**CENTRE UNIVERSITAIRE - SALHI AHMED - DE NAAMA**
**INSTITUT DES SCIENCES & TECHNOLOGIES**
**DEPARTEMENT DE MATHEMATIQUES ET INFORMATIQUE**

# MÉMOIRE DE FIN D'ETUDES

Pour l'obtention du diplôme

## MASTER EN INFORMATIQUE

Spécialité : Systèmes d'Informations (SI)

## Thème

# Conception et réalisation d'une application Java Android pour la Gestion de stock

**Présenté par :**

- ARBAOUI Siham

**Membres du jury :**

- Mr BOUCHIHA Djelloul **Encadreur**
- Mr BOUAGUADA Benameur **Président**
- Mr BOUZIENE Abdelghani **Examinateur**

**Année Universitaire** : 2017-2018

# Remerciements

Je tiens à exprimer ma gratitude

à mon directeur de recherche monsieur BOUCHIHA Djelloul

qui a accepté de suivre ce travail, je tiens à le remercier

pour ces conseils, ses encouragements, sa disponibilité et son orientation.

Merci également aux membres du jury qui ont accepté d'évaluer mon travail.

Je remercie tous mes enseignants qui ont contribué à ma formation, le long de ces années, trouvent ici l'expression de ma gratitude et de ma reconnaissance.

Enfin, je tiens à remercier tous ceux qui, de près ou de loin, ont contribué à l'aboutissement de ce travail.

# Dédicace

Je dédie ce modeste travail en témoignage de ma reconnaissance :

C'est avec profonde gratitude et sincères mots

Que nous dédions ce modeste de fin d'étude

A mes chers parents, qui ont sacrifie leurs vie pour

notre réussite et nous ont éclairé le chemin par

leurs conseils judicieux

A ma grande mère

A mes frères et mes sœurs

A ma petite famille : Djamel, Ansem ,Mohamed et Arwa

A ma famille ARBOUI

A toute la promotion M2 informatique.

Et à mes chères amis : Fatima, Yakoute, Soumia ,Amina et Fatiha.

Siham

# Table des matières

# Résumé

Le thème abordé dans ce projet consiste à la conception et réalisation d'une application java Android pour la gestion du magasin du centre universitaire SALHI Ahmed de NAAMA.

L'application implémentée prend en charge des tâches importantes dans la gestion des produits. Le magasiner doit réaliser deux tâches principales :

- La réception des produits entrants à partir des fournisseurs, et les enregistrer dans un registre d'entrée et une fiche de stock.
- La livraison des produits sortants aux services demandeurs, et les enregistrer dans un registre de sortie, et une fiche de stock.

Pour réaliser ce travail nous avons utilisé la méthode de conception UML. Pour le langage de programmation, nous avons opté pour Java-Android avec son module de base de données.

Les avantages apportées par notre solution est l'amélioration du service sur deux plans :

L'aspect temporel qui sera amélioré nettement par rapport à l'ancien système, et l'aspect de qualité du service qui se distingue clairement en le comparant l'ancienne méthode de travail utilisée.

Mots clés : Gestion de stock, Java, Android, UML.

# Abstract

This project aims at designing implementing an Android java application for the store management of the Ahmed SALHI University Center of Naama.

The application, when implemented, can greatly support important tasks for the store management. The storekeeper must receive incoming products from suppliers and save them in an entry register and a stock form. He must also deliver outgoing products to the requesting services, and save them in an outgoing register and a stock form.

To carry out this work we used the unified modeling language UML for designing our system. For the programming language, we opted for Java-Android with its basic data module.

The advantages brought by our solution is the improvement of the service on two levels: the temporal aspect which will be improved clearly compared to the performances of the old system, and the quality of the service aspect which is clearly distinguished in comparison with the old working method.

**Keywords:** Store Management, Java, Android, UML

# ملخص


يتناول هدا المشروع تصميم وانجاز تطبيق جافا أندرويد لتسيير مخزن المركز الجامعي صالحي أحمد بالنعامة

.

من المهام الاساسية التي يقوم بها هدا التطبيق في تسيير البضائع ان العامل يتوجب عليه القيام بمهمتين اساسيتين:

استقبال البضائع الواردة من قبل الموزعين وتدوينها في سجل الواردات وكذلك في بطاقة المخزون.

توزيع البضائع المطلوبة الى مختلف المصالح المركز الجامعي ثم تدوينها في سجل الصادرات وبطاقة المخزون.

لقد استعملنا لتصميم هذا العمل UML أما فيما يتعلق بلغة البرمجة فقد اعتمدنا على لغة جافا اندرويد

من المحاسن التي يتميز بها هذا التطبيق هو تحسين نوعية الخدمات من جهتين

-تحسين الجانب الزمني الذي يتجلى بوضوح بالمقارنة بالتصميم القديم

-تحسين نوعية الخدمات التي تختلف بشكل واضح بالمقارنة مع طريقة العمل المستعملة سابقا



الكلمات المفتاحية : تسيير المخزون ،جافا ،أندرويد،UML.

# Liste des figures

# Liste des tableaux

# Acronymes

| Notion | Signification |
|---|---|
| **UML** | Unified Modeling Language |
| **OMT** | Object Modeling Technique |
| **OOSE** | Object Oriented Software Engineering |
| **OMG** | Object Management Group |
| **OOA** | Object-oriented analysis |
| **OOD** | Object-oriented design |
| **IBM** | International Business Machines |
| **HP** | Hewlett-Packard |
| **OS** | Operating System |
| **OHA** | Open Handset Alliance |
| **SDK** | Software Development kit |
| **IOS** | Internetwork Operating System |
| **API** | Application Programming Interface |
| **PC** | Personal Computer |
| **AVD** | Android Virtual Device |
| **NumE** | Numéro de l'état de besoin |
| **CCP** | Compte de Code Postal |
| **NTel** | Numéro de Téléphone |
| **NumFiche** | Numéro de fiche de stock |
| **RIB** | Le Relevé d'Identité Bancaire |
| **NumA** | Identifiant du Produit |
| **NumF** | Numéro du Fournisseur |
| **NEB** | Numéro de l'Etat de Besoin |
| **NumB** | Numéro du Bon de Livraison |

# Introduction générale

Vu les avantages d'automatisation des systèmes d'information, l'exploitation des technologies avancées de traitement de l'information est devenue une nécessité dans la gestion des entreprises.

Par ailleurs, le développement d'une entreprise en matière de technologies de l'information exige de nouveaux moyens et supports pour échanger et diffuser l'information dans le but de réduire les contraintes de temps, d'espace et du coût, et facilite la gestion de l'entreprise.

L'objectif de ce travail de recherche est d'intégrer les technologies actuelles (Smartphones, tablettes,…etc.) pour faciliter les tâches du personnels (comptable, magasinier et les agents de saisies) du service magasin au niveau du centre universitaire **Salhi Ahmed-Naama**. Après avoir constaté que l'ensemble des traitements effectués dans ce service se fait de manière manuelle. Ce qui engendre un certain nombre de problèmes tels que la lenteur dans l'accès aux données et le risque de perte d'informations. Nous avons implémenté une application Java Android qui permettra d'automatiser et par conséquent faciliter et accélérer les tâches des agents du magasin, par une application accessible à partir de leur mobile.

En effet, nous avons répartit notre mémoire en quatre chapitres principaux : on commence par quelques généralités au premier chapitre intitulé «**UML**», comprenant une approche globale de la méthode utilisée. Le second chapitre intitulé «**Android**» fera l'objet de la description de l'environnement du travail. Une conception détaillée de notre système est présentée dans le troisième chapitre intitulé « **Conception du système** » : on commencera par le diagramme de cas d'utilisation et le diagramme de classes, et on passera par la suite aux diagrammes d'activités et diagrammes de séquence. Nous terminerons par la partie réalisation dans le dernier chapitre intitulé « **Implémentation** », où nous présenterons un aperçu de l'application ainsi quelques interfaces de notre application. Enfin, on clôture notre mémoire par une conclusion et des perspectives.

# Chapitre I:

# UML

## 1. Introduction

Dans le cadre de ce chapitre, nous allons définir quelques généralités portant sur la méthode et outils mettant en évidence la réalisation de notre projet. Nous allons commencer par présenter le langage de modélisation UML (Unified Modeling Language)

## 2. Historique

Pour aller plus loin dans le rapprochement, James Rumbaugh et Grady Booch se sont retrouvés au sein de la société Rational Software, et ont été ensuite rejoints par Ivar Jacobson en se donnant comme objectif de fusionner leur méthode et créer UML (Unified Methode Language).
UML est donc une norme du langage de modélisation objet qui a été publiée, dans sa première version, en novembre 1997 par l'OMG (Object Management Group), instance de normalisation internationale du domaine de l'objet (Joseph Gabayet David Gabay, 2008).
Les grandes étapes de la diffusion d'UML peuvent se résumer dans le tableau suivant :

**Tableau I.1 :** Historique d'UML

| Période | Description |
|---|---|
| Dans les années 80 | Les méthodes utilisées pour organiser la programmation impérative (notamment Merise). |
| Début des années 90 | La nécessité d'une méthode qui lui soit adaptée devient évidente. |
| Entre 1994 et 1995 | rapprochement des méthodes OMT, BOOCH et OOSE et naissance de la première version d'UML. |
| En 1995 | Booch et Rumbaugh (et quelques autres) se sont mis d'accord pour construire une méthode unifiée, Unified Method 0.8. |
| En 1996 | Jacobson les a rejoints pour produire UML 0.9 (notez le remplacement du mot méthode par le mot langage, plus modeste). Les acteurs les plus importants dans le monde du logiciel s'associent alors à l'effort (IBM, Microsoft, Oracle, DEC, HP, Rational, Unisys, etc.) et UML 1.0 est soumis à l'OMG. |
| 23 novembre 1997 | version 1.1 d'UML adoptée par l'OMG. |
| Entre 1998-1999 | sortie des versions 1.2 à 1.3 d'UML |
| Entre 2000-2001 | sortie des dernières versions suivantes 1.x. |
| Entre 2002-2003 | préparation de la V2. |
| 10 octobre 2004 | sortie de la V2.1 |
| 5 février 2007 | La version d'UML en 2008 est UML 2.1.1 et les travaux d'amélioration se poursuivent. |

## 3. Points forts d'UML

UML est un langage formel et normalisé, Il permet ainsi :

1. Gain de précision
2. Gage de stabilité
3. Utilisation d'outils

UML est un support de communication performant. Il cadre l'analyse et facilite la compréhension de représentations abstraites complexes. Son caractère polyvalent et sa souplesse lui ont font un langage universel.

## 4. Définition

UML (Unified Modeling Language) est un langage de modélisation orienté objet développé en réponse à l'appel à propositions lancé par l'OMG (Object Management Group) dans le but de définir une notation standard pour la modélisation des applications construites à l'aide d'objets

UML est une méthode utilisant une représentation graphique. L'usage d'une représentation graphique est un complément excellent à celui de représentions textuelles. En effet, l'une comme l'autre sont ambigues mais leur utilisation simultanée permet de diminuer les ambigüités de chacune d'elles. Un dessin permet bien souvent d'exprimer clairement ce qu'un texte exprime difficilement et un bon commentaire permet d'enrichir une figure.

## 5. Les différents types de diagrammes UML

Il existe 2 types de vues du système qui comportent chacune leurs propres diagrammes :

- Vues statiques (diagrammes structurels) :
  - diagrammes d'objets
  - diagrammes de classes
  - diagrammes de composants
  - diagrammes de déploiement
  - diagrammes des structures composites
  - diagrammes de paquetages
- Vues dynamiques (diagrammes de comportement) :
  - diagrammes de collaboration
  - diagrammes de séquence
  - diagrammes d'états-transitions
  - diagrammes d'activités
  - diagrammes de cas d'utilisation
  - diagrammes de temps
  - diagrammes de communications

La figure suivante montre les deux types de diagrammes : diagrammes structurels et diagrammes de comportement :

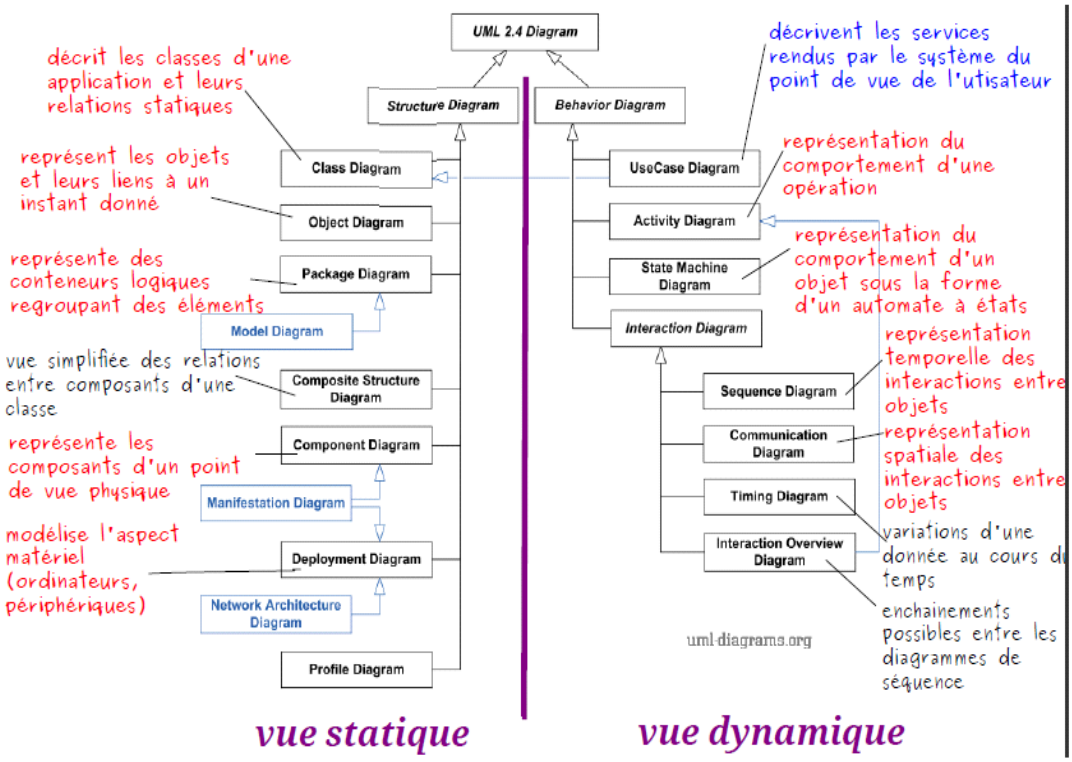



**Figure I.1 :** Les diagrammes UML (Olivier Augereau, 2012)

## 6. Les diagrammes statiques

Le but de la conceptualisation est de comprendre et structurer les besoins du client. Il ne faut pas chercher l'exhaustivité, mais clarifier, filtrer et organiser les besoins.

### 6.1. Diagramme de classes

En effet, ce diagramme permet de donner la représentation statique du système à développer. Cette représentation est centrée sur les concepts de classe et d'association. Chaque classe se décrit par les données et les traitements dont elle est responsable, pour elle-même et vis-à-vis des autres classes. Les traitements sont matérialisés par des opérations (Joseph Gabayet David Gabay)..
La description du diagramme de classe est fondée sur :

- le concept d'objet,
- le concept de classe comprenant les attributs et les opérations,
- et les différents types d'association entre classes.

Un diagramme de classes se compose donc de :

☞ **Objet**

Un objet est un concept, une abstraction ou une chose qui a un sens dans le contexte du système à modéliser. Chaque objet a une identité et peut être distingué des autres sans considérer a priori les valeurs de ses propriétés.

Un objet est caractérisé par les valeurs de ses propriétés qui lui confèrent des états significatifs suivant les instants considérés. Le formalisme de représentation d'un objet est donné après celui d'une classe.

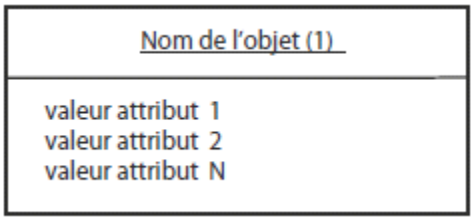



**Figure I.2 :** Formalisme de représentation d'un objet.

☞ **Classe**

Une classe décrit un groupe d'objets ayant les mêmes propriétés (attributs), un même comportement (opérations), et une sémantique commune (domaine de définition).
La classe représente l'abstraction de ses objets. Au niveau de l'implémentation, c'est-à-dire au cours d'exécution d'un programme, l'identificateur d'un objet correspond à une adresse mémoire. Une classe se représente à l'aide d'un rectangle comportant plusieurs compartiments. Les trois compartiments de base sont :

- la désignation de la classe,
- la description des attributs,
- et la description des opérations.

Deux autres compartiments peuvent être aussi indiqués :

- la description des responsabilités de la classe
- et la description des exceptions traitées par la classe.

La Figure I.3 montre le formalisme général des compartiments d'une classe :

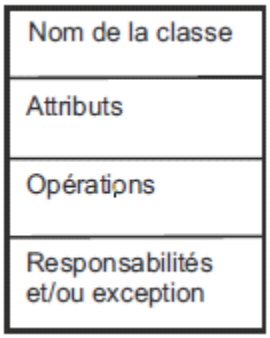



**Figure I.3 :** Formalisme de représentation d'une classe.

☞ **Attribut**

Un attribut est une propriété élémentaire d'une classe. Pour chaque objet d'une classe, l'attribut prend une valeur (sauf cas d'attributs multi-valués).

Les attributs d'une classe se représentent comme suit :

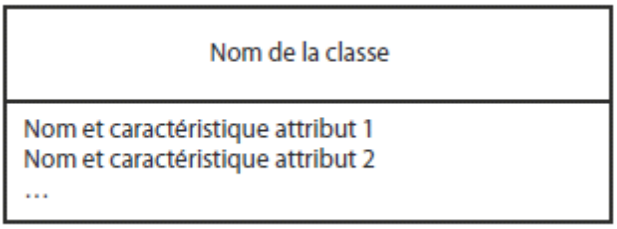



**Figure I.4 :** Formalisme d'attributs de classe

Un attribut se décrit comme suit :

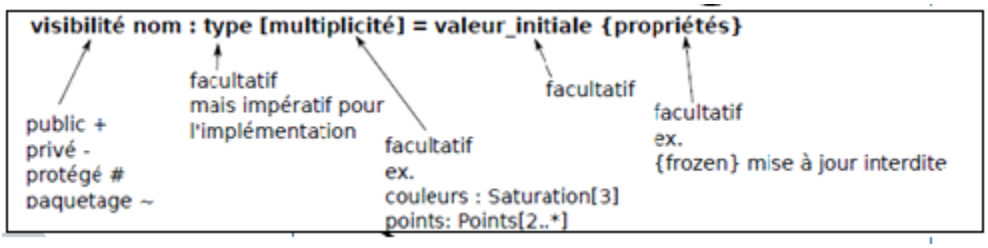



**Figure I.5 :** Format de description d'un attribut

☞ **Opération**

Une opération est une fonction applicable aux objets d'une classe. Une opération permet de décrire le comportement d'un objet.
Voici le format de description d'une opération :

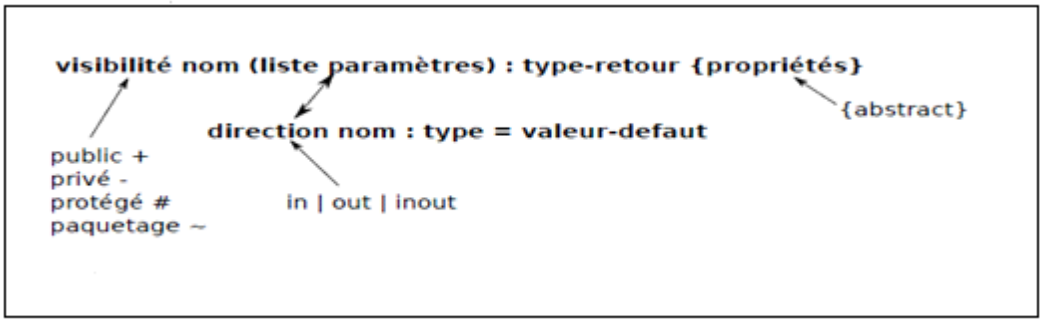



**Figure 1.6 :** Format de description d'une opération

☞ **Les relations**

Les différents types de relations qui peuvent exister dans un diagramme de classes sont :

**a) Association**

- Relation entre classes.
- Décrit les connexions structurelles entre les instances des classes associées.

On distingue

- Association binaire : elle relie deux classes.

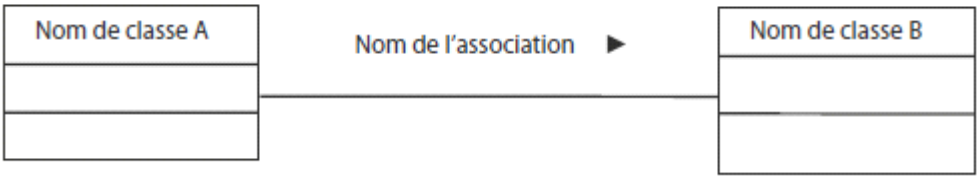



**Figure 1.7 :** Notations d'une association binaire

- Association n-aire : une association de dimension supérieure à 2 se représente en utilisant un losange permettant de relier toutes les classes concernées.

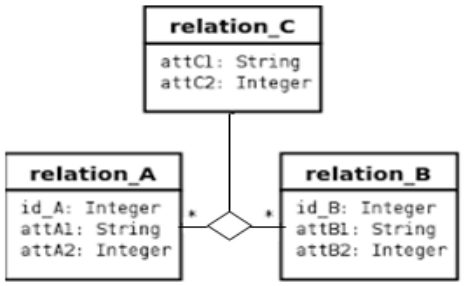



**Figure 1.8 :** Notations d'une association n-aire

### b) Classe-association

Une classe-association permet de décrire soit des attributs soit des opérations propres à l'association. Cette classe-association est elle-même reliée par un trait en pointillé au losange de connexion. Une classe-association peut être reliée à d'autres classes d'un diagramme de classes.

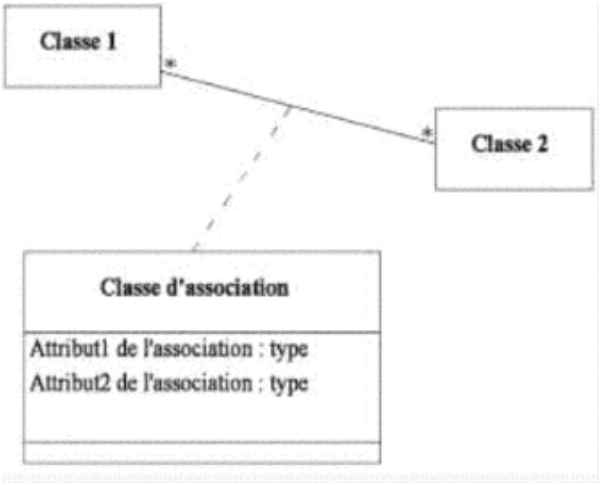



**Figure 1.9 :** Notations d'une class association

### c) Agrégation

Une agrégation est une association qui représente une relation d'inclusion structurelle ou comportementale d'un élément dans un ensemble

La Figure I.10 donne le formalisme général de l'agrégation.

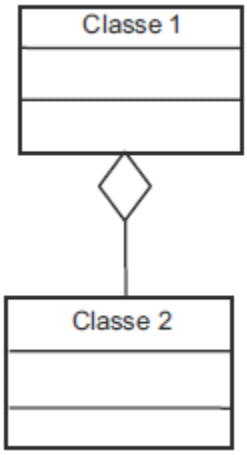



**Figure 1.10 :** Formalisme général de l'agrégation.

### d) Composition

La composition est un cas particulier de l'agrégation dans laquelle la vie des composants est liée à celle des agrégats. Elle fait souvent référence à une contenance physique. Dans la composition, l'agrégat ne peut être multiple.

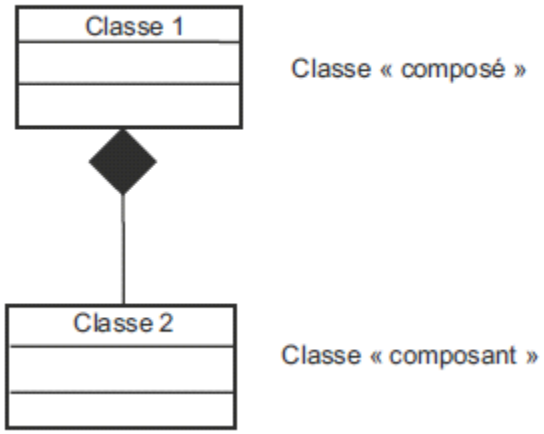



**Figure 1.11 :** Formalisme de la composition

### e) Généralisation/Spécialisation et l'héritage simple

La généralisation est la relation entre une classe et deux autres classes ou plus, partageant un sous-ensemble commun d'attributs et/ou d'opérations.

La classe qui est affinée s'appelle super-classe, les classes affinées s'appellent sous-classes. L'opération qui consiste à créer une super-classe à partir de classes s'appelle la généralisation. Inversement la spécialisation consiste à créer des sous-classes à partir d'une classe (Joseph Gabayet David Gabay,2008).

La Figure 1.12 montre le formalisme de la généralisation-spécialisation.

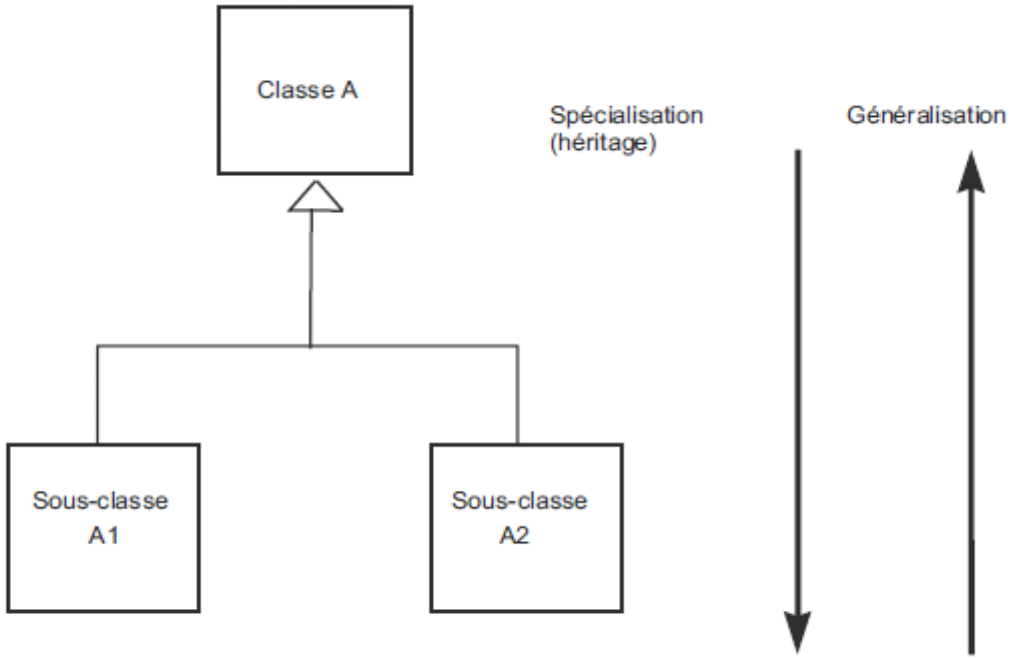



**Figure 1.12 :** Formalisme de la Généralisation-Spécialisation

- la sous-classe A1 hérite de A, c'est une spécialisation de A .
- la sous-classe A2 hérite de A, c'est une spécialisation de A.

L'héritage permet à une sous-classe de disposer des attributs et opérations de la classe dont elle dépend.

### f) Relation de dépendance

Les relations de dépendance sont utilisées lorsqu'il existe une relation sémantique entre plusieurs éléments qui n'est pas de nature structurelle. Une relation de dépendance définit une relation unidirectionnelle entre un élément source et un élément cible.

Une dépendance est une relation entre deux éléments de modélisation dans laquelle toute modification effectuée sur un élément de modélisation (l'élément influent) affecte l'autre élément (élément dépendant).

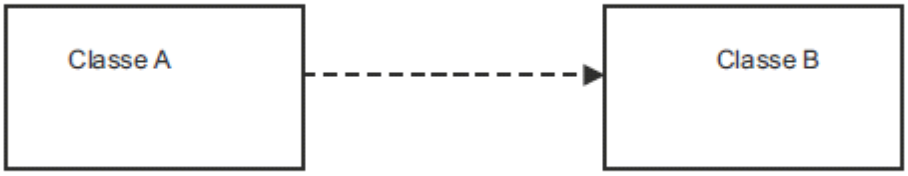



**Figure 1.13 :** Formalisme de représentation d'un lien de dépendance

### g) L'interface

Une interface définit le comportement visible d'une classe. Ce comportement est défini par une liste d'opérations ayant une visibilité « public ». Aucun attribut ou association n'est défini pour une interface.
Une interface est en fait une classe particulière (avec le stéréotype « interface » (J. Steffe, 2016).

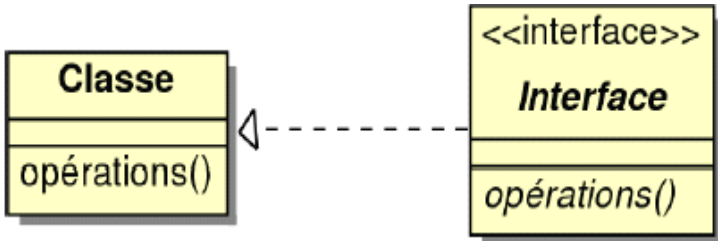



**Figure 1.14 :** Formalisme de représentation de l'interface

### h) La multiplicité

Elle indique un domaine de valeurs pour préciser le nombre d'instances d'une classe vis-à-vis d'une autre classe pour une association donnée. La multiplicité peut aussi être utilisée pour d'autres usages, comme par exemple un attribut multi-valué.

### i) Qualification

La qualification d'une relation entre deux classes permet de préciser la sémantique de l'association et de qualifier de manière restrictive les liens entre les instances. Seules les instances possédant l'attribut indiqué dans la qualification sont concernées par l'association. Cet attribut ne fait pas partie de l'association.

## 6.2. Diagramme d'objet

Le diagramme d'objets permet de mettre en évidence des liens entre les objets. Les objets instances de classes, sont reliés par des liens, instances d'associations.
Le diagramme d'objets utilise les mêmes concepts que le diagramme de classes. Ils sont essentiellement utilisés pour comprendre ou illustrer des parties complexes d'un diagramme de classe.

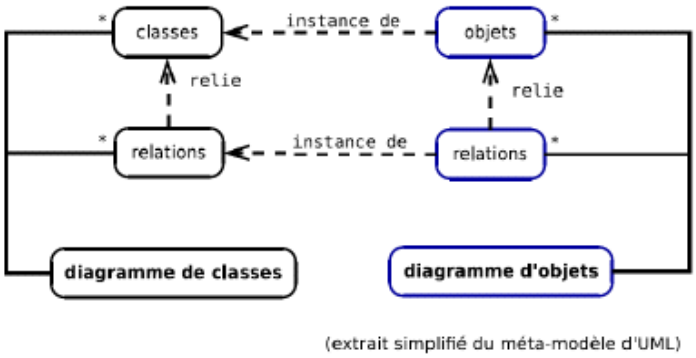



**Figure I.15 :** Extrait simplifié du méta-modèle d'UML

Le diagramme d'objets est utilisé pour :

- Préciser une structure complexe trop difficile à comprendre avec un diagramme de classes.
- Illustrer un modèle de classes en montrant un exemple qui l'explique
- Préciser certains aspects du système.
- Exprimer une exception en modélisant des cas particuliers ou des connaissances non généralisables.
- Prendre une image (snapshot) d'un système à un moment donnée.
- Documenter des cas de test, analyser des exemples.
- Montrer un contexte (collaborations sans messages) (Laetitia Matignon, 2012).

### 6.3. Diagramme de composant

Le diagramme de composant permet de :

- représenter les composants logiciels d'un système ainsi que les liens existant entre ces composants.
- Structurer une architecture logicielle à un niveau de granularité moindre que les classes.
- Spécifier l'intégration de briques logicielles tierces (composants EJ CORBA, COM+ ou .Net, WSDL...).
- Identifier les composants réutilisables.

Un composant est un espace de noms qu'on peut employer pour organiser les éléments de conception, les cas d'utilisation, les interactions et les artefacts de code.

### 6.4. Diagramme de déploiement

Le diagramme de déploiement permet de représenter l'architecture physique supportant l'exploitation du système. Cette architecture comprend des nœuds correspondant aux supports physiques (serveurs, routeurs…) ainsi que la répartition des artefacts logiciels (bibliothèques, exécutables…) sur ces nœuds. C'est un véritable réseau constitué de nœuds et de connexions entre ces nœuds qui modélisent cette architecture

**Tableau I.2 :** Eléments d'un diagramme de déploiement

| Elément | Définition |
|---|---|
| Nœud   | Un nœud correspond à une ressource matérielle de traitement sur laquelle des artefacts seront mis en œuvre pour l'exploitation du système. Les nœuds peuvent être interconnectés pour former un réseau d'éléments physiques. |
| Artefact | la spécification d'un élément physique qui est utilisé ou produit par le processus de développement du logiciel ou par le déploiement du système. |

### 6.5. Le diagramme de structure composite

Un diagramme de structure composite est un diagramme qui affiche la structure interne d'un classificateur, y compris ses points d'interaction avec d'autres parties du système.

Une structure composite est un ensemble d'éléments interconnectés collaborant dans un but commun lors de l'exécution d'une tâche. Chaque élément se voit attribuer un rôle dans la collaboration.

Le diagramme des structures composites est apparu dans la spécification d'UML 2.0. Les éléments clés du diagramme de structure composite sont les classifieurs structurés, les parties, les ports, les connecteurs et les collaborations :

- Un classifieur structuré représente une classe, dans la plupart des cas une classe abstraite, dont le comportement peut être décrit complètement ou partiellement par le biais d'interactions entre parties. Un classifieur encapsulé est une forme de classifieur structuré contenant des ports.
- Le port est un point d'interaction qui peut être utilisé pour connecter un classifieur structuré avec ses parties ou son environnement. Les ports peuvent accessoirement spécifier les services qu'ils fournissent ainsi que les services qu'ils peuvent requérir d'autres parties du système. Les ports sont symbolisés par un carré sur le diagramme. Les ports peuvent déléguer les requêtes reçues à des parties internes ou au contraire les délivrer directement à la partie qui possède le port en question. Les ports ayant un statut public sont dessinés à cheval sur la bordure de la partie. À l'inverse, les ports protégés (non visibles par l'environnement) sont contenus dans la partie.
- Les connecteurs relient plusieurs entités, leur permettant d'interagir entre elles lors de l'exécution. Un connecteur est représenté par une ligne reliant une combinaison de parties, de ports ou de classifieurs structurés.
- Une collaboration est en général d'un niveau d'abstraction plus élevé qu'un classifieur. Elle est représentée par un ovale en pointillé contenant les rôles joués par chaque instance dans la collaboration lors de l'exécution.


**Figure I.16 :** Exemple de représentation de collaboration d'instances par un diagramme de structure composite

## 6.6. Diagramme de paquetages

Un paquetage est un conteneur logique permettant de regrouper et d'organiser les éléments dans le modèle UML. Le Diagramme de paquetage sert à représenter les dépendances entre paquetages, c'est-à-dire les dépendances entre ensembles de définitions.
Il est possible de représenter les éléments du système appartenant au paquetage :


**Figure I.17 :** Les éléments du système appartenant au paquetage

La dépendance entre paquetages peut être qualifiée par un niveau de visibilité qui est soit public soit privé. Par défaut le type de visibilité est public.
À chaque type de visibilité est associé un lien de dépendance. Les deux types de dépendances entre paquetages sont :

- Dépendance de type « import » : Ce type de dépendance permet, pour un paquetage donné, d'importer l'espace de nommage d'un autre paquetage. Ainsi tous les membres du paquetage donné ont accès à tous les noms des membres du paquetage importé sans avoir à utiliser explicitement le nom du paquetage concerné. Ce type de dépendance correspond à un lien ayant une visibilité « public ».

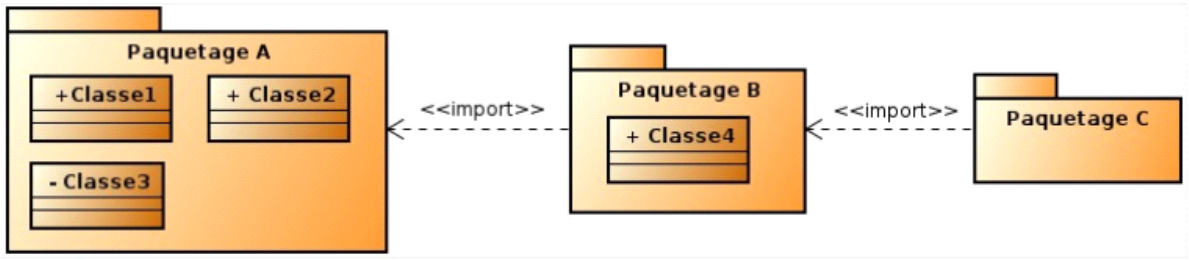



**Figure I.18 :** Dépendance de type « import »

Le paquetage B importe Classe1 et Classe2 (pas Classe3 qui a une visibilité de type privée). Classe1 et Classe2 ont une visibilité de type public dans paquetage B. Le paquetage C importe Classe1, Classe2 et Classe4

- Dépendance de type «access» : Ce type de dépendance permet, pour un paquetage donné, d'avoir accès à l'espace de nommage d'un paquetage cible. L'espace de nommage n'est donc pas importé et ne peut être transmis à d'autres paquetages par transitivité. Ce type de dépendance correspond à un lien ayant une visibilité « privé ». La dépendance de type « access » est représentée par une flèche pointillée muni du stéréotype <<access>>.

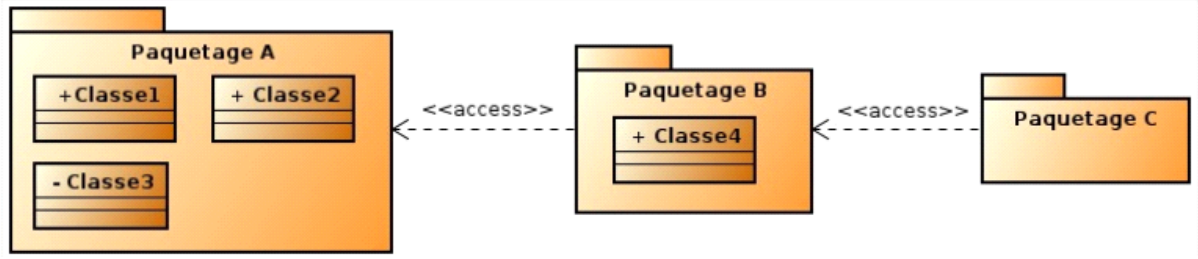



**Figure I.19 :** Dépendance de type « access »

## 7. Diagrammes de comportement (diagrammes dynamiques)

Les diagrammes comportementaux sont focalisés sur la description de la partie dynamique du système à modéliser.

### 7.1. Diagramme de cas d'utilisation

Les cas d'utilisation ont été définis initialement par Ivar Jacobson en 1992 dans sa méthode OOSE. Les cas d'utilisation constituent un moyen de recueillir et de décrire les besoins des acteurs du système. Ils peuvent être aussi utilisés ensuite comme moyen d'organisation du développement du logiciel, notamment pour la structuration et le déroulement des tests du logiciel( Joseph Gabay et David Gabay ,2008).

La représentation d'un cas d'utilisation met en jeu trois concepts : l'acteur, le cas d'utilisation et l'interaction entre l'acteur et le cas d'utilisation :

- L'acteur : La première étape de modélisation consiste à définir le périmètre du système, à

définir le contour de l'organisation et à le modéliser. Toute entité, qui est en dehors de cette organisation, et qui interagit avec elle, est appelée acteur selon UML. Un acteur représente un rôle joué par une personne ou une chose qui interagit avec le système. (la même personne physique peut donc être représentée par plusieurs acteurs en fonction des rôles qu'elle joue). Pour identifier les acteurs, il faut donc se concentrer sur les rôles joués par les entités extérieures au périmètre. Il existe 4 catégories d'acteurs :

- ☞ les acteurs principaux : les personnes qui utilisent les fonctions principales du système
- ☞ les acteurs secondaires : les personnes qui effectuent des tâches administratives ou de maintenance.
- ☞ le matériel externe : les dispositifs matériels incontournables qui font partie du domaine de l'application et qui doivent être utilisés.
- ☞ les autres systèmes : les systèmes avec lesquels le système doit interagir.

- Le cas d'utilisation : Le cas d'utilisation (ou use case) correspond à un objectif du système, motivé par un besoin d'un ou plusieurs acteurs. L'ensemble des use cases décrit les objectifs (le but) du système.
- La relation : Elle exprime l'interaction existant entre un acteur et un cas d'utilisation.

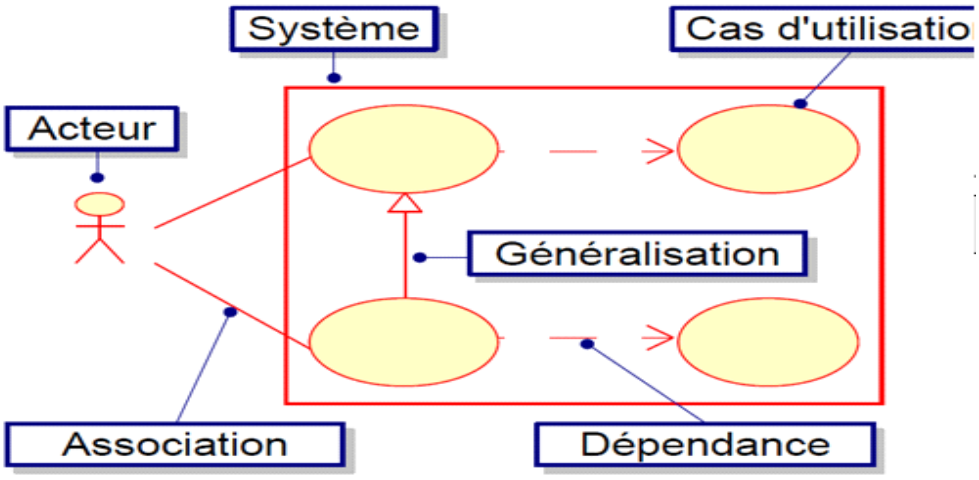



**Figure I.20 :** Notations du diagramme de cas d'utilisation

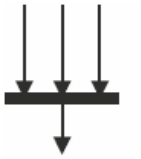
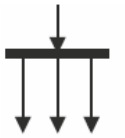
### 7.2. Diagramme d'activité

Les diagrammes d'activités permettent de représenter graphiquement le comportement d'une méthode ou l'exécution d'un scénario de cas d'utilisation.

La représentation des diagrammes d'activités est proche de celle des diagrammes d'états transitions, mais leur sémantique diffère.

Un diagramme d'activité a toujours un et un seul point de départ (initial) mais peut avoir plusieurs points finaux (terminaux).

**Tableau I.3 :** Notations du diagramme d'activité

| Elément/Notation | Description |
|---|---|
| Début  | Point de départ |
| Action/Activité | fait quelque chose. Une transition automatique se déclenche à son achèvement.<br>Une transition prend en charge la modélisation du flot de contrôle. |
| Transition | Une transition représente le passage d'une activité à une autre. |
| Nœud d'objet | L'objet produit ou utilisé par des actions/activités permet de modéliser les flots de données ou des flots d'objets. |
| Nœud de jonction (synchronisation)  | permet à partir de plusieurs flots concurrents en entrée de la synchronisation, de produire un flot unique sortant |
| Nœud de bifurcation (fourche)  | permet à partir d'un flot unique entrant de créer plusieurs flots concurrents en sortie de la barre de synchronisation. |
|  | nœud de fin flot (état de sortie) |
| Nœud de test-décision | permet de faire un choix entre plusieurs flots sortants en fonction des conditions de garde de chaque flot. |
| Etat terminal | fin du processus |

### 7.3. Diagramme d'interaction

Le diagramme global d'interaction ou diagramme d'interactivité est un diagramme UML utilisé pour rendre compte de l'organisation spatiale des participants à l'interaction.

Les diagrammes globaux d'interaction définissent des interactions par une variante de diagrammes d'activité, d'une manière qui permet une vue d'ensemble de flux de contrôle.

Ils se concentrent sur la vue d'ensemble de flux de contrôle où les nœuds sont des interactions ou InteractionUses. Les lignes de vie et les messages n'apparaissent pas à ce niveau de vue d'ensemble.

Le diagramme global d'interaction utilise les concepts du diagramme d'activité auquel on ajoute deux compléments :

- Les fragments d'interaction du diagramme de séquence : Il s'agit comme le montre la Figure I.21 de la notion de fragment d'interaction.

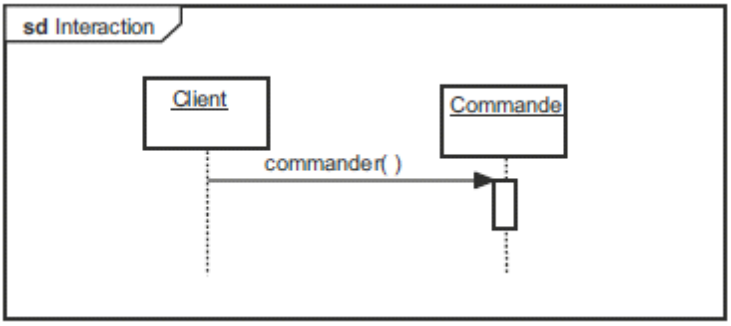



**Figure I.21 :** Exemple de fragment d'interaction

- Les utilisations de fragments d'interaction : Il est aussi possible de faire appel à des fragments d'interaction à l'aide de l'opérateur ref comme le montre la Figure I.22


**Figure I.22 :** Exemple de fragment d'interaction avec l'opérateur ref

### 7.4. Diagramme de séquence

Le diagramme de séquence permet de représenter les interactions entre objets en indiquant la chronologie des échanges.

Les diagrammes de séquence permettent de modéliser les interactions entre les objets :

- Communications synchrones (flèches pleines).
- Communications asynchrones (flèches pointillés).
- Les données transmises entre les objets (annotation des flèches).

Ils permettent aussi de connaître pour chaque objet :

- Ses dates de création et de destruction. (Figure I.23)
- Sa durée de vie, si la création est externe au diagramme (ligne verticale en pointillés).

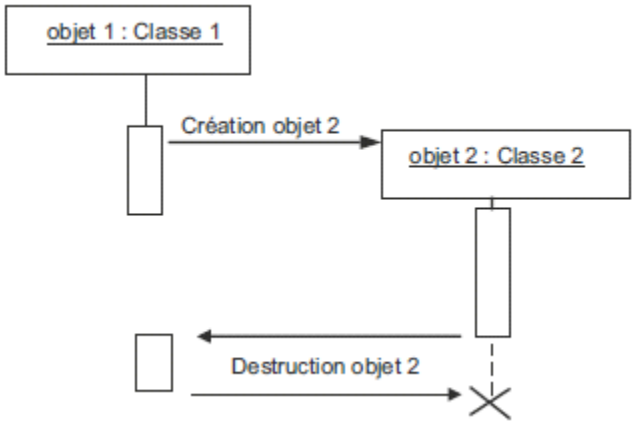



**Figure I.23 :** Exemple type de création et de destruction d'objet

La figure suivante illustre le formalisme général du diagramme de séquence :

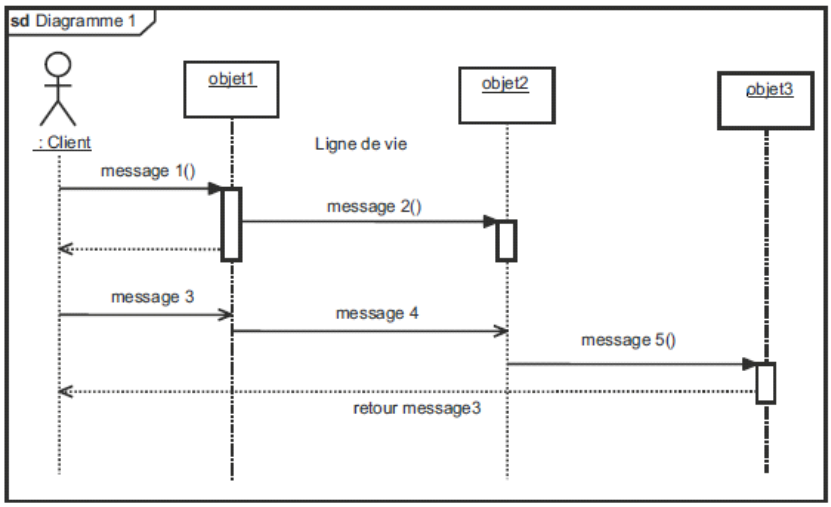



**Figure I.24 :** Formalisme du diagramme de séquence

## 7.5. Diagramme de communication

Le diagramme de communication constitue une autre représentation des interactions que celle du diagramme de séquence. En effet, le diagramme de communication met plus l'accent sur l'aspect spatial des échanges que l'aspect temporel (Joseph Gabay, David Gabay.2008).

Le diagramme de communication utilise trois types de concepts :

- des instances de classes (objets du système étudié) qui interviennent lors du scénario,
- des acteurs qui reçoivent ou émettent des événements lors du scénario,
- et les appels aux méthodes ou flux d'information vers un acteur externe (C. Johnen, 2017).

Chaque participant à un échange de message, correspondant à une ligne de vie dans le diagramme de séquence, se représente sous forme d'un rôle dans le diagramme de communication.

Un rôle est identifié par : <nom de rôle> : <nom du type>
Les rôles correspondent à des objets. Le lien entre les rôles est représenté par un trait matérialisant le support des messages échangés. La Figure I.25 donne le formalisme de base du diagramme de communication.


**Figure I.25 :** Formalisme de base du diagramme de communication

### 7.6. Diagramme d'états-transitions

Le diagramme d'états-transitions décrit le comportement interne d'un objet. Il présente les séquences possibles d'états et d'actions qu'une instance de classe peut traiter au cours de son cycle de vie, en réaction à des événements discrets (de type signaux, invocations de méthode).

**Tableau I.4 :** Notations du diagramme d'activité

| Elément/Notation | Description |
|---|---|
|   | Abstraction d'un moment de la vie d'une entité pendant lequel elle satisfait un ensemble de conditions. |
| État composite  | état décomposé en régions contenant chacune un ou plusieurs sous-état(s). |
|  | Changement d'état. |
|  | Point de choix |
|  | Point de sortie (particulier) |
|  | État initial (à sa création, un objet entre dans un pseudo-état : l'état initial) |
|  | État final est un pseudo-état qui indique la fin d'un diagramme d'états |

### 7.7. Diagramme de temps

Le diagramme de temps permet de représenter les états et les interactions d'objets dans un

contexte où le temps a une forte influence sur le comportement du système à gérer. Autrement dit, le diagramme de temps permet de mieux représenter des changements d'états et des interactions entre objets liés à des contraintes de temps.

Pour cela, le diagramme de temps utilise en plus des lignes de vie, les concepts suivants :

- Des états ou des lignes de temps conditionnées avec deux représentations graphiques possibles.
- Des représentations propres aux aspects temporels : échelle de temps, contrainte de durée, événements… (Joseph Gabay, David Gabay,2008).

La figure suivante représente un exemple illustratif d'un diagramme de temps, qui décrit les variations d'une donnée au cours du temps.

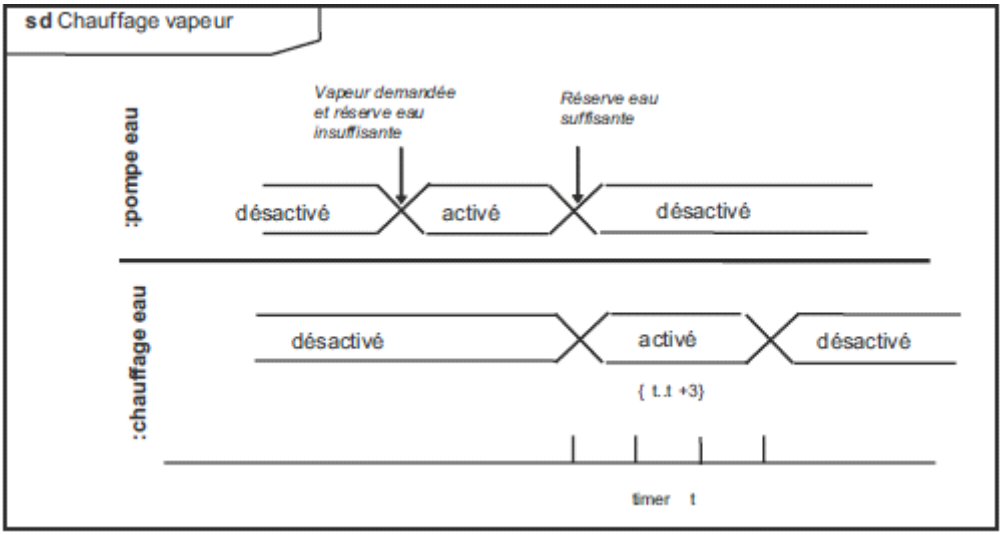



**Figure I.26 :** Exemple de diagramme de temps avec une représentation linéaire

## 8. Conclusion

Dans ce premier chapitre, nous avons présenté le langage de modélisation UML et ses différents diagrammes, passant maintenant au chapitre suivant concernant l'Android.

# Chapitre II:
# Android

## 1. Introduction

Avec l'explosion des ventes de Smartphones ces dernières années, Android a pris une place importante dans la vie quotidienne ;
Android est un OS (Operating System ou System d'exploitation Open Source pensé pour les téléphones mobiles et développé par l'Open Handset Alliance (OHA) sous l'autorité de Google .L'OHA est composé d'environ 80 sociétés telles que Samsung, HTC, SFR, Orange, Asus, Qualcomm

## 2. Historique

L'histoire d'Android commence en octobre 2003, lorsqu' Andy Rubin, Rich Miner,Nick Sears et Chris White créent la société Android à Palto Alto(Californie).
Google a racheté la société en Aout 2005.Deux ans plus tard, l'open handset Alliance est annoncée et Androïde devient officiellement Open source.

La première version du SDK Android1.0 sert en 2008 avec le 1er téléphone sous Android(HTC Dream).

En avril 2009, la version 1.5 (API 3) d'Android sort. Cette version baptisée Cupcake (petit gâteau) inaugure les nouveaux noms des versions d'Android.
Android a connu plusieurs mises à jour depuis sa première version. Ces mises à jour servent généralement à corriger des bugs et à ajouter de nouvelles fonctionnalités, le tableau suivant résume quelques versions de l'android : (SENOUSSAOUI & LAKEHAL, 2013)

Tableau **II.1** : Historique d'Android

| La version | La date d'apparition |
|---|---|
| Android 1.0 | Septembre 2008 |
| Android 1.1 | Février 2009 |
| Android 1.5 dit « Cupcake » | Avril 2009 |
| Android 1.6 dit « Donut » | Septembre 2009 |
| Android 2.0 dit « Eclair » | Octobre 2009 |
| Android 2.0.1 | Décembre 2009 |
| Android 2.1, 2.2, 2.3 | Janvier - Décembre 2010 |
| Android 2.3.3, 2.3.4, 2.3.5, 2.3.6,2.3.7 | Février-septembre 2011 |
| Android 3.0 | Février 2011 |
| Android 3.1 | Mai 2011 |
| Android 3.2 | JUILLET 2011 |
| Android 3.2.1, 3.2.2 | Septembre 2011 |
| Android 4.0 | Octobre2011 |
| Android 5.0 | Octobre2014 |
| Android 5.1 | Mars 2015 |
| Android 6.0 | Octobre 2015 |
| Android 7.0 | Août 2017 |

## 3. La philosophie et les avantages d'Android

Le contrat de licence pour Android respecte l'idéologie open-source, c'est-à-dire que vous pouvez à tout moment télécharger les sources et les modifier selon vos goûts

***Gratuit (ou presque)***

Android est gratuit, autant pour vous, que pour les constructeurs.

***Facile à développer***

Toutes les API mises à disposition facilitent et accélèrent grandement le travail. Ces APIs sont très complètes et très faciles d'accès.

***Facile à vendre***

Le Play Store (anciennement Android Market) est une plateforme immense et très visitée ; c'est donc une mine d'opportunités pour quiconque possède une idée originale ou utile.

***Flexible***

Le système est extrêmement portable, il s'adapte à beaucoup de structures différentes. Les smartphones, les tablettes.

***Ingénieux***

L'architecture d'Android est inspirée par les applications composites, et encourage par ailleurs leur développement. Ces applications se trouvent essentiellement sur internet et leur principe est que vous pouvez combiner plusieurs composants totalement différents pour obtenir un résultat surpuissant. (Apollidore, 2005).

## 4. Les Concurrents d'Android

- Apple avec iOS
- Research In Motion (RIM) avec BlackBerry OS
- Microsoft avec Windows Phone
- Samsung avec Bada (même si Samsung utilise aussi Android)
- HP avec Palm webOS devenu webOS (Arrêté en 2011)
- Nokia avec Symbian OS (Arrêté en 2011, Nokia utilisera d´désormais Windows Phone). (Gauthier Picard, 2012)

## 5. Les Smartphones vendus selon leur OS

La figure suivante montre Les Smartphones vendus selon leur système d'exploitation (Gauthier Picard, 2012)

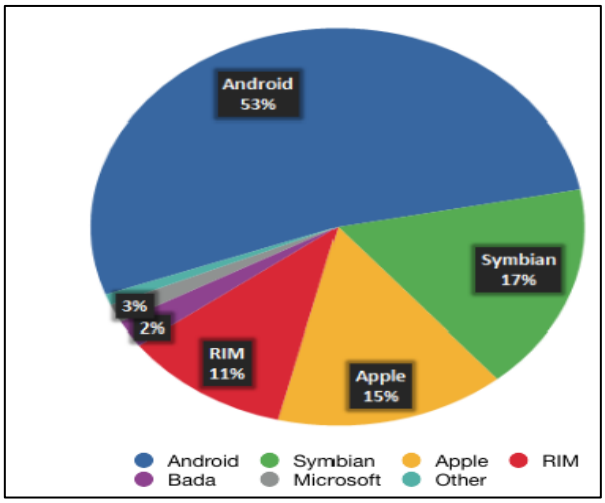



**Figure II.1**: Les Smartphones vendus selon leur OS

Apres l'étude des différents systèmes d'exploitation des Smartphones, j'ai constaté qu'Android est le plus évolué.

**Tableau II.2 :** Ventes de Smartphones aux utilisateurs finals par système d'exploitation 2011(BEN SIDIA Marwa,2012)

| **Système d'exploitation** | **Système d'exploitation** | **2011 Part du marché (%)** | **2010 Unités** | **2010 Part du marché (%)** |
|---|---|---|---|---|
| Android | 46,775.9 | 43.4 | 10,652.7 | 17.2 |
| Symbian | 23,853.2 | 22.1 | 25,386.8 | 40.9 |
| Ios | 19,628.8 | 18.2 | 8,743.0 | 14.1 |
| Research In Motion | 12,652.3 | 11.7 | 11,628.8 | 18.7 |
| Bada | 2,055.8 | 1.9 | 577.0 | 0.9 |
| Microsoft | 1,723.8 | 1.6 | 3,058.8 | 4.9 |
| Autres | 1,050.6 | 1.0 | 2,010.9 | 3.2 |
| **Total** | **107,740.4** | **100.0** | **62,058.1** | **100.0** |

## 6. Architecture de l'Android

Android est basé sur un kernel linux 2.6.xx, au-dessus du kernel il y a l'hardware abstraction layer" qui permet de séparer la plateforme logique du matériel.
Au-dessus de cette couche d'abstraction on retrouve les librairies C/C++ utilisées par un certain nombre de composants du système Android.
Au-dessus des librairies on retrouve l'Android Runtime, cette couche contient les librairies cœurs du Framework ainsi que la machine virtuelle exécutant les applications.
Au-dessus la couche "Android Runtime" et des librairies cœurs on retrouve le Framework permettant au développeur de créer des applications. Enfin au-dessus du Framework il y a les applications.
La figure suivante illustre les composants principaux du système d'exploitation Android :

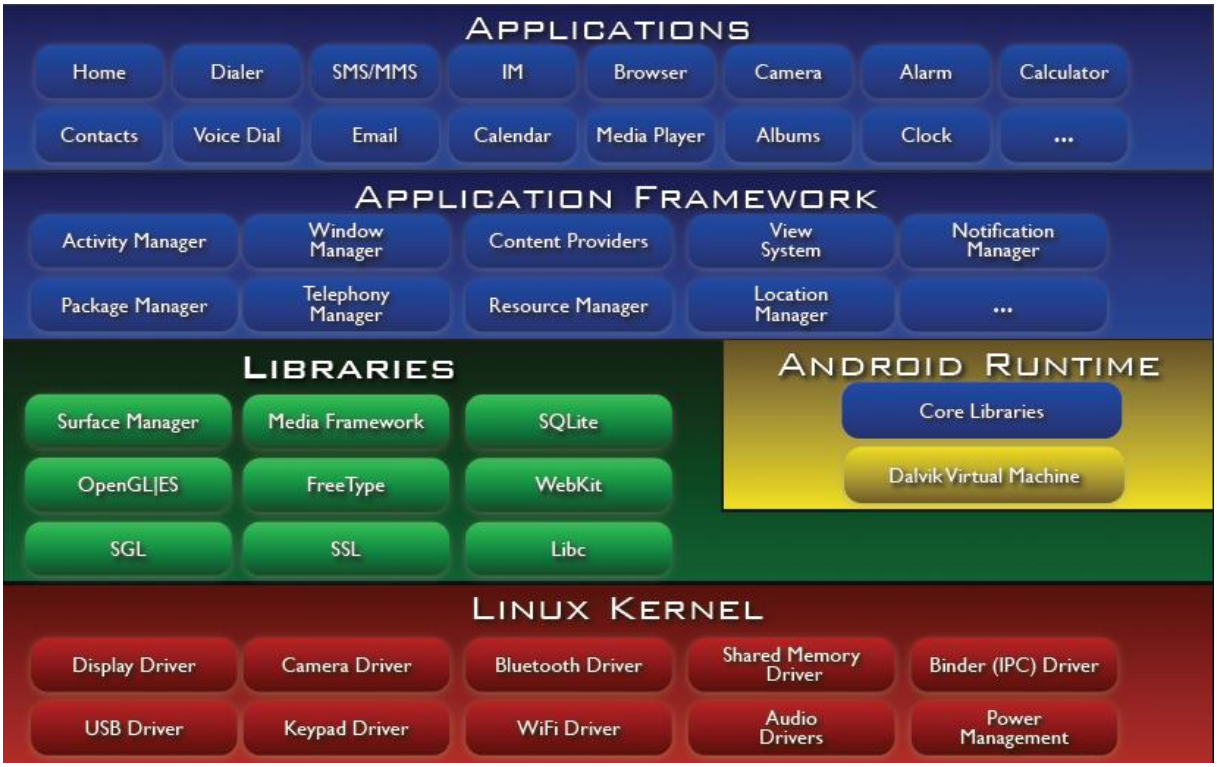



**Figure II.2 :** Architecture de l'android

## 7. Les éléments d'une application

Une application Android est composée des éléments suivants:

• des activités (**android.app.Activity**): il s'agit d'une partie de l'application présentant une vue à l'utilisateur
• des services (**android.app.Service**): il s'agit d'une activité tâche de fond sans vue associée
• des fournisseurs de contenus (**android.content.ContentProvider**): permettent le partage d'informations au sein ou entre applications
• des widgets (**android.appwidget.***): une vue accrochée au Bureau d'Android
• des Intents (**android.content.Intent**): permettent d'envoyer un message pour un composant externe sans le nommer explicitement
• des récepteurs d'Intents (**android.content.BroadcastReceiver**): permettent de déclarer, être capable de répondre à des Intents
• des notifications (**android.app.Notifications**): permettent de notifier l'utilisateur de la survenue d'événements.(Serge Ungare).

# 8. Les projets Android

## 8.1 Structure du projet vue sous Eclipse

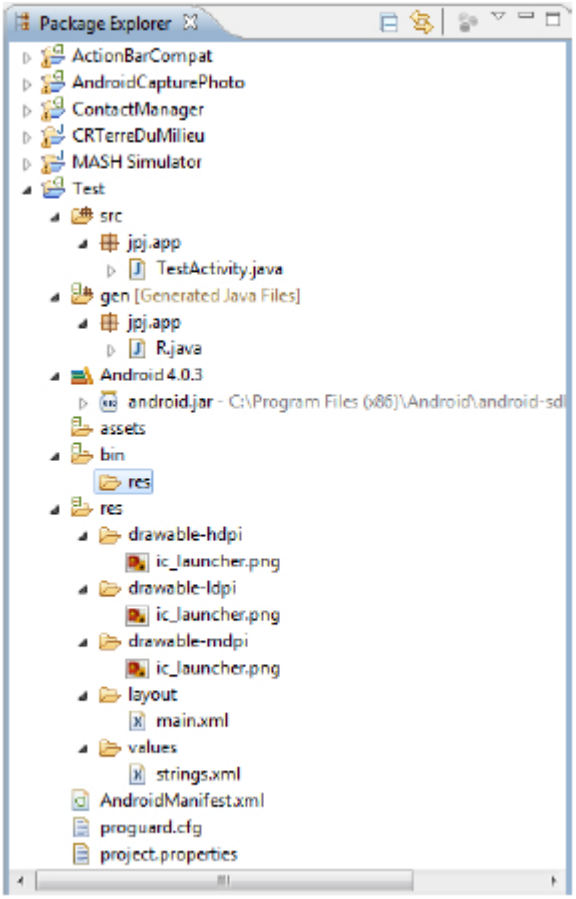



**Figure II.3 :** Structure du projet vue sous Eclipse

## 8.2 Répertoire

- manifest.xml : Le fichier manifest qui d´ ecrit l'application
- build.xml : Le script Ant qui permet de compiler l'application et de l'installer sur le terminal
- default.properties et local.properties : Deux fichiers de propriétés utilises par le script Ant
- bin/ : Répertoire qui contient l'application compilée
- gen/ : Répertoire qui contient le code source produit par les outils de compilation Android
- libs/ : Répertoire qui contient les fichiers JAR extérieurs `a l'application
- src/ : Répertoire qui contient code source Java de l'application
- res/ : Répertoire qui contient les ressources (icones, layouts...)

- tests/ : Répertoire qui contient un projet Android complétement distinct qui permet de tester celui qui est crée
- assets/ : Répertoire qui contient les autres fichiers statiques fournis avec l'application pour son d´déploiement sur le terminal

### 8.3 Les ressources

- res/drawable/ : Répertoire qui contient les images (JPG, PNG...)
- res/layout/ : Répertoire qui contient les descriptions XML de la composition de l'IHM (les layouts)
- res/menu/ : Répertoire qui contient les descriptions XML des menus
- res/raw/ : Répertoire qui contient les fichiers généraux (un fichier CSV contenant les informations de compte par exemple)
- res/values/ : Répertoire qui contient les messages, les dimensions...
- res/xml/ : Répertoire qui contient les autres fichiers XML que vous souhaitez fournir

### 8.4 les exécutables

**Contenu du répertoire bin/**

- bin/classes/ : Répertoire qui contient les classes java compilées
- bin/classes.dex : Répertoire qui contient l'ex ´ecutable crée a partir des classes compil´ees
- bin/votreApp.ap : Répertoire qui contient les ressources de l'application (fichier zip)
- bin/votreApp-debug.apk : Répertoire qui contient la véritable application Android.

## 9. Une activité

Une activité est la composante principale pour une application Android. Elle représente l'implémentation et les interactions d'une interface.

### 9.1 Les méthodes d'une activité

**onStart()** est lancée tout de suite après le onCreate() lorsque l'activité va devenir visible à l'utilisateur. Vous allez pouvoir charger des données, ...Le système pourra ensuite lancer onResume() ou onStop().

**onResume()** est lancée lorsque votre application est passée en avant plan. Vous pourrez donc (re)lancer vos threads, ... Le système pourra ensuite lancer onPause()

**onPause()** est lancée lorsqu'une autre activité va s'afficher à l'avant plan. C'est le moment de sauver toutes les données/informations saisies par l'utilisateur pour l'activité courante. **onStop()** est lancée lorsque votre activité n'est plus visible

**onRestart()** est lancée lorsque votre activité en sommeil et redémarre. Cette méthode lancera ensuite la méthode onStart().

**onCreate()** devra être exécutée pour obtenir à nouveau l'activité.

Enfin**, onDestroy()**, qui met fin à au cycle de vie de l'acitivité.

Voici un schéma qui représente le cycle de vie d'une activité

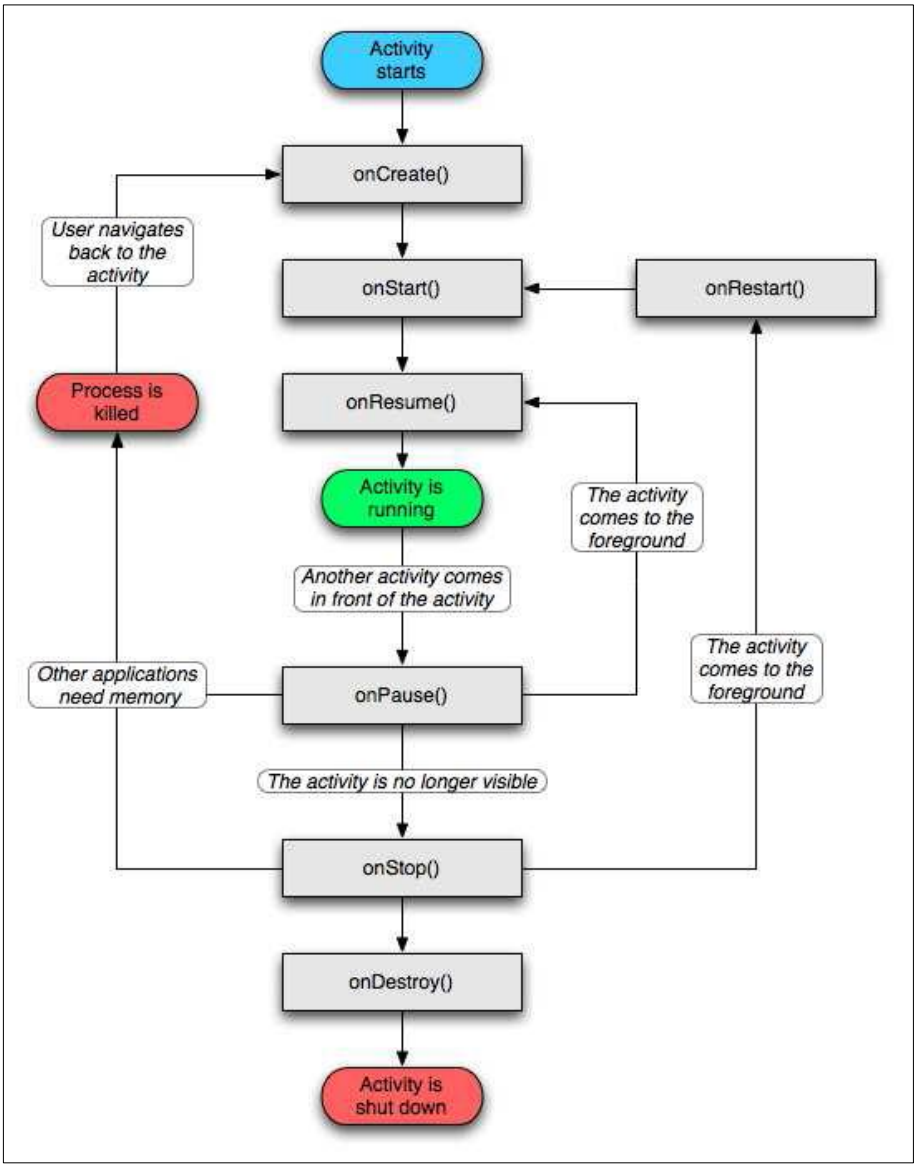



**Figure II.4:** Le cycle de vie d'une activité

## 10. La technologie mobile

La technologie mobile change le monde plus rapidement et plus profondément que toute autre innovation. Sur les marchés émergents, l'ampleur du phénomène et l'impact sur les communautés locales sont trop importants pour être ignorés. (Espiau, 2017)

## 10.1 Chronologie de l'histoire du développement de téléphone mobile

Les figures suivantes nous montrent l'histoire de développement de téléphone mobile. Les premières radios voiture à voiture utilisés en 1921 nous offrent la possibilité d'utiliser le téléphone mobile. Ainsi, le premier téléphone avec la mélodie est inventé par la société Nokia en 1994, de lors le téléphone mobile est entré dans notre vie quotidienne. Pendant les dix années dernières, le téléphone devient de plus en plus avancé avec le développement de technologie (Edraw, 2018).

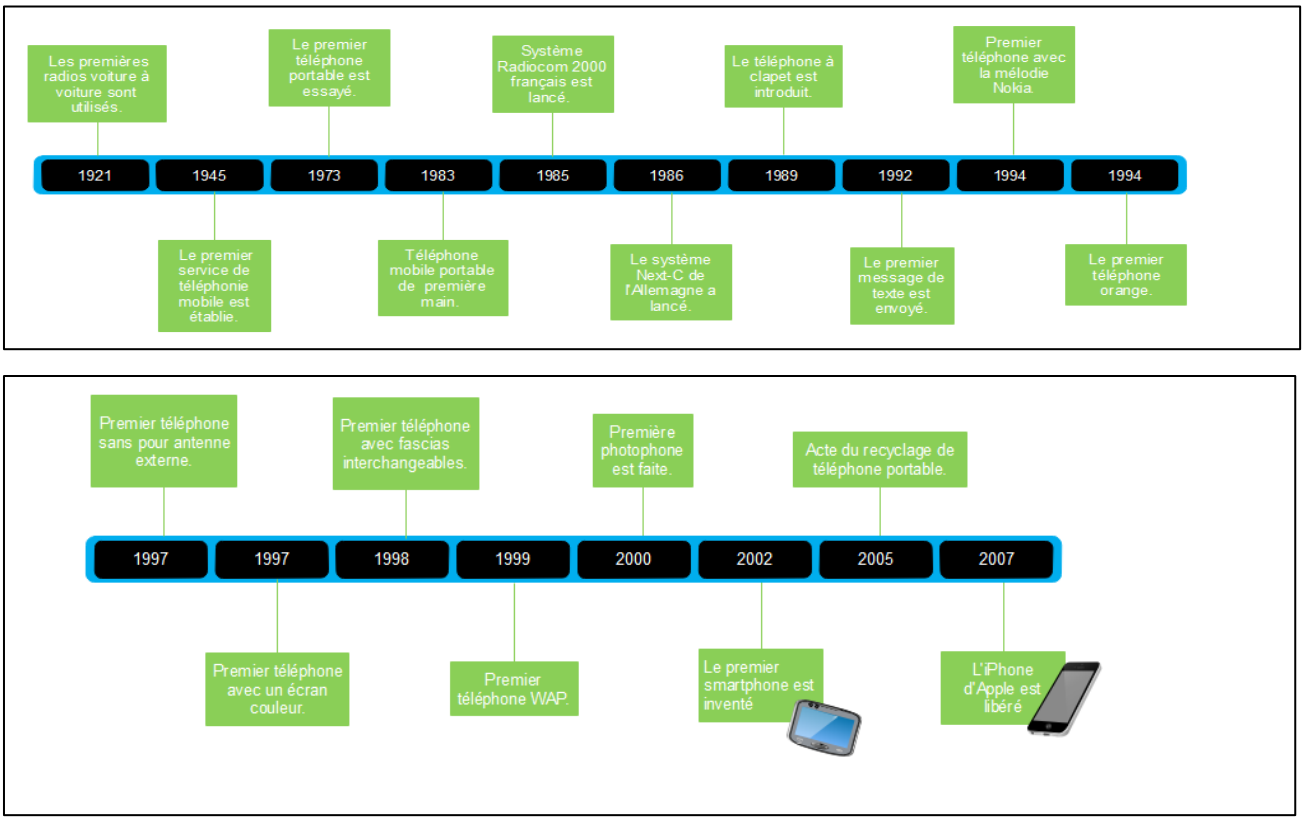



**Figure II.5:** Chronologie de l'histoire du développement de téléphone mobile (Edraw,2018).

## 10.2 Les différentes forces et faiblesses des applications Web mobiles

Voici un résumé des différentes forces et faiblesses des applications Web mobiles (Activités et initiatives du gouvernement du Canada.2013)

### a. Forces

- Elles peuvent offrir une expérience semblable à celle des applications mobiles natives.
- Elles peuvent être exploitées sur de nombreuses plateformes (interface utilisateur adaptable).
- Elles sont contrôlées et tenues à jour à l'interne.
- Elles sont indépendantes du système d'exploitation et de la boutique d'applications.
- Elles offrent des capacités d'analyse Web.

- Elles n'exigent aucune installation (très commode).
- Leur contenu peut être trouvé à l'aide des moteurs de recherche.

### b. Faiblesses

- Elles ne peuvent pas (encore) exploiter toutes les capacités des appareils.
- Bien qu'elles ne s'appliquent pas à un appareil en particulier, la performance varie d'une plateforme à l'autre.
- Elles exigent généralement une connexion 3G ou plus ou une connexion Wi-Fi.
- Le langage HTML5 est toujours en cours d'élaboration (certaines fonctions s'ajoutent en cours de route).

## 10.3 Les générations

La figure suivante décrit les différentes générations de la technologie mobile. (High Tech info, 2016)

| Génération | Acronyme | Description | Intitulé | Débit indicatif (download) en bits/s | Durée de download d'un DivX (700 Mo) |
|---|---|---|---|---|---|
| 1G | Radiocom 2000 | | Radiocom 2000 (analogique) de France Télécom, SFR2000 de SFR | analogique | |
| 2G | GSM | Échanges de type voix uniquement | Global System for Mobile Communication | 9,05 kbps | 7.2 jours |
| 2.5G | GPRS | Échange de données ou (exclusif) voix | Global Packet Radio Service | 171,2 kbps | 9.1 heures |
| 2.75G | EDGE | Basé sur réseau GPRS existant | Enhanced Data Rate for GSM Evolution | 384 kbps | 4 heures |
| 3G | UMTS | Voix + données | Universal Mobile Telecommunications System | 144 kbps rurale, 384 kbps urbaine, 1,9 Mbps point fixe | 10.8 heures, 4 heures, 49 min |
| 3.5G | HSPA | Évolution de l'UMTS | High Speed Packet Access (HSDPA/HSUPA) | 14,4 Mbps | 6 min 28 |
| 3.75G | HSPA+ Dual Carrier | Évolution de l'UMTS | High Speed Packet Access + | 42 Mbps | 2 min 13 |
| 4G | LTE | | Long Term Evolution | 150 Mbps | 37 secondes |
| 4G ou 4G+ | LTE-Advanced | Agrégation de spectre | Long Term Evolution Advanced | 1 Gbps à l'arrêt, 100 Mbps en mouvement | 56 secondes, 5,6 secondes |

**Figure II.6:** les différentes générations de la technologie mobile

# 11. Programmation java sous Android

## 11.1 Outils nécessaires pour programmer Java Android

La figure suivante décrit les différents outils nécessaires pour programmer Java Android

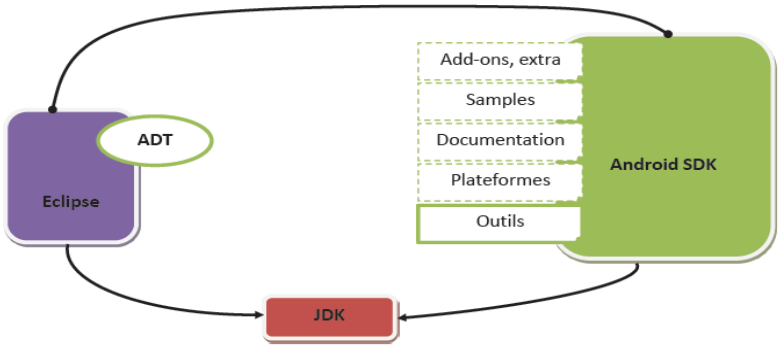



**Figure II.7:** Outils nécessaires pour programmer Java Android.

## 11.2 Sdk Manager (Software Developement kit)

Un SDK, c'est-à-dire un ***Kit de Développement*** , est un ensemble d'outils que met à disposition un éditeur afin de vous permettre de développer des applications pour un environnement précis. Le SDK Android permet donc de développer des applications pour Android et uniquement pour Android.

Android SDK manager est fourni par google est contient plusieurs éléments :

- Un compilateur JAVA pour Android (l'outil dx).
- Un logiciel de virtualisation pour les émulateurs Android
- Des exemples de projets avec des codes sources
- Un outils de communication entre l'ordinateur et l'émulateur
- Les images (iso) des différentes versions d'Android
- Un gestion de versions du SDK

**Android SDK** est téléchargeable ici : http://developer.android.com/sdk/index.html

Au premier lancement du SDK, il s'affichera :

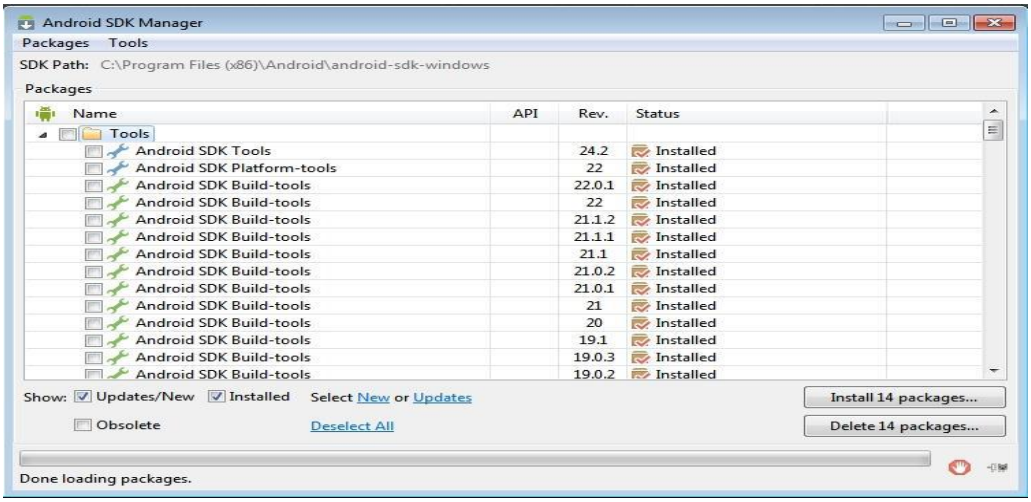



**Figure II.8:** Android Sdk Manager

### 11.3 L'émulateur

Le SDK propose un émulateur Android. Il permet de lancer sur la machine du développeur un terminal virtuel représentant à l'écran un téléphone embarquant Android. C'est bien évidemment un outil indispensable pour le développement mobile. A chaque version d'Android est associée une version de l'émulateur, permettant au développeur de voir exactement à quoi ressemblera son application sur un matériel réel.


**Figure II.9:** émulateur d'éclipse

### 11.4 Eclipse et plug-in ADT

**Eclipse** est un environnement de développement spécialisé dans le développement Java gratuit, disposant d'une importante collection de plug-in, il est tout indiqué pour le développement d'applications Android. Pour ce faire, il faut lui adjoindre le **plug-in ADT**, qui intègre de nombreuses fonctionnalités spécifiques au développement Android. (Hebuternr et Reochon).

### 11.5 Création d'un projet

Nous allons décrire les étapes de création d'un simple projet Android sous Eclipse (l'environnement de développement est installé).

1. Démarrer Eclipse.
2. En cliquant sur : File -> New->Project et nous sélectionnons « Android Project ».

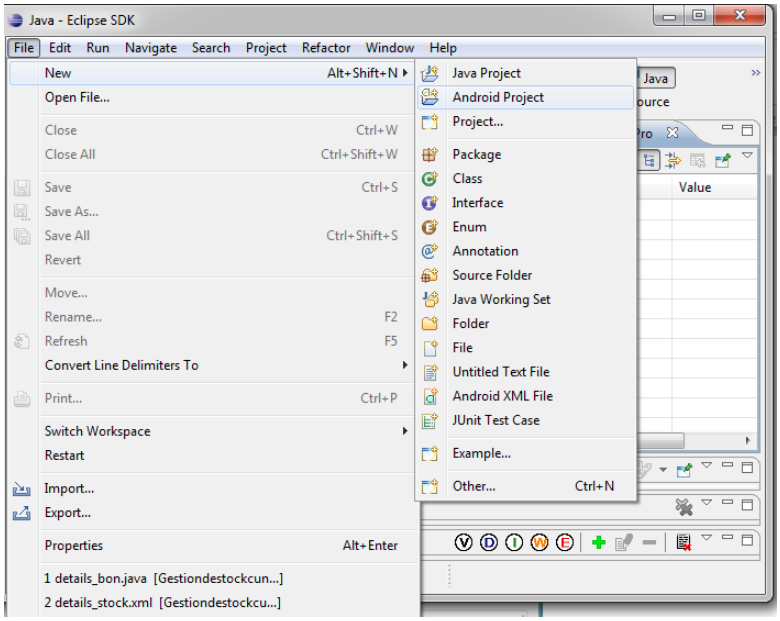



**Figure II.10 :** Structure du projet vue sous Eclipse

3. Choisir essai_android comme nom du projet.
4. Et mettre les informations suivantes :

Application name : HelloAndroid
Package name : com.android.helloandroid
Create Activity : HelloAndroid

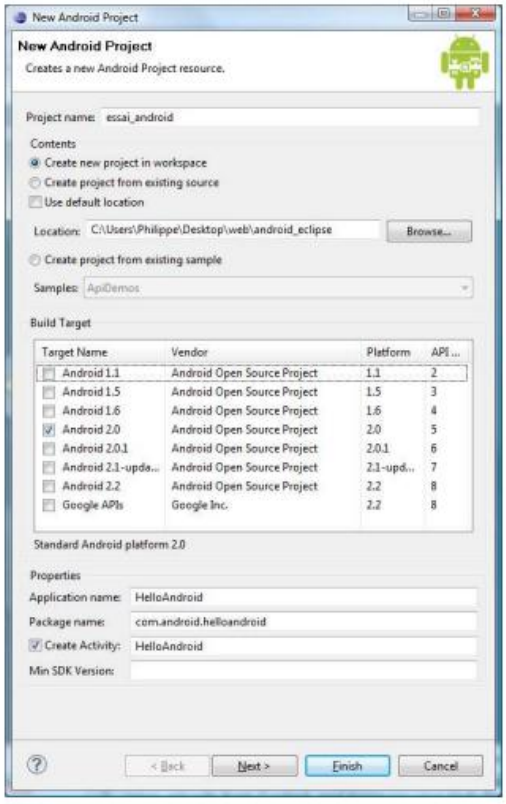



**Figure II.11:** Paramètres du projet HelloAndroid

5. Nous cliquons sur Finish.
6. Ceci devrait donner :

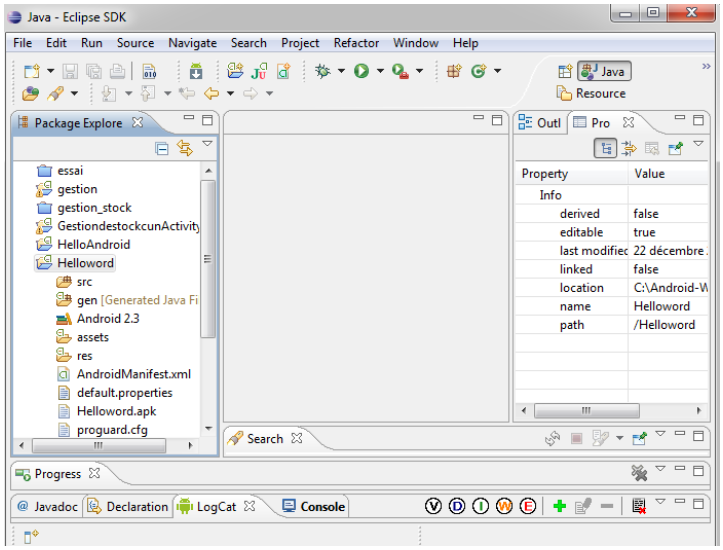



**Figure II.12 :** Création d'un nouveau projet Android

7. Le projet se présente comme ceci :

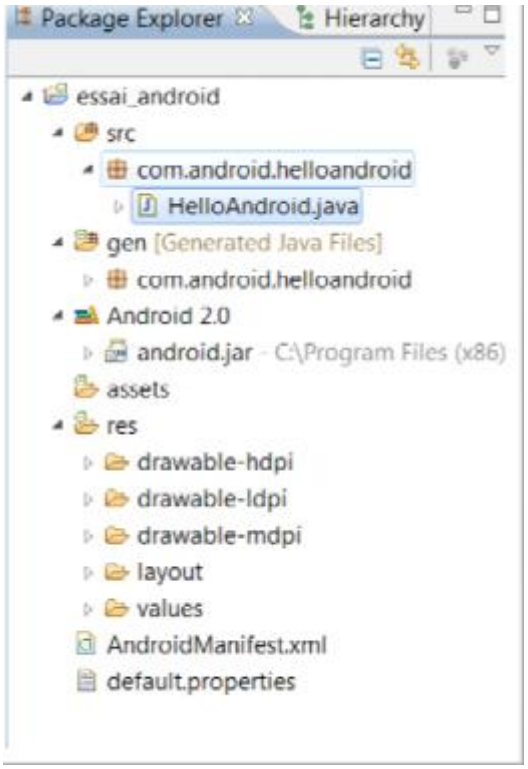



**Figure II.13 :** Structure du nouveau projet vue sous Eclipse

8. Modifier le code du fichier HelloAndroid.java.

```
package com.android.helloandroid;
import android.app.Activity;
 import android.os.Bundle;
 import android.widget.TextView;
public class HelloAndroid extends Activity {
 /** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
TextView tv = new TextView(this);
 tv.setText("Hello, Android");
setContentView(tv); } }
```

9. Faire Run / Run

Le gestionnaire d'AVD permet alors de choisir la machine virtuelle à utiliser. Si tout se passe bien, l'écran de votre ordinateur se présente maintenant comme suit :

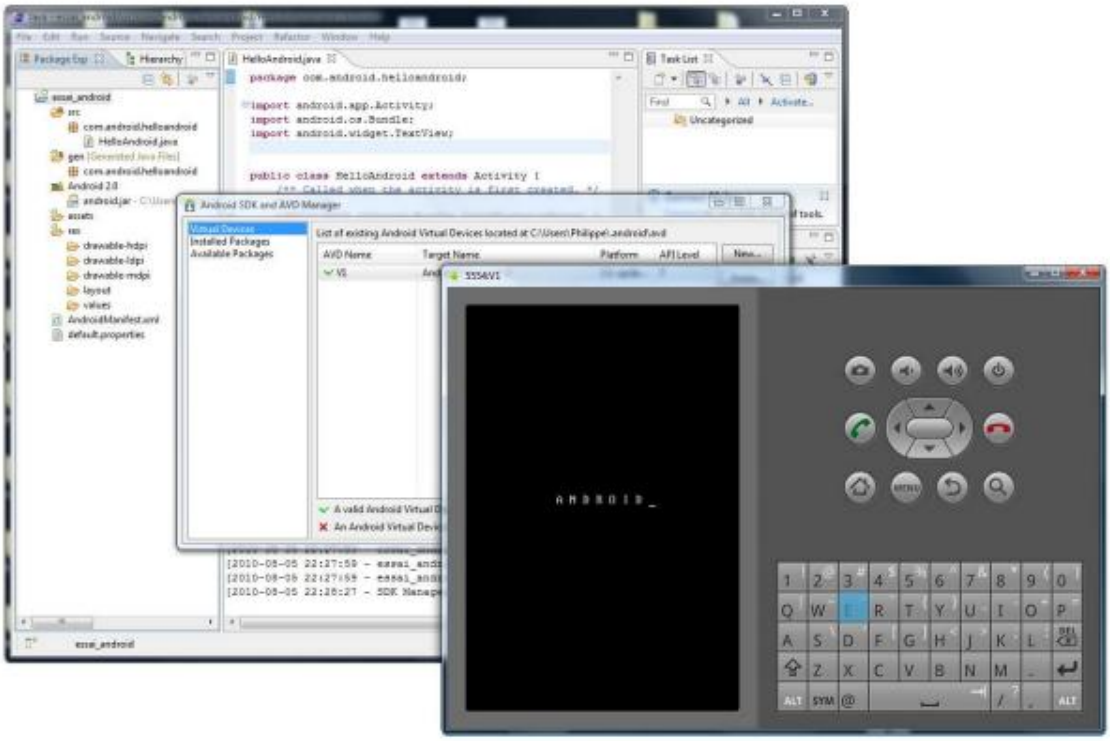


**Figure II.14 :** Exécution de l'application

10. Attendre que le système Android démarre sur le téléphone.

Ce qui donne :


**Figure II.15:** Résultat d'exécution de l'application.

## 12 Conclusion

Dans ce chapitre, nous avons vu quelques informations sur Android et la technologie mobile, passant maintenant au chapitre suivant concernant la conception.

# Chapitre III: Conception

## 1. Introduction

Ce chapitre est consacré à la conception de notre application, la première section nous avons présenté une description de notre cas d'étude « service de magasin »

La deuxième section nous permettre de modéliser notre problème on utilise le langage de modélisation unifié UML ; on commencera avec les diagrammes de cas d'utilisation et diagramme de classes et on passera par la suite aux diagrammes d'activité et diagrammes de séquence.

## 2. Définition des taches de Magasinier

Le magasiner doit réaliser les tâches suivantes :

- Stocker les produits en entrée du magasin.
- La sortie des produits pour les différents services de centre de l'université.
- Remplir les fiches stock.
- La mise à jour du registre des entrées
- Assurer la bonne organisation des produits dans le magasin.
- La vérification de l'état des produits en entrée du magasin.
- Le dénombrement des produits en entrée du stock magasin.
- La restitution du matériel en état vétuste.
- Classer les bons de sorties.
- Remplir des décharges en cas du besoin.
- Réaliser l'inventaire des produits pour chaque année budgétaire.
- Signaler les produits en rupture du stock.
- Classer les fiches de stock pour tous les produits affectés aux différents services.

Un magasinier doit en plus assurer la répartition des produits :

- Matériel technique et scolaire, ameublement, mobilier.
- Matériel d'éclairage ; de chauffage et d'électricités.
- Matériel d'hébergement (matelas, armoires individuelles ou collectives, table de nuit).
- Vêtement de travail, les blouses.
- Matière détergent (eau de javel, savon, Désodorisant, Isis …).
- Matériel d'informatique.
- Produits chimique de laboratoire.
- Fourniture de bureau.

Sur le tableau ci-dessous, sont recensés les différents documents en circulation dans le service de magasin :

**Tableau III.1**: les différents documents en circulation dans le service

| N° | Désignations |
|---|---|
| 01 | Bon de commande |
| 02 | Bon de sortie |
| 03 | Bon de livraison |
| 04 | Décharge |
| 05 | Etat de besoins |

## 3. Critiques de L'existant

L'objectif de cette étape consiste à faire ressortir toutes les anomalies, pour tenter d'étudier les causes profondes et chercher des solutions adéquates.

### 3.1 Synthèse de l'anomalie constatée

- La majorité du travail ce fait manuellement.
- L'absence de l'utilisation de l'outil informatique dans certaines tâches.
- Multiplicité des documents qui cause une lenteur administrative considérable.
- Difficulté dans la recherche des documents.
- La perte des documents ;
- Existence de nombreux fichiers manuels.
- Existence des documents répétitifs.
- Difficultés dans l'inventaire périodique.
- La redondance dans le saisi des données

### 3.2 Proposition des solutions nouvelles

- Acquisition ou conception d'un logiciel Android de gestion.
- Utiliser l'outil informatique pour le traitement de certains documents de gestion en absence d'une application.
- Eliminer la redondance en matière de l'information.
- Annuler les documents répétitifs.
- Faciliter l'accès au document on introduisant des outils informatique.

## 4. Conception

Cette partie porte sur la conception du système, nous allons détailler quatre étapes :

- Décrire la structure statique de notre système par le diagramme de classe.
- Spécifier les besoins de notre système en définissant le diagramme de cas d'utilisation.
- Le diagramme de cas d'utilisation va être détaillé en plusieurs diagrammes de séquences.
- Les diagrammes d'activités pour l'aspect dynamique.

## 4.1 Diagramme de cas d'utilisation

Un cas utilisation modélise une interaction entre le système informatique a développer et un utilisateur ou acteur interagissent avec le système.
Ce diagramme est destiné à présenter les besoins des utilisateurs par rapport au système.

La figure suivante présente le diagramme de cas d'utilisation :

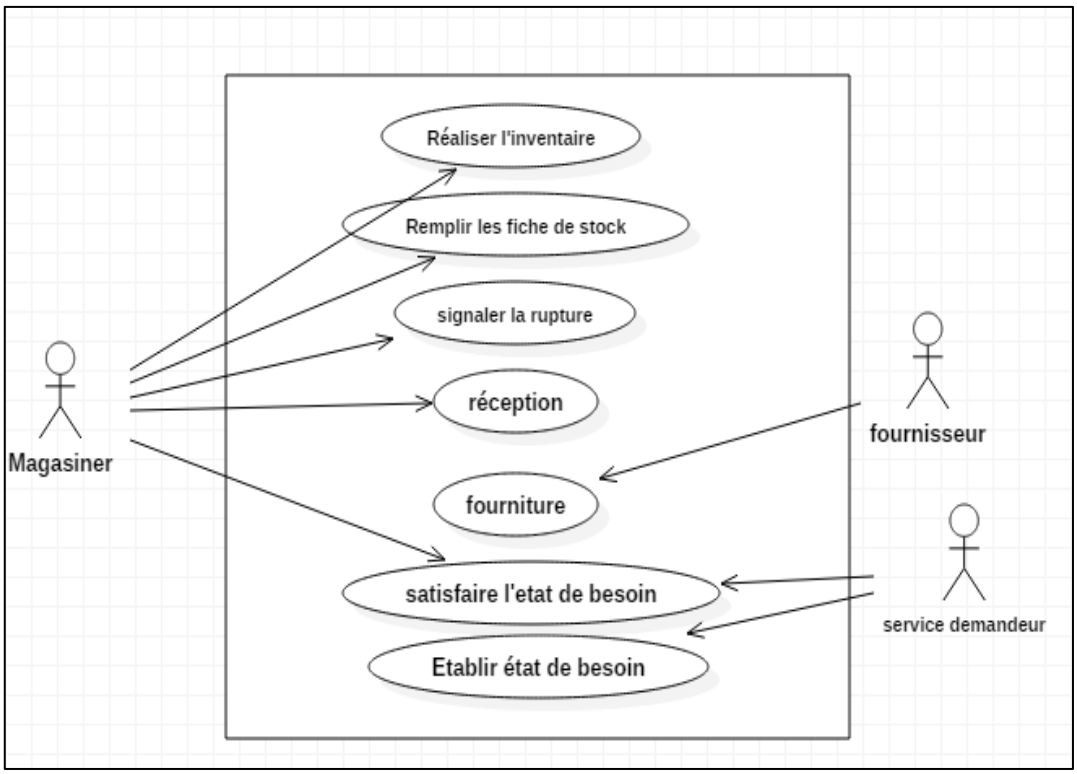



**Figure III.1**: Diagramme de cas utilisation

## 4.2 Diagramme de classes

C'est une représentation statique des éléments qui composent un système et de leurs relations.

La figure suivante présente le diagramme de classes élaboré pour ce projet :

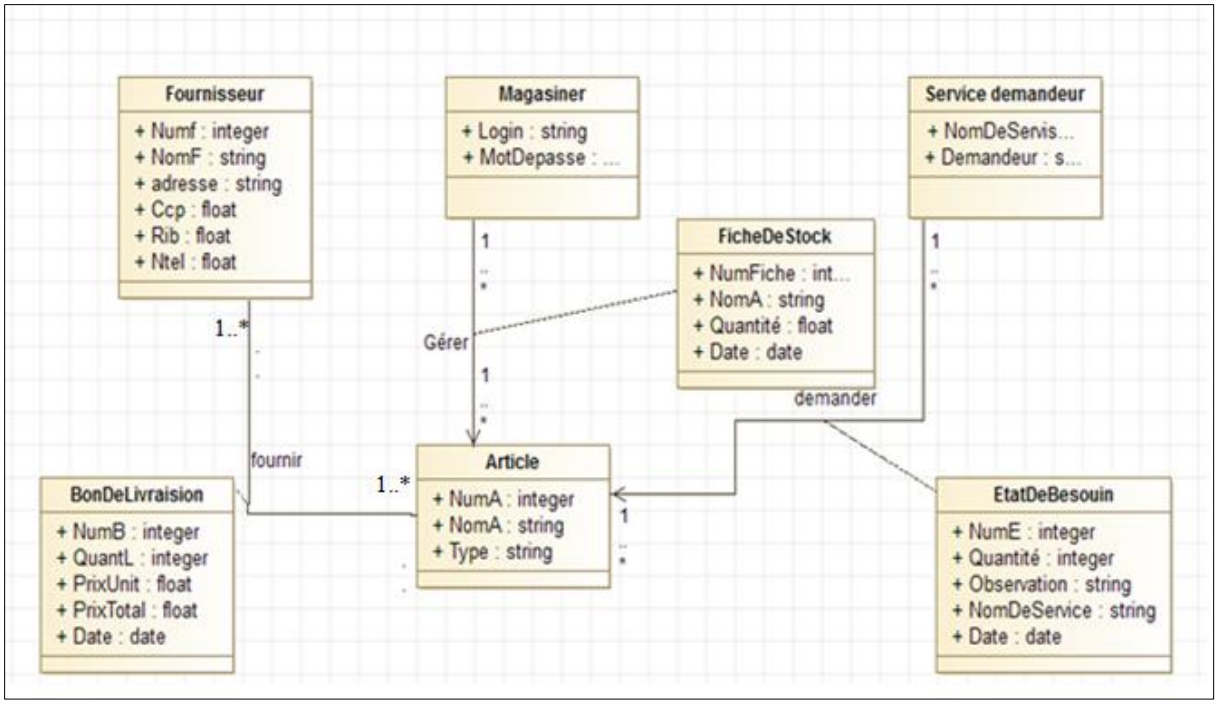



**Figure III.2**: Diagramme de classes.

### a) Les règles de passage

1. Chaque classe devient une relation. Les attributs de la classe deviennent attributs de la relation. L'identifiant de la classe devient clé primaire de la relation.

2. Chaque association 1-1 est prise en compte en incluant la clé primaire d'une des relations comme clé étrangère dans l'autre relation.

3. Chaque association 1-N est prise en compte en incluant la clé primaire de la relation dont la multiplicité maximale est * comme clé étrangère dans l'autre relation.

4. Chaque association M-N est prise en compte en créant une nouvelle relation dont la clé primaire et la concaténation des clés primaires des relations participantes. Les attributs de la classe d'association sont insérés dans cette nouvelle relation si nécessaire. (Jérôme Darmont, 2001**).**

### b) Schéma relationnel

Fournisseur(Numf,Nomf,adresse,Ccp,Rib,Ntel)

Article(NumA,NomA,Type)

BonDelivraison(NumB,QuantL,Prixunit, PrixTotal Date,Numf,NumA)

Magasiner(Login,MotDepasse)

FicheDeStock(NumFiche,NomA,Quantite,Date,NumA)

Service demendeur(NomDeservice,Demandeur)

EtatBebesion(NumE,Quantite,Observation,NomDeService,date,NumA)

## 4.3 Diagramme d'activité

Le diagramme d'activité permet de modéliser le comportement du système, dont la séquence des actions et leurs conditions d'exécution.

### a)Diagramme d'activité entre le fournisseur et le magasiner

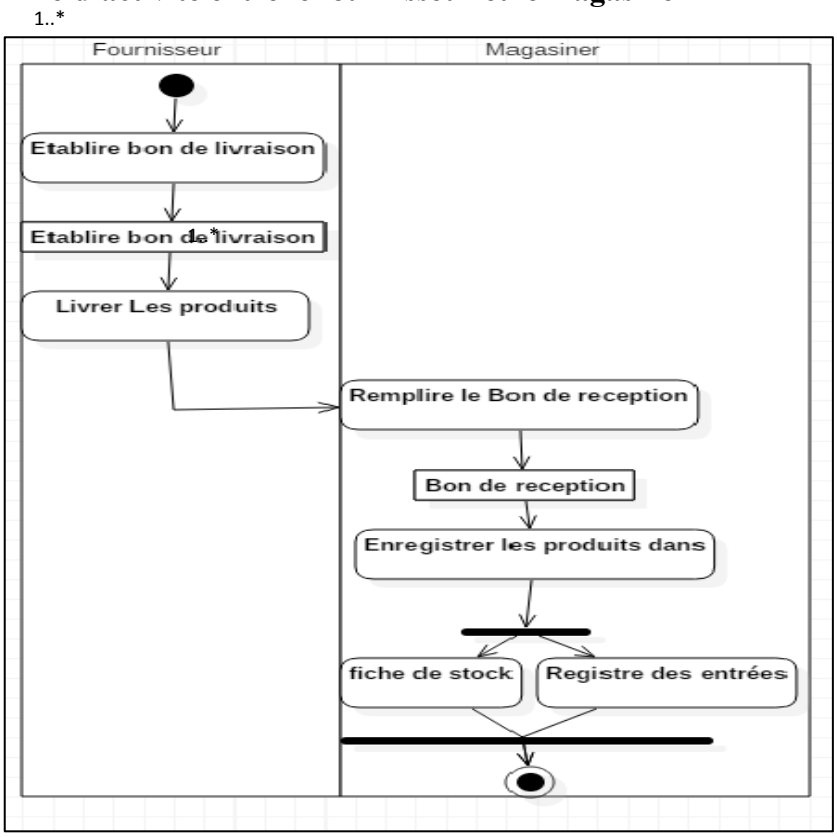



**Figure III.3**: Diagramme d'activité entre le fournisseur et le magasiner

### b) Diagramme d'activité entre le service demandeur et le magasiner

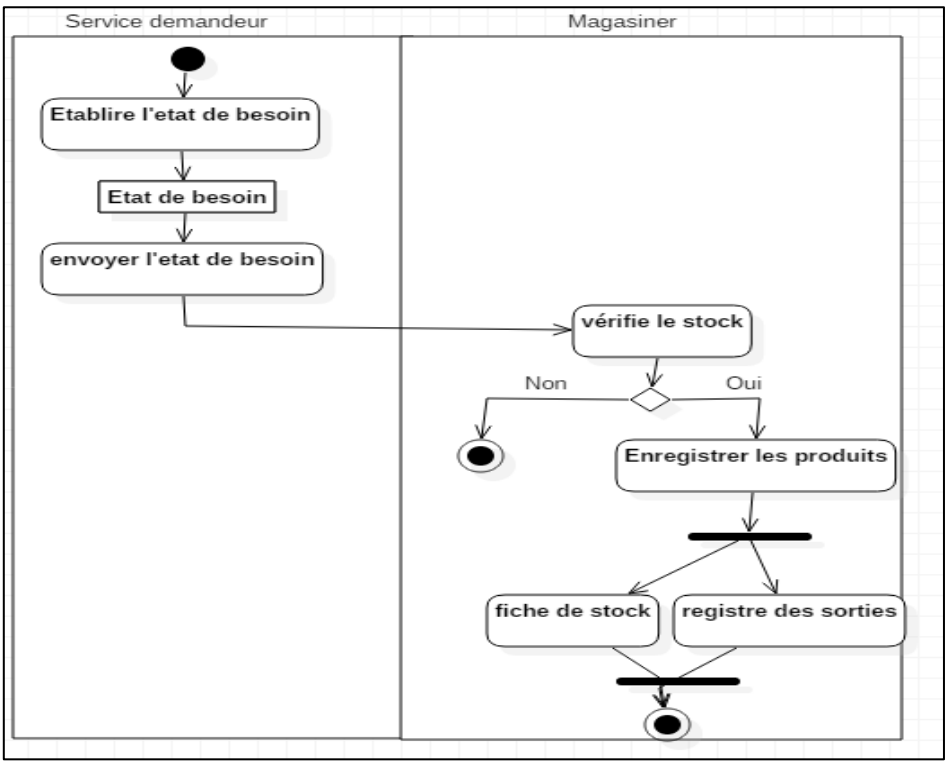



**Figure III.4**: Diagramme d'activité entre le service demandeur et le magasiner

## 4.4 Diagramme de séquence

### a) Diagramme de séquence d'authentification

La figure suivante présente la séquence des opérations permettant de réaliser le cas d'utilisation «s'authentifier » :

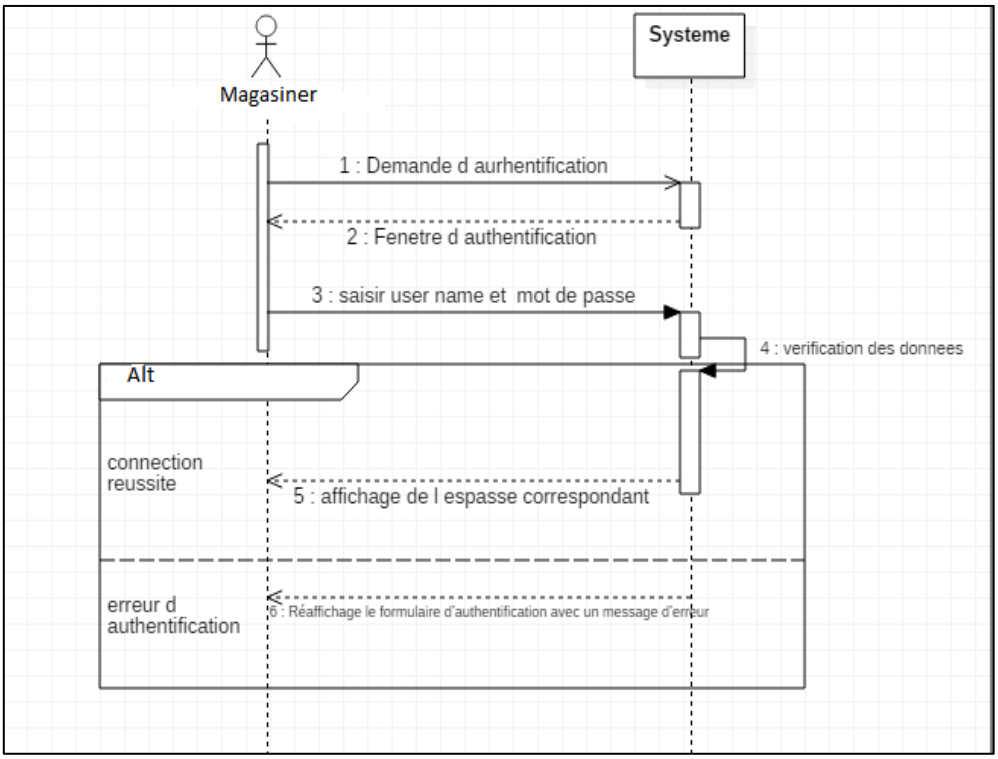



**Figure III.5**: Diagramme de séquence du cas d'utilisation « s'authentifier»

### b) Diagramme de séquence pour gérer les fournisseurs

La figure suivante présente le diagramme du cas d'utilisation entre le magasiner et les fournisseurs

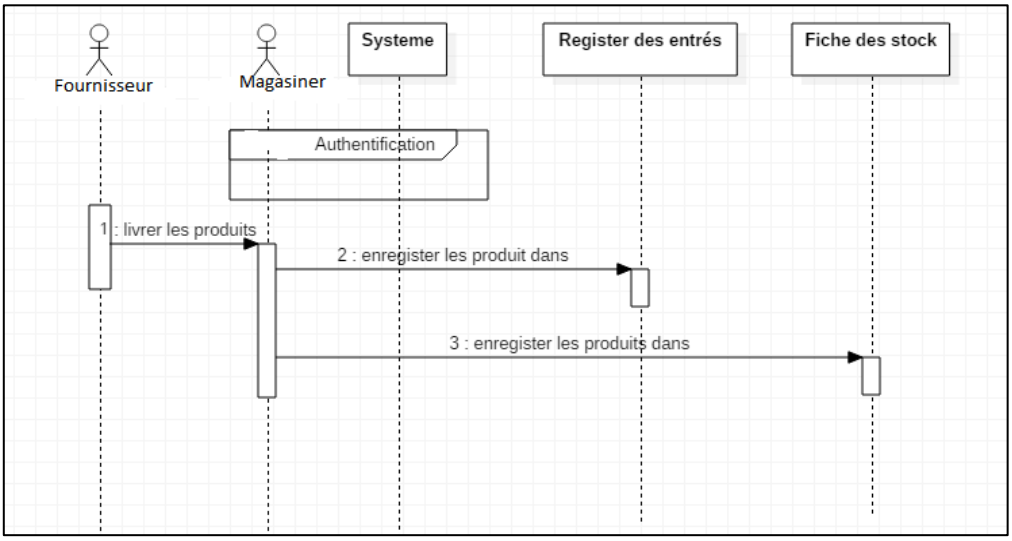



**Figure III.6**: Diagramme de séquence gérer les fournisseurs

### c) Diagramme de séquence pour gérer les demandes des services

La figure suivante présente le diagramme du cas d'utilisation entre le magasiner et les services demandeurs

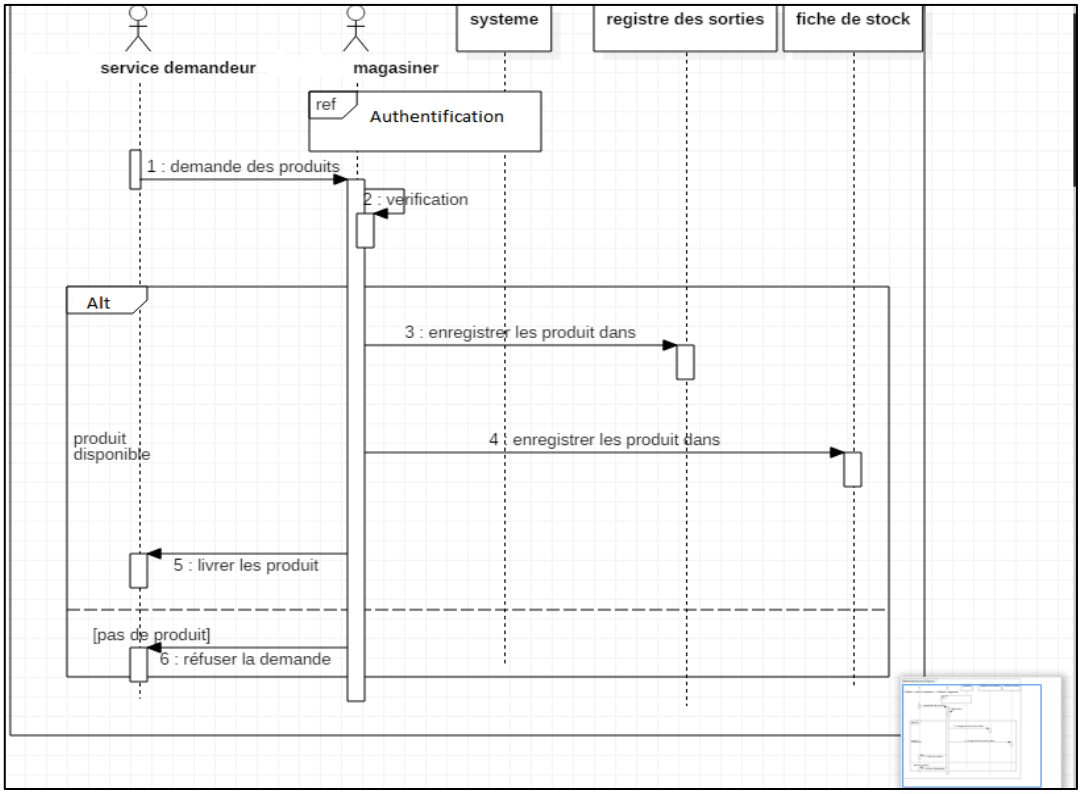



**Figure III.7:** Diagramme de séquence gérer les demandes des services

## 5. Conclusion

Nous avons vu dans ce chapitre l'ensemble des diagrammes important pour traduire les besoins en spécifiant comment l'application pourra les satisfaire avant de procéder à son implémentation avec des outils de développement appropriés.

# Chapitre IV:
# Implémentation

## 1. Introduction

Après l'analyse et la conception de notre application, nous entamons la phase de réalisation et la mise en œuvre de notre application. Nous allons présenter les différentes interfaces de l'application, donnant ainsi un aperçu général sur la phase pratique qui consiste à décrire les différentes fonctionnalités disposées lors de l'utilisation de notre logiciel. Ce détail est donné après avoir introduit une brève explication de la méthode de connexion à une BDD.

## 2. Architecture du système

Il est impossible de se connecter à une base de données directement sous Android. La méthode la plus répandue pour se connecter à une base de données MySQL à distance à partir d'un appareil Android, est de mettre en place un script PHP, et d'exécuter ce script en utilisant le protocole HTTP et le format JSON. La figure suivante montre comment connecter une application Android à une base de données :


**Figure IV.1** : Architecture du système.

## 3. JSON

JSON (JavaScript Object Notation) est un format léger d'échange de données. Il est facile pour les humains de le lire et l'écrire. Il est facile pour les machines de l'analyser et de le générer. Il est basé sur un sous-ensemble du langage de programmation JavaScript , Standard ECMA-262 3rd Edition - Décembre 1999, et qui a été popularisé par Douglas Crockford

### 3. 1 Comment va être utilisé JSON dans notre application ?

- Lorsque l'application Android va s'exécuter, il se connectera au script PHP.
- Le script PHP va récupérer les données depuis la base de données MySQL.
- Ensuite les données seront encodées au format JSON, et envoyées au système Android.
- Ensuite, l'application Android va obtenir ces données codées. Elle les analysera et les affichera sur l'appareil Android.

### 4. Mise en œuvre de l'application

Nous allons maintenant entamer la phase pratique qui consiste à décrire les différentes fonctionnalités de notre application :

Quand vous lancez le logiciel, l'interface principale de l'application apparaitra sur l'écran. L'utilisateur doit s'identifier par son nom d'utilisation et son mot de passe pour pouvoir accéder au menu principal. Cette étape a pour rôle de sécuriser les données et protéger l'application :

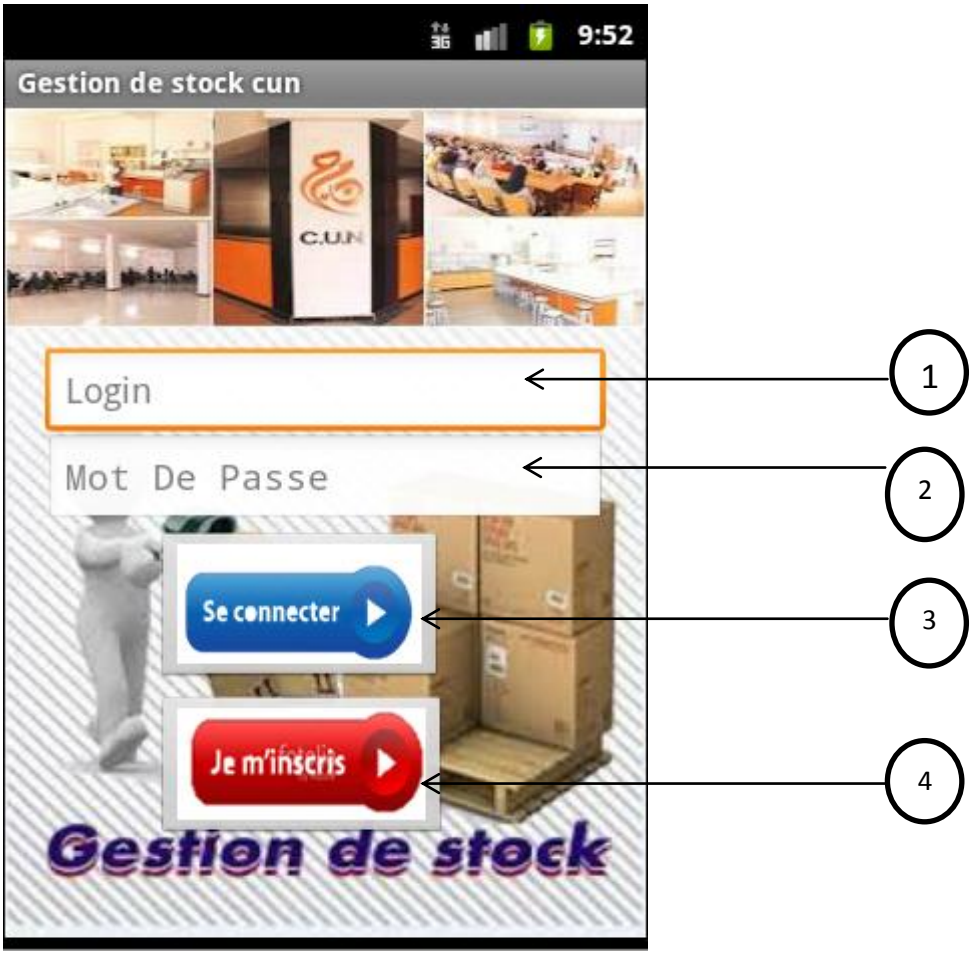



**Figure IV.2:** L'interface principale de l'application.

**(1)** : Saisir le nom de l'utilisateur.

**(2)** : Saisir le mot de passe de l'utilisateur.

**(3)** : Accéder à l'application.

**(4)** : Créer un nouveau compte (Inscription).

Si on clique sur le bouton « se connecter »:


Si les informations ne sont pas correctes le système affiche un message d'erreur sinon le système affiche le menu principal. Et on obtient l'interface suivante :


**Figure IV.3:** L'interface du menu principale.

Si on clique sur « Je m'inscris » de la Figure IV.2 :


On obtient l'écran suivant :

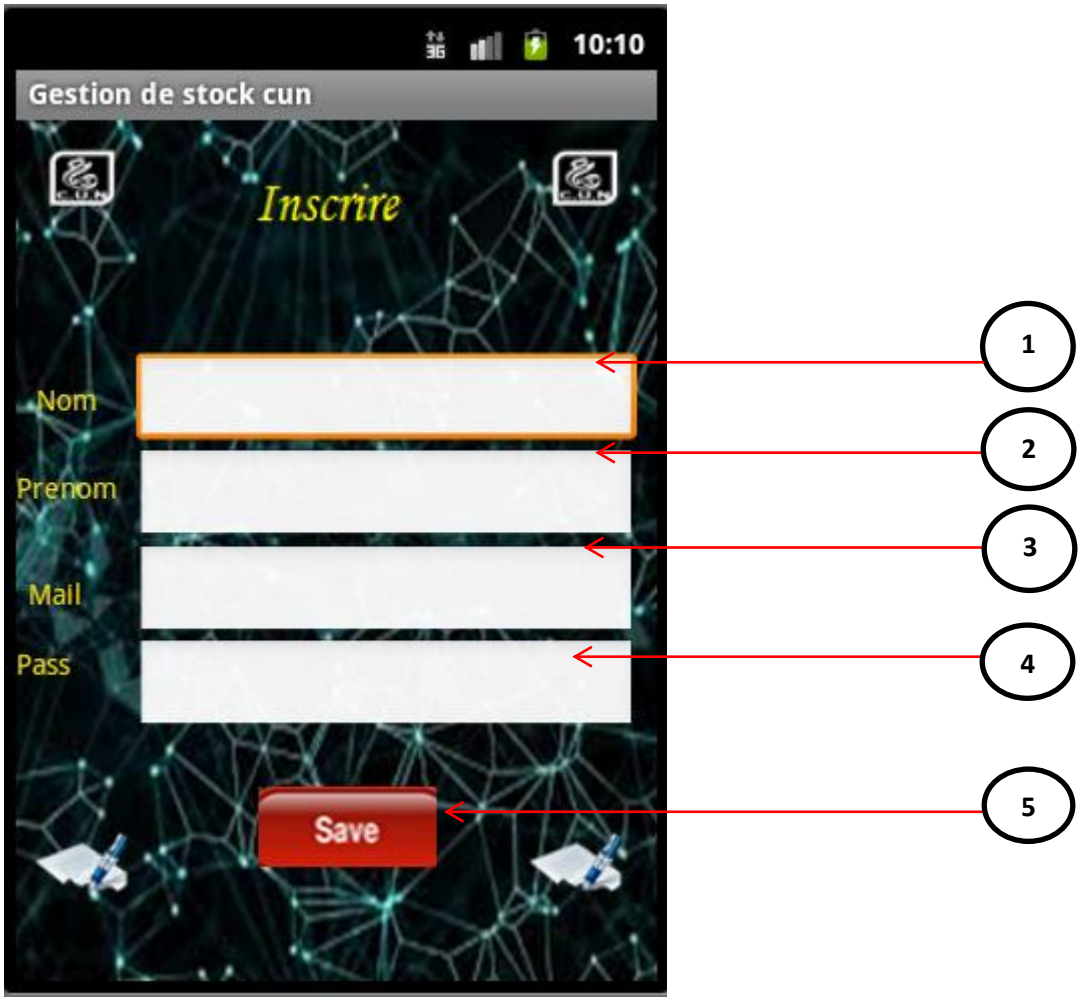



**Figure IV.4**: Inscription pour avoir un nouveau compte

**(1)** : Saisir le nom de l'utilisateur.

**(2)** : Saisir le prénom de l'utilisateur

**(3)** : Saisir l'email de l'utilisateur.

**(4)** : Saisir un mot de passe.

**(5)** : Enregistrer les données.

Si on clique sur le bouton


On obtient :

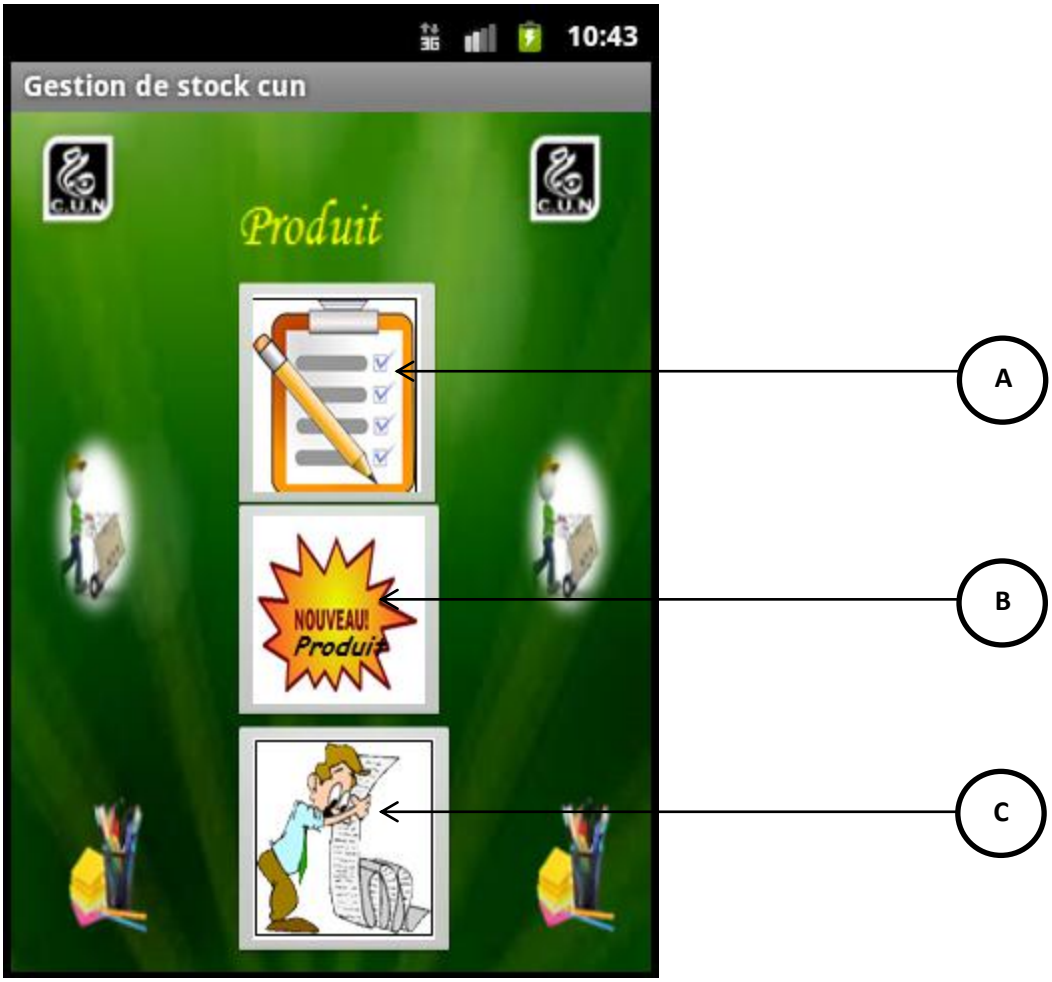



**Figure IV.5:** Ecran permettant la manipulation d'un produit.

(A) : Afficher la liste des produits.

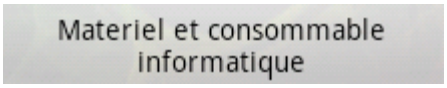
(B) : Créer un nouveau produit.

(c) : Afficher l'écran d'inventaire.

Si on clique sur le bouton (A) « La liste des produits » de la Figure IV.5, on obtient l'interface suivante:


**Figure IV.6**: Ecran de choix d'un type de produit

Si on clique sur l'un des boutons, il nous affiche la liste des produits indiqués sur le bouton Par exemple :




On obtient l'interface suivante :

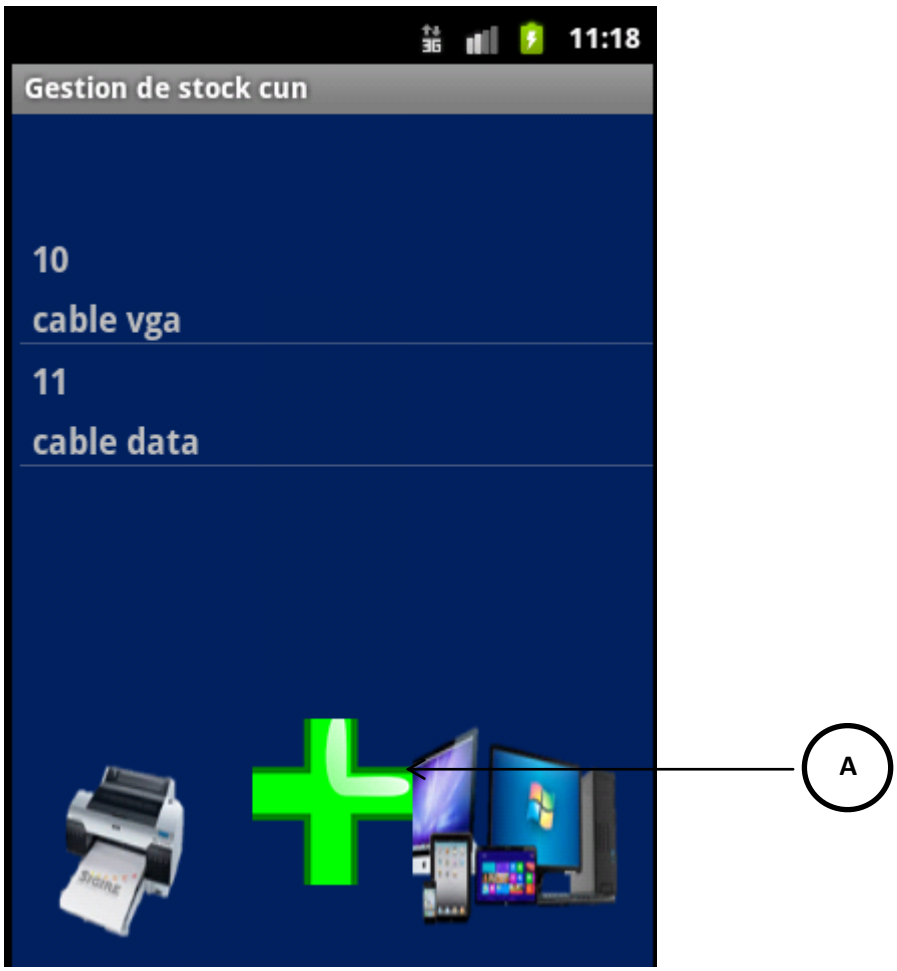



**Figure IV.7:** Interface de la liste de produit de type « matériel et consommable informatique ».

**(A)** : Ajouter un nouveau produit.

Si on clique sur un produit on va avoir la figure ci-dessous permettant la modification des informations relatives à un produit en cas de besoin.


**Figure IV.8:** Ecran de la modification d'un produit.

(1) : Modifier le nom du produit.

(2) : Modifier le type du produit.

(3) : Supprimer le produit.

(4) : Enregistrer le changement dans les champs d'un produit.

Si on clique sur le bouton (B) « La liste de produits » de la Figure IV.5, on obtient l'interface suivante:


**Figure IV.9:** Création d'un nouveau produit.

**(1)** : Saisir le nom du produit.

**(2)** : Saisir le type du produit.

**(3)** : Enregistrer le nouveau produit.

Si on clique sur le bouton **(c)** « Inventaire » de la Figure IV.5, on obtient l'interface suivante :


**Figure IV.10:** Ecran permettant la manipulation de l'inventaire.

**(1)** : Le produit le plus consommable.

**(2)** : Le produit le moins consommable.

**(3):** L'état actuel de stock.

**(4)** : Le produis non consommable.

Si on clique sur le bouton


On obtient l'interface suivante :

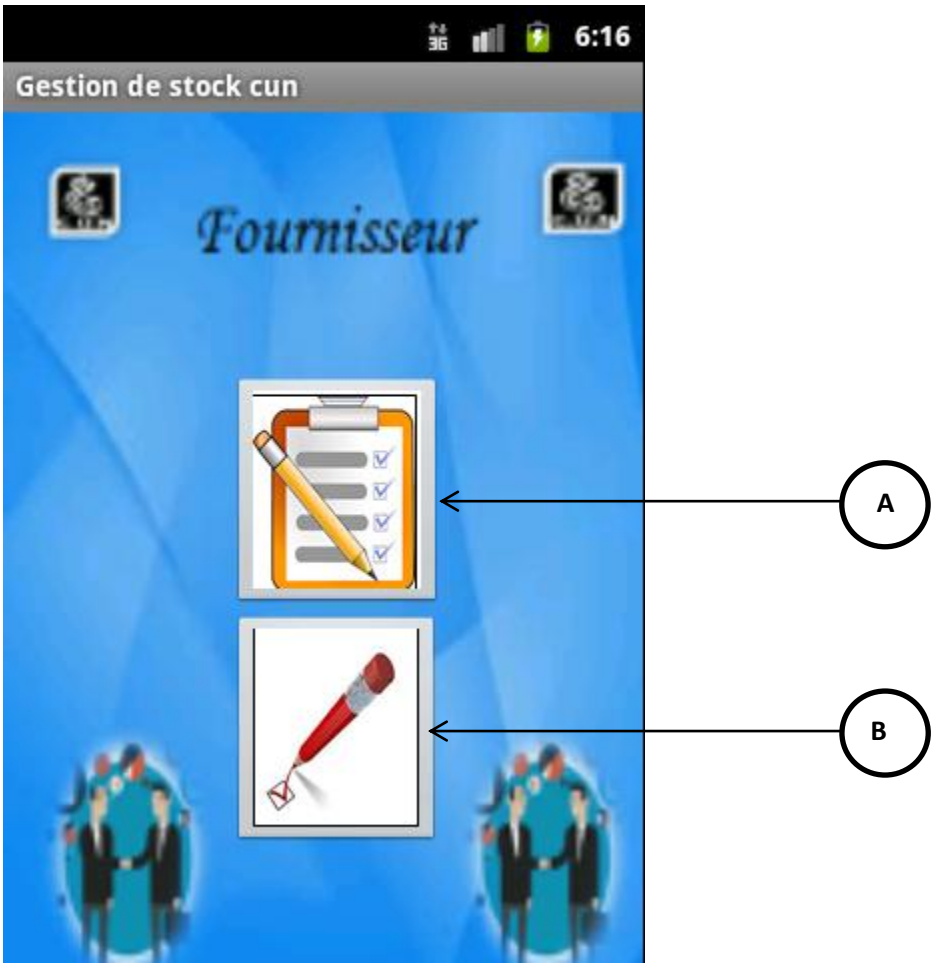



**Figure IV.11**: Ecran permettant la manipulation d'un fournisseur

**(A)** : Afficher la liste des fournisseurs.

**(B)** : Créer un nouveau fournisseur.

Lorsqu'on clique sur le bouton (A) de la Figure IV.12, la liste des fournisseurs s'affichera, et quand on veut modifier les informations du fournisseur, il suffit une clique sur le fournisseur qu'on veut modifier, et on obtient la figure suivante :


**Figure IV.12**: Modification d'un fournisseur.

**(1)** : Modifier le nom du fournisseur.

**(2)** : Modifier l'adresse du fournisseur.

**(3)** : Modifier le CCP du fournisseur (Compte de Code Postal).

**(4)** : Modifier le RIB (Le relevé d'Identité Bancaire).

**(5)** : Modifier le numéro de téléphone du fournisseur

**(6)** : Enregistrer les modifications dans les champs du fournisseur.

(**7)** : Supprimer le fournisseur.

Si on clique sur le bouton (B) de la Figure IV.12, on obtient l'interface suivante permettant d'ajouter un nouveau fournisseur.


**Figure IV.13** : Création d'un nouveau fournisseur**.**

Si on clique sur le bouton


On obtient :

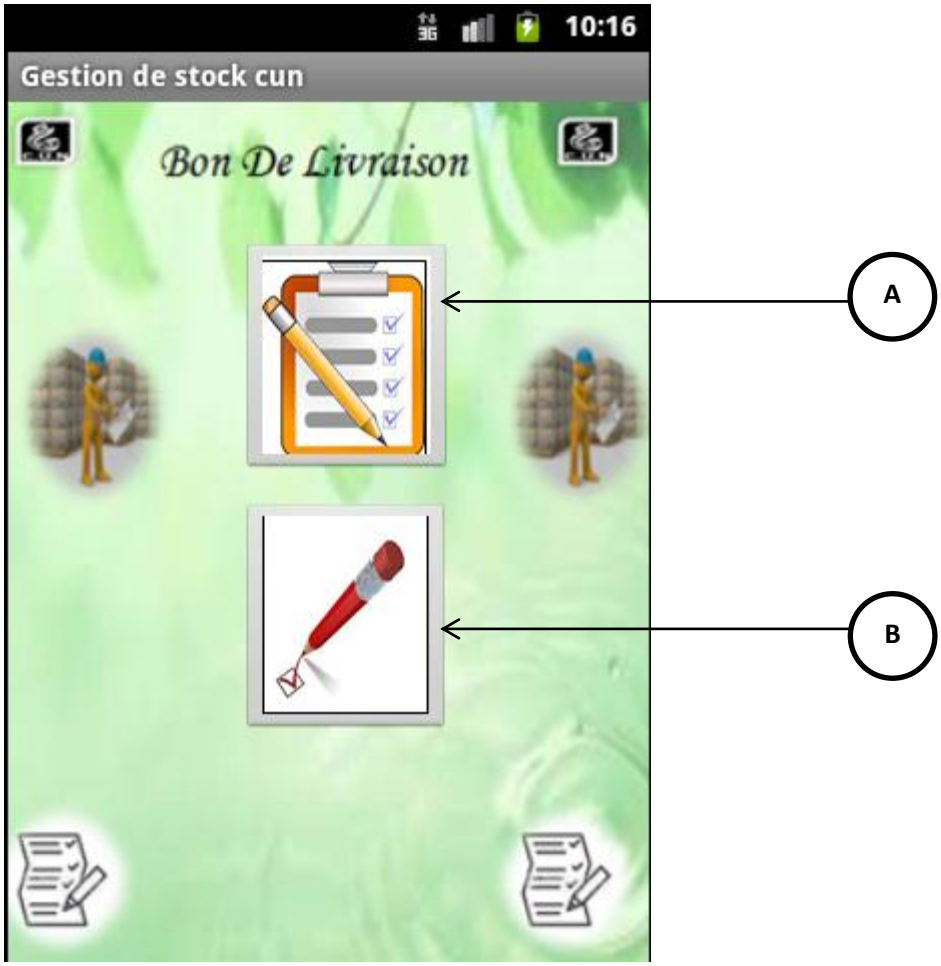



**Figure IV.14**: Gestion du bon de livraison.

**(A)** : Afficher la liste des bons de livraison.

**(B)** : Créer un nouveau bon de livraison.

Si on clique sur le bouton (A) de la Figure IV.15, on obtient la liste des bons de livraison.

Pour la modification, il faut tout simplement cliquer sur le bon de livraison qu'on veut modifier, et on obtient la figure suivante :

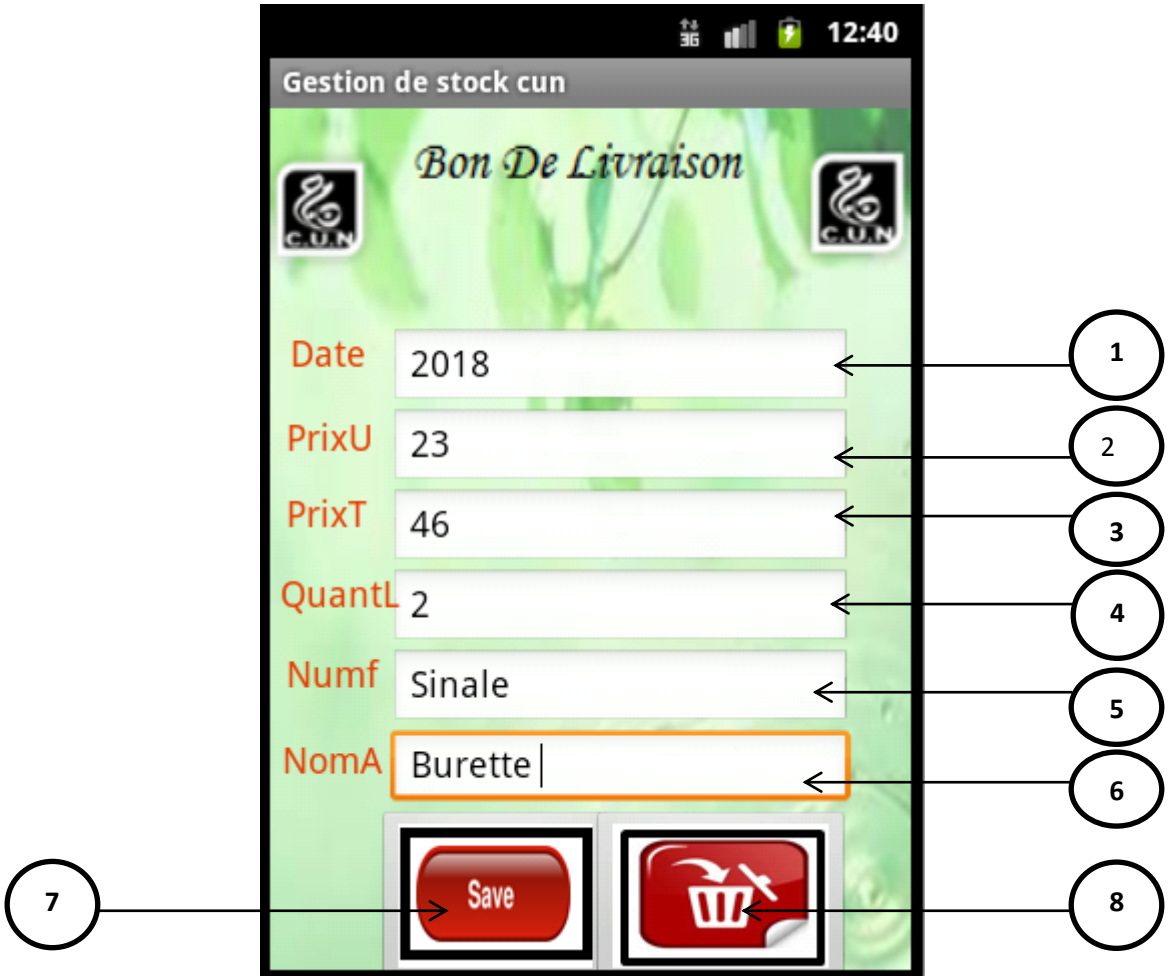



**Figure IV.15**: Modification d'un bon de livraison.

**(1)** : Modifier la date.

**(2)** : Modifier le prix unitaire du produit.

**(3)** : Modifier le prix total du produit.

**(4)** : Modifier la quantité livrée du produit.

**(5)** : Modifier le numéro du fournisseur.

**(6)** : Modifier le nom du produit.

**(7)** : Enregistrer les changements dans les champs du bon de livraison.

**(8)** : Supprimer le bon de livraison.

Si on clique sur le bouton (B) de la Figure IV.15, on obtient l'interface suivante pour remplir les champs d'un nouveau bon de livraison.


**Figure IV.16** : Création d'un nouveau bon de livraison.

Si on clique sur le bouton :


On obtient :

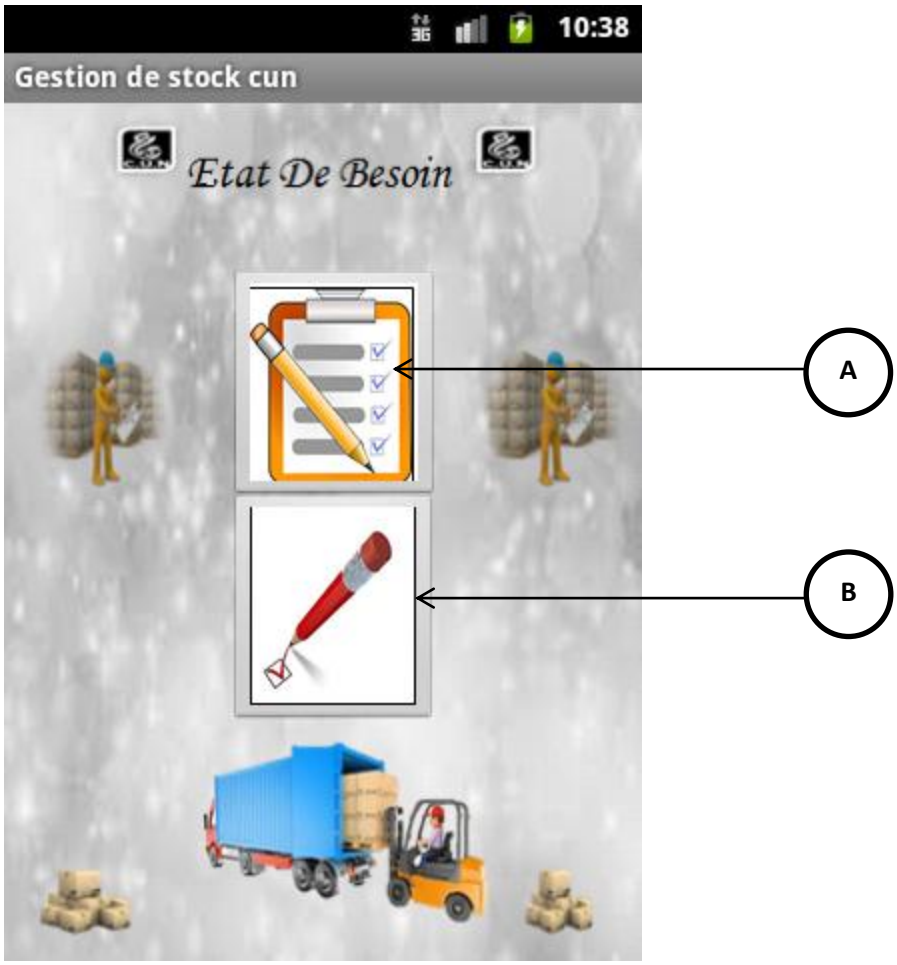



**Figure IV.17**: Gestion d'un état de besoin

**(A)** : Afficher la liste des états de besoin.

**(B)** : Créer un nouveau état de besoin.

Si on clique sur le bouton (A) de la Figure IV.18, on obtient la liste des états de besoin.

Pour la modification, il faut tout simplement cliquer sur l'état de besoin qu'on veut changer, et on obtient la figure suivante :


**Figure IV.18** : Modification d'un état de besoin.

***(1):*** Modifier le numéro de produit.

**(2)** : Changer la quantité des produits.

**(3)** : Modifier les observations.

**(4)** : Modifier le nom du service demandeur.

**(5)** : Changer la date de l'état de besoin.

**(6):** Modifier le numéro de produit

**(7)** : Supprimer l'état de besoin.

**(8)** : Enregistrer les changements dans les champs de l'état de besoin.

Si on clique sur le bouton (B) de la Figure IV.18, on obtient l'interface suivante pour remplir les champs d'un nouvel état de besoin.


**Figure IV.19** : Création d'un état de besoin.

Si on clique sur le bouton :


On obtient :

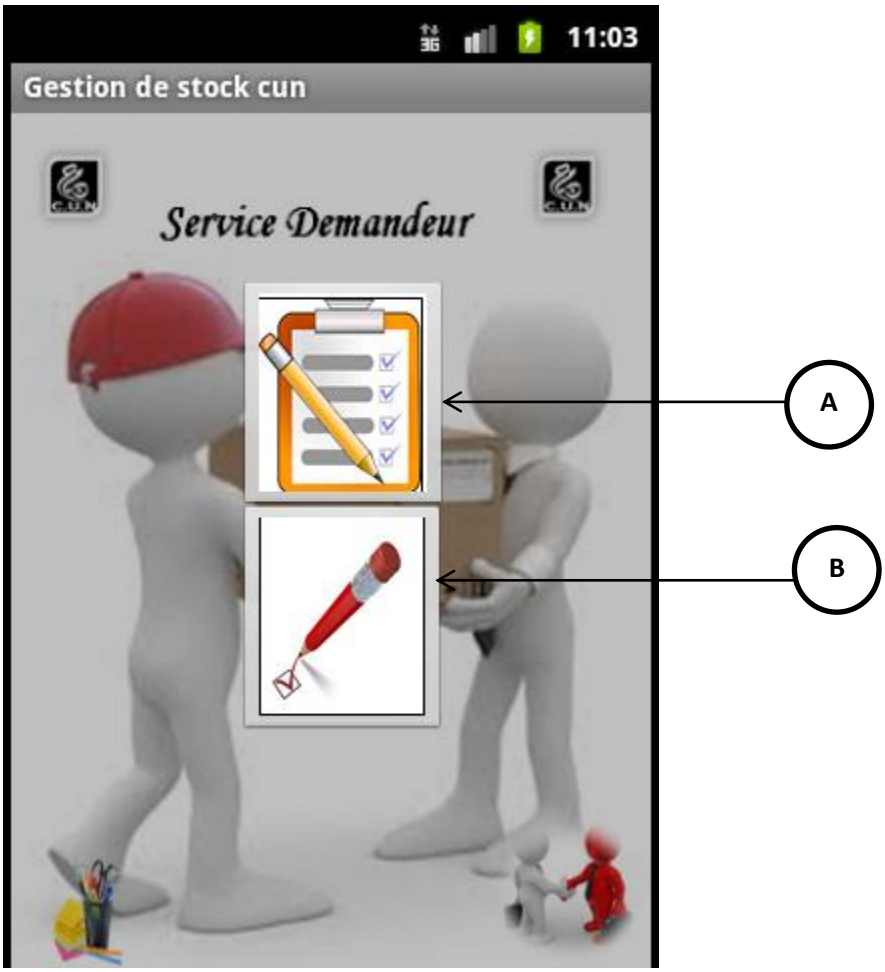



**Figure IV.20 :** Gestion d'un service demandeur.

**(A)** : Afficher la liste des services demandeur.

**(B)** : Créer un nouveau service.

Si on clique sur le bouton (A) de la Figure IV.21, on obtient la liste des services demandeur.

Pour la modification, il faut tout simplement cliquer sur le service qu'on veut changer, et on obtient la figure suivante :


**Figure IV.21 :** Modification d'un service demandeur.

**(1):** Modifier le nom du service demandeur.

**(2)** : Modifier le nom de demandeur.

**(3)** : Supprimer le service demandeur.

**(4)** : Enregistrer les changements dans les champs de service demandeur.

Si on clique sur le bouton (B) de la Figure IV.20, on obtient l'interface suivante pour remplir les champs d'un nouveau service demandeur *:*


**Figure IV.22** : Création d'un service demandeur.

## 5. Conclusion

Dans ce chapitre, nous avons présenté un guide d'utilisation de l'application, en représentant ses différentes interfaces.

# Conclusion Générale

Rappelons que l'objectif de ce travail est de concevoir et d'implémenter une application Java Android pour la gestion de stock au niveau du centre universitaire NAAMA. Pour cela, nous avons réalisé une application adaptée à tous les appareils téléphoniques mobiles (Smartphones ou tablettes), permettant de faciliter le travail de l'agent du magasin et d'évaluer avec plus souplesse l'exécution de ses tâches à partir de son appareil.

Ce projet a été très bénéfique pour nous, car il nous a permis de renforcer et enrichir nos connaissances théoriques dans le domaine de la conception, et de mettre en application nos connaissances acquises le long de nos études. Il nous a encore donné l'occasion de maîtriser le langage de programmation Java, et le système d'exploitation Android.

Comme perspectives, nous envisageons intégrer internet dans cette application afin de permettre à l'utilisateur d'accéder à cette plateforme sans les contraintes spatio-temporelles.

# Références bibliographiques


- Apollidore (2015).créez des applications pour Android. Teste de savoir.29 octobre 2015. Disponible sur le lien( http://www.adobe.com/support/downloads/product.jsp) Dernier accès 21/12/2017.

- C. Johnen (Vu 19/10/2018). *UML Cours*, Diagramme de communication. IUT de Bordeaux, Ver 2. Disponible sur le lien : (http://www.labri.fr/perso/johnen/pdf/IUT-Bordeaux/UMLCours/diagramme-Communication.pdf).

- Edraw(2018). *Chronologie de l'histoire du développement de téléphone mobile*.2018. Disponible sur le lien : (https.02://www.Edrawsoft.com/fr). Dernier accès 31/1/2018.

- Frédéric Espiau (2017). L'univers Android. OpenClassrooms. 2 Mars 2017. Disponible sur le lien : (https://openclassrooms.com/courses/creez-des-applications-pour-android/l-univers-android).

- Ikram SENOUSSAOUI et Asma LAKEHAL (2015). Création d'une application android via une autre application android (partie serveur). Mémoire de licence. Université Abou Bakr Belkaid- TLEMCEN. Annéé universitaire 2014-2015.

- Jenny Geneviève (2014). UML 2. Introduction générale à UML. 26 aout 2014.Disponible sur le lien : (http://jenny.bourdiol.org/frbook/uml-2x).

- J.STEFFE (2016). Cours UML. ENITA de Bordeaux. 12.02.2016. Disponible sur le lien : (http://www.anor.fr/fichiers/1.pdf).

- Joseph Gabay et David Gabay (2008). UML 2 analyse et conception. DUNOD, Paris.2008.

- Laetitia Matigno(2012). UML Unified Modeling Longuage diagrammes statiques.2012/20.13, Département Informatique - Polytech Lyon, Université Claude Bernard Lyon 1. Dernier accès 24/12/2017.

- Marwa BEN SIDIA (2012). Application Android "RiyadTowing". Mémoire de Master. Université virtuelle de Tunis. Année universitaire 2011-2012. Disponible sur le lien : (http://pf-mh.uvt.rnu.tn/806/1/Application-Android.pdf).

- Olivier Augereau(2012). Formation UML.12/04/2012.Disponible sur le lien : (https.02://www.enseirb.fr). Dernier accès (22/12/2017)

- Serge Ungar. Développer une application android : programmation java sous Eclipse.2013, edition ENI. Disponible sur le lien : (https://www.decitre.fr/livres/develppez-une application-android)

- Stéphane Crozat, Passage UMLRelationnel : classes et associations, 2 octobre 2016. Disponble sur le lien : (https://stph.scenari-community.org).

- Sylvain Hébuternr et Sébastien Perochon(2014).guide de développement d'application java Android pour Smartphones et tablettes .Eni .juin 2014. Dernier accès 28/1/2018.