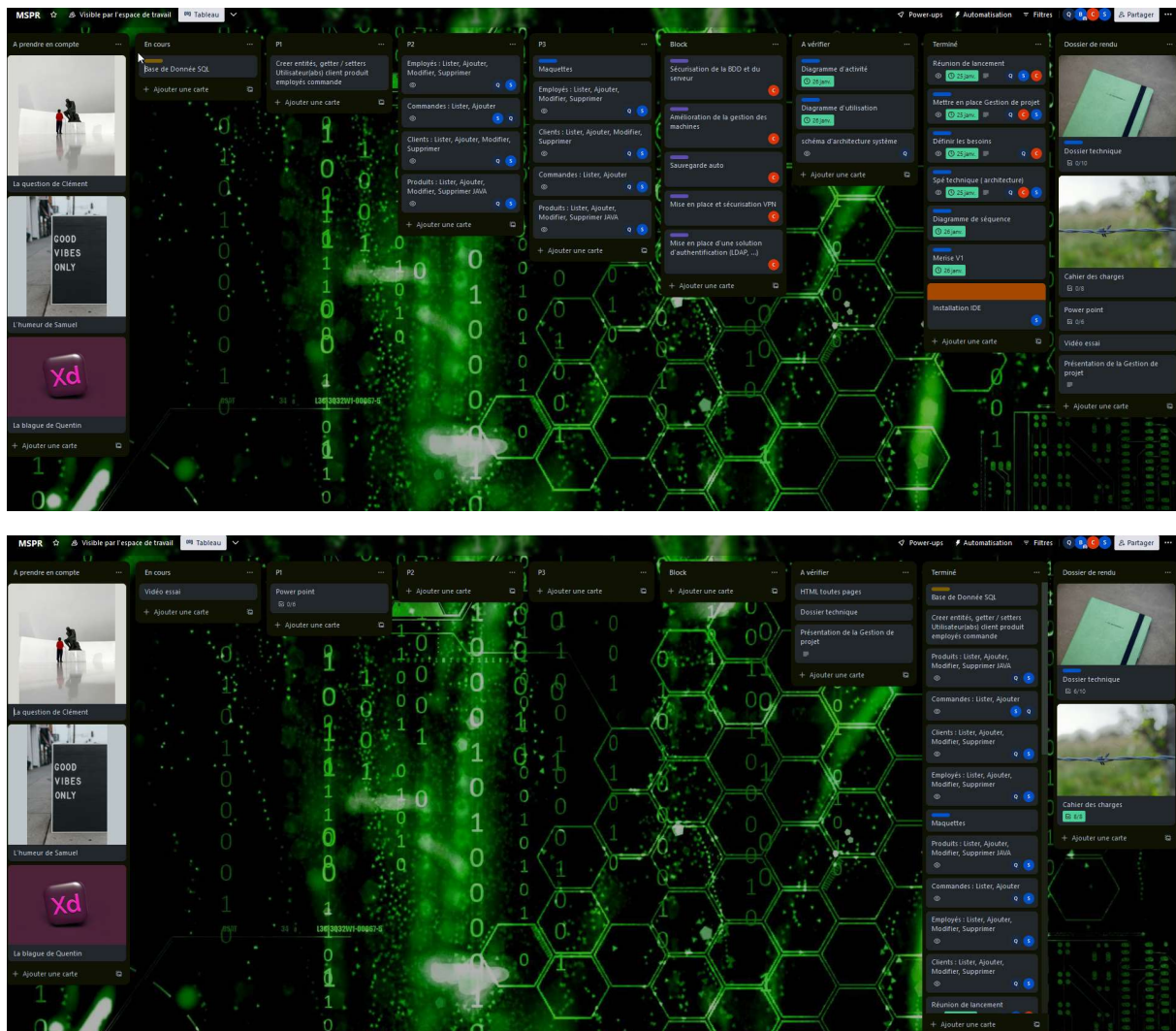


Figure 12 : outil de gestion de projet Trello

Bilan du projet

Au cours de ces trois derniers mois nous avons pu dans notre gestion de projet, impulser une réelle dynamique de groupe travaillant à la fois sur des aspects du développement, de modélisation et d'infrastructure, ...

Nous avons réussi à travailler de concert pour fournir une application non seulement fonctionnelle mais également allant plus loin que les attentes fournies dans les consignes.

Certaines difficultés se sont fait ressentir notamment lors de la mise en place du projet. En effet, lors de l'initialisation du projet, nos connaissances étaient insuffisantes du fait de la précocité du démarrage vis-à-vis des enseignements qui devaient nous être dispensés. Au fil des semaines nous avons donc pu non seulement appliquer des notions vu lors des différents cours mais également approfondir chez nous certaines notions tout en posant des questions aux intervenants sur l'application de ces notions sur notre projet.

Les difficultés ont donc toutes été plus ou moins dépassées avec cependant quelques problématiques restant à approfondir si nous disposions de plus de temps ou à revoir s'il fallait refaire le même projet.

L'une de ces difficultés réside dans l'utilisation d'outils essentiels pour la bonne gestion de projet comme GitHub qui ne faisait pas partie du programme de la formation. Nous avons donc tous perdu du temps afin de maîtriser autant que possible ce nouvel outil.

Le manque de notions de développement lors de la phase de modélisation nous a paru également être une difficulté étant donné le manque d'informations dont nous disposions sur la forme qu'allait revêtir le CRM.

Une difficulté qui s'est présentée lors de cette gestion de projet découle d'une des forces de notre groupe. Le champ d'expertise bien distinct des différents membres. D'un côté cela nous a permis d'avancer rapidement de façon passionnée sur le projet mais cela a pu impacter notre compréhension des différentes parties des uns et des autres qui aurait pu être préjudiciable si nous n'étions pas curieux de la partie des autres.

Nous aurions également souhaité approfondir certaines choses si nous avions disposé de plus de temps. C'est le cas notamment d'un attribut de prix unitaire à la commande produit pour stocker un prix à un moment donné. C'est le cas également de la gestion de toutes les exceptions et les erreurs (gérer tous les inputs, mettre en forme via des Regex, etc, ...). Un travail davantage approfondi sur la mise en page et le style aurait également été apprécié.

Pour conclure ce projet a réellement été positif et nous a permis de mettre en application de nombreuses notions vu en cours et même, dans de nombreux cas, de les dépasser tant sur le développement que sur la partie infrastructure et cybersécurité.

Annexes

Bibliographie

<https://openclassrooms.com/fr/>

<https://www.youtube.com/>

<https://www.baeldung.com/>

<https://frontbackend.com/thymeleaf/>

<https://stackoverflow.com/>

<https://stackabuse.com/>