LITZLER Marco

DE ABREU Quentin

ROLLAND Julia

Rapport Projet gl52   
Prototype Gestion Magasin ÉLECTROMÉNAGER

UV GL52           P21

Professeur : Abderrafiaa Koukam

Table des matières

[1- Introduction 6](#_Toc74902760)

[Buts et destinataires 6](#_Toc74902761)

[Présentations générale du document 6](#_Toc74902762)

[2- Description générale 7](#_Toc74902763)

[Environnement ou contexte du système 7](#_Toc74902764)

[Contexte : Spécification en langage naturel 7](#_Toc74902765)

[La vente 7](#_Toc74902766)

[Le service après-vente (SAV) 7](#_Toc74902767)

[Le stock 7](#_Toc74902768)

[Diagramme de contexte 8](#_Toc74902769)

[SAV 8](#_Toc74902770)

[Vente 8](#_Toc74902771)

[Stock 9](#_Toc74902772)

[Caractéristique des utilisateurs 9](#_Toc74902773)

[Le client 9](#_Toc74902774)

[Les employés 9](#_Toc74902775)

[Les contraintes principales de développement 9](#_Toc74902776)

[Hypothèse de travail 9](#_Toc74902777)

[3- Besoin fonctionnels 10](#_Toc74902778)

[Nom du cas d'utilisation : Diagramme cas d’utilisation 10](#_Toc74902779)

[SAV 10](#_Toc74902780)

[Vente 16](#_Toc74902781)

[Stock 19](#_Toc74902782)

[4- Spécification des structures de données 20](#_Toc74902783)

[Package MODEL 20](#_Toc74902784)

[Package service 21](#_Toc74902785)

[Package controller 22](#_Toc74902786)

[Package UI 23](#_Toc74902787)

[Projet complet 24](#_Toc74902788)

[5- Spécifications des interfaces externes 25](#_Toc74902789)

[Interface matériel/logiciel 25](#_Toc74902790)

[Interface logiciel/logiciel 25](#_Toc74902791)

[Interface Homme/logiciel 27](#_Toc74902792)

[Vente 27](#_Toc74902793)

[SAV 28](#_Toc74902794)

[Stock 29](#_Toc74902795)

[Gestionnaire client 30](#_Toc74902796)

[6- Les besoins en performance 31](#_Toc74902797)

[7- Les contraintes de développement 31](#_Toc74902798)

[8- Références 31](#_Toc74902799)

[9- Index 32](#_Toc74902800)

[Package ui 32](#_Toc74902801)

[File App 32](#_Toc74902802)

[File controller\AbstractController 32](#_Toc74902803)

[File controller\GestionnaireClientController 32](#_Toc74902804)

[File controller\GestionnaireFactureController 32](#_Toc74902805)

[File controller\GestionnaireSavController 32](#_Toc74902806)

[File controller\GestionProduitController 33](#_Toc74902807)

[File controller\MainSceneController 33](#_Toc74902808)

[Package model 33](#_Toc74902809)

[File Client 33](#_Toc74902810)

[File Employe 34](#_Toc74902811)

[File Produit 34](#_Toc74902812)

[File sav\Demande 35](#_Toc74902813)

[File sav\Reparation 35](#_Toc74902814)

[File sav\SAV 35](#_Toc74902815)

[File sav\facture\Facture 35](#_Toc74902816)

[File sav\facture\LigneFacture 36](#_Toc74902817)

[File stock\EntreeStock 36](#_Toc74902818)

[File vente\LigneVente 36](#_Toc74902819)

[File vente\Vente 36](#_Toc74902820)

[Package controller 36](#_Toc74902821)

[File ClientController 36](#_Toc74902822)

[File EmployeeController 37](#_Toc74902823)

[File ProduitController 37](#_Toc74902824)

[File sav\DemandeController 37](#_Toc74902825)

[File sav\ReparationController 37](#_Toc74902826)

[File sav\SAVController 37](#_Toc74902827)

[File sav\facture\FactureController 38](#_Toc74902828)

[File sav\facture\LigneFactureController 38](#_Toc74902829)

[File stock\EntreeStockController 38](#_Toc74902830)

[File vente\LigneVenteController 38](#_Toc74902831)

[File vente\VenteController 38](#_Toc74902832)

[Package service 38](#_Toc74902833)

[File ClientService 38](#_Toc74902834)

[File EmployeeService 38](#_Toc74902835)

[File IConnectDbService 39](#_Toc74902836)

[File ProduitService 39](#_Toc74902837)

[File sav\DemandeService 39](#_Toc74902838)

[File sav\ReparationService 39](#_Toc74902839)

[File sav\facture\FactureService 39](#_Toc74902840)

[File sav\facture\LigneFactureService 39](#_Toc74902841)

[File stock\EntreeStockService 39](#_Toc74902842)

[File vente\VenteService 40](#_Toc74902843)

[File vente\LigneVenteService 40](#_Toc74902844)

[10- Annexes 40](#_Toc74902845)

[Outils 40](#_Toc74902846)

[Environnement de développement 40](#_Toc74902847)

# Introduction

## Buts et destinataires

L’objectif de ce projet est de proposer un prototype d’un logiciel permettant la gestion d’un magasin d’électroménager. Celui-ci doit faciliter le bon déroulement de la vente et du service après-vente du magasin.

L’organisme commanditaire souhaite remettre au goût du jour son outil déjà existant, dans l’objectif d’avoir une meilleure gestion de leurs ressources et de simplifier l’exécution des différentes tâches de leurs employés.

Les trois parties renouveler sur l’application seront la vente, le service après-vente et le stock du magasin.

## Présentations générale du document

Dans ce document, nous pouvons voir dans un premier temps, une description générale du projet. Celle-ci est composé d’une brève description complétée par plusieurs diagrammes UML tels que des diagrammes de contexte qui modélise des situations réalisables avec l’application ou encore de cas d’utilisation pour bien décrire l’application.

Dans une deuxième partie, nous décrivons les différents besoins fonctionnels, les structures de données qui composent notre application, les contraintes de développement que nous avons rencontrées durant ce semestre.

Enfin dans une dernière partie, nous répertorions toutes les références et indexons tous les packages et fonctions que nous avons utilisées. La partie annexes vient compléter notre rapport en détaillant les outils et l’environnement de développement du projet.

# Description générale

## Environnement ou contexte du système

### Contexte : Spécification en langage naturel

Le logiciel permet de gérer un magasin d’électroménager en trois parties : la vente, le SAV, le stock.

#### La vente

Concernant la partie vente. Un client veut acheter un article disponible dans le magasin. Pour ce faire, un membre du personnel prend les références de l’article et du client (S’il n’est pas déjà dans le fichier client). Le client peut payer par carte ou en espèces.

#### Le service après-vente (SAV)

Concernant la partie SAV. Un client a un problème ou souhaite rendre le produit qu’il a acheté. Un membre du personnel prend les références de l’article et du client. Puis il vérifie la date d’achat du produit, si celui-ci est garanti ou non. Le produit est ensuite étudié  pour voir s’il est neuf (pour un retour) ou s’il est défectueux comme indiqué par le client. En fonction de l’état du produit, on a plusieurs cas possibles :

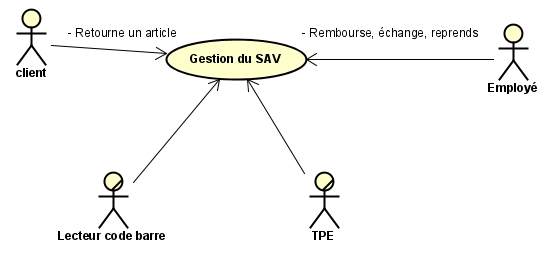
* Il est repris et on rembourse ou donne un avoir au client.
* Il est envoyé en réparation puis donné au client.
* Il n’est pas réparable. S’il en reste en stock, on donne un nouveau produit au client sinon on le rembourse ou lui donne un avoir.
* Il n’a pas de défaut, on le rend au client.
* Possibilité de prêter un produit pendant la durée de la réparation.

#### Le stock

Concernant la gestion du stock. Possibilité de voir le stock en temps réel de l’ensemble du magasin. Possibilité de modifier le stock par exemple lors d’un inventaire.

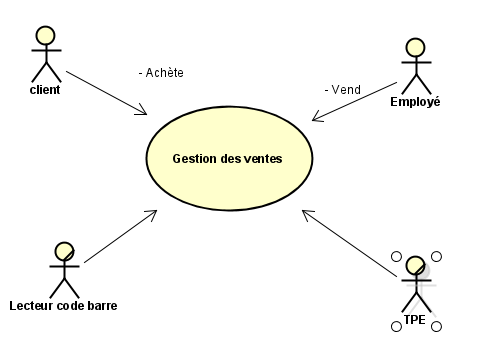
### Diagramme de contexte

#### SAV



Dans ce diagramme de contexte représentant le SAV du magasin, nous pouvons voir qu’il y a deux acteurs : l’employé et le client. Le lecteur de code barre et le TPE sont des instances utilisées par notre application. Ici, nous pouvons voir que le client doit pouvoir retourner un article et l’employé du magasin doit, en fonction de l’état de l’article retourner, rembourser/échanger/reprendre l’article du client. Il utilise le lecteur de code barre pour enregistrer l’article retourné et le TPE pour rembourser si besoin le client.

#### Vente



Dans ce diagramme de contexte représentant la partie Vente du magasin, nous pouvons retrouver encore les deux mêmes acteurs que pour le SAV : le client et l’employé, ainsi que le lecteur de code barre et le TPE. La gestion des ventes est simple, le client sélectionne plusieurs articles et les achète auprès du magasin. L’employé lui scanne les articles à l’aide du lecteur de code barre et les vends grâce au TPE qui va recueillir l’argent du client et enregistrer la vente.

#### Stock



Dans ce diagramme de contexte représentant la partie Stock, nous avons qu’un seul acteur : un employé. En effet, le client n’a pas accès à cette partie du magasin. Comme aucune vente ni interaction n’est réalisée avec le client, il n’y a que le lecteur de code barre qui est représenté comme instance. Dans cette partie, l’employé peut voir le stock du magasin, c’est-à-dire la liste de tous les produits vendus mais peut également le modifier (faire des nouvelles commandes, supprimer des produits, etc.).

## Caractéristique des utilisateurs

### Le client

Il n’est pas l’utilisateur direct du système, ses informations transitent par un employé de la boutique avant d'être intégrées dans le logiciel. Il est un acteur extérieur du système.

### Les employés

Les employés de l’entreprise sont polyvalents et assurent différents rôles liés à différentes fonctions du logiciel (vente, SAV, gestion du stock).

## Les contraintes principales de développement

Aucune contrainte n’est réellement présente, la réalisation de ce projet est particulièrement libre.

## Hypothèse de travail

Plusieurs hypothèses sont susceptibles de remettre en cause la réalisation des spécifications du projet, en voit ci :

* L’utilisation d’un client léger type page web, très chronophage:

→ Solution : utilisation d’un client lourd exemple JavaFx

* Le périmètre d'action du projet est très vaste.
* Lien avec une base de données client - server

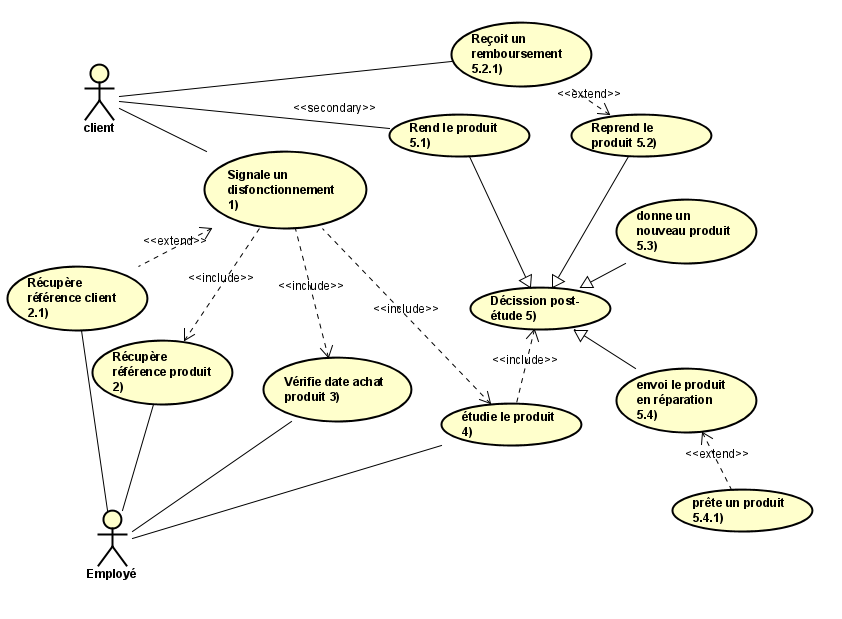
→ Solution : utilisation d’une base de données embarquée : SQLite 3

* Librairie restrictive en termes d’accès et d’utilisation.

# Besoin fonctionnels

## Nom du cas d'utilisation : Diagramme cas d’utilisation

### SAV



|  |  |
| --- | --- |
| Use Case ID | 1 |
| Nom | Signale un dysfonctionnement |
| Premier Acteur | Client |
| Deuxième Acteur | Employé |
| Description | Le client contact le service après-vente pour demander une réparation car son produit à un dysfonctionnement |
| Pré-Conditions | / |
| Post-Conditions | L’employé récupère le produit, les références du client, vérification de la date d’achat, étude du problème |
| Événements principaux | a. Recherche client dans la base  b. Ajout de la demande dans la base avec les informations |
| Evènements exceptionnels | a. Le client est inconnu  b. Le produit n’a pas été acheté au magasin |

|  |  |
| --- | --- |
| Use Case ID | 2 |
| Nom | Récupère référence produit |
| Premier Acteur | Employé |
| Deuxième Acteur | / |
| Description | L’employé récupère les informations du client |
| Pré-Conditions | Signale un dysfonctionnement |
| Post-Conditions | L’employé récupère les références du client, vérifie de la date d’achat, étude du problème |
| Événements principaux | Sélection produit dans la base |
| Evènements exceptionnels | a. Le produit est inconnu  b. Le produit n’a pas été acheté au magasin |

|  |  |
| --- | --- |
| Use Case ID | 2.1 |
| Nom | Récupère référence client |
| Premier Acteur | Employé |
| Deuxième Acteur | / |
| Description | L’employé récupère les informations du client |
| Pré-Conditions | Récupère référence produit |
| Post-Conditions | L’employé vérifie de la date d’achat, étude du problème |
| Événements principaux | Sélection client dans la base  ou  Création du produit dans la base |
| Evènements exceptionnels | a. Le client est inconnu  b. Le produit n’a pas été acheté au magasin |

|  |  |
| --- | --- |
| Use Case ID | 3 |
| Nom | Vérifie date achat produit |
| Premier Acteur | Employé |
| Deuxième Acteur | / |
| Description | L’employé vérifie qu’une vente est associé au produit |
| Pré-Conditions | Récupère référence client |
| Post-Conditions | étude du problème |
| Événements principaux | Vérification de la vente dans la base |
| Evènements exceptionnels | b. Le produit n’a pas été acheté au magasin |

|  |  |
| --- | --- |
| Use Case ID | 4 |
| Nom | Etudie le produit |
| Premier Acteur | Employé |
| Deuxième Acteur | / |
| Description | L’employé étudie les dysfonctionnements du produit |
| Pré-Conditions | Vérifie date achat produit |
| Post-Conditions | Décision post-étude |
| Événements principaux | 1. étude de l’état du produit |
| Evènements exceptionnels |  |

|  |  |
| --- | --- |
| Use Case ID | 5 |
| Nom | Décision post-étude |
| Premier Acteur | Employé |
| Deuxième Acteur | / |
| Description | L’employé rend son verdict sur l’état du produit |
| Pré-Conditions | Etude du produit |
| Post-Conditions | Rend le produit, reprend le produit, envoi le produit en réparation, donne un nouveau produit |
| Événements principaux | 1. annonce du verdict 2. Mène une action associée |
| Evènements exceptionnels |  |

|  |  |
| --- | --- |
| Use Case ID | 5.1 |
| Nom | Rend le produit |
| Premier Acteur | Employé |
| Deuxième Acteur | Client |
| Description | Le produit n’a pas de problème, il est rendu au client |
| Pré-Conditions | Décision post-étude |
| Post-Conditions | / |
| Événements principaux | 1. Le produit est retourné au client. |
| Evènements exceptionnels |  |

|  |  |
| --- | --- |
| Use Case ID | 5.2 |
| Nom | Reprend le produit |
| Premier Acteur | Employé |
| Deuxième Acteur | Client |
| Description | L’employé récupère le produit. |
| Pré-Conditions | Décision post-étude |
| Post-Conditions | Reçoit un remboursement |
| Événements principaux | 1. retour du produit |
| Evènements exceptionnels |  |

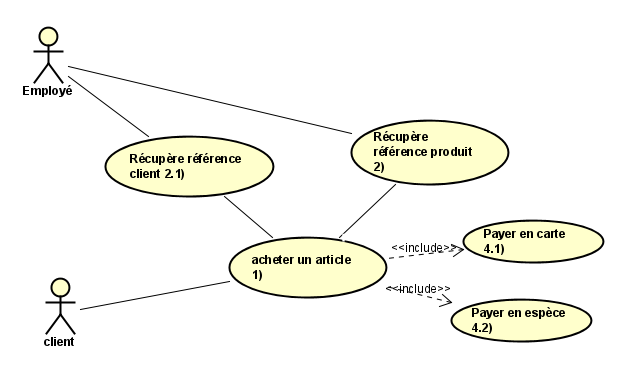
|  |  |
| --- | --- |
| Use Case ID | 5.2.1 |
| Nom | Reçoit un remboursement |
| Premier Acteur | Client |
| Deuxième Acteur | Employé |
| Description | Le client reçoit un remboursement pour le produit qu’il vient de retourner. |
| Pré-Conditions | Reprend le produit |
| Post-Conditions | / |
| Événements principaux | 1. Le client récupère la somme due suite à son retour 2. L’employé effectue le remboursement |
| Evènements exceptionnels |  |

|  |  |
| --- | --- |
| Use Case ID | 5.3 |
| Nom | Donne un nouveau produit |
| Premier Acteur | Employé |
| Deuxième Acteur | Client |
| Description | L’employé remplace le produit défectueux du client. |
| Pré-Conditions | Décision post-étude |
| Post-Conditions | / |
| Événements principaux | 1. L’employé récupère un second produit dans le stock 2. le client récupère un nouveau produit |
| Evènements exceptionnels |  |

|  |  |
| --- | --- |
| Use Case ID | 5.4 |
| Nom | Envoi le produit en réparation |
| Premier Acteur | Employé |
| Deuxième Acteur | / |
| Description | Le produit défectueux peut être réparé. |
| Pré-Conditions | Décision post-étude |
| Post-Conditions | / |
| Événements principaux | 1. Le produit est récupéré 2. Une demande de réparation est créée |
| Evènements exceptionnels |  |

|  |  |
| --- | --- |
| Use Case ID | 5.4.1 |
| Nom | Prête un produit |
| Premier Acteur | Employé |
| Deuxième Acteur | Client |
| Description | L’entreprise prête un produit durant la durée de la réparation |
| Pré-Conditions | Envoi le produit en réparation |
| Post-Conditions | / |
| Événements principaux | 1. L’employé récupère un produit en stock 2. Le client se voit pourvoir d’un produit de remplacement |
| Evènements exceptionnels |  |

### Vente



|  |  |
| --- | --- |
| Use Case ID | 1 |
| Nom | Acheter un article |
| Premier Acteur | Client |
| Deuxième Acteur | Employé |
| Description | Le client a choisi un article dans le magasin et passe à la caisse pour l’acheter. |
| Pré-Conditions | / |
| Post-Conditions | Récupérer référence produit |
| Événements principaux | * + - 1. Le client choisit un produit dans le magasin.       2. Le client va à la caisse pour l’acheter. |
| Evènements exceptionnels | a. L’article est inconnu ou n’est plus disponible dans le stock. |

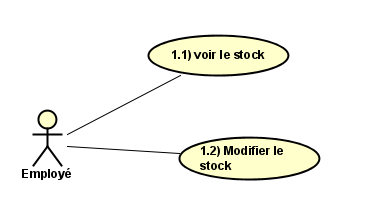
|  |  |
| --- | --- |
| Use Case ID | 2 |
| Nom | Récupérer la référence d’un produit |
| Premier Acteur | Employé |
| Deuxième Acteur | / |
| Description | L’employé parcours le stock du magasin pour voir si le produit est disponible à la vente et le sélectionne pour le client. |
| Pré-Conditions | Acheter un article |
| Post-Conditions | Récupérer référence client |
| Événements principaux | * + - 1. L’employé cherche dans le stock le produit désiré par le client. |
| Evènements exceptionnels | * + - 1. La référence du produit n’existe pas. |

|  |  |
| --- | --- |
| Use Case ID | 2.1 |
| Nom | Récupérer la référence d’un client |
| Premier Acteur | Employé |
| Deuxième Acteur | Client |
| Description | L’employé cherche dans la base de données du magasin si le client a déjà un code client. |
| Pré-Conditions | Récupérer référence produit |
| Post-Conditions | Payer en carte  Payer en espèce |
| Événements principaux | * + - 1. L’employé cherche dans la base de données le client.       2. L’employé sélectionne le client pour la vente. |
| Evènements exceptionnels | a. Le client est encore inconnu au magasin, on l’ajoute |

|  |  |
| --- | --- |
| Use Case ID | 4.1 |
| Nom | Payer en carte |
| Premier Acteur | Client |
| Deuxième Acteur | Employé |
| Description | Le client règle son achat par carte bancaire. |
| Pré-Conditions | Récupérer référence client |
| Post-Conditions | / |
| Événements principaux | * + - 1. Le client donne l’argent.       2. L’employé encaisse le client via le TPE.       3. Le client récupère son article acheté. |
| Evènements exceptionnels |  |

|  |  |
| --- | --- |
| Use Case ID | 4.2 |
| Nom | Payer en espèce |
| Premier Acteur | Client |
| Deuxième Acteur | Employé |
| Description | Le client règle son achat en espèce. |
| Pré-Conditions | Récupérer référence client |
| Post-Conditions | / |
| Événements principaux | * + - 1. Le client donne l’argent.       2. L’employé récupère la somme et encaisse le client sur le TPE.       3. Le client récupère son article acheté. |
| Evènements exceptionnels |  |

### Stock



|  |  |
| --- | --- |
| Use Case ID | 1.1 |
| Nom | Voir le stock |
| Premier Acteur | Employé |
| Deuxième Acteur | / |
| Description | L’employé peut voir tous les produits et leur description dans le stock du magasin. |
| Pré-Conditions | / |
| Post-Conditions | / |
| Événements principaux | a. L’employé parcours le stock. |
| Evènements exceptionnels |  |

|  |  |
| --- | --- |
| Use Case ID | 1.2 |
| Nom | Modifier le stock |
| Premier Acteur | Employé |
| Deuxième Acteur | / |
| Description | L’employé modifie le stock : il ajoute des nouveaux produits, augmentent la quantité/prix/etc. d’un produit déjà existant ou supprimer un produit du stock. |
| Pré-Conditions | / |
| Post-Conditions | / |
| Événements principaux | a. L’employé modifie le stock. |
| Evènements exceptionnels |  |

# Spécification des structures de données

## Package MODEL

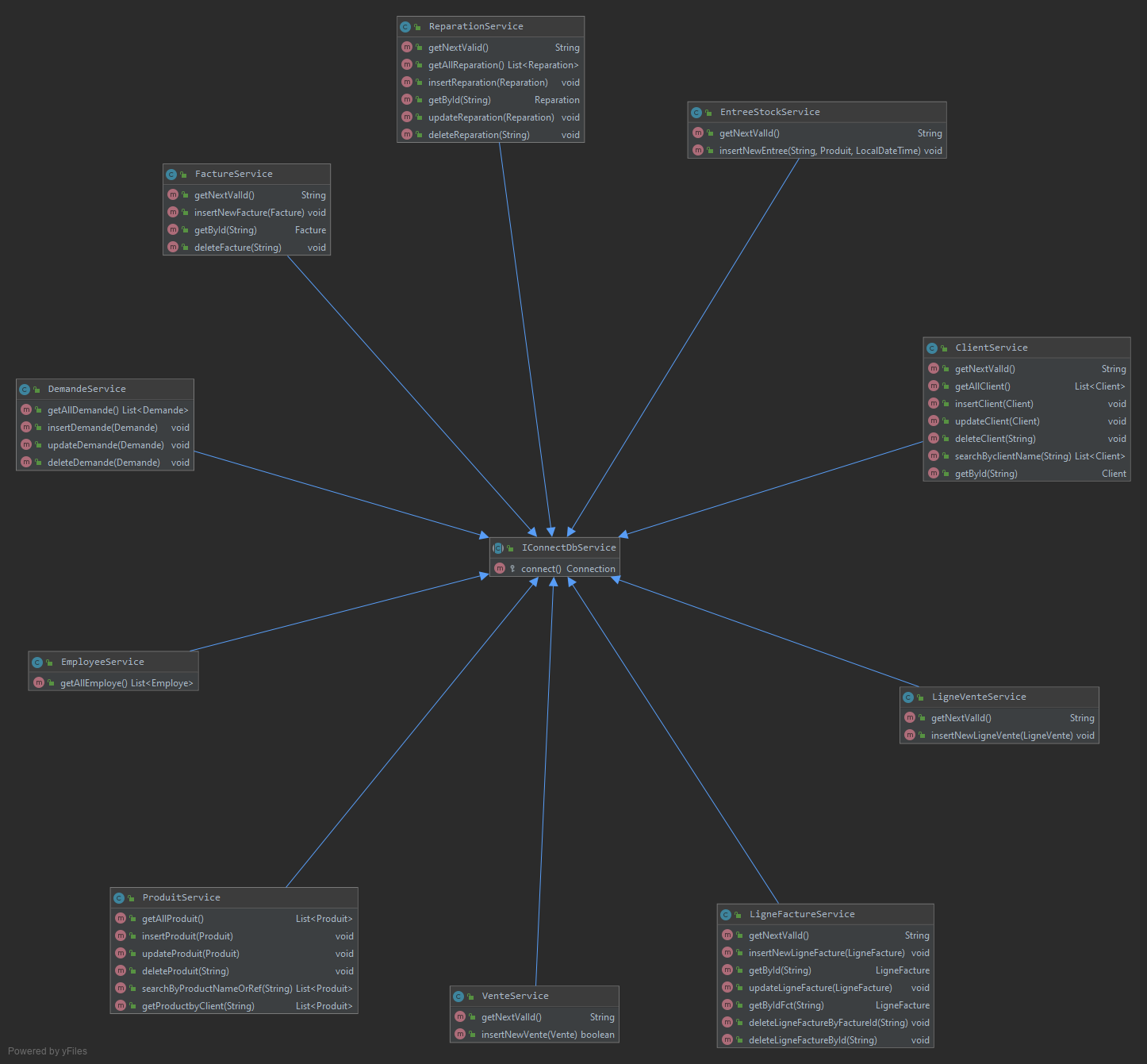
Une image contenant texte, noir, métal, capture d’écran

Description générée automatiquement

**Figure 1 : Diagramme de classe - package model**

La structure des classes composant le package model est toujours la même, à savoir un Java beans. C’est-à-dire que chaque classe possède ses attributs propres ainsi que les getters et setters. De plus les méthodes hashcode(), equals() et toString() sont également implémentées. Toutes ces classes sont bien évidemment en relation l’une avec les autres, mais seulement par l’intermédiaire de “services” que nous verrons juste après.

## Package service



**Figure 2 : diagramme de classe - package service**

Les classes du package service ont toutes la même particularité. En effet, chaque entité hérite d’une seule et même classe abstraite nommée “*IConnectDbService”* en voici plus de détails :

***IConnectDbService***

*Cette classe abstraite contient une unique méthode permettant d’ouvrir et de fermer la connexion à la base de données et donc par la suite de faire des requêtes SQL.*

*Méthode : protected Connection connect();*

*Comme expliqué précédemment, c'est au sein de cette méthode que le projet peut communiquer à l’aide de requêtes SQL. Ainsi la méthode contient les chaînes et les drivers de connexion à la base de données. Cette méthode retourne un objet* ***Connection*** *qui sera le point de départ des futures requêtes SQL.*Pour finir, chaque méthode de chaque classe correspond à une requête SQL.

## Package controller

Une image contenant texte, noir, capture d’écran

Description générée automatiquement

**Figure 3 : diagramme de classe - package controller**

Pour le package controller, chaque classe permet d’accéder aux données du model et donc de la base de données. Tout cela de manière contrôlée afin de garantir la sécurité et de respecter le principe d’encapsulation. Les méthodes de chaque classe correspondent aux services vus précédemment.

## Package UI

Une image contenant texte

Description générée automatiquement

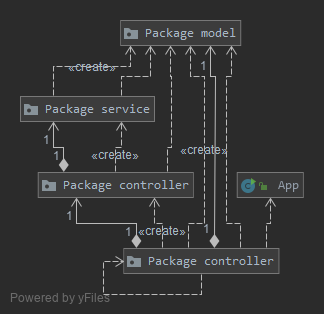
**Figure 4 : diagramme de classe - package ui**

Le package ui (user interface) est vraiment le centre du projet puisque c’est dans les classes de ce package que l’appel des contrôleurs est effectué, mais également la mise à jour de l’interface.

Pour ce faire, l’interface est découpée en plusieurs fenêtres, il y a donc pour chaque fenêtre un contrôleur (du type contrôleur d’interface graphique différent des contrôleurs présentés précédemment).

Nous pouvons retrouver en attributs les éléments graphiques appelé widget disposé sur l’interface utilisateur et les méthodes permettant la mise à jour des données de l’interface.

## Projet complet



**Figure 5 : diagramme montrant les relations entre chaque package**

Pour finir sur la structure du projet, nous trouvons ici la façon dont tous les packages sont connectés entre eux.

# Spécifications des interfaces externes

## Interface matériel/logiciel

Le projet est en lien avec aucun autre élément matériel, cependant dans une application concrète du logiciel celui-ci est en relation avec quatre éléments matériels :

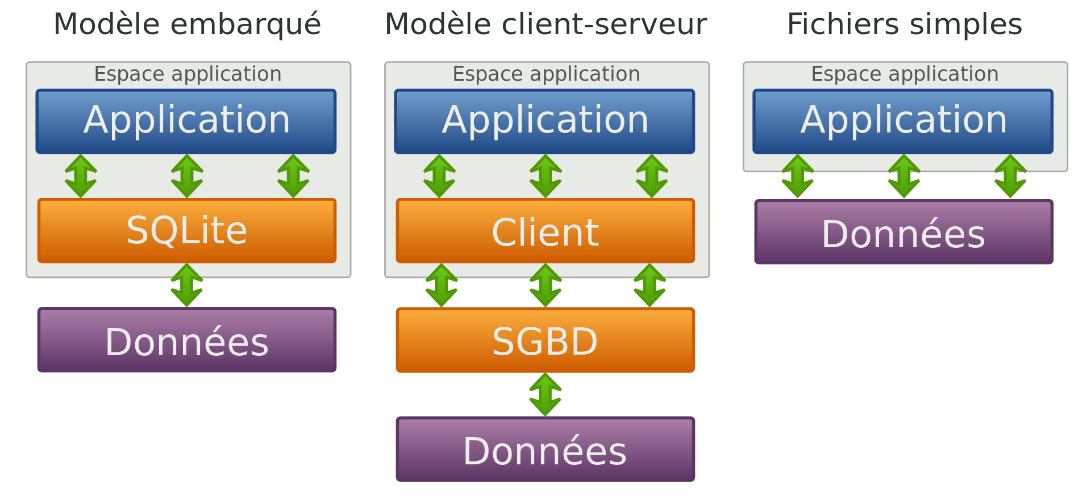
* Un serveur situé au sein même de l’entreprise. Ce serveur héberge l’ensemble de la base de données du magasin.
* Un lecteur de code-barre. L’échange se fait par câble ou bien par Bluetooth puisque chaque poste dispose d’un lecteur.
* Un TPE pour les paiements par carte bancaire. Celui-ci est relié par câble à chaque poste de l’entreprise.
* Une caisse enregistreuse. Puisqu’il est possible de payer en liquide, l’ouverture et la fermeture peuvent se commander depuis le logiciel.

## Interface logiciel/logiciel

SQLite 3.31.1 :



La partie base de données a été réalisée avec SQLite qui est un moteur de base de données qui permet une intégration directe à l’application sans système client/serveur plus classique. SQLite, au contraire, est directement intégrée dans l'application qui utilise sa bibliothèque logicielle, avec son moteur de base de données. L'accès à une [base de données](https://fr.wikipedia.org/wiki/Base_de_donn%C3%A9es) avec SQLite se fait par l'ouverture du fichier correspondant à celle-ci : chaque base de données est enregistrée dans un fichier qui lui est propre, avec ses déclarations, ses tables et ses index mais aussi ses données. De par son extrême légèreté (moins de 600 [Kio](https://fr.wikipedia.org/wiki/Octet#Multiples_normalis.C3.A9s)), il est également très populaire sur les [systèmes embarqués](https://fr.wikipedia.org/wiki/Syst%C3%A8me_embarqu%C3%A9). De plus, c'est également un outil open source, ce qui permet de l’utiliser sans licence.

  
Architecture de SQLite

Javafx 11.0.2 :



JavaFX est un [framework](https://fr.wikipedia.org/wiki/Framework) et une [bibliothèque d'interface utilisateur](https://fr.wikipedia.org/wiki/Biblioth%C3%A8que_graphique) issue du projet [OpenJFX](https://fr.wikipedia.org/wiki/OpenJFX), qui permet aux développeurs [Java](https://fr.wikipedia.org/wiki/Java_(langage)) de créer une [interface graphique](https://fr.wikipedia.org/wiki/Interface_graphique) pour des applications de bureau, des [applications internet riches](https://fr.wikipedia.org/wiki/Rich_desktop_application) et des applications smartphones et tablettes tactiles.

Cette API est composée de nombreux composants et est un projet open source.

Nous avons choisi cette API notamment pour la possibilité d’utiliser FXML. FXML est un [format de données](https://fr.wikipedia.org/wiki/Format_de_donn%C3%A9es) textuelles, dérivé du format [XML](https://fr.wikipedia.org/wiki/Extensible_Markup_Language), qui permet de décrire une [interface utilisateur](https://fr.wikipedia.org/wiki/Interface_graphique). Il s'agit d'une alternative à la conception d'interfaces réalisés par l'écriture de lignes de codes, en découplant l'interface graphique du code source qui le contrôle.

Maven 3.8.1:



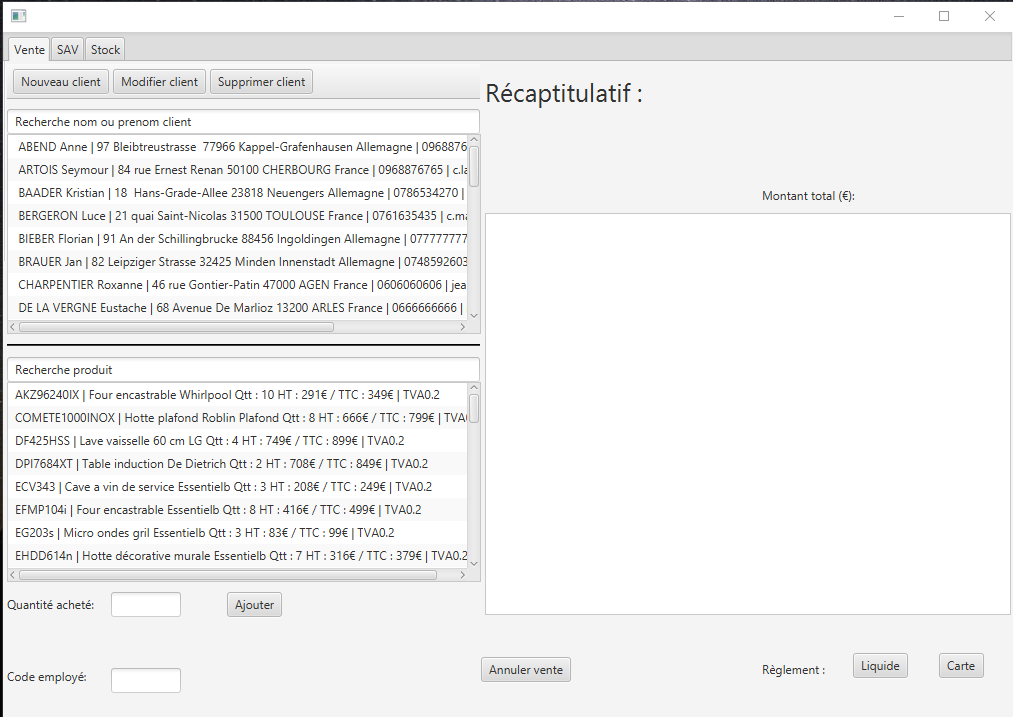
Apache Maven est un outil de gestion et de compréhension de projets logiciels. Basé sur le concept d'un modèle d'objet de projet (POM), Maven peut gérer la construction, le reporting et la documentation d'un projet à partir d'une information centrale.

L'objectif recherché est de produire un logiciel à partir de ses sources, en optimisant les tâches réalisées à cette fin et en garantissant le bon ordre de fabrication.

Chaque projet ou sous-projet est configuré par un POM qui contient les informations nécessaires à Maven pour traiter le projet (nom du projet, numéro de version, dépendances vers d'autres projets, bibliothèques nécessaires à la compilation, noms des contributeurs, etc.). Ce POM se matérialise par un fichier pom.xml à la racine du projet. Cette approche permet l'héritage des propriétés du projet parent. Si une propriété est redéfinie dans le POM du projet, elle recouvre celle qui est définie dans le projet parent. Ceci introduit le concept de réutilisation de configuration. Le fichier pom du projet principal est nommé pom parent. Il contient une description détaillée de votre projet, avec en particulier des informations concernant le versionnage et la gestion des configurations, les dépendances, les ressources de l'application, les tests, les membres de l'équipe, la structure et bien plus.

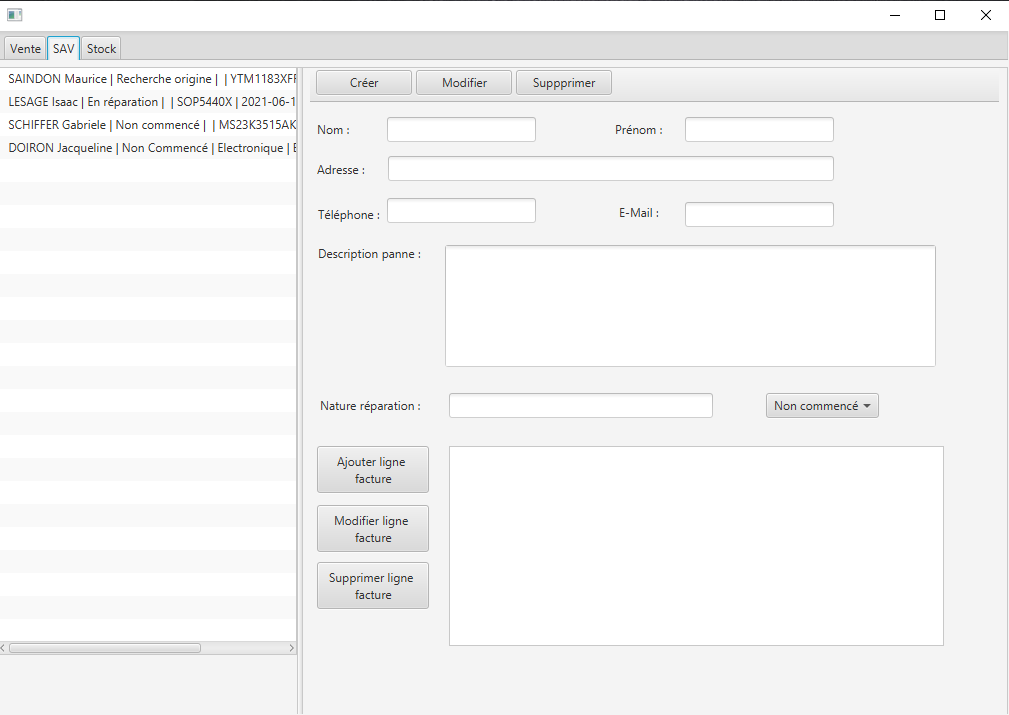
## Interface Homme/logiciel

### Vente



Dans cette partie l’utilisateur peut ajouter, modifier et supprimer un client. De plus, il peut créer une nouvelle vente en sélectionnant un client, un produit, une quantité, un code employé et un moyen de paiement. Si l’un de ces éléments est invalide ou est manquant alors la vente n’est pas créée sans message d’erreur.

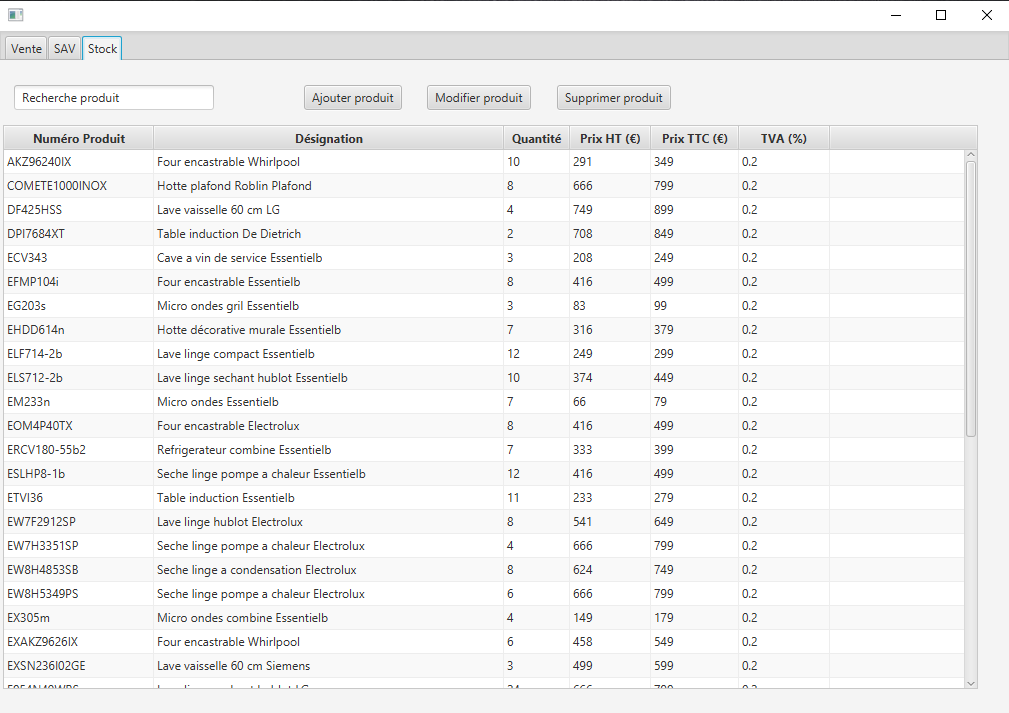
### SAV



Dans cette partie l’utilisateur peut créer, modifier et supprimer une demande de services après-vente. Une demande de sav doit contenir : le nom, prénom, numéro de téléphone, adresses postale et e-mail d’un client. Cette demande peut également contenir une description de la panne, la nature de la réparation ainsi que son état.

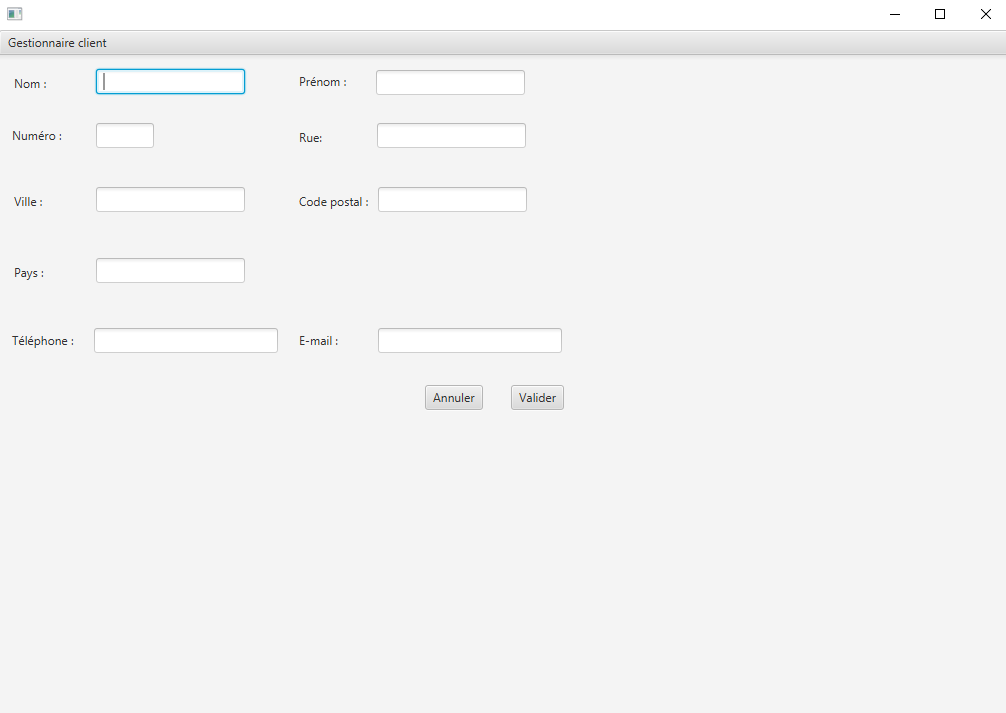
C’est également sur cet état que l’utilisateur peut créer une facture pour la demande de réparation.

### Stock



Dans cette partie l’utilisateur peut visualiser les stocks, rechercher, ajouter, modifier et supprimer des produits.

### Gestionnaire client



Cet écran est celui affiché lorsque l’utilisateur souhaite créer un nouveau client (des écrans analogues existent pour la gestion des demandes de réparations et la gestion des produits).

# Les besoins en performance

Le programme n’a pas de réelle contrainte de performance puisqu’il s’agit seulement d’ajout, de modification ou de suppression de données, ce qui n’est pas réellement gourmand en termes de performances. L’outil peut être utilisé de multiple fois sans souci d’atteindre un nombre maximal de terminaux.

Dans le cas d’une utilisation avec une base de données hébergée sur un serveur, il faudrait bien évidemment instaurer un pool de connexion pour pouvoir gérer le nombre de connexion et le nombre de transaction vers la base de données.

Enfin ce projet ne possède pas de contrainte concernant l ‘environnement ou bien le temps de réponse.

# Les contraintes de développement

Pour ce qui est de la fiabilité, le programme continue de fonctionner lorsque celui-ci fait face à une erreur. Cependant nous avons estimé que dans le cadre professionnel ou le prototype est utilisé, les utilisateurs sont aptes à rentrer les bonnes informations, sous le bon format, dans la bonne cellule. De ce fait, le programme acceptera tout format de données sans vérification. Néanmoins, certaines conditions sont tout même nécessaires pour effectuer des actions comme la sélection d’un élément pour pouvoir le modifier.

Par rapport à la sécurité, nous contrôlons l’accès aux données par le biais de services spécifiques (*expliqué dans la partie Spécification des structures de données).* Cependant nous avons jugé qu’il n’était pas nécessaire d’implémenter un système d’authentification puisque les seuls utilisateurs de cet outil sont les employés de l’entreprise.

# Références

*JavaFX 8*. (s. d.). Docs Oracle JavaFX 8. Consulté le 16 juin 2021, à l’adresse <https://docs.oracle.com/javase/8/javafx/api/toc.htm>

*MavenRepository*. (s. d.). Maven Repository. Consulté le 16 juin 2021, à l’adresse <https://mvnrepository.com/>

# Index

## Package ui

### File App

private static Parent loadFXML(String fxml) throws IOException

public static void main(String[] args)

public static void setRoot(String fxml) throws IOException

public void start(@SuppressWarnings("exports") Stage stage) throws IOException

public static FXMLLoader getFxmlLoader()

### File controller\AbstractController

public void switchToMainScene() throws IOException

### File controller\GestionnaireClientController

private Client setClient(String id)

private boolean checkTextFieldIsNull()

private void handleClientAdresse(ActionEvent event)

private void handleClientCp(ActionEvent event)

private void handleClientEmail(ActionEvent event)

private void handleClientFirstName(ActionEvent event)

private void handleClientLastName(ActionEvent event)

private void handleClientPays(ActionEvent event)

private void handleClientPhone(ActionEvent event)

private void handleClientRue(ActionEvent event)

private void handleClientVille(ActionEvent event)

private void handleClientnum(ActionEvent event)

private void isNew() throws IOException

private void modifyClient() throws IOException

private void newClient() throws IOException

public void initialize(URL location, ResourceBundle resources)

public void setDataForUpdate(@SuppressWarnings("exports") Client client)

public void switchToMainScene() throws IOException

public void switchToMainSceneValidate() throws IOException

### File controller\GestionnaireFactureController

private LigneFacture setLigneFacture(String id)

private void handleDesRepTxtField(ActionEvent event)

private void handlePrixRepTxtField(ActionEvent event)

private void modifyLigneFacture() throws IOException

private void newLigneFacture() throws IOException

public void setDataForUpdate(@SuppressWarnings("exports") LigneFacture ligneFacture)

public void setNumFct(String numFct)

### File controller\GestionnaireSavController

private void handleReparationDescription(ActionEvent event)

private void handleReparationNature(ActionEvent event)

private void handleReparationNumEmployee(ActionEvent event)

private void newSAV() throws IOException

public void getSearchClientVente()

public void setClientProduct()

### File controller\GestionProduitController

private Produit setProduit() throws IOException

private void handleProductDescription(ActionEvent event)

private void handleProductHTPrice(ActionEvent event)

private void handleProductQuantity(ActionEvent event)

private void handleProductReference(ActionEvent event)

private void handleProductTTCPrice(ActionEvent event)

private void modifyProduit() throws IOException

private void newProduit() throws IOException

public void setDataForUpdate(@SuppressWarnings("exports") Produit produit)

### File controller\MainSceneController

private Vente setSell()

private void setSellLine(String numVte, String refProd, String qtt)

private void switchToGestionnaireClient() throws IOException

public MainSceneController()

public void addFactureLine() throws IOException

public void cancelSell()

public void changeMontantVenteValue()

public void changeSavTextFieldValues()

public void createClient() throws IOException

public void createProduit() throws IOException

public void createReparation() throws IOException

public void deleteClient() throws IOException

public void deleteFactureLine()

public void deleteProduit() throws IOException

public void deleteReparation()

public void getSearchProductStock()

public void getSearchProductVente()

public void modifyClient() throws IOException

public void modifyFactureLine() throws IOException

public void modifyProduit() throws IOException

public void modifyReparation() throws IOException

public void selectClientVenteInfo()

public void sell()

## Package model

### File Client

public Client()

public String getAdresse()

public String getCpCli()

public String getMailCli()

public String getNomCli()

public String getNumCli()

public String getPaysCli()

public String getPrenomCli()

public String getRueCli()

public String getTelCli()

public String getVilleCli()

public String toString()

public void setCpCli(String cpCli)

public void setMailCli(String mailCli)

public void setNomCli(String nomCli)

public void setNumCli(String numCli)

public void setPaysCli(String paysCli)

public void setPrenomCli(String prenomCli)

public void setRueCli(String rueCli)

public void setTelCli(String telCli)

public void setVilleCli(String villeCli)

### File Employe

public Employe()

public String getCpEmp()

public String getMailEmp()

public String getNomEmp()

public String getNumEmp()

public String getPaysEmp()

public String getPrenomEmp()

public String getRueEmp()

public String getTelEmp()

public String getVilleEmp()

public void setCpEmp(String cpEmp)

public void setMailEmp(String mailEmp)

public void setNomEmp(String nomEmp)

public void setNumEmp(String numEmp)

public void setPaysEmp(String paysEmp)

public void setPrenomEmp(String prenomEmp)

public void setRueEmp(String rueEmp)

public void setTelEmp(String telEmp)

public void setVilleEmp(String villeEmp)

### File Produit

public Produit()

public Produit(String referenceProd, String descriptionProduct, String quantity, String prixHTT, String prixTTC, String tVA)

public String getDesProd()

public String getPrixHTProd()

public String getPrixTTCProd()

public String getQteProd()

public String getRefProd()

public String getTVAProd()

public boolean equals(Object obj)

public final String getDesProdProperty()

public final String getPrixHTProdProperty()

public final String getPrixTTCProdProperty()

public final String getQteProdProperty()

public final String getRefProdProperty()

public final String getTVAProdProperty()

public int hashCode()

public void setDesProd(String desProd)

public void setPrixHTProd(String prixHTProd)

public void setPrixTTCProd(String prixTTCProd)

public void setQteProd(String qteProd)

public void setRefProd(String refProd)

public void setTVAProd(String tVAProd)

### File sav\Demande

public Demande()

public Demande(String numRep, String numCli, String natureRep, String descRep, String refProd, String numFct)

public String getDescRep()

public String getNatureRep()

public String getNumFct()

public String getNumRep()

public void setDescRep(String descRep)

public void setNatureRep(String natureRep)

public void setNumRep(String numRep)

### File sav\Reparation

public Reparation()

public Reparation(String numRep, String etatRep)

public String getEtatRep()

public void setEtatRep(String etatRep)

### File sav\SAV

public Client getClient()

public Demande getDemande()

public Facture getFacture()

public Reparation getReparation()

public SAV()

public SAV(Client client, Demande demande, Reparation reparation, Facture facture)

public void setClient(Client client)

public void setDemande(Demande demande)

public void setFacture(Facture facture)

public void setReparation(Reparation reparation)

### File sav\facture\Facture

public Facture()

public Facture(String numFct, String dateFct, String numEmp)

public String getDateFct()

public void setDateFct(String dateFct)

### File sav\facture\LigneFacture

public LigneFacture()

public LigneFacture(String idLigneFct, String numFct, String mttLigneFct, String desLigneFct)

public String getDesLigneFct()

public String getIdLigneFct()

public String getMttLigneFct()

public void setDseLigneFct(String desLigneFct)

public void setIdLigneFct(String idLigneFct)

public void setMttLigneFct(String mttLigneFct)

### File stock\EntreeStock

public EntreeStock()

public EntreeStock(String idEntr, String qteEnStock, String refProduct, String dateEntree)

public String getDateEntree()

public String getIdEntr()

public String getQteEnStock()

public String getRefProduct()

public void setDateEntree(String dateEntree)

public void setIdEntr(String idEntr)

public void setQteEnStock(String qteStock)

public void setRefProduct(String refProduct)

### File vente\LigneVente

public LigneVente()

public LigneVente(String id,String qteLigne, String refProd, String numVte)

public String getId()

public String getNumVte()

public String getQteLigne()

public void setId(String id)

public void setNumVte(String numVte)

public void setQteLigne(String qteLigne)

### File vente\Vente

public String getDate()

public String getMontantVte()

public Vente()

public Vente(String numVte, String numCli, String numEmp, String date, String montantVte)

public void setDate(String date)

public void setMontantVte(String montantVte)

## Package controller

### File ClientController

public Client getById(String numCli)

public ClientController()

public List<Client> getAllClient()

public List<Client> searchByClientName(String searchName)

public String getNextValId()

public void deleteClient(Client client)

public void insertNewClient(Client client)

public void modifyClient(Client client)

### File EmployeeController

public EmployeeController()

### File ProduitController

public void modifyProduit(Produit produit)

public void modifyProduitList(List<Produit> producList)

public List<Produit> getAllProduit()

public List<Produit> getProductbyClient(String numCli)

public List<Produit> searchByProductNameOrRef(String searchName)

public ProduitController()

public void deleteProduit(String referenceProd)

public void insertNewProduit(Produit produit)

### File sav\DemandeController

public DemandeController()

public List<Demande> getAllDemande()

public void deleteDemande(Demande demande)

public void insertNewDemande(Demande demande)

public void modifyDemande(Demande demande)

### File sav\ReparationController

public Reparation getById(String numRep)

public ReparationController()

public void deleteReparation(String numRep)

public void insertNewReparation(Reparation reparation)

public void modifyReparation(Reparation reparation)

### File sav\SAVController

public List<Produit> getProductbyClient(String numCli)

public List<SAV> getAllSav()

public void createSAV(Client client, Produit produit , String descRep, String natureRep, String numEmp)

public void deleteSAV(Demande demande)

public void modifySAV(Reparation reparation, Demande demande)

### File sav\facture\FactureController

public Facture getById(String numFct)

public FactureController()

public void deleteFacture(String numFct)

public void insertNewFacture(Facture facture)

### File sav\facture\LigneFactureController

public LigneFacture getById(String idLigneFct)

public LigneFacture getByIdFct(String numFct)

public LigneFactureController()

public void deleteLigneFactureByFactureId(String numFct)

public void deleteLigneFactureById(String idLigneFct)

public void insertNewLigneFacture(LigneFacture ligneFacture)

public void updateLigneFacture(LigneFacture ligneFacture)

### File stock\EntreeStockController

public EntreeStockController()

public void insertNewEntree(Produit produit)

### File vente\LigneVenteController

public LigneVenteController()

public void insertNewLigneVente(String numVte, String refProd, String qtt)

### File vente\VenteController

public Vente insertNewVente(Client client, String numEmp, float mttTotal)

public VenteController()

## Package service

### File ClientService

public ClientService()

public List<Client> searchByclientName(String searchName)

public int compare(Client c1, Client c2)

public void deleteClient(String numCli)

public void insertClient(Client client)

public void updateClient(Client client)

### File EmployeeService

public EmployeeService()

public List<Employe> getAllEmploye()

### File IConnectDbService

protected Connection connect()

### File ProduitService

public ProduitService()

public int compare(Produit c1, Produit c2)

public void deleteProduit(String refProd)

public void insertProduit(Produit produit)

public void updateProduit(Produit produit)

### File sav\DemandeService

public DemandeService()

public int compare(Demande d1, Demande d2)

public void insertDemande(Demande demande)

public void updateDemande(Demande demande)

### File sav\ReparationService

public List<Reparation> getAllReparation()

public ReparationService()

public String getNextValid()

public int compare(Reparation r1, Reparation r2)

public void insertReparation(Reparation reparation)

public void updateReparation(Reparation reparation)

### File sav\facture\FactureService

public FactureService()

public Facture getById(String numFct)

public void deleteFacture(String numFct)

public void insertNewFacture(Facture facture)

### File sav\facture\LigneFactureService

public LigneFactureService()

public LigneFacture getById(String idLigneFct)

public LigneFacture getByIdFct(String numFct)

public void deleteLigneFactureByFactureId(String numFct)

public void deleteLigneFactureById(String idLigneFct)

public void insertNewLigneFacture(LigneFacture ligneFacture)

public void updateLigneFacture(LigneFacture ligneFacture)

### File stock\EntreeStockService

public EntreeStockService()

public void insertNewEntree(String nextValId, Produit produit, LocalDateTime now)

### File vente\VenteService

public VenteService()

public boolean insertNewVente(Vente vente)

### File vente\LigneVenteService

public LigneVenteService()

public int compare(LigneVente c1, LigneVente c2)

public void insertNewLigneVente(LigneVente ligneVente)

# Annexes

## Outils

* Maven
* Javafx
* SqLite Database 3

## Environnement de développement

* Eclipse IDE for Enterprise Java and Web Developers - 2021-03
* IntelliJ IDEA 2019.2.4 x64
* JavaFx Scene Builder 2.0