



*Institut supérieur de formation continue d'Etterbeek*

*Enseignement de Promotion Sociale*

**Natural Corner, l'App au service du bio :**

**une application full-stack avec backend en PHP et frontend AngularJS au moyen du service cloud Google App Engine**

Daniel Imeri

Directeur de suivi : Christophe Lemaigre

Année scolaire : 2015-2016

*Travail de fin d'études présenté en vue  
de l'obtention du titre de Bachelier en informatique  
de gestion*

# Table des matières

<b>Table des matières</b>	<b>2</b>
<b>1 Introduction</b>	<b>5</b>
<b>I Méthodologie et architecture</b>	<b>7</b>
<b>2 Méthodologie</b>	<b>8</b>
2.1 UP simplifié . . . . .	8
<b>3 Architecture</b>	<b>10</b>
3.1 Choix technologiques . . . . .	10
3.1.1 Choix du langage . . . . .	10
3.1.2 Base de données . . . . .	11
3.2 Framework . . . . .	11
3.2.1 MVC . . . . .	11
3.2.2 Framework "fait maison" . . . . .	13
3.3 Outils logiciels . . . . .	14
<b>II Fonctionnalités de l'application Natural Corner</b>	<b>15</b>
<b>4 Expression initiale des besoins</b>	<b>16</b>
4.1 Vision du projet . . . . .	16
4.2 Analyse métier . . . . .	16
4.2.1 Natural Corner . . . . .	16
4.2.2 Logique métier . . . . .	17
4.3 Exigences fonctionnelles . . . . .	18
4.4 Exigences non fonctionnelles . . . . .	18
4.5 Exigences de performance . . . . .	18
4.6 Contraintes de conception . . . . .	19
<b>5 Cas d'utilisation</b>	<b>20</b>
5.1 Use case des utilisateurs . . . . .	20
5.2 Use case des employés . . . . .	21
5.3 Use case secondaire . . . . .	22
5.3.1 API . . . . .	22
5.4 Classement des cas d'utilisation et planification du projet en itération . . . . .	23
<b>6 Maquette</b>	<b>24</b>
6.1 Captures d'écran . . . . .	24
<b>III Spécifications détaillées des exigences</b>	<b>26</b>
<b>7 Descriptions des cas d'utilisation et diagrammes de séquence système</b>	<b>27</b>
7.1 UC Créer son compte client . . . . .	27

7.1.1	Description . . . . .	27
7.1.2	Diagramme de séquence système . . . . .	29
7.2	UC Gérer son compte client . . . . .	30
7.2.1	Description . . . . .	31
7.2.2	Diagramme de séquence système . . . . .	32
7.3	UC Recherche des produits . . . . .	32
7.3.1	Description . . . . .	32
7.3.2	Diagramme de séquence système . . . . .	34
7.4	UC Gérer son panier . . . . .	35
7.4.1	Description . . . . .	35
7.4.2	Diagramme de séquence système . . . . .	37
7.5	UC Commander . . . . .	38
7.5.1	Description . . . . .	38
7.5.2	Diagramme de séquence système . . . . .	38
7.6	UC Maintenir le catalogue . . . . .	39
7.6.1	Description . . . . .	39
7.6.2	Diagramme de séquence système . . . . .	41
<b>IV</b>	<b>Classes d'analyse et base de données</b>	<b>42</b>
<b>8</b>	<b>Conception objet des classes d'analyse</b>	<b>43</b>
8.1	Classes d'analyse . . . . .	43
8.1.1	Identification des concepts du domaine . . . . .	43
8.1.2	Ajout des associations et des attributs . . . . .	44
<b>9</b>	<b>Base de données</b>	<b>48</b>
9.1	Conception . . . . .	48
9.2	Modèle conceptuel de données (MCD) . . . . .	49
9.3	Modèle logique de données (MLD) . . . . .	50
9.4	Réflexions sur la conception de la base de données . . . . .	51
<b>V</b>	<b>Réalisation des cas d'utilisation</b>	<b>53</b>
<b>10</b>	<b>Conception et implémentation des UC</b>	<b>54</b>
10.1	UC "Créer son compte client" et "Gérer son compte client" . . . . .	54
10.1.1	Diagramme des classes participantes (DCP) . . . . .	54
10.1.2	Diagramme de navigation <sup>1</sup> . . . . .	56
10.1.3	Conception objet préliminaire . . . . .	57
10.1.4	Implémentation . . . . .	60
10.1.5	Difficultés rencontrées, réflexions et conclusions provisoires . . . . .	66
10.1.6	Première livraison du logiciel et feedback du client. . . . .	68
10.2	UC "Recherche des produits" . . . . .	68
10.2.1	Diagramme des classes participantes (DCP) . . . . .	68
10.2.2	Diagramme de navigation . . . . .	71
10.2.3	Conception objet préliminaire . . . . .	71
10.2.4	Implémentation . . . . .	77
10.2.5	Difficultés rencontrées, réflexions et conclusions provisoires . . . . .	79
10.3	UC "Gérer son panier" . . . . .	81
10.3.1	Diagramme des classes participantes (DCP) . . . . .	81
10.3.2	Diagramme de navigation . . . . .	82
10.3.3	Conception objet préliminaire . . . . .	83
10.3.4	Conception objet détaillée . . . . .	85
10.3.5	Tests unitaires . . . . .	86

1. Voir Annexe B

10.3.6	Code . . . . .	86
10.3.7	Difficultés rencontrées, réflexions et conclusions provisoires . . . . .	87
10.4	UC "Commander" . . . . .	87
10.4.1	Diagramme des classes participantes (DCP) . . . . .	87
10.4.2	Diagramme de navigation . . . . .	88
10.4.3	Conception objet préliminaire . . . . .	89
10.4.4	Conception objet détaillée . . . . .	90
10.5	UC "Maintenir catalogue" . . . . .	90
10.5.1	Diagramme des classes participantes (DCP) . . . . .	90
10.5.2	Diagramme de navigation . . . . .	92
10.5.3	Conception objet préliminaire . . . . .	93
10.5.4	Tests unitaires . . . . .	94
10.5.5	Code . . . . .	95
<b>11</b>	<b>Création de l'API et utilisation de AngularJS</b>	<b>97</b>
11.1	API REST . . . . .	97
11.2	AngularJS . . . . .	97
<b>12</b>	<b>Conclusion</b>	<b>100</b>
<b>Bibliographie</b>		<b>101</b>
<b>A</b>	<b>Dump du code MySQL généré par JMerisse (et corrigé manuellement)</b>	<b>102</b>
<b>B</b>	<b>Convientions pour les diagrammes de navigation</b>	<b>105</b>
<b>C</b>	<b>Créer les tests avec PHPUnit</b>	<b>106</b>
<b>D</b>	<b>Déployer une application en PHP sur Google App Engine</b>	<b>113</b>

# Chapitre 1

## Introduction

Natural Corner est un magasin bio du centre ville ouvert depuis environ 7 ans et qui propose une large variété d'articles bio allant des légumes et fruits aux produits cosmétiques en passant par les laits aux amendes<sup>1</sup>. Cette boutique bio jouit d'une jolie réputation et ne cesse d'améliorer son offre.

Lorsque j'ai eu une première entrevue avec le gérant du magasin Natural Corner, il m'a fait part de son envie de pouvoir un peu moderniser son magasin par le biais du numérique et proposer une nouvelle image résolument moderne à la clientèle. Je lui ai alors suggéré de créer une application web ou smartphone dans l'optique de l'écriture de mon travail de fin d'études. L'idée était que les clients aimeraient bénéficier du confort d'une précommande en ligne avant d'aller récupérer leur panier. Enthousiasmé par cette ébauche de projet, je l'ai rapidement proposé comme sujet de mémoire.

Les deux raisons principales qui m'ont amené à choisir le sujet d'une application web pour la vente en ligne d'un magasin bio sont d'une part, d'un point de vue technique, mon désir d'approfondir le domaine du développement web, et d'autre part, mon intérêt pour ce nouveau marché des produits bio proposant des alternatives de consommation intéressantes et des opportunités indéniables au niveau commercial.

Avec de nouvelles technologies proposées, la programmation web semble être arrivée à une certaine maturité. Les services PaaS offrent aux développeurs web des outils leur permettant de se passer d'une bonne partie de l'infrastructure et de sa configuration pour se concentrer sur le code et l'expérience des utilisateurs. Je perçois dans ces technologies la possibilité d'une très grande autonomie pour le développeur. Il est aussi intéressant de constater qu'une série de framework orienté web en langage javascript, aussi bien frontend que backend, innondent le marché (meteorjs, angularjs, nodejs, backbonejs,...) et qu'il s'agit sans doute d'un signe d'engouement pour ce type de programmation. La puissance croissante des moteurs javascript des navigateurs et la pile HTML5 proposant des services très nombreux, la possibilité d'être, à l'instar des applications Java, dans une certaine mesure "cross-plateforme", et la nécessité de pouvoir offrir une expérience UI similaire sur les différents appareils (Android, IOS, Windows phone, ChromeBook, laptop) poussent les technologies web à s'améliorer sans cesse.

Dans un tel contexte, j'aimerais expérimenter un ensemble de choix de langages qui peuvent apporter une solution élégante à une question récurrente dans le développement des applications web. Comment envisager la création d'une application web qui pourra rester évolutive et "maintenable" lors d'une migration vers une application pour smartphone sous Android ou IOS ? La réponse à cette question est en partie l'objet de mon TFE. La solution réside de le découplage du front-end et du back-end par l'intermédiaire d'un service web REST (qui envoie sous forme de requête http des objets au format JSON).

A ce jour le marché du bio répond à une forte demande. Bien qu'encore élitiste, ce marché semble grandir sans cesse, permettant aux consommateurs exigeants quant à la qualité des produits et à l'impact écologique de leur mode de vie de consommer conformément à leurs principes. Etant personnellement attentif aux problèmes d'environnement, je vois dans le marché bio la possibilité d'une remise en question de notre mode de consommation sans entrer en contradiction avec le commerce tel qu'il fonctionne traditionnellement. De plus, l'idée de proposer aux clients des produits frais et sains n'est pas pour me déplaire.

La création d'une application web multiplateforme, cependant, semble un challenge difficile à relever. En effet, il s'agit non seulement de créer une application dont la logique serveur propose une persistence des données et les calculs nécessaires au service de vente, mais aussi une partie cliente qui devra proposer sous une forme conviviale le catalogue des produits du magasin Natural Corner. Après des semaines de développements et d'analyses, il a été finalement

---

1. <http://www.naturalcorner.be/>

décidé de créer une partie serveur pouvant servir de base multiplateforme pour n'importe quelle application cliente de façon modulaire.

L'enjeu de ce travail de fin d'étude est de proposer une solution complète « full stack », c'est-à-dire allant de la base de donnée à une application cliente, le backend au frontend, munie d'un déploiement sur le Cloud qui permettra de répondre à la demande initiale du gérant du magasin. Tout cela passera par la mise en place d'un framework orienté objet pour assurer la maintenabilité de l'application.

Il est toutefois important de ne pas avoir les yeux plus grands que le ventre et de rester humble dans ses objectifs. Je n'ai pas l'ambition d'égaler une équipe de développement de plusieurs développeurs et analystes spécialisés dans des techniques pointues. Mon objectif principale est de créer les bases solides d'une application WEB qui sera réutilisable et aisément maintenable. Sa capacité à proposer un frontend sera réalisée par la mise en place d'une API de type REST suffisamment flexible pour accueillir par exemple une application Android, iOS ou AngularJS.

Comment réaliser ce programme en partant de zéro sans prendre trop de risques ? Il s'agit de mener l'analyse depuis le recueil des besoins jusqu'à l'écriture des diagrammes UML dans l'optique d'arriver à une description fidèle de l'application. Alors seulement l'écriture du code devient aisée et limpide car elle correspond aux différents points de vue conceptuels de l'analyste, transformés de la manière la plus adéquate en code par le programmeur.

La coopération avec le gérant du magasin Natural Corner fut très profitable pour l'écriture de ces diagrammes UML, qui ne sont que la traduction de la logique métier du magasin. J'ai eu plusieurs interview dans lesquels j'ai récupéré et consignés ses attentes.

Il s'agit donc avant tout d'un travail d'analyse en ce sens que son résultat doit être indépendant de tout langage, framework ou architecture logicielle. Cette démarche ne présuppose tout au plus que l'équipe de développement (dans ce cas, le rédacteur de ces lignes) se mette d'accord sur le design pattern d'architecture MVC pour l'écriture du logiciel.

L'implémentation passe par la mise en oeuvre d'un framework Model-View-Controller qui n'existe pas en tant que tel dans le langage de programmation choisi. Ce fut la part du travail la plus ardue mais aussi la plus intéressante en ce sens qu'elle est la jonction entre le mode de pensée orienté objet et son abstraction intrinsèque à la réalité technique du langage de programmation. Ce travail a dû se plier à l'exigence du travail patient du programmeur, fait d'essais et d'erreur, d'égarements plus ou moins longs et de persévérance dans ses objectifs. Au final, c'est une solution viable de l'implémentation de l'analyse et de ses diagrammes UML qui a vu le jour. Ce travail peut donc être considéré, en plus d'être l'application du magasin Natural Corner, comme la création d'un framework MVC PHP. Le déploiement, quant à lui, se fera sur le Cloud. Plus exactement sur un PaaS bien connu nommé Google App Engine. Ce PaaS<sup>2</sup> propose une API complète et permet de déployer rapidement dans un contexte professionnel une application web.

Pour résumer ce travail, il s'agira du résultat de la création d'un framework MVC codé à partir de zero, pouvant s'utiliser avec un PaaS, permettant de créer une application côté serveur et muni d'une API basée sur la technologie REST. Une telle approche a l'ambition de répondre à la question actuelle du développement « full stack ». Elle correspond à la philosophie orientée objet en ce sens que le code n'est que l'aboutissement d'une réflexion d'analyse et l'architecture qu'une manière flexible d'implémenter par un procédé technique cette réflexion. Cette application va aussi profiter des services Cloud à faible coût donnant au développeur des résultats se rapprochant le plus possible de la demande initiale du client<sup>3</sup>.

---

2. Rappelons qu'un Palteforme As A Service est un ensemble d'outils en ligne se basant sur la technologie Cloud pour accélérer la productivité du développeur d'une part, et de permettre une grande capacité d'adaption au traffic de l'application, d'autre part. Il est aussi possible de faire des choix technologiques rapides sans passer par des configurations complexes (comme passer d'une base de donnée SQL à une base de donnée Big Data No SQL par exemple).[https://fr.wikipedia.org/wiki/Plate-forme\\_en\\_tant\\_que\\_service](https://fr.wikipedia.org/wiki/Plate-forme_en_tant_que_service)

3. Le code de l'application se trouve sur le Git public <https://github.com/userdanydan/NaturalCorner/tree/GoodVersion>

**Première partie**

**Méthodologie et architecture**

## Chapitre 2

# Méthodologie

### 2.1 UP simplifié

La méthodologie utilisée est celle préconisée par le livre de Pascal Roques "UML2 Modéliser une application web"<sup>1</sup> qui consiste à utiliser la méthode Unified Process dans une optique simplifiée pour s'adapter idéalement à l'analyse et la conceptualisation d'une application web. Une grande partie des schémas de ce travail sont directement inspirés de ce livre. Ce fut une occasion unique pour me familiariser avec les diagrammes UML et de les assimiler progressivement. La pratique m'a permis d'intégrer la logique sous-jacente de ces diagrammes et leurs connexions avec les besoins du client, la vision du logiciel et le code. Les premiers diagrammes pourront apparaître comme de pâles copies des diagrammes du livre mais sont en fait une version adaptée à l'application Natural Corner. Plus j'avancais dans le travail, plus j'arrivais à me détacher du livre pour produire des schémas entièrement originaux (voir l'UC Maintenir catalogue), prouvant à mon sens l'acquisition de la compétence nécessaire pour le métier d'analyste-développeur.

---

1. Roques, P., UML2 Modéliser une application web, Eyrolles 4ème édition, 2008

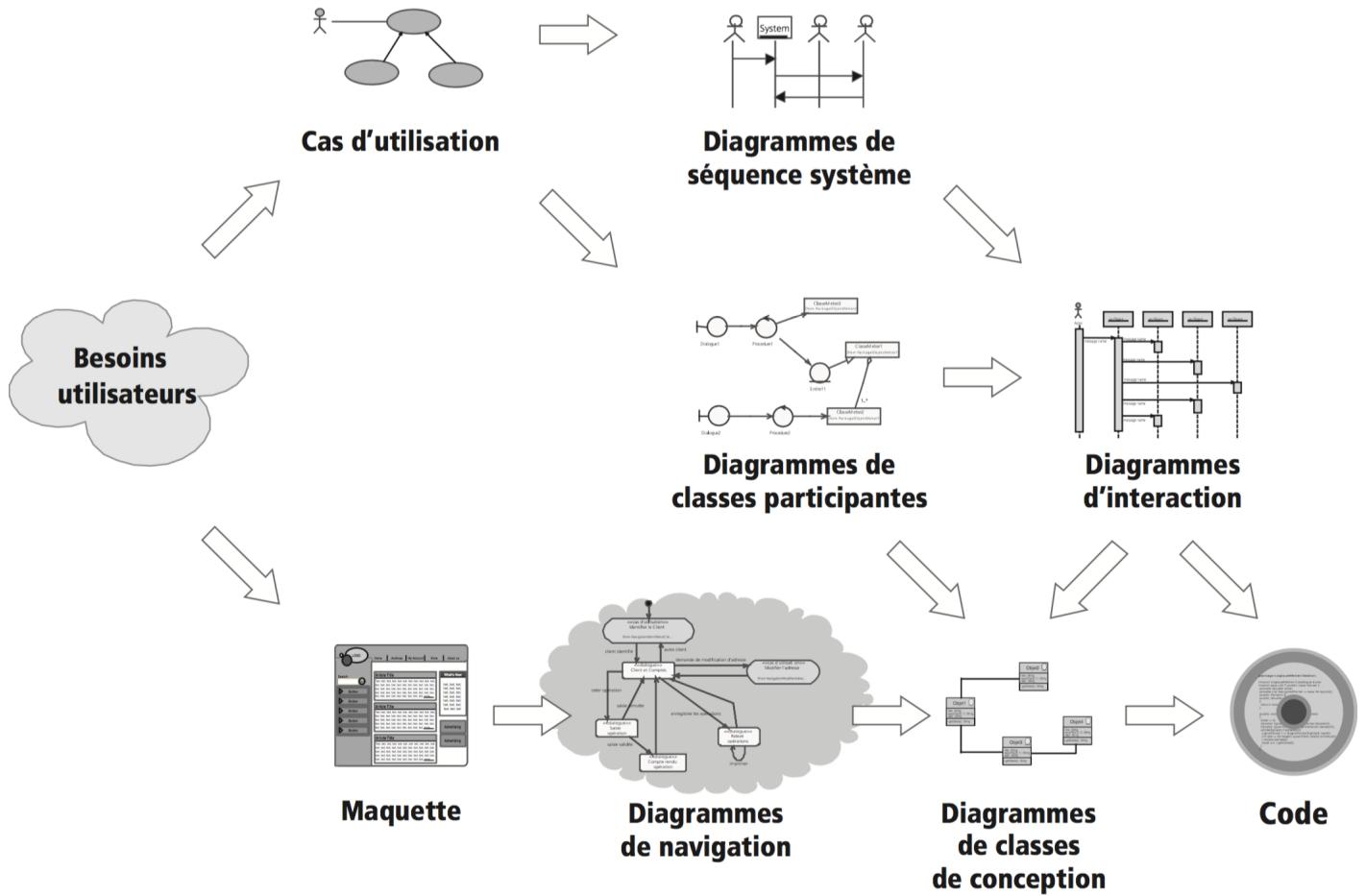


FIGURE 2.1.1 – Méthodologie

Ce schéma résume la démarche du livre mentionné. En partant du recueil des besoins de l'utilisateur, il est possible de définir les cas d'utilisation, un diagramme de séquence système et une description détaillée. C'est l'objet de la partie "Fonctionnalités de l'application Natural Corner". Une maquette peut aussi être réalisée mais elle n'a pas été nécessaire de perdre du temps pour une activité plutôt destinée à un graphiste. Ensuite, on peut produire un Diagramme des Classes Participantes qui est une première ébauche de la vision statique de l'application. A partir de ce DCP, on commence à réfléchir plus profondément à la dynamique interne à l'aide des diagrammes d'interaction. Un diagramme de navigation issu en principe de la maquette va être utile pour comprendre la manière dont la partie Vue du logiciel sera utilisée. Enfin, l'ensemble de ces points de vue va aboutir à l'écriture des diagrammes de classes de conception. Dans ma démarche, j'ajoute à cela l'écriture des tests unitaires à partir de ce schéma qui comporte le nom des méthodes des différentes classes. Finalement le code peut être écrit et l'application créée. Par ailleurs, la lecture du livre de référence en programmation orientée objet de Graig Larman<sup>2</sup> a guidé la problématique d'assignation des responsabilités aux objets de l'analyse représentés sous forme de diagrammes UML.

2. Larman, G., Applying UML and Patterns , An introduction to object-oriented analysis and design and unified process, Seconde édition, 2001

# Chapitre 3

## Architecture

### 3.1 Choix technologiques

#### 3.1.1 Choix du langage

J'ai choisi le langage PHP pour plusieurs raisons. Sa popularité (Facebook est programmé en PHP), la possibilité de programmer en orienté objet, le fait d'être libre de créer mon propre framework (par opposition à JavaEE ou .NET). C'est un langage de script, interprété, peu typé, laissant un certain laxisme pour la programmation. Le développement TDD<sup>1</sup> orienté par les tests permet d'éviter certains écueils (Pas de compilateur pour corriger avant le runtime les erreurs de syntaxe). En contre-partie les performances sont moins bonnes qu'avec les langages compilés. Enfin, j'ajouterais mon expérience personnelle dans ce langage et le fait que le service PaaS que j'utilise autorise l'usage de PHP 5.5 (Google App Engine).

**PHP contre JavaEE** JavaEE très bon pour un grosse équipe car il sépare le travail entre différents groupes avec netteté. Les page .jsp nuisent à ma façon de créer le framework point de vue de la courbe d'apprentissage. Par ailleurs, il faut apprendre un framework Strut ou Spring MVC car JavaEE seul est rarement utilisé.

**PHP contre .NET** Très bon pour grosse équipe. Le XAML et le patron de conception architectural MVVM<sup>2</sup> sous jacent avec le langage C# qui est sans doute un des meilleurs langages orientés objet. Sa courbe d'apprentissage est acceptable mais quid d'une migration vers un nouvel appareil ? Il y a aussi l'obligation de se plier au patron MVVM qui n'est pas le MVC, qui me semble plus adéquat pour l'analyse. Il y a aussi le fait qu'il faut utiliser les produits Microsoft (Visual Studio, I2S), ...

**PHP avec Angularjs, Android et iOS** Si la vue utilise un web service REST, il est découpé avec la frontend et permet de développer sans code PHP la vue. Angularjs<sup>3</sup> est un framework frontend qui permet d'étendre le HTML. Il utilise le patron MVC et le langage javascript. Il peut être utilisé en fragment en laissant de côté certains aspects techniques plus complexes. Une simple instruction en HTML permet de charger cette bibliothèque et ne demande donc rien de particulier pour les outils de développement. La documentation est excellente et l'équipe de Google dans une optique de promotion de ce framework fait un gros travail pour faciliter la courbe d'apprentissage. Si j'arrive à découpler convenablement le frontend du backend, je pourrai envisager une version Android voire IOS de cette application. Il ne s'agira en fait que d'une vue permettant aux utilisateurs de communiquer avec le backend écrit en PHP.

**Bootstrap<sup>4</sup>** Ce framework CSS est très utile et donne des rendus impressionnantes avec un minimum d'effort. Il est "responsive" et permet de développer "mobile-oriented".

---

1. [https://fr.wikipedia.org/wiki/Test\\_driven\\_development](https://fr.wikipedia.org/wiki/Test_driven_development)  
2. <https://msdn.microsoft.com/en-us/library/hh848246.aspx>  
3. <https://angularjs.org/>  
4. <https://getbootstrap.com/>

### 3.1.2 Base de données

**MySQL** Très populaire, s'interface bien avec PHP, documentation claire, excellent tutoriel sur openclassroom<sup>5</sup> (ancienne étudiante de l'ULB), stackoverflow<sup>6</sup> déborde de questions et réponses sur ce gestionnaire de base de données, autant de bonnes raisons pour l'adopter. J'ai hésité car PostgreSQL est orienté objet et permet donc de se passer du mapping relationnel-objet. J'ai finalement décidé de ne pas utiliser d'ORM et de mettre en place les requêtes SQL adéquates. J'étais aussi attiré par une base de donnée no SQL comme mongoDB. Cependant, je pense que les données que je vais utiliser demande d'être structurées (par opposition aux enregistrements clé-valeur s'imbriquant et pouvant laisser la place à une certaine difficulté pour la maintenance et les requêtes), en particulier les lignes d'achat et les commandes, je préfère donc rester en SQL relationnel.

## 3.2 Framework

Pour rappel un Framework est un cadre de travail qui permet au programmeur de développer une application logicielle selon une certaine architecture basée sur un patron de conception. Cette architecture est munie la plupart du temps d'une API( pour les utilisateurs ou les formulaires par exemple) permettant d'augmenter la productivité du programmeur. Un travail professionnel dans le domaine de l'informatique de gestion suppose de nos jours pratiquement à chaque fois l'utilisation d'un framework. En PHP, il existe Symfony<sup>7</sup> ou Zend<sup>8</sup> (parmi des dizaines d'autres). Ils sont très demandés dans l'industrie logiciel. Cependant, ici, il n'est pas vraiment question de rentrer dans les détails d'un framework particulier, d'en acquérir les connaissances utiles pour le monde du travail et de rester ainsi bloquer dans un créneau pour le reste de sa carrière (pour autant que ce framework n'ait pas été dépassé par un autre). Il s'agit plutôt de comprendre la logique générale d'un framework en le créant soi-même, afin de rester maître en tout situation de l'outil technologique utilisé. En soi, les frameworks, mêmes s'ils diffèrent en langage de programmation et/ou en pattern de conception architectural, répondent toujours à la même logique fondamentale. Une fois comprise, il est possible de s'adapter rapidement (sans être trop optimiste toutefois, un framework comme Struts et JavaEE demande des mois d'études) à n'importe quel outils de création de logiciels. En adoptant cette logique, il s'agit de mettre à profit mes études d'informatique de gestion et de rester dans une démarche rationnelle quant aux outils de développement.

### 3.2.1 MVC

Le site du professeur B. Estellon<sup>9</sup> fournit pour ses étudiants une framework rudimentaire qui s'apparente à ce qu'on appelle un micro-framework<sup>10</sup>. Selon l'article de Wikipedia, quatre fonctionnalités sont assurées : 1) les comptes ; 2) l'abstraction de la base de données ; 3) la vérification des données introduites par l'utilisateur et 4) le template web<sup>11</sup>. Eh bien, il s'agit des quatre fonctions que vont remplir ce framework. Voici un schéma expliquant le fonctionnement générale :

---

5. <https://openclassrooms.com/courses/administrez-vos-bases-de-donnees-avec-mysql>

6. <https://stackoverflow.com/>

7. <https://symfony.com/>

8. <http://framework.zend.com/>

9. [http://pageperso.lif.univ-mrs.fr/~bertrand.estellon/web2/cours\\_mvc.pdf](http://pageperso.lif.univ-mrs.fr/~bertrand.estellon/web2/cours_mvc.pdf)

10. <https://en.wikipedia.org/wiki/Microframework>

11. « It lacks most of the functionality which is common to expect in a full fledged web application framework, such as ; Accounts, authentication, authorization, roles, etc. Database abstraction via a object-relational mapping. Input validation and input sanitation. Web template engine. »><https://en.wikipedia.org/wiki/Microframework>

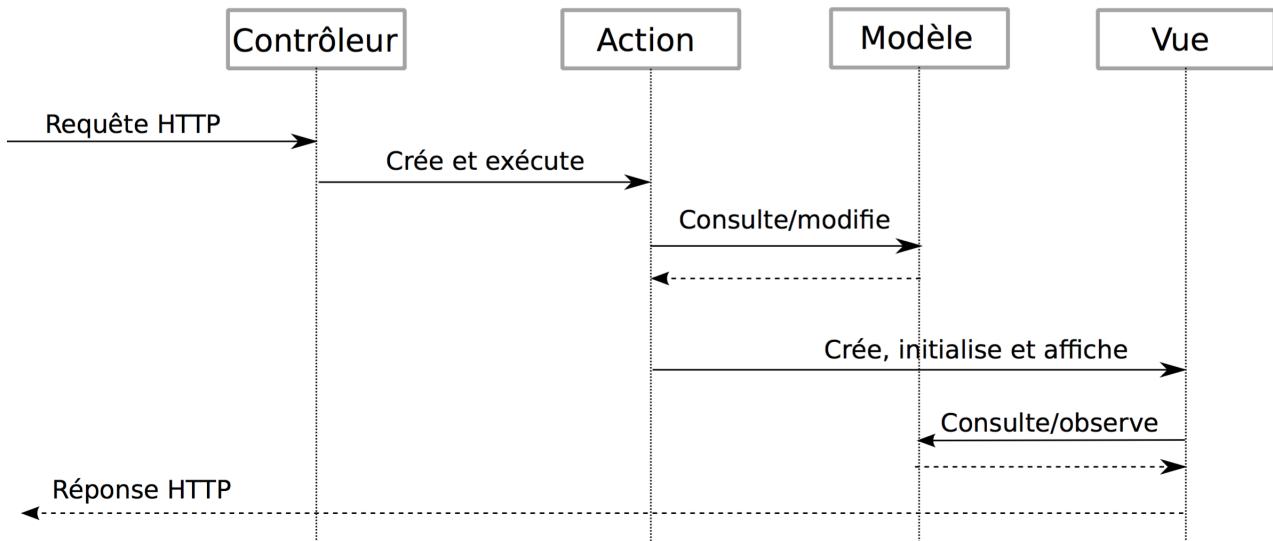


FIGURE 3.2.1 – MVC

En PHP, le fichier index.php ressemble à ceci :

```

5    session_start();
6    $action = getAction();
7    $action->run();
8    $view = $action->getView();
9    $view->setLogin($action->getSessionLogin());
10   $view->run();
11   function getActionByName($name)
12   {
13       $name .= 'Action';
14       include("actions/$name.inc.php");
15       $action1 = new $name();
16       return $action1;
17   }
18   function getViewByName($name)
19   {
20       /* Factory */
21       $name .= 'View';
22       include("views/$name.inc.php");
23       $view1 = new $name();
24       return $view1;
25   }
26   function getAction()
27   {
28       /* Factory */
29       $actions = array('Default', 'Login', 'Logout', 'Inscription',
30                       'Enregistrement', 'Accueil', 'UpdateUser',
31                       'VoirCompte', 'UserJSON', 'RechercheRapide',
32                       'RechercheAvancee', 'Panier', 'Catalogue',
33                       'Gerant', 'ModifierArticle', 'Commande');
34
35       $path = parse_url($_SERVER['REQUEST_URI'], PHP_URL_PATH);
36       $path = ltrim($path, '/');
37       if (!in_array($path, $actions))
38           $action = 'Default';
39       else
40           $action = $path;
41       return getActionByName($action);
42   }

```

FIGURE 3.2.2 – index.php

C'est la réalisation du patron de conception Front controller<sup>12</sup>. Il va centraliser les requêtes du protocole HTML. Chaque requête est en même temps séparée par une action et une vue. Les lignes 35 à 40 ont été ajoutées pour récrire les URL. J'aimerais attirer l'attention du lecteur sur le fait que le langage PHP permet une très large marge de manœuvre comparativement aux autres langages web comme JavaEE ou C# et donne vraiment un sentiment de liberté au développeur. Une idée est toujours réalisable avec un peu de recherche personnelle.

### 3.2.2 Framework "fait maison"

Un framework<sup>13</sup> fait par soi-même contient plusieurs avantages<sup>14</sup> :

- Plus facile à mettre à jour et à maintenir puisque je l'ai construit.
- Pas de problèmes de licences.

12. [https://en.wikipedia.org/wiki/Front\\_controller](https://en.wikipedia.org/wiki/Front_controller)

13. <http://pageperso.lif.univ-mrs.fr/~bertrand.estellon/web2/>

14. Michael Peacock, PHP 5 Social Networking, Packt Publishing, 2010, p.14-15

- Pousse à une connaissance étendue de la conception orientée objet et des technologies WEB (PHP, requêtes http, css, javascript, HTML5, ...). D'une part, les choix d'implémentation doivent être réfléchis en fonction de la conception objet et de l'analyse, d'autre part des efforts de compréhensions supplémentaires sont nécessaires pour appréhender les différents concepts, que ce soit du point de vue du langage web (ici PHP et la pile HTML5) ou du déploiement sur l'internet. Rien n'est laissé à une couche logicielle pré-construite, très utile en terme de productivité certes mais occultant souvent les raisonnements sous-jacents et le bien fondé de l'utilisation de tel ou tel patron de conception. Il s'agit à mon avis d'un must dans une optique d'apprentissage.
- Le code est plus efficient puisqu'il ne s'embarasse pas des différents modules et couches logiciels des frameworks de l'industrie.

### 3.3 Outils logiciels

- Eclipse PHP<sup>15</sup> (version Mars) me convient car j'ai l'habitude de l'utiliser. Il est particulièrement populaire, open-source et agréable à utiliser. L'auto-completion est assurée et il prévoit des outils de développement pour le javascript et le html-css. Je peux utiliser si je désire le débuggeur (ici X-DEBUG). J'ai accès à un CVS (avec Github) grâce à un module téléchargé et ajouté, je peux aussi faire des tests unitaires avec PHPUnit qui est interfacé dans ce EDI.
- Navigateur Chrome<sup>16</sup> : offre des outils pour le développeurs web, notamment une console javascript, une possibilité de voir le rendu sur différents devices (smartphones, tablettes)
- StarUML<sup>17</sup> : ce logiciel permet de créer des diagrammes UML de qualité.
- Google App Engine<sup>18</sup> : Ce service PaaS de Google est très intéressant car il permet de déployer sur le web une application très facilement. Il propose une base de données Google Cloud SQL (MySQL) et un certain nombre d'APIs plus ou moins intéressantes. Tout ce qui j'apprends en déployant et maintenant une application web avec ce service cloud est utilisable sur n'importe quel autre service Cloud. Je pense profondément qu'il s'agira d'un plus indéniable pour ma carrière de développeur.
- On peut ajouter GitHub<sup>19</sup>, serveur open source et partagé pour git, permettant de garder un historique et des "forks" de mes versions de l'application.

---

15. <https://eclipse.org/pdt/>

16. <https://www.google.com/chrome/browser/desktop/index.html>

17. <http://staruml.io/>

18. <https://cloud.google.com/appengine/docs>

19. <https://github.com/userdanydan/NaturalCorner>

## **Deuxième partie**

### **Fonctionnalités de l'application Natural Corner**

## Chapitre 4

# Expression initiale des besoins

### 4.1 Vision du projet

#### Demande initiale

Le gérant du magasin Natural Corner, spécialisé en produit bio, aimerait offrir à sa clientèle une application web utilisable sur tablette et smartphone pour leur permettre de pré-commander les produits frais. L'objectif fondamental est de permettre aux clients de chercher les produits frais par catégorie ou par mot-clé, en connaître le prix, de créer un panier virtuel et les pré commander éventuellement. L'application doit aussi permettre au gérant du magasin de faciliter l'écoulement des invendus en proposant une promotion journalière sur ces produits frais. Enfin, les produits devront afficher des informations en relation avec la fraîcheur, la qualité et le développement durable.

#### Positionnement

L'application Natural Corner devra être la plateforme de commande des produits frais. Il existe déjà un site internet et ne devra pas le concurrencer. Le but du projet consiste à créer une étape supplémentaire dans le logique "bio" du magasin en rationalisant la commandes des produits frais en relation avec les fermes et producteurs proches du magasin. Une gestion à flux tendu et en relation avec les récoltes se rapproche de la philosophie "bio" et du développement durable. Se différencier des autres magasins "bio" en proposant un plus contemporain qui plaira à la jeune clientèle .

### 4.2 Analyse métier

#### 4.2.1 Natural Corner

Le 26 avril 2009<sup>1</sup>, Natural Corner ouvre son enseigne au milieu de la ville de Bruxelles dans une situation idéale. Aboudi Ei, ancien restaurateur dans un restaurant de la place de la vieille halle au blé a été éduqué culinairement à l'utilisation des produits locaux. Pendant ses années de travail dans la restauration, il a appris que la qualité des aliments est au moins aussi importante que la maîtrise de l'art de cuisiner. Il s'est pris de passion pour le bio et a voulu changer la direction de sa carrière professionnelle en réalisant le projet d'ouverture d'un magasin bio. Les autres passionnés pourront ainsi plus facilement se procurer des aliments de qualité.

A. Ei avait constaté en effet le manque criant de boutique bio dans le centre ville. Il n'y avait alors guère que le magasin bio de la rue des Chartreux pouvant proposer des produits bio labelisés. Le coin de la rue de l'escalier donnant sur la place de la vieille halle au blé était alors à vendre. Les locaux étaient un endroit idéal pour créer le magasin. Ce fut un succès quasi immédiat. Après quatre mois de travail ardu, à la rentrée, la clientèle s'est manifesté avec enthousiasme et le succès n'a fait qu'augmenter depuis. Ce succès vient du service et de l'écoute des clients. Le catalogue a changé de 90 pour cent depuis l'ouverture. Ce sont les discussions avec les clients, leurs attentes et leurs désirs qui ont été mis en avant. Le choix des produits était à chaque fois à l'origine d'une demande. La clientèle est particulièrement diversifiée, toutes les tranches d'âge viennent se procurer au magasin Natural Corner.

---

1. Cette section est le fruit d'une discussion avec le gérant de Natural Corner

#### 4.2.2 Logique métier

Du point de vue du business, on peut dire que Natural Corner fonctionne selon un circuit court à flux tendu, le stock est à son état minimum et les commandes maximisées de manière à garantir la fraîcheur des produits. Les dates doivent être les meilleures proposées, aussi les quantités commandées sont petites. Cela représente beaucoup de travail physique, on peut compter plus de 50 commandes par semaine et les commandes de produits frais ont lieu tous les jours. Pour garantir la fraîcheur, les bacs sont régulièrement changés, les dates vérifiées systématiquement. Une ristourne à l'avantage du client est proposé lorsqu'un produit va être sur le point de perdre sa fraîcheur, ce qui facilite le roulement du stock.

Pour accomplir ce travail, le staff est représenté par un gérant et trois vendeurs. Le magasin Natural Corner bénéficie du label Biogarantie<sup>2</sup>.



FIGURE 4.2.1 – Label Biogarantie de Natural Corner

Deux points de vue se dégagent : le point de vue des clients du magasin Natural Corner, et celui des clients du marché belge utilisant déjà une application de vente en ligne pour commander des produits bio.

Les clients de Natural Corner sont des habitués recherchant des produits spécifiques. Ils aiment en général l'idée d'avoir un magasin bio dans le quartier. Cependant, le site internet du magasin n'offre que des informations d'ordre général. Il serait intéressant de leur proposer une application leur permettant de faire leur course en ligne et d'éviter de chercher dans les rayons.

En ce qui concerne le marché bio, il existe une série de site web proposant les achats en ligne. En faisant une simple recherche Google, on trouve une grande quantité de site français qui propose la commande en ligne de produits bio. Il s'agit souvent de sites traditionnels ne proposant pas d'app IOS ou Android et n'étant pas responsive. Il y a <http://www.greenweez.com/> dont le site est responsive et qui livre en Belgique, par exemple.

Le plus gros concurrent, à mon sens, est sans doute Collect&Go du groupe Colruyt sous le nom Bioplanet qui propose une app et qui est responsive pour les tablettes et smartphones. Après avoir exprimé cette opinion au gérant, il m'a expliqué que le positionnement n'est en fait pas le même. Dans le cas des grandes enseignes comme Bio Planet ou Farm, l'assortiment proposé correspond au minimum des exigences bio, l'important restant le prix et la rentabilité<sup>3</sup>. On peut prendre l'exemple connu du thé Yogitheia qui a disparu des rayons parce que son prix avait subitement augmenté en raison de la variation du prix du marché du thé. Par opposition, un magasin bio de proximité continue à proposer un produit à ses clients si la demande existe.

Notons l'existence de Delhaize qui propose son assortiment bio en ligne et Carrefour avec son Carrefour-Drive.

Natural Corner doit se dégager de cette concurrence en proposant une approche familiale des commandes en ligne. Tout l'enjeu est de garder l'esprit du magasin tout en se modernisant par une application web.

Une des idées qui est ressortie pendant mes interviews avec le gérant du magasin est de mettre en relation la page Facebook du magasin, des recettes de cuisine et les promotions. Le créneau sur lequel cette application se situe n'est pas celui des grands sites de vente en ligne impersonnels, il s'agit de faire participer la communauté du quartier et les fans de produits bio. Plus qu'une simple opportunité d'augmenter ses parts de marché, il est surtout question de rendre la démarche de consommation bio agréable et amicale.

2. <http://www.biogarantie.be/fr>

3. A titre informatif, le label d'un magasin ou d'un produit doit toujours être vérifié en priorité pour tout ce qui relève de l'agriculture biologique [http://ec.europa.eu/agriculture/organic/index\\_fr.htm](http://ec.europa.eu/agriculture/organic/index_fr.htm)

### 4.3 Exigences fonctionnelles

Cette application permettra à l'utilisateur de rechercher un produit, d'en découvrir les spécificités, de le choisir et de le précommander.

#### Rechercher un produit

Le catalogue des produits frais permet d'y rechercher un produit. Il existera plusieurs méthodes de recherche. Le critère pourra être le nom, le mot-clé, la saison, le prix, ... Les résultats seront facilement consultables et reclassables. Il y aura par défaut une offre lui permettant de voir les produits sélectionnables.

#### Informations sur le produit

Le produit se présentera au client en affichant : Une image le prix et la disponibilité la saison son origine une information de nature didactique sur les vertus curatives par ex.

#### Choix du produit

Il y aura un panier qui se remplira au fur et à mesure des sélections sur client. Une fois rempli le panier, celui-ci pourra être validé sur une page spécifique. Ce panier pourra être changé ou supprimé par la suite.

#### Commande

Un formulaire de bon de commande confirme le panier. En fonction de la disponibilité le client est invité à venir chercher son panier au magasin.

#### Promotions du jour

Une promotion journalière sera envoyé aux clients par notification. Les employés du magasin seront chargés d'envoyer cette notification.

#### Facebook

Le client devra "liker" la page Facebook de la boutique au moins une fois pour accéder aux services de l'application.

#### Messages sociaux et durables

L'application devra permettre aux employés du magasin d'informer par un système de news sur les nouveaux produits.

### 4.4 Exigences non fonctionnelles

#### Qualité professionnelle

L'application devra montrer un bon niveau de qualité afin de renvoyer un certain standing.

#### Graphisme

Les logos et couleurs de l'enseigne doivent être gardés.

### 4.5 Exigences de performance

Le niveau de performance n'est pas essentiel puisqu'il vise une centaine de clients pour commencer. Cependant il faudra tenir en compte la possibilité de monter en charge dans les mois suivants.

## 4.6 Contraintes de conception

### Base de données des produits

La base de données des produits sera créée avec les catalogues des fournisseurs. Le gérant du magasin aura la possibilité d'ajouter ou de retirer des produits.

### Base de données des clients

Les clients donneront des informations personnelles qui seront enregistrées dans la base de données. Ces informations seront sécurisées et confidentielles.

### Panier

Le panier sera enregistré sous forme de cookies pour une durée d'un an. Le client pourra ainsi retrouver son panier tel quel dans une nouvelle session.

### Support

L'application sera compatible sur les tablettes et smartphone IOS et Android.

# Chapitre 5

## Cas d'utilisation

Nous allons essayer de dégager à qui et pour quoi faire les informations vont être montrées<sup>1</sup>. Nous allons identifier les acteurs et les cas d'identification.

### 5.1 Use case des utilisateurs

L'acteur est l'utilisateur de l'application. Les cas d'utilisation ont été mis en évidence par l'expression des besoins préliminaire : Créer son compte, gérer son compte, chercher des produits, gérer son panier, commander.

---

1. Roques, P., UML2 Modéliser une application web, Eyrolles 4ème édition, 2008, p.40

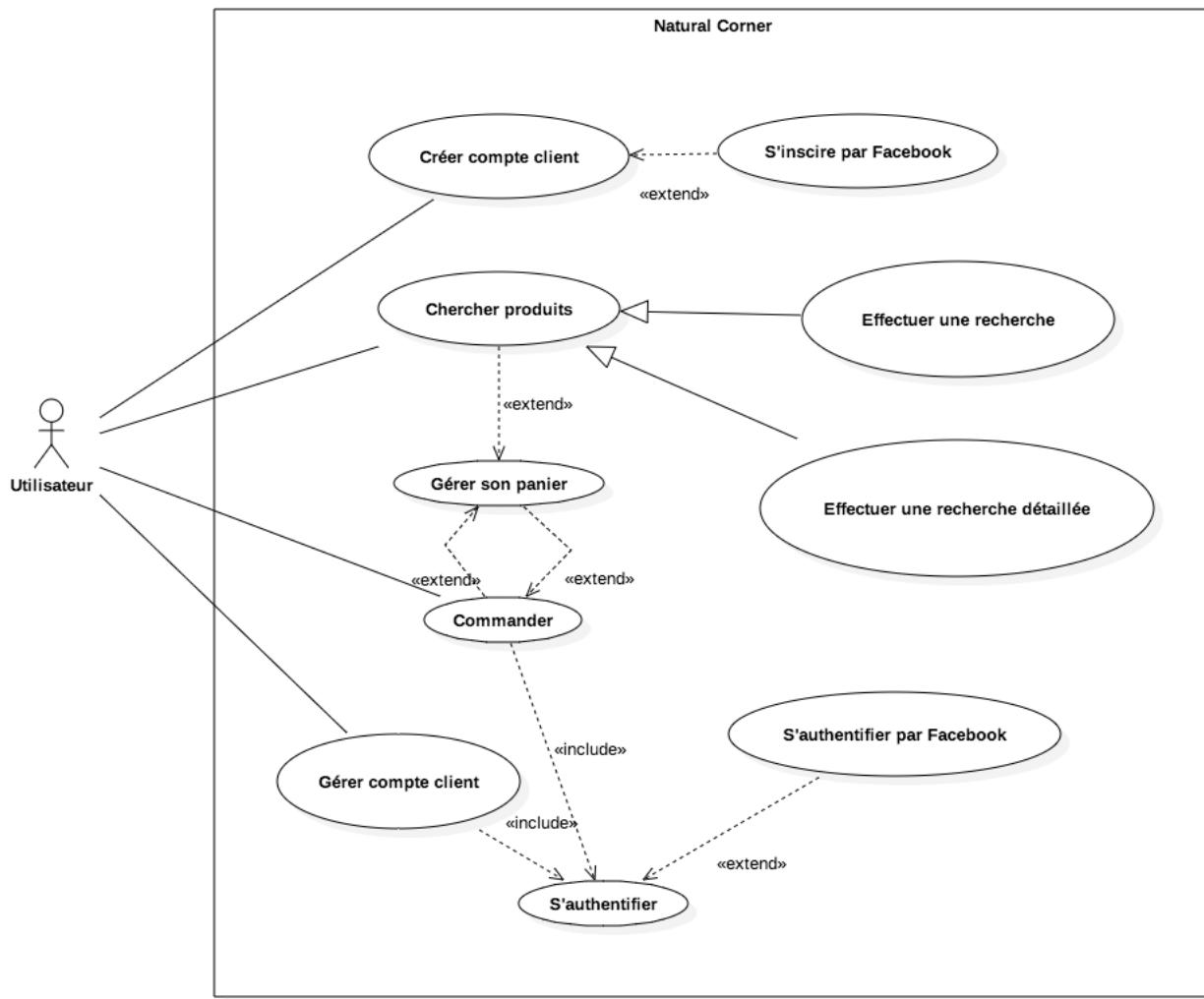


FIGURE 5.1.1 – Use case des utilisateurs

## 5.2 Use case des employés

Un gérant hérite d'un vendeur parce qu'il peut réaliser son travail et a, en plus, la charge de la mise à jour du catalogue. Les cas d'utilisation sont maintenir le catalogue et envoyer des promos.

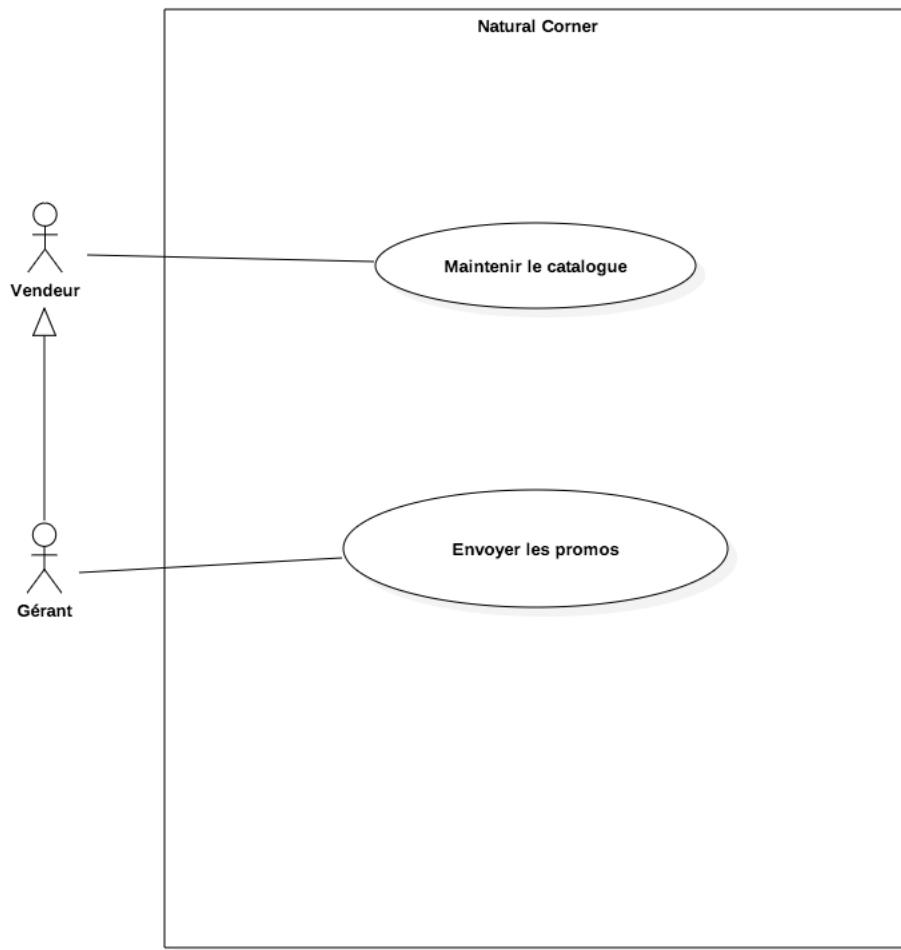


FIGURE 5.2.1 – Use case des employés

### 5.3 Use case secondaire

#### 5.3.1 API

Il s'agit d'un cas d'utilisation secondaire qui correspond à la capacité de l'application à être utilisée par une autre application. Son développement aura lieu au chapitre 11.

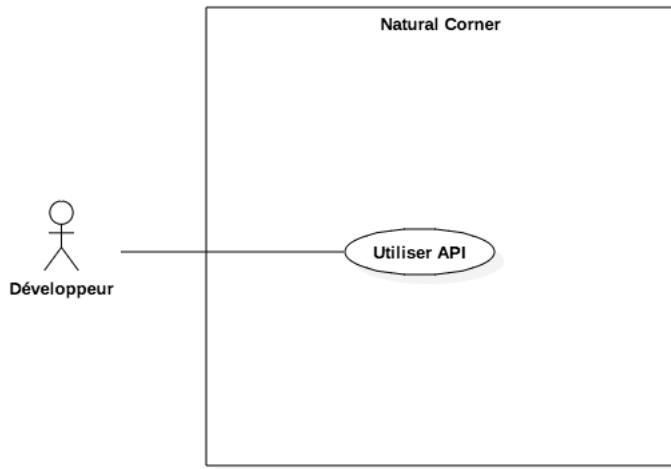


FIGURE 5.3.1 – Use case secondaire - API

#### 5.4 Classement des cas d'utilisation et planification du projet en itération

Pour réaliser le logiciel, j'ai décidé de donner une priorité aux cas d'utilisation en fonction des risques et de la priorité. Chaque cas d'utilisation correspond à une itération selon la méthodologie UP.

Cas d'utilisation	Priorité	Risque	Itération
Créer un compte client	Haute	Moyen	1
Gérer son compte client	Moyenne	Moyen	2
Recherche des produits	Haute	Moyen	3
Gérer son panier	Haute	Haute	4
Commander	Moyenne	Haut	5
Maintenir le catalogue	Haute	Moyen	6
Envoyer les notifications promotionnelles	Haute	Haut	7
Developper une API	Moyenne	Bas	8

# Chapitre 6

## Maquette

La maquette a été créée en langage statique HTML. L'application doit garder le logo du magasin et ses couleurs en général. Des diagrammes de navigation ont été élaborés pour les différents cas d'utilisation dans la partie de réalisation des cas d'utilisation.

### 6.1 Captures d'écran

Il est important de garder une allure correspondant aux application Android et iOS. On voit la possibilité de se logguer avec Facebook et aussi avec un email et une mot de passe. Le catalogue des fruits et légumes reste simple, sans surcharge.

The screenshot displays a mobile application interface for a grocery store. At the top, there are three icons: a user profile, a shopping cart, and a search function. The main content area is a table listing three items:

Sélection	Image	Dénomination	Prix Unitaire	Commentaire	Quantité désirée
■		orange	2.00 €	bio de Valence	<input type="button" value="1"/>
■		poire	1.20 €	poire bio	<input type="button" value="1"/>
■		carotte	2.30 €	carotte super bio	<input type="button" value="1"/>

On the right side of the screen, there is a sidebar with a menu icon. The menu items are: Voir Compte (Account), Voir Panier (View Cart), Chercher (Search), and Déconnexion (Logout). The "Voir Panier" item is highlighted in blue.

FIGURE 6.1.1 – Navigation

Voici ce que donne l'application dans un format correspond à un smartphone.

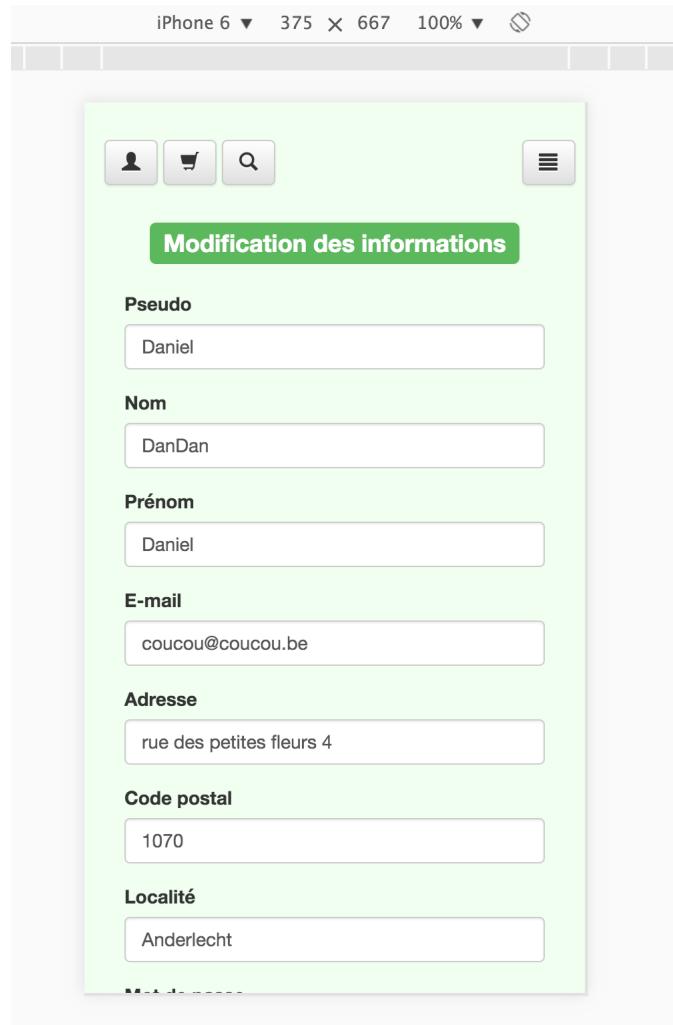


FIGURE 6.1.2 – Mobile-first design

## **Troisième partie**

# **Spécifications détaillées des exigences**

## Chapitre 7

# Descriptions des cas d'utilisation et diagrammes de séquence système

Ce chapitre va décrire de manière détaillée les cas d'utilisation et à chaque fois remplir une fiche-type pour chaque cas<sup>1</sup>. Il s'agit de décrire un scénario nominal en se mettant à la place de l'utilisateur dans son rôle vis-à-vis de l'application. Ensuite vient le diagramme de séquence système qui va considérer l'application comme une boîte noire avec laquelle l'acteur interagit.

### 7.1 UC Créer son compte client

#### 7.1.1 Description

Description du UC 'Créer son compte client'	
<b>Acteur principal</b>	— Le visiteur.
<b>Acteurs secondaires</b>	— Néant.
<b>Objectifs</b>	— Le visiteur veut pouvoir accéder au contenu de l'application en indiquant son adresse e-mail et un mot de passe. Il peut aussi s'il le désire indiquer d'autres informations (nom, nom de famille, etc...).
<b>Préconditions</b>	— L'application est déployée avec une base de données prête.
<b>Postconditions</b>	— La base de données contient un utilisateur en plus et celui-ci peut utiliser l'application.
<b>Scénario nominal</b>	<ol style="list-style-type: none"><li>1. Le visiteur appuie sur le bouton "Inscription" de la page d'accueil.</li><li>2. La page d'inscription s'affiche.</li><li>3. Le visiteur indique son email et un mot de passe qu'il confirme.</li><li>4. Le visiteur est enregistré et dirigé vers la page d'accueil.</li></ol>

1. Roques, P., UML2 Modéliser une application web, Eyrolles 4ème édition, 2008, p.57

**Alternatives**

- 1a Le visiteur se loggue avec Facebook.
- 1. Le visiteur appuie sur le bouton “Facebook”.
- 2. Un compte client est automatiquement créé.
- 3.a Le visiteur indique un mot de passe ne correspondant pas aux critères de sécurité.
- 1. Après avoir appuyé sur “envoyer”, l’application lui transmet un message d’erreur.
- 3.b Le visiteur confirme son mot de passe avec un autre mot de passe.
- 1. Après avoir appuyé sur “envoyer”, l’application lui transmet un message d’erreur.
- 3.c Le visiteur décide de remplir les autres champs non obligatoires.
- 1. Si un des champs ne correspond pas aux critères, l’application lui transmet un message d’erreur.

### 7.1.2 Diagramme de séquence système

L'inscription laisse le choix entre une inscription dite classique et une inscription par l'intermédiaire du réseau social Facebook. Plusieurs champs sont optionnels, en fait tous les champs sauf l'adresse e-mail et le mot de passe. Veuillez noter que la flèche représentant le message vers le système Natural Corner n'est pas rempli en noir, non pas que, comme l'indique la documentation UML 2, il s'agit d'un signal asynchrone (qui n'arrête pas l'exécution du process en attente d'une réponse) mais plutôt qu'il n'est pas utile de s'arrêter à ce niveau de détails pour un diagramme de séquence système. Nous voulons seulement indiquer qu'il existe une interaction séquentielle, peu importe qu'elle soit synchrone ou asynchrone.

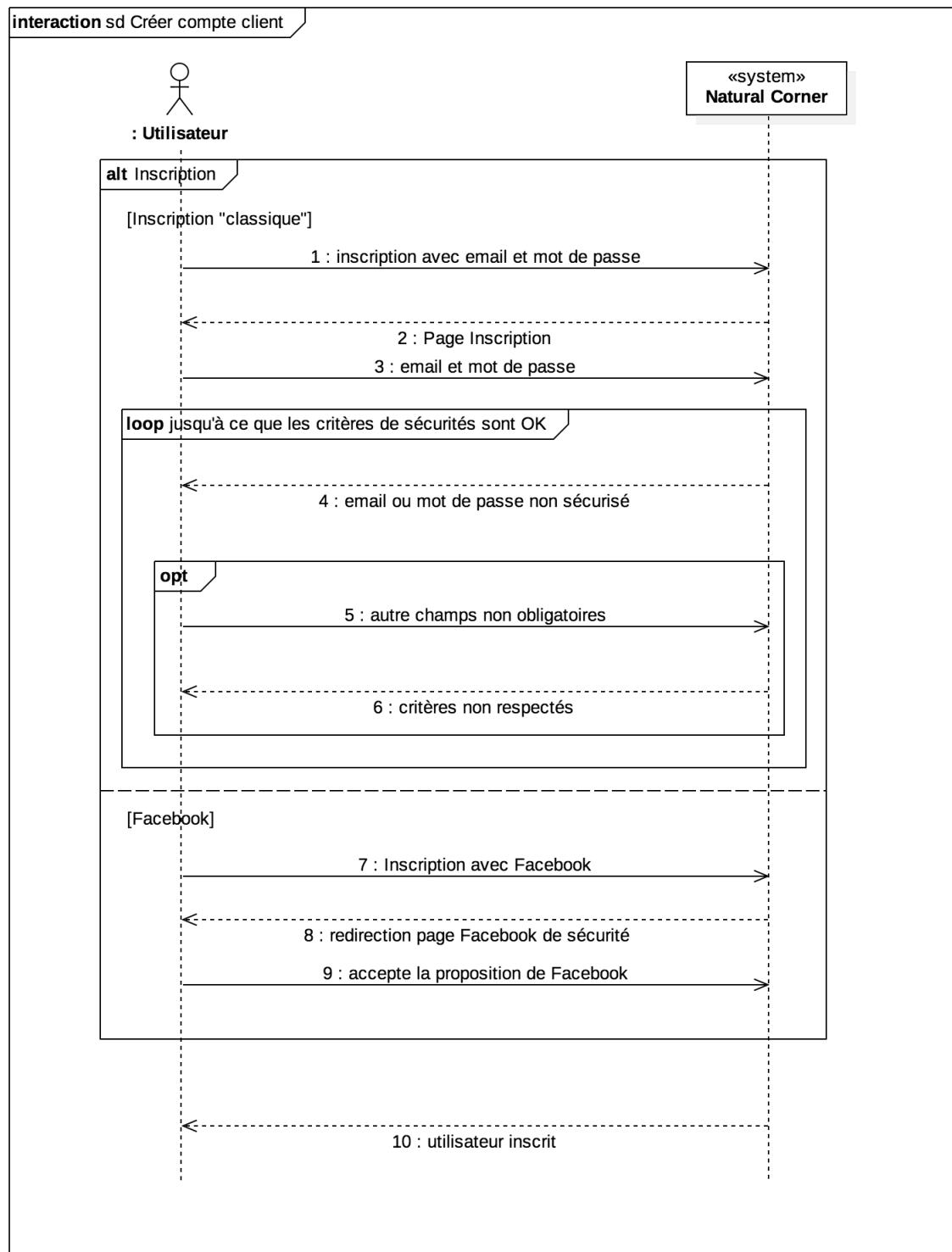


FIGURE 7.1.1 – Diagramme de séquence système "Créer son compte client"

## 7.2 UC Gérer son compte client

Ce Use Case est très similaire au précédent. Il s'agit de permettre à l'utilisateur de changer ses données.

### 7.2.1 Description

<b>Description du UC 'Gérer son compte client'</b>	
<b>Acteur principal</b>	— Le client.
<b>Acteurs secondaires</b>	— Néant.
<b>Objectifs</b>	— Le client veut changer son compte client avec des nouvelles informations.
<b>Préconditions</b>	— Le client est déjà enregistré et loggué.
<b>Postconditions</b>	— La base de données contient des nouvelles informations sur le client.
<b>Scénario nominal</b>	<ol style="list-style-type: none"> <li>1. Le client appuie sur l'icône du petit bonhomme ou dans le burger menu sur "Voir Compte".</li> <li>2. La page du client avec ses informations s'affiche.</li> <li>3. Le client appuie sur le bouton "Modifier" et est dirigé vers un formulaire prérempli contenant ses information.</li> <li>4. Le client modifie ses information selon son envie.</li> <li>5. Le client appuie sur le bouton rouge "modifier", est dirigé vers la page d'accueil et un message lui indique que ses informations ont été mises à jour.</li> </ol>
<b>Alternatives</b>	<ul style="list-style-type: none"> <li>— 5.a Le client indique un mot de passe ne correspondant pas aux critères de sécurité.</li> <li>1. Après avoir appuyé sur "Modifier", l'application lui transmet un message d'erreur.</li> <li>— 5.b Le client confirme son mot de passe avec un autre mot de passe.</li> <li>1. Après avoir appuyé sur "Modifier", l'application lui transmet un message d'erreur.</li> <li>— 5.c Le client décide de remplir les autres champs.</li> <li>1. Si un des champs ne correspond pas aux critères, l'application lui transmet un message d'erreur.</li> </ul>

### 7.2.2 Diagramme de séquence système

On voit l'ordre de séquencement dans le cas d'utilisation. L'app Natural Corner est vraiment considérée du point de vue de l'utilisateur sans se soucier du fonctionnement internet de l'application. "Loop" signifie que l'utilisateur introduit les données jusqu'à ce qu'il se conforme à ce que l'application lui renvoie comme indications.

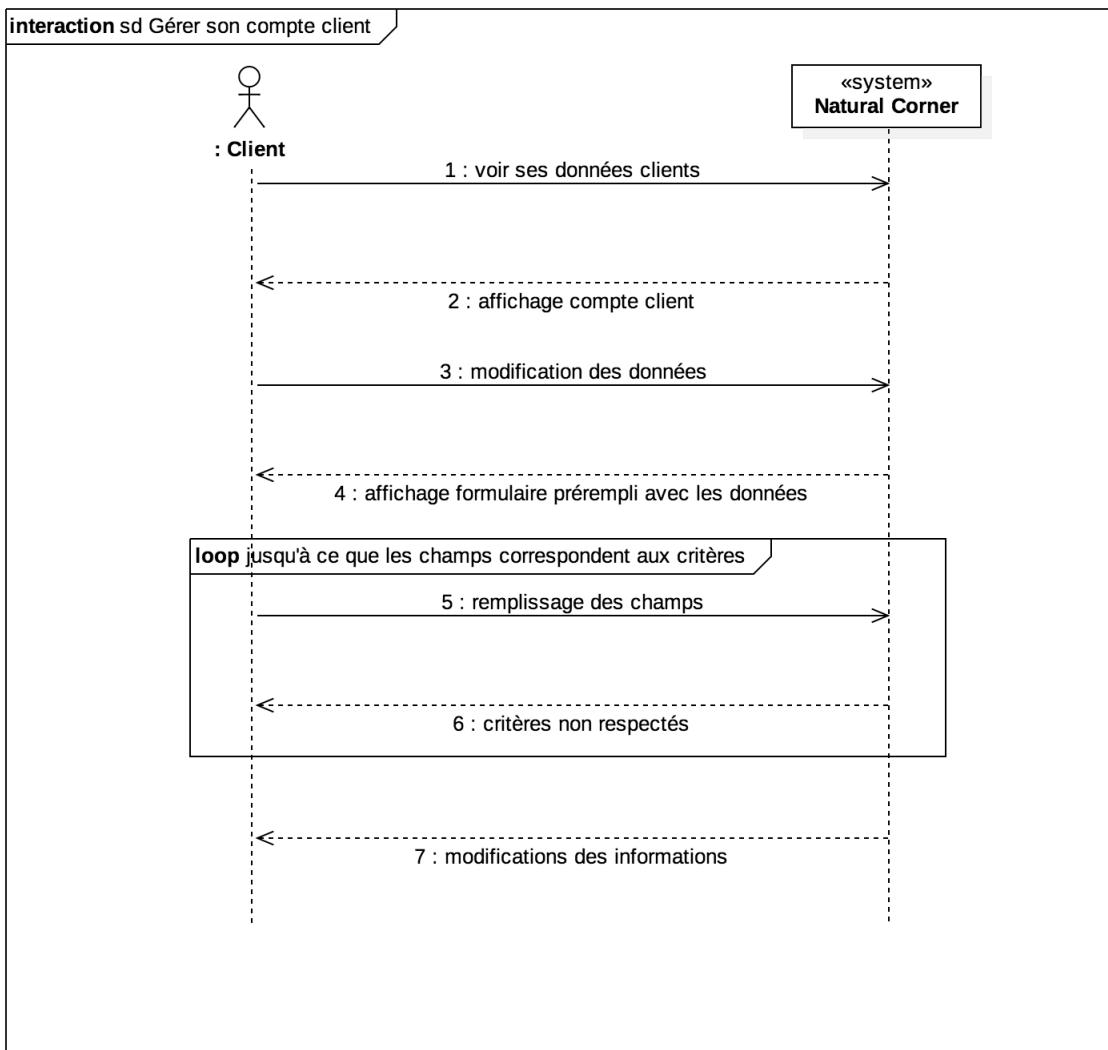


FIGURE 7.2.1 – Diagramme de séquence système "Gérer son compte client"

### 7.3 UC Recherche des produits

Ce UC est un fondamental de la commande en ligne. Que serait une app de vente sans la facilité de se renseigner sur un produit et de chercher à son aise.

#### 7.3.1 Description

Description du UC 'Recherche des produits'
--

<p><b>Acteur principal</b></p> <ul style="list-style-type: none"><li>— L'utilisateur (qu'il soit déjà client, ou simple visiteur)</li></ul> <p><b>Objectifs</b></p> <ul style="list-style-type: none"><li>— L'utilisateur veut trouver le plus rapidement possible un produit précis dans l'ensemble du catalogue. Il veut également pouvoir flâner comme il le ferait dans le magasin Natural Corner et chercher des produits avec des critères variés.</li></ul> <p><b>Préconditions</b></p> <ul style="list-style-type: none"><li>— Le catalogue est disponible (voir le cas d'utilisation Maintenir le catalogue).</li></ul> <p><b>Postconditions</b></p> <ul style="list-style-type: none"><li>— L'utilisateur a trouvé le produit précis qu'il cherchait, ou un produit qui l'intéresse, voire plusieurs.</li></ul>	<p><b>Scénario nominal</b></p> <ol style="list-style-type: none"><li>1. L'utilisateur lance une recherche à partir de mots-clés : un nom de produit, un rayon, un aliment de la composition, une catégorie de fruit ou de légume.</li><li>2. L'application affiche une page de résultat. Les produits sont classés par défaut par catégorie.</li><li>3. L'utilisateur sélectionne un produit.</li><li>4. L'application lui présente une description détaillée pour le produit sélectionné. On y trouvera en particulier :<ul style="list-style-type: none"><li>— une image (pour la majorité des produits).</li><li>— une description bio, la saison, la catégorie, le rayon.</li><li>— son prix et sa disponibilité,</li></ul></li></ol>
---	---

**Alternatives**

- 1a L'utilisateur n'a pas d'idée préconçue et préfère flâner dans les rayons du magasin bio virtuel. Pour cela, l'application lui propose un ensemble de pages telles que : nouveautés, meilleures ventes, recettes, promotions.
  1. L'utilisateur navigue dans ces pages et peut enchaîner sur l'étape 3 du scénario nominal.
- 2a L'application n'a pas trouvé de produits correspondant à la recherche.
  1. L'application signale l'échec à l'utilisateur et lui propose d'effectuer une nouvelle recherche. Le cas d'utilisation redémarre à l'étape 1 du scénario nominal.
- 2b L'application a trouvé de très nombreux produits.
  1. L'application signale le nombre de produits à l'utilisateur et lui affiche une première page de résultats. Les autres pages sont accessibles directement ou par des symboles Suivante et Précédente.
  2. L'utilisateur navigue dans ces pages et enchaîne éventuellement sur l'étape 3 du scénario nominal. Il peut également reclasser les produits obtenus par différents critères : une description bio, la saison, la catégorie, le rayon, etc
- 3a L'utilisateur n'est pas intéressé par les résultats.
  1. L'utilisateur revient à l'étape 1 du scénario nominal pour lancer une nouvelle recherche.
  2. L'utilisateur abandonne la recherche. Le cas d'utilisation se termine en échec.
- 1 L'utilisateur est intéressé par le résultat et met un produit dans le panier.
  1. L'application affiche le panier de l'utilisateur (voir le cas d'utilisation Gérer son panier).

**Exigences supplémentaires**

- La qualité visuelle doit permettre de bien voir les produits et rendre la recherche agréable.

### 7.3.2 Diagramme de séquence système

L'instruction "alt" signifie qu'une alternative aura lieu selon que l'utilisateur trouve un produit ou non. S'il le désire, il mettra le produit ainsi trouvé dans son panier ("opt").

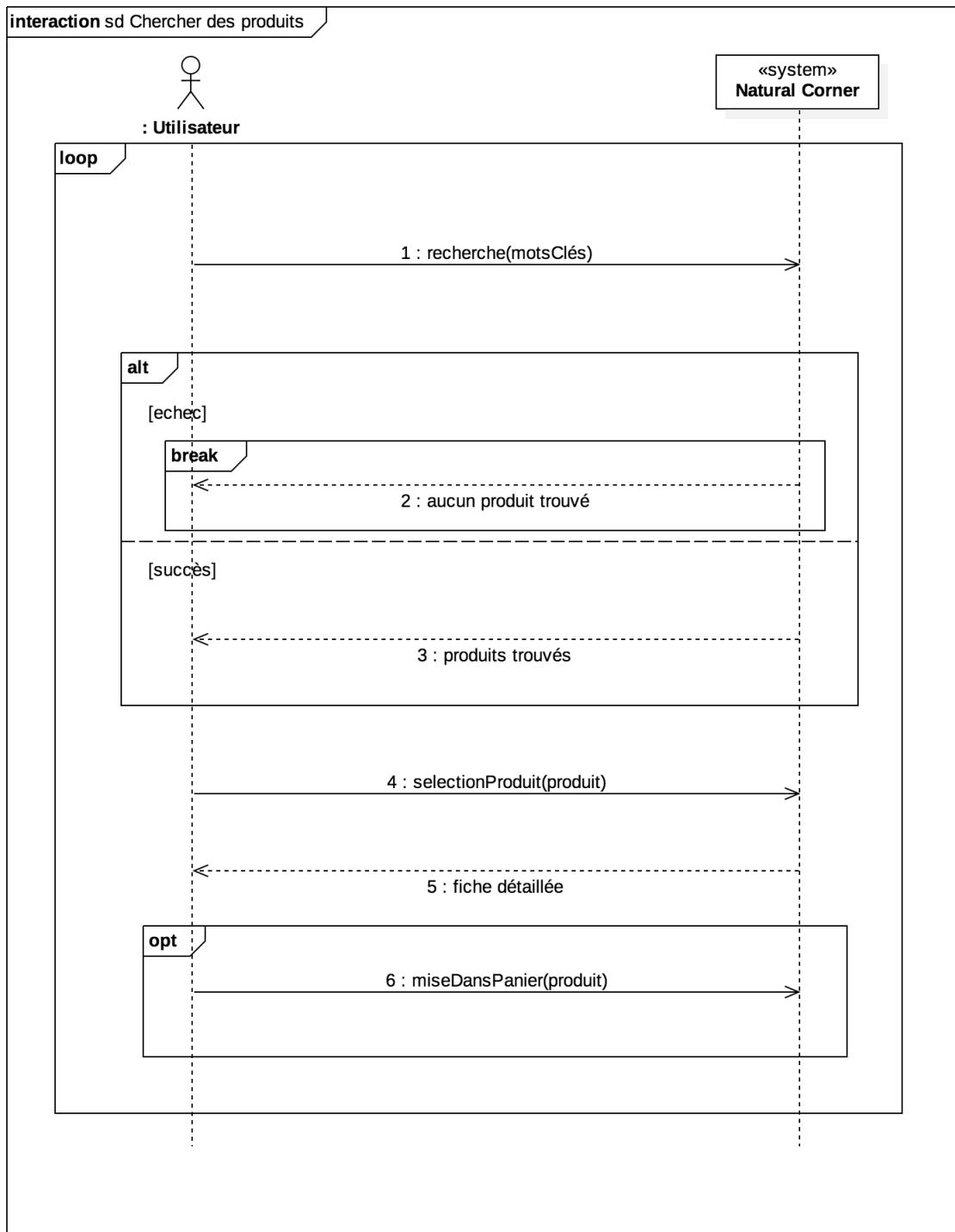


FIGURE 7.3.1 – Diagramme de séquence système "Recherche de produits"

## 7.4 UC Gérer son panier

### 7.4.1 Description

Description du UC 'Gérer son panier'
<p><b>Acteur principal</b></p> <ul style="list-style-type: none"> <li>— Le client.</li> </ul> <p><b>Objectifs</b></p> <ul style="list-style-type: none"> <li>— Lorsque le client est intéressé par un produit, il faut qu'il puisse l'enregistrer dans un panier virtuel. Ensuite, il doit pouvoir ajouter d'autres aliments, en supprimer ou encore en modifier les quantités avant de passer commande.</li> </ul> <p><b>Préconditions</b></p> <ul style="list-style-type: none"> <li>— Le client est déjà enregistré et loggué.</li> </ul> <p><b>Postconditions</b></p> <ul style="list-style-type: none"> <li>— Néant.</li> </ul>
<p><b>Scénario nominal</b></p> <ol style="list-style-type: none"> <li>1. Le client enregistre les produits qui l'intéressent dans un panier virtuel (voir le cas d'utilisation Chercher des produits).</li> <li>2. L'application lui affiche l'état de son panier. Chaque produit qui a été préalablement sélectionné est présenté sur une ligne, avec son titre, sa catégorie et son rayon. Son prix unitaire est affiché, la quantité est positionnée à '1' par défaut, et le prix total de la ligne est calculé. Le total général est calculé par l'application et affiché en bas du panier, avec le nombre d'articles.</li> <li>3. Le client continue ses achats (voir le cas d'utilisation Chercher des produits).</li> </ol>
<p><b>Alternatives</b></p> <ul style="list-style-type: none"> <li>— 2a Le panier est vide.</li> </ul> <ol style="list-style-type: none"> <li>1. Le l'application affiche un message d'erreur au client (« Votre panier est vide ») et lui propose de revenir à une recherche de produit (voir le cas d'utilisation Chercher des produits).</li> <li>— 4a Le client modifie les quantités des lignes du panier, ou en supprime.</li> </ol> <ol style="list-style-type: none"> <li>1. Le client revalide en demandant la mise à jour du panier.</li> <li>2. Le cas d'utilisation reprend à l'étape 2 du scénario nominal.</li> <li>— 1 Le client souhaite commander en ligne.</li> </ol> <ol style="list-style-type: none"> <li>1. L'application l'amène sur la page de validation de commande.</li> </ol> <p><b>Exigences supplémentaires</b></p> <ul style="list-style-type: none"> <li>— S'assurer que l'intégrité des données soit assurée (calculs, contenu du panier, sauvegarde de celui-ci).</li> </ul>

### 7.4.2 Diagramme de séquence système

On part du principe que l'utilisateur est déjà passé par le UC "Chercher des articles" ("ref").

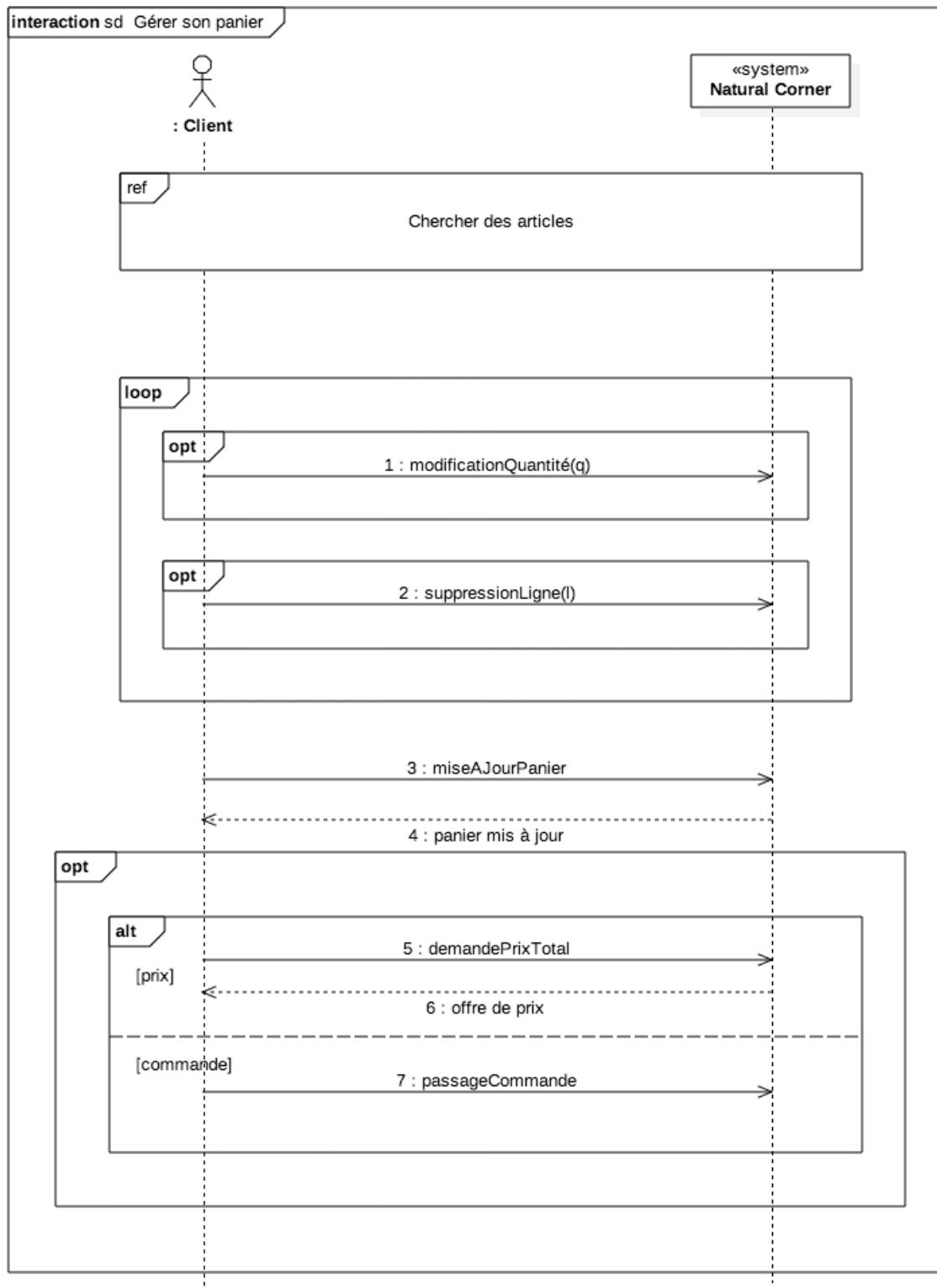


FIGURE 7.4.1 – Diagramme de séquence système "Gérer son panier"

## 7.5 UC Commander

### 7.5.1 Description

Description du UC 'Commander'
<p><b>Acteur principal</b></p> <ul style="list-style-type: none"> <li>— Le Client.</li> </ul> <p><b>Objectifs</b></p> <ul style="list-style-type: none"> <li>— À tout moment, le client doit pouvoir accéder à la page de commande, dans lequel il peut saisir ses coordonnées et les informations nécessaires au retrait en magasin.</li> </ul> <p><b>Préconditions</b></p> <ul style="list-style-type: none"> <li>— Le panier du client n'est pas vide.</li> </ul> <p><b>Postconditions</b></p> <ul style="list-style-type: none"> <li>— Une commande a été enregistrée et transmise à Natural Corner.</li> </ul>
<p><b>Scénario nominal</b></p> <ol style="list-style-type: none"> <li>1. Le Client saisit l'ensemble des informations nécessaires à la garantie de venir chercher son panier, à savoir : les coordonnées de l'adresse du client (nom, prénom, adresse postale complète, téléphone).</li> <li>2. L'application affiche un récapitulatif des adresses indiquées et du panier à venir chercher.</li> <li>3. L'application confirme la prise de commande au client.</li> <li>4. L'application envoie la commande validée aux vendeurs de Natural Corner.</li> <li>5. L'application confirme la commande au client.</li> </ol>
<p><b>Alternatives</b></p> <ul style="list-style-type: none"> <li>— 1-3a Le Client annule sa commande.</li> <li>1. L'application revient sur l'affichage du panier et le cas d'utilisation se termine en échec.</li> </ul>

### 7.5.2 Diagramme de séquence système

Une précondition a été ajoutée, le panier doit être rempli par au moins un article pour être commandé. Le vendeur intervient comme utilisateur secondaire.

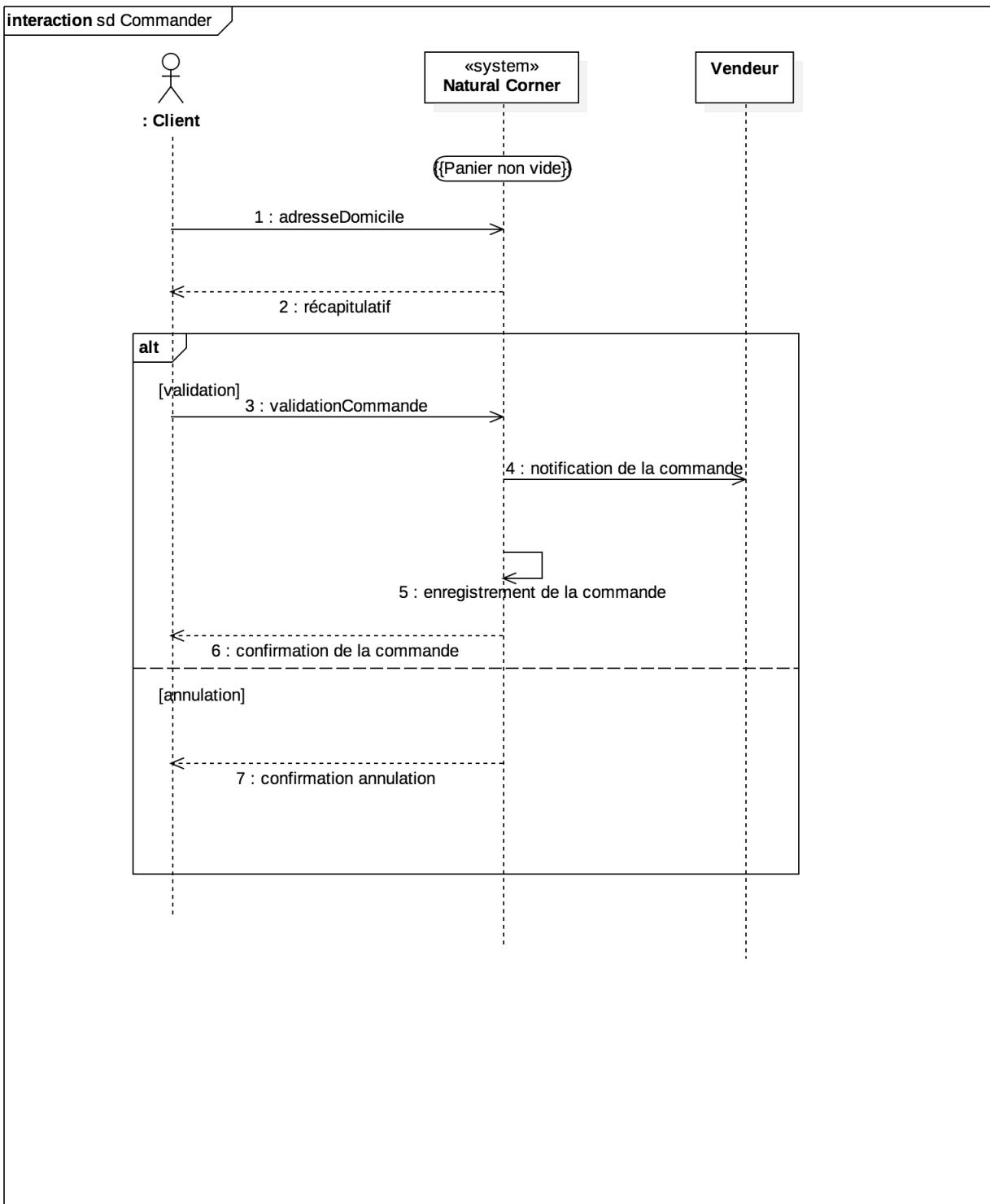


FIGURE 7.5.1 – Diagramme de séquence système "Commander"

## 7.6 UC Maintenir le catalogue

### 7.6.1 Description

Description UC Maintenir le catalogue
<p><b>Acteur principal</b></p> <ul style="list-style-type: none"><li>— Le vendeur.</li></ul>
<p><b>Acteur secondaire</b></p> <ul style="list-style-type: none"><li>— Le catalogue.</li></ul>
<p><b>Objectifs</b></p> <ul style="list-style-type: none"><li>— Le vendeur veut mettre à jour le catalogue avec de nouveaux produits et certaines informations sur les produits.</li></ul>
<p><b>Préconditions</b></p> <ul style="list-style-type: none"><li>— Le vendeur s'est authentifié.</li><li>— Le catalogue est accessible.</li></ul>
<p><b>Postconditions</b></p> <ul style="list-style-type: none"><li>— Une nouvelle version du catalogue est disponible.</li></ul>
<p><b>Scénario nominal</b></p> <ol style="list-style-type: none"><li>1. Le vendeur introduit un nouveau produit dans le catalogue.</li></ol>
<p><b>Alternatives</b></p> <ul style="list-style-type: none"><li>— 1a Le vendeur met à jour un produit.</li></ul>

### 7.6.2 Diagramme de séquence système

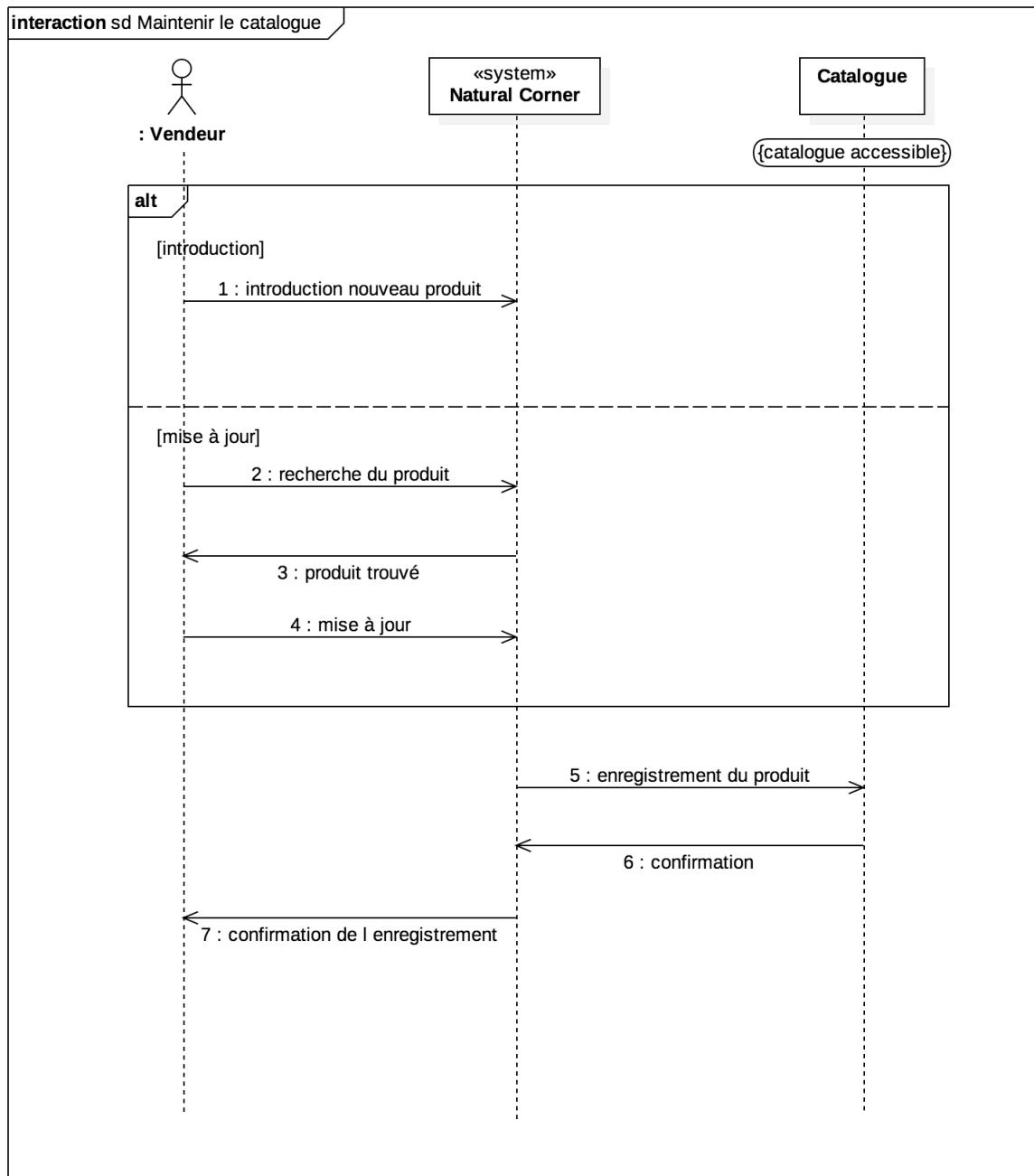


FIGURE 7.6.1 – Diagramme de séquence système "Maintenir catalogue"

## **Quatrième partie**

### **Classes d'analyse et base de données**

## Chapitre 8

# Conception objet des classes d'analyse

## 8.1 Classes d'analyse

### 8.1.1 Identification des concepts du domaine

Quels sont les concepts métiers qui correspondent à ces cas d'utilisation ? Pour le cas d'utilisation "Créer son compte", il va de soi que le concept "Utilisateur" est pertinent. Il faut de plus prendre en compte l'existence des différents utilisateurs du système "Natural Corner", c'est à dire les vendeurs, le gérant, l'internaute simple visiteur, l'internaute client identifié. L'analyse des Uses Cases nous a permis de différencier le système Natural Corner depuis le point de vue d'un client (potentiel ou actuel) et du point de vue du gérant et des vendeurs. Un choix d'analyse consiste à créer une classe "Utilisateur" qui concerne un utilisateur identifié, qu'il soit client ou non. Il ne s'agit pas ici de fonctionner à la manière d'un site web d'achat traditionnel mais plutôt d'inciter le client à utiliser l'application calquée sur les "Apps" pour smartphone. Les utilisateurs sont désormais habitués à s'identifier d'emblée avant la première utilisation d'une application. Un utilisateur sera donc un concept dont pourront dériver les concepts de vendeur et gérant car ils vont partager des attributs similaires.

Un autre concept intéressant est celui d'adresse qui permet de factoriser le concept d'utilisateur, ce concept possédant plusieurs caractéristiques aisément identifiables en attributs. Ne perdons pas de vue par ailleurs que le gérant du magasin Natural Corner a clairement demandé lors de l'analyse des besoins une identification par le réseau social Facebook. Voici les concepts métiers qui se dégagent :

- utilisateur,
- vendeur,
- gérant,
- adresse

Le cas d'utilisation "Recherche produits" suggère le concept d'article. Il faut de plus pouvoir ranger un article dans un rayon comme l'indique le fichier transmis par le gérant du magasin et, enfin, il est agréable pour un utilisateur de l'application de chercher ses articles favoris par catégorie. Voici les concepts métiers identifiés :

- article,
- rayon,
- catégorie

Nous apprenons du UC "Gérer panier" qu'un candidat à l'univers conceptuel de l'application Natural Corner est celui de panier. Il s'agit du panier qu'utilisera un client dans un magasin d'alimentation. Ce panier est rempli d'articles. Pour donner de l'ordre à ce panier et pouvoir différencier les articles, connaître leurs nombres et en calculer le prix total, il est intéressant d'utiliser le concept de ligne de panier. Ainsi,

- panier,
- ligne panier

De l'UC "Commander", le concept de commande apparaît presque immédiatement :

- commande

### 8.1.2 Ajout des associations et des attributs

Un utilisateur peut se caractériser par les attributs prénom, nom, pseudonyme, mot de passe, adresse e-mail. On peut lui ajouter une date d'inscription pour avoir des informations supplémentaires et aussi son adresse IP à des fins statistiques (pays à partir duquel l'utilisateur se connecte à l'application par exemple).

Une adresse contient une rue (place, chaussée, avenue, etc...), une numéro, une code postal, une localité et un pays (par défaut Belgique). Un utilisateur n'utilise qu'une seule adresse et même si une même adresse peut être utilisée par plusieurs personnes (familles, colocataires), on limitera l'univers conceptuel à un utilisateur par adresse et une adresse par utilisateur . La classe Adresse est une factorisation clarifiant la classe Utilisateur et qui servira le cas échéant à d'autres classes à venir.

Nous résumons ces réflexion par une relation de composition entre les classes Utilisateur et Adresse. Un vendeur est un utilisateur qui a certains droits d'accès supplémentaires du point de vue du catalogue des produits et du système Natural Corner. Un gérant est un vendeur qui peut éditer des commentaires informatifs et notifier les clients des nouvelles promos. Voici un premier jet des classes d'analyse :

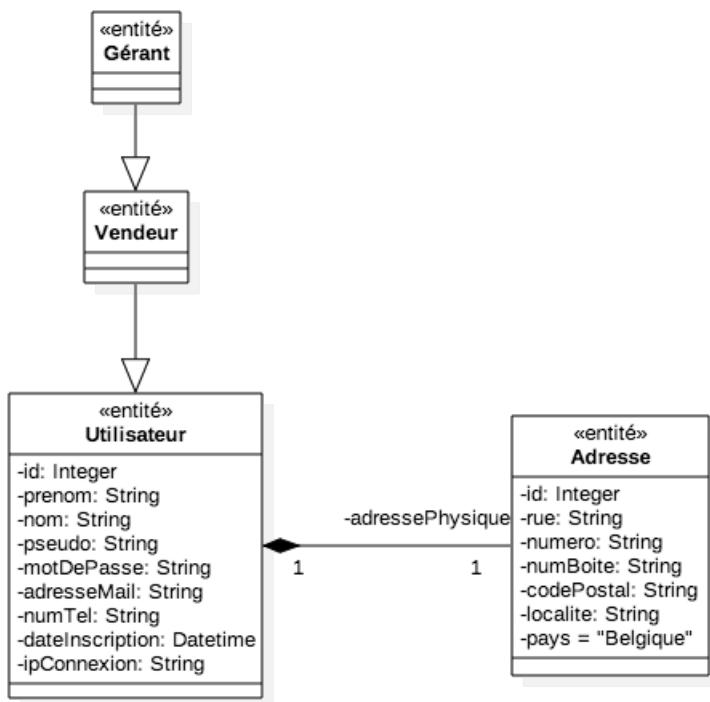


FIGURE 8.1.1 – Classes d'analyse des UC "Créer son compte client" et "Gérer son compte client"

Le choix du type des classes obéit à une logique de facilité d'implémentation et de flexibilité. Un numéro de maison peut être "6 a", ce qu'un type integer ne peut contenir. Un code postal peut commencer par zéro en France (exemple : 01330, Amberieu-en-dombes, Ain, Rhône-Alpes), de même pour un numéro de téléphone. Initialiser à "Belgique" par défaut à la variable "pays" pour faciliter l'encodage. Un horodatage est plus précis qu'une date pour la date d'inscription.

Le schématisation UML2 permet de représenter directement un attribut de la classe Utilisateur à l'aide d'une relation avec la classe Adresse. Une adresse contient aussi un numéro de boîte. Les classes Vendeur et Gérant sont des classes vides à ce niveau d'analyse mais pourront être plus sophistiquées dans le futur. Le concept d'héritage permet dans ce cas ci d'enrichir les classes Vendeur et Gérant en fonction de l'analyse des besoins ultérieure sans devoir toucher à la classe Utilisateur. La navigabilité va dans le sens de la classe Utilisateur vers la classe Adresse. La classe Adresse n'a pas à avoir accès aux données de la classe Utilisateur.

Pour trouver les entités intéressantes, je me suis inspiré d'un document remis par le gérant du magasin. On peut y voir le mot Porte 1 qui correspond au rayon. Le numéro de de cinq chiffres n'a pas été gardé pour l'application car

elle ne semble pas intéressante pour la logique métier. La catégorie a par contre été ajoutée pour l'application afin de faciliter la recherche du client dans le catalogue.

	Firma:BIOF Afdeling:SKW	Tariefkode:TTO	sam- jeud	qtté min
1			qtté min	ven w-e
2	Porte 1			
3	23234 BIOMOMO GINGEMBRE CONFIT		6	
4	54850 Sojade nature grand 400g (com par 6)	6 STK	18	30
5	54851 Sojade abricot grand 400g (com par 6)	6 STK	6	6
6	54857 sojade ananas 400 gr	6 STK	6	6
7	54854 Sojade myrtille grand 400gr (com par 6)	6 STK	6	6
8	54847 Sojade Cranberry 400 gr		SUR COM	SUR COM
9	54855 Sojade banane grand 400gr (com par 6)	6 STK	6	6
10				

FIGURE 8.1.2 – Echantillon de la liste des produits frais

Pour l'UC "Rechercher un produit", un article a une dénomination, un prix unitaire, peut être en promo ou indisponible et possède un commentaire écrit par le gérant du magasin. Un rayon a une dénomination correspondant aux portes ou l'emplacement, comme l'on peut le voir sur le fichier cvs fournit par le gérant du magasin "Natural Corner". La catégorie a une dénomination correspondant aux catégories habituelles dans le secteur agro-alimentaire pour les fruits et légumes et autres produits frais.

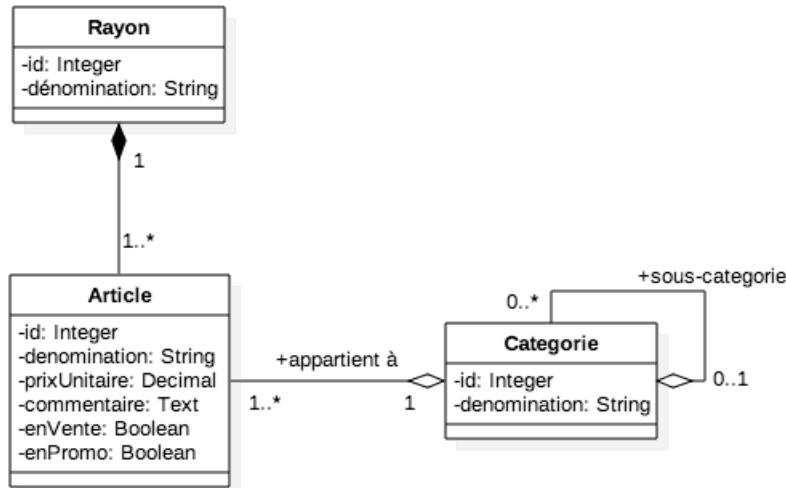


FIGURE 8.1.3 – Classes d'analyse de l'UC "Chercher produit"

Il a été décidé après une lecture de différents forums en ligne que la variable prix unitaire sera de type Integer lors de l'implémentation<sup>1</sup>. L'argument semble convaincant. En effet, un float n'est jamais qu'une approximation et les tests de type "if" sont biaisés avec ce type. Utiliser le type Integer qu'on multiplie par cent par la suite (centimes vers Euros), est une manière de ne pas perdre l'intégrité de l'information du système.

Un article peut appartenir à plusieurs catégories selon le point de vue d'un magasin bio (le concombre est un légume mais aussi un produit frais, un légume vert, un légume prévu pour des shakes santé et même un cosmétique). Une catégorie contient parfois des sous-catégorie (produits frais > légumes de saison > légumes verts > ...). Nous introduisons cette idée à l'aide d'une association réflexive nommée "sous-catégorie". Les rayons étant séparés spatialement, nous pouvons établir une relation de composition entre Article et Rayon. Un rayon possède plusieurs articles, un article n'est rangé que dans un seul rayon.

1. <https://stackoverflow.com/questions/900677/how-do-i-safely-perform-money-related-calculations-in-php>

Un panier a un certain nombre d'article et une valeur totale. Ces deux attributs sont en réalité dérivés car ils se calculent à partir des lignes de panier. Il en va de même pour l'attribut montant de la classe LignePanier, le montant d'une ligne est le produit du nombre d'articles de la ligne et du prix unitaire de cet article.

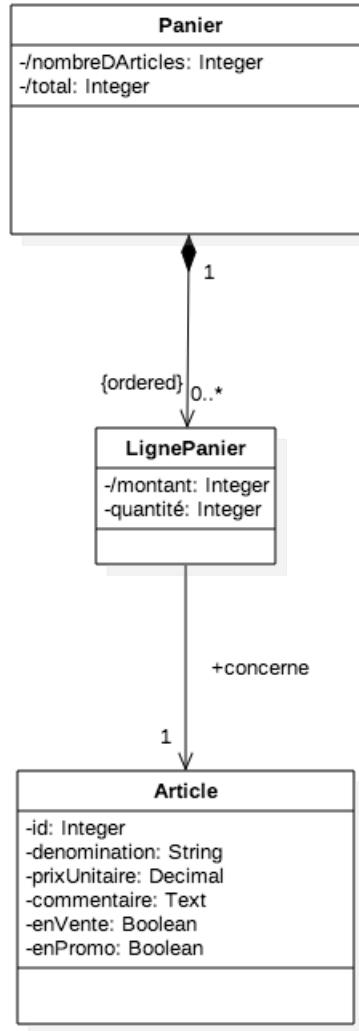


FIGURE 8.1.4 – Classes d'analyse de l'UC "Gérer panier"

La navigabilité rend étanche le sens de navigation d'article vers lignePanier. En effet, la conception objet nous suggère de limiter le plus possible la connaissance des classes entre elles. "Ordered" car il s'agit d'une liste numérotée de lignes de Panier. Une relation de composition avec Panier au lieu d'une agrégation puisque un Panier décide de construire une ligne de panier et aussi de sa destruction.

La classe commande possède un horodatage correspondant au moment de la commande, un délais de collecte pour le client et un montant total dérivé et calculé à partir du panier.

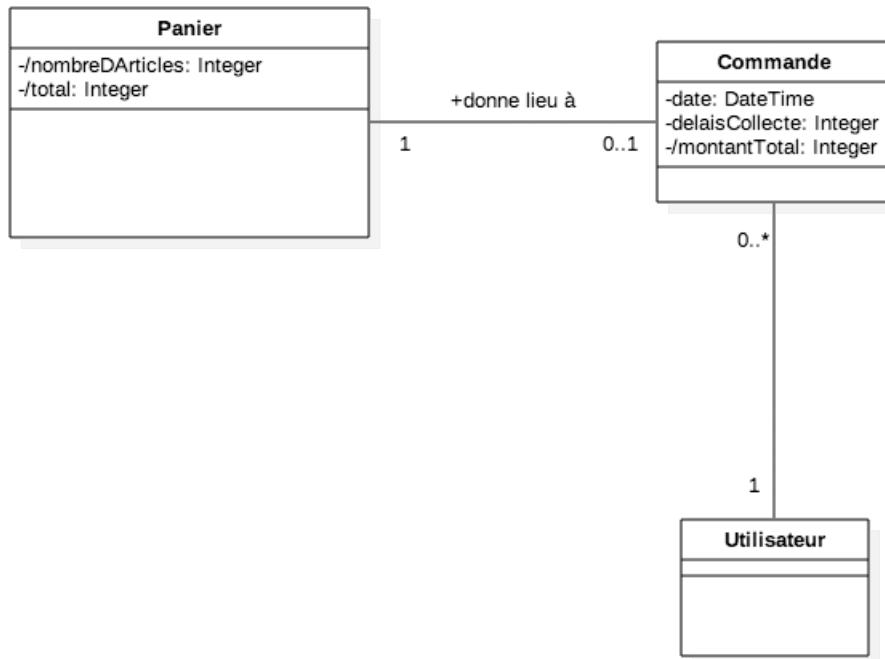


FIGURE 8.1.5 – Classes d'analyse de l'UC "Commander"

# Chapitre 9

## Base de données

### 9.1 Conception

Tout l'enjeu de cette base de données est de donner des tables permettant de tester le backend et la persistance des données en respectant le modèle objet des classes d'analyse. En effet, "la correspondance des données entre le modèle relationnel et le modèle objet doit faire face à plusieurs problèmes : le modèle objet propose plus de fonctionnalités : héritage, polymorphisme, ... les relations entre les entités des deux modèles sont différentes un objet ne possède pas d'identifiant en standard (hormis son adresse mémoire qui varie d'une exécution à l'autre). Dans le modèle relationnel, chaque occurrence devrait posséder un identifiant unique"<sup>1</sup>. Comme nous nous passons d'une couche logicielle permettant d'effectuer le pont entre le modèle objet de l'application et le modèle relationnel de la base de données (ORM), c'est en amont du code, au moment de la conception de la base de données, qu'il faut y penser de manière adéquate pour éviter des problèmes d'implémentation par la suite<sup>2</sup>.

Reprendons : un objet est instancié par son constructeur et, à ce moment, un premier choix d'implémentation consiste à fournir à l'attribut `_id` la valeur d'un attribut de classe `static_id` automatiquement incrémenté. Chaque nouvelle instantiation de la classe se verra attribuer un nouvel identifiant. Au moment de l'insertion de l'objet dans la base de données, on introduit l'`id` en lieu et place du champ ID. Cependant le choix conceptuel du modèle relationnel impose l'utilisation d'une clé auto-incrémentée. Etant donné qu'il n'y a pas de translation directe entre cette auto-incrémantation garantissant l'unicité des lignes de la table et l'identifiant arbitraire des objets de l'application, l'insertion rate.

Il faut donc trouver une solution convenable pour garantir la liaison objet-relationnelle. En PHP, les objets sont passés par référence en paramètre aux méthodes. La méthode se chargeant d'ajouter un objet dans la base de données utilisera son mutateur `setId($id)`. L'objet, après avoir été enregistré dans la base, sera utilisé par l'application avec le même `id` que celui de la base de données.

La conception de la base de données suit de près la conception objet de l'application Natural Corner. Voici la description détaillée de la modélisation :

- Un utilisateur passe une commande
  - Deux entités ("utilisateurs" et "commandes") reliées par une association ("passer").
  - Un utilisateur peut ne passer aucune commande (Cardinalité 0).
  - Un utilisateur peut passer plusieurs commandes (Cardinalité n).
  - Une commande est passée par un et un seul utilisateur (Cardinalité min 1 et max 1).
- Un panier donne lieu à une commande
  - Deux entités ("paniers" et "commandes") reliées par une association ("donner lieu à").
  - Un panier peut donner lieu à aucune commande (Cardinalité 0).
  - Un panier peut donner lieu au plus à une commande (Cardinalité 1).
  - Une commande a été donné lieu par un et un seul panier (Cardinalité min 1 et max 1).
- Un panier contient des lignes de panier.

1. <http://www.jmdoudoux.fr/java/dej/chap-persistence.htm>

2. Gruau C., Conception d'une base de données, site developpez.com 1ère édition corrigée, 13/7/2006, p. 41

- Deux entités ("paniers" et "lignes panier") reliées par une association ("contenir").
- Un panier peut ne pas contenir de ligne de panier (Cardinalité 0).
- Un panier peut contenir plusieurs lignes de panier (Cardinalité n).
- Une ligne de panier est contenu par un et un seul panier (Cardinalité min 1 et max 1).
- Un article concerne une ligne de panier.
  - Deux entités ("lignes panier" et "articles") reliées par une association ("concerner").
  - Un article peut ne concerner aucune ligne de panier (Cardinalité 0).
  - Un article peut être concerné par plusieurs lignes de panier (Cardinalité n).
  - Une ligne de panier est concerné par un et un seul article (Cardinalité min 1 et max 1).
- Un article appartient à un rayon
  - Deux entités ("rayons" et "articles") reliées par une association ("appartenir à").
  - Un article appartient à un et un seul rayon (Cardinalité min 1 et max 1).
  - Un rayon peut n'avoir aucun article (Cardinalité 0).
  - Un rayon contient plusieurs articles (Cardinalité n).
- Un article appartient à une catégorie
  - Deux entités ("catégorie" et "articles") reliées par une association ("appartenir à").
  - Un article appartient au moins à une catégorie (Cardinalité 1).
  - Un article appartient peut appartenir à plusieurs catégories (Cardinalité n).
  - Une catégorie peut ne s'appliquer à aucun article (Cardinalité 0).
  - Une catégorie peut s'appliquer à plusieurs articles (Cardinalité n).
- Une catégorie catégorise une autre catégorie.
  - Une entité ("catégorie") reliée par une association ("catégorise").
  - Une catégorie peut ne pas appartenir à une autre catégorie (Cardinalité 0).
  - Une catégorie peut appartenir à plusieurs autres catégories (Cardinalité n).
  - Une catégorie peut ne catégoriser aucune catégorie (Cardinalité 0).
  - Une catégorie peut catégoriser plusieurs autres catégories (Cardinalité n).
- Un vendeur est un utilisateur.
  - Deux entités ("utilisateur" et "vendeurs") reliés par une relation d'héritage.
- Un gérant est un vendeur.
  - Deux entités ("gérant" et "vendeurs") reliés par une relation d'héritage.

## 9.2 Modèle conceptuel de données (MCD)

Une traduction conceptuelle comme fruit de ces réflexions.

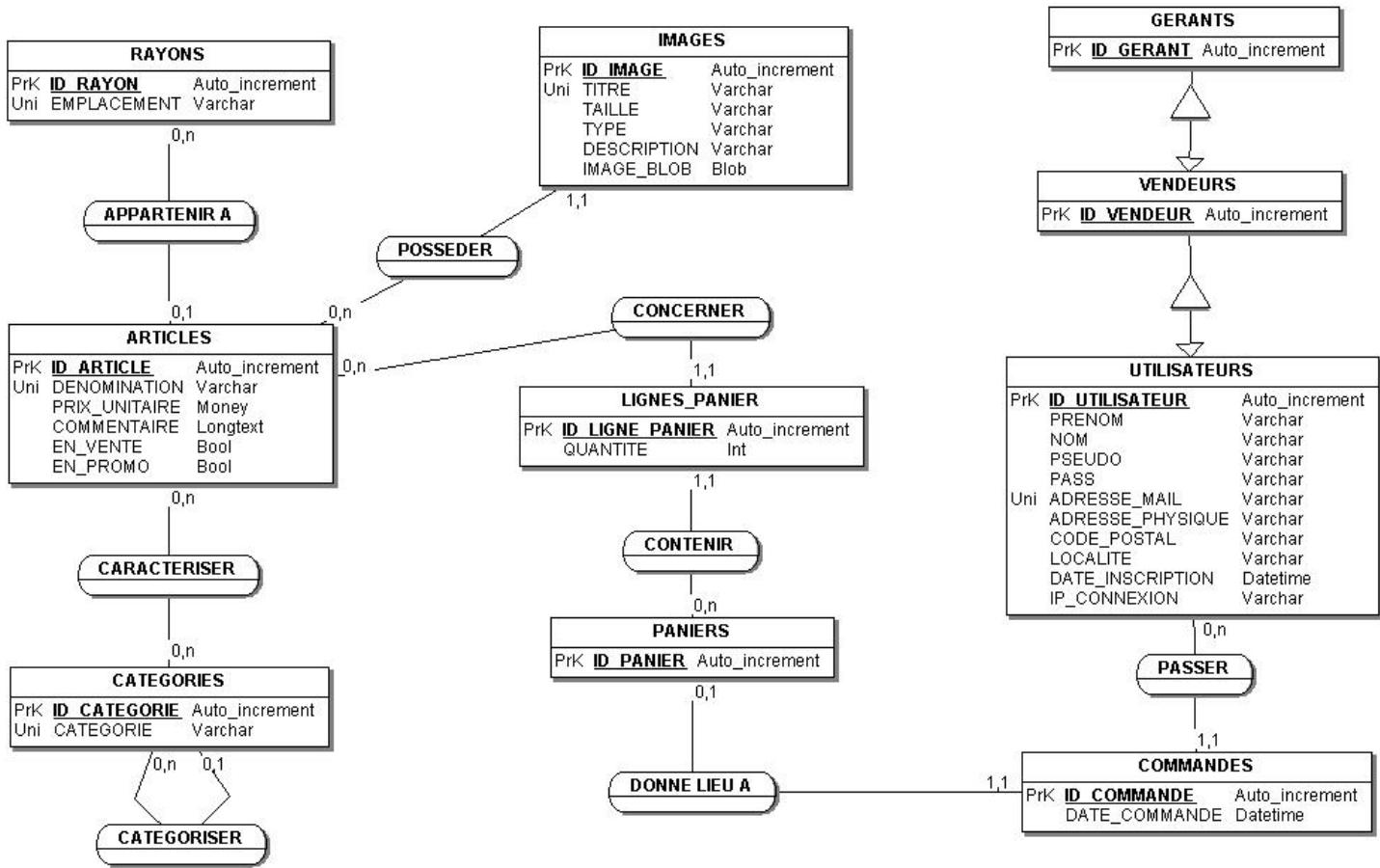


FIGURE 9.2.1 – MCD de la base de données

### 9.3 Modèle logique de données (MLD)

La traduction a été effectuée en partie par le logiciel JMerise<sup>3</sup>.

3. <http://www.jfreesoft.com/JMerise/>

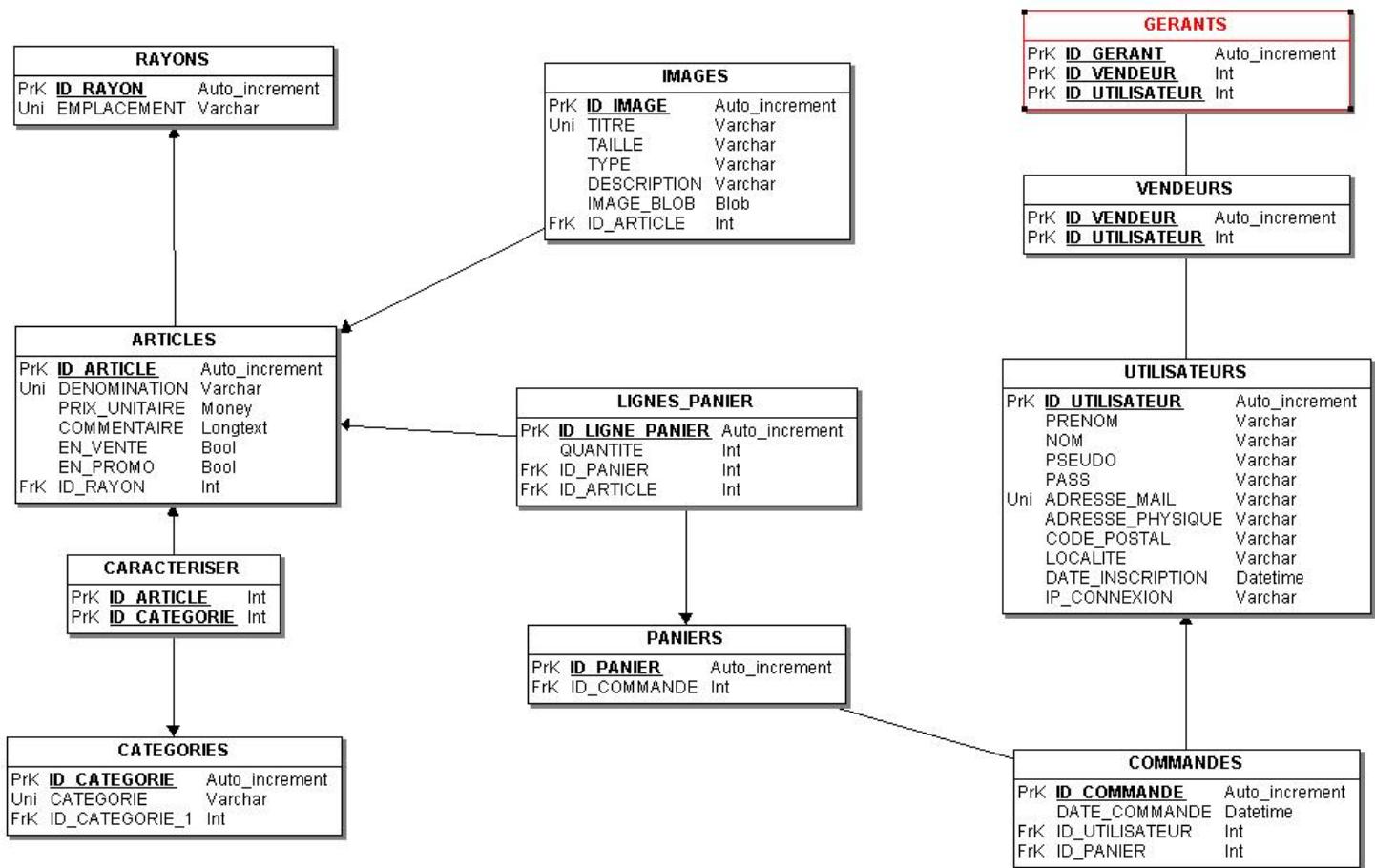


FIGURE 9.3.1 – MLD de la base de données

#### 9.4 Réflexions sur la conception de la base de données

Il n'a pas été aisé de réaliser la conception. Il fallait notamment que la base de données corresponde à la conception orientée objet. D'autre part, la réalisation de l'application étant incrémentale et se perfectionne au fur et à mesure des itérations, il a fallu remanier à plusieurs reprises la base de données. Par exemple, il n'y avait pas de classe Adresse au début de la conception et l'application a d'abord été écrite en n'utilisant qu'un seul champ pour la rue ou le numéro de boîte. Après analyse et perfectionnement, il a semblé plus judicieux de créer une classe Adresse réutilisable. La difficulté provient du fait qu'une mauvaise cardinalité va influencer l'implémentation. Est-ce qu'une adresse doit d'abord être créée avant un utilisateur ou l'inverse ? Après avoir relu ma documentation sur la conception d'une base de données, je me suis aperçu que je n'avais pas normalisé les relations<sup>4</sup>. En fait, le modèle objet s'accorde très bien de l'existence de deux classes mais pas le modèle relationnel. J'ai donc repensé la base de données et ajouté les champs de la table Adresses à la table Utilisateurs.

4. Gruau C., Conception d'une base de données, Site developpez.com 1ère édition corrigée, 13/7/2006, p.12

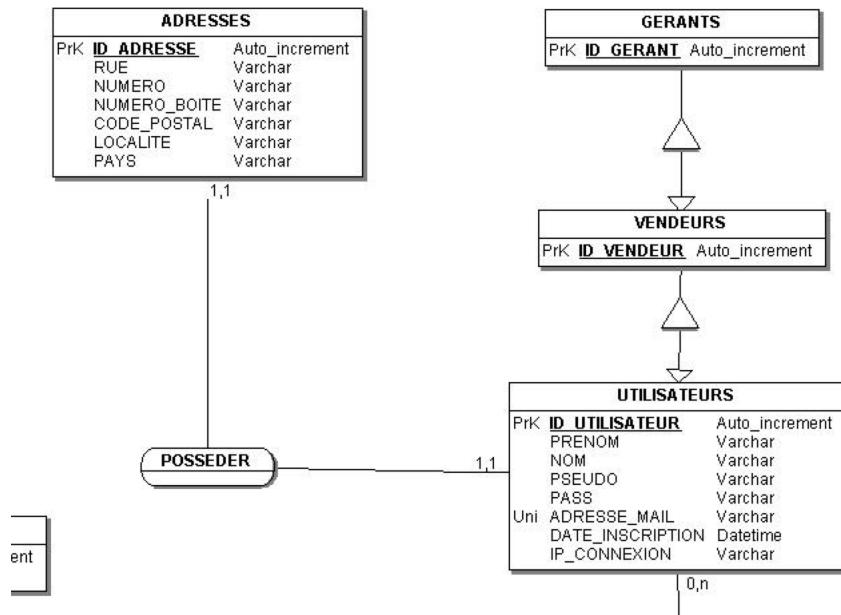


FIGURE 9.4.1 – Erreur de conception

Le logiciel utilisé JMerise<sup>5</sup> est très bien pour créer des diagrammes Merise mais donne des résultats erronés lorsqu'on utilise la traduction automatique vers le schéma relationnel. De fait, lors de la traduction du schéma entité-relation vers le schéma relationnel, il traduit les relations 1 :1 par une intégrité référentielle sur les clés primaires des deux côtés de la relation, rendant le modèle logique inutilisable pour l'application. Comment mettre à jour un article s'il faut d'abord créer un rayon qui ne se met à jour que si un article est créé ? J'ai donc dû réaliser "à la main" la traduction du schéma entité-relation en suivant la méthode classique<sup>6</sup>.

5. <http://www.jfreesoft.com/JMerise/>

6. Gruau C., Conception d'une base de données, Site developpez.com 1ère édition corrigée, 13/7/2006, p.24

**Cinquième partie**

**Réalisation des cas d'utilisation**

## Chapitre 10

# Conception et implémentation des UC

### 10.1 UC "Créer son compte client" et "Gérer son compte client"

#### 10.1.1 Diagramme des classes participantes (DCP)

##### Réflexions sur le UC "Créer son compte client"

Un utilisateur a besoin de s'inscrire pour utiliser l'application. Il est obligé de fournir une adresse e-mail et un mot de passe (qu'il confirme) pour avoir accès à l'application "Natural Corner". Le dialogue correspondant sera un formulaire reprenant les champs utiles pour un compte utilisateur. Le contrôleur a la responsabilité de vérifier les données fournies, du point de vue de la syntaxe et du type, mais aussi du point de vue de la sécurité afin de se prémunir des failles de type injection de code javascript , entre autres. La méthode "verifierInput()" aura cette responsabilité. Une méthode "envoyerFormulaire()" se charge de récolter les informations et de les fournir au contrôle "CtrlCompte". Un diagramme de communication permet d'illustrer cette question.

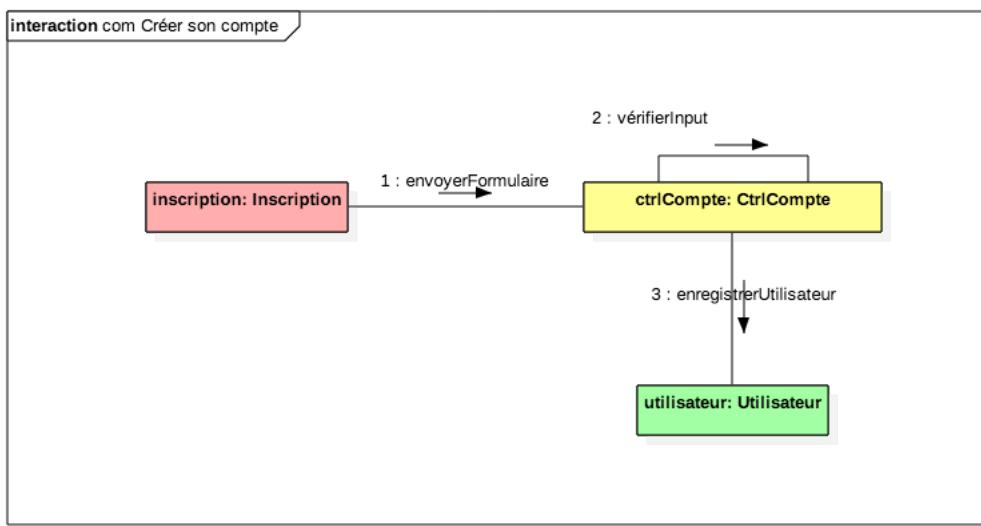


FIGURE 10.1.1 – Diagramme de communication "Créer son compte client"

##### Réflexions sur le UC "Gérer son compte client"

L'utilisateur qui a un compte client peut à tout moment modifier ce compte. Pour ce faire, nous allons créer une vue MiseAJourUtilisateur qui sera un formulaire de mise à jour des informations. Il pourra compléter ses informations personnelles et aussi changer son mot de passe. La logique étant sensiblement la même que pour le UC "Créer un compte client", nous nous bornerons à changer la classe de dialogue Incription en la classe MiseAJourUtilisateur. La méthode miseAJourUtilisateur() est ajoutée à la classe CtrlCompte.

### Diagramme des classes participantes

Voici le diagramme des classes participantes réalisant le use case "Créer son compte client" suivi de celui de "Gérer son compte client".

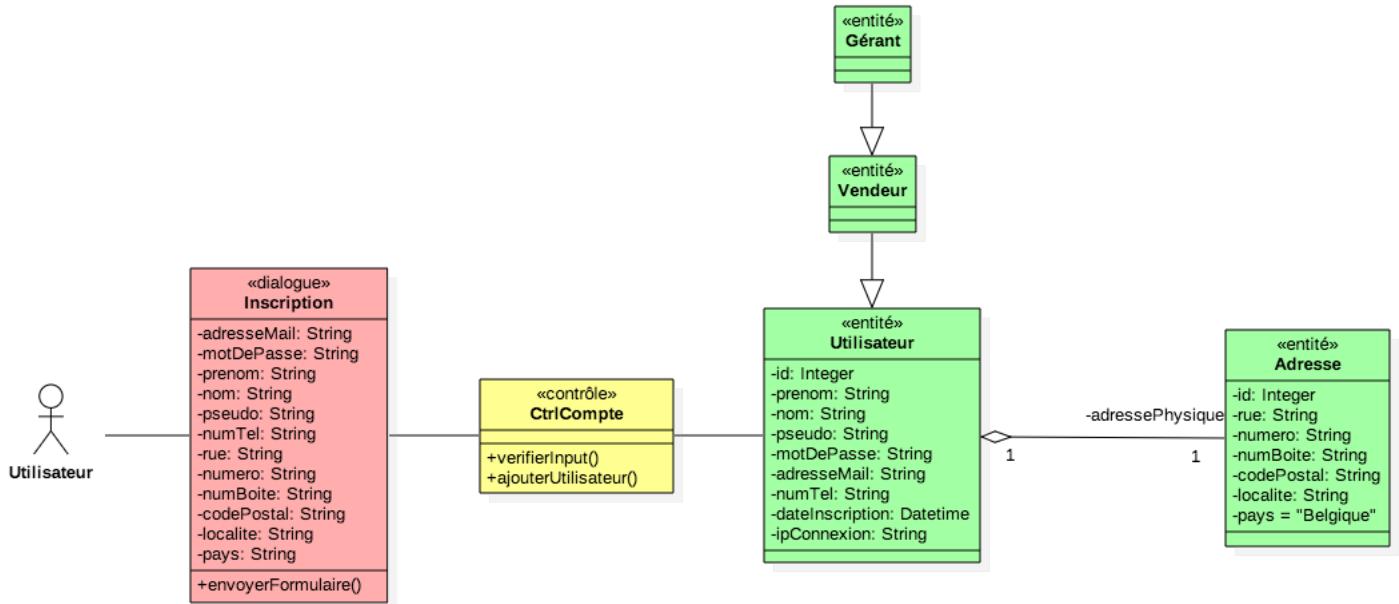


FIGURE 10.1.2 – Crée son compte client

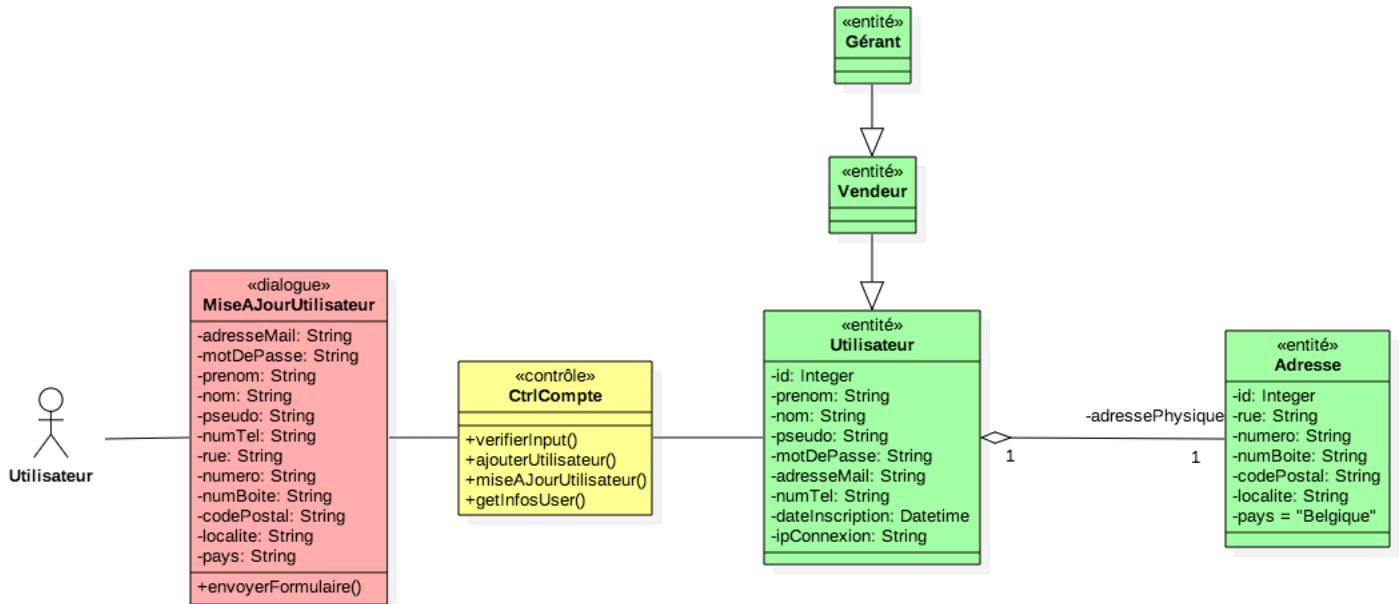


FIGURE 10.1.3 – Gérer son compte client

### 10.1.2 Diagramme de navigation<sup>1</sup>

Un utilisateur accède à l'application et a ensuite le choix entre se logguer avec ou sans Facebook ou créer un compte. Il ne peut pas entrer dans l'application sans s'être identifié. Il peut créer un compte s'il le désire. Il est envoyé vers la page de l'inscription. Il remplit les champs jusqu'à ce qu'il se conforme aux règles de validation. Il voit un message « compte créé » et est redirigé vers la page principale de l'application.

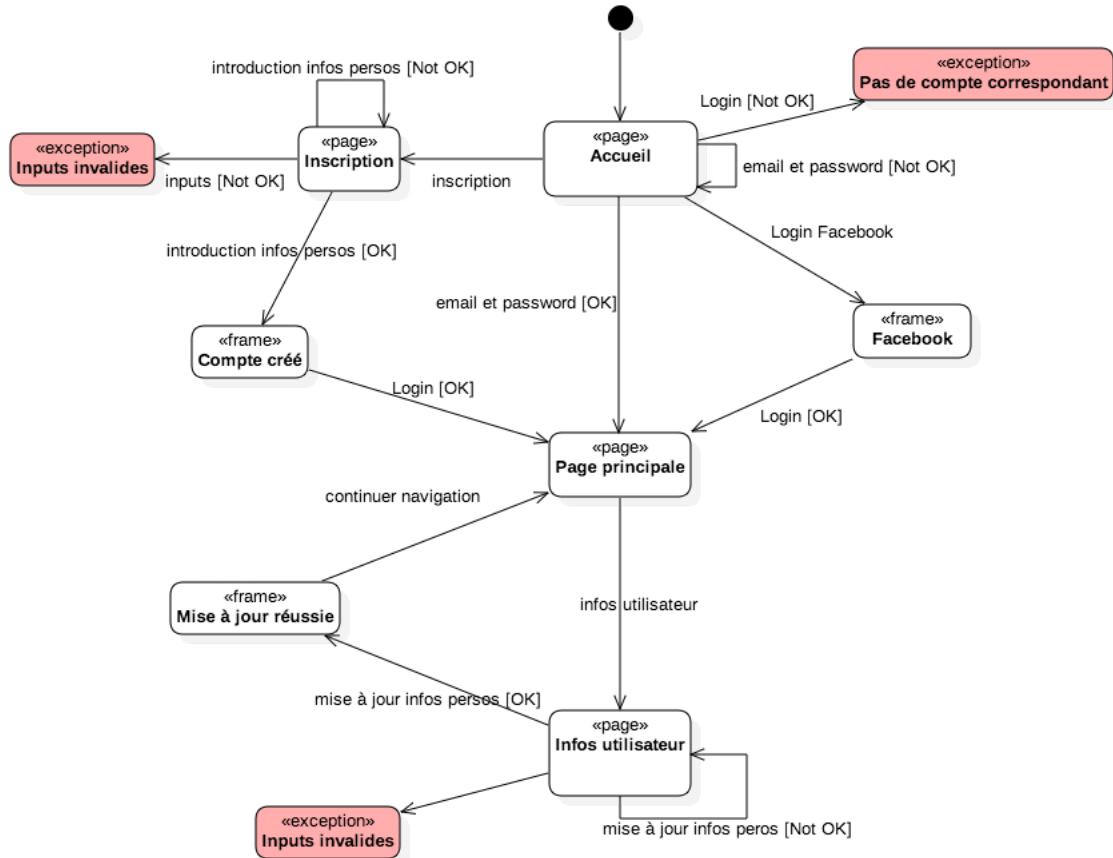


FIGURE 10.1.4 – Diagramme de navigation pour "Créer son compte" et "Gérer son compte"

1. Voir Annexe B

### 10.1.3 Conception objet préliminaire

Diagrammes d'interaction pour le UC "Créer son compte client"

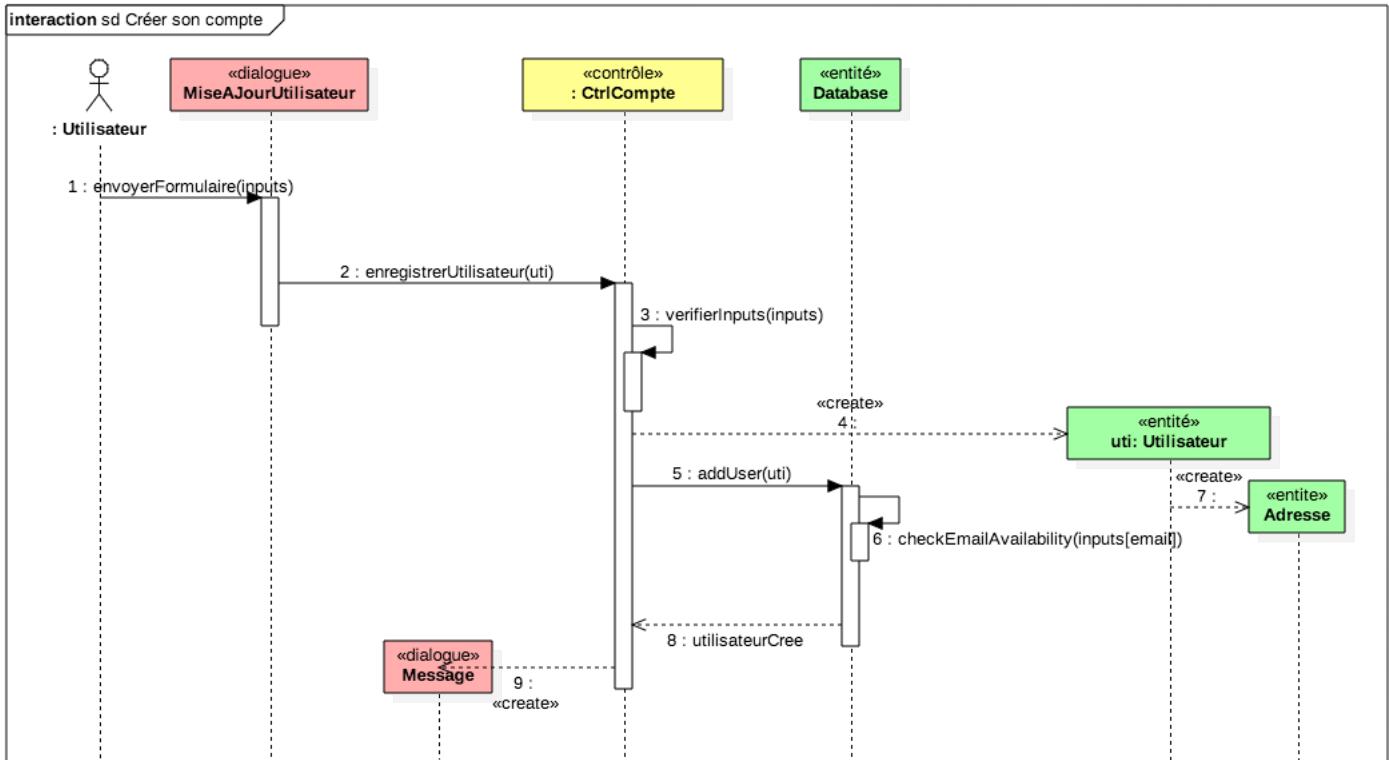


FIGURE 10.1.5 – Diagramme d'interaction "Créer son compte"

La réflexion à partir du diagramme de séquence nous a amené à penser qu'un dialogue Message était nécessaire pour alerter l'utilisateur de la création de son compte. La méthode addUser(uti) introduit l'objet 'uti' créé suite à un message d'un contrôleur CtrlCompte. Une instance d'adresse est immédiatement créée à la suite de la création d'une instance d'utilisateur du fait de leur relation de composition.

L'écriture d'un diagramme de séquence prenant en compte le cas où l'input ne correspond pas au critère ne semble pas pertinent car trop rudimentaire, en effet un message d'erreur sera propagé et l'utilisateur recommencera l'opération.

La méthode checkEmailAvailability(email) va consulter la base de données pour savoir si l'email est déjà pris.

## Diagrammes d'interaction pour le UC "Gérer son compte client"

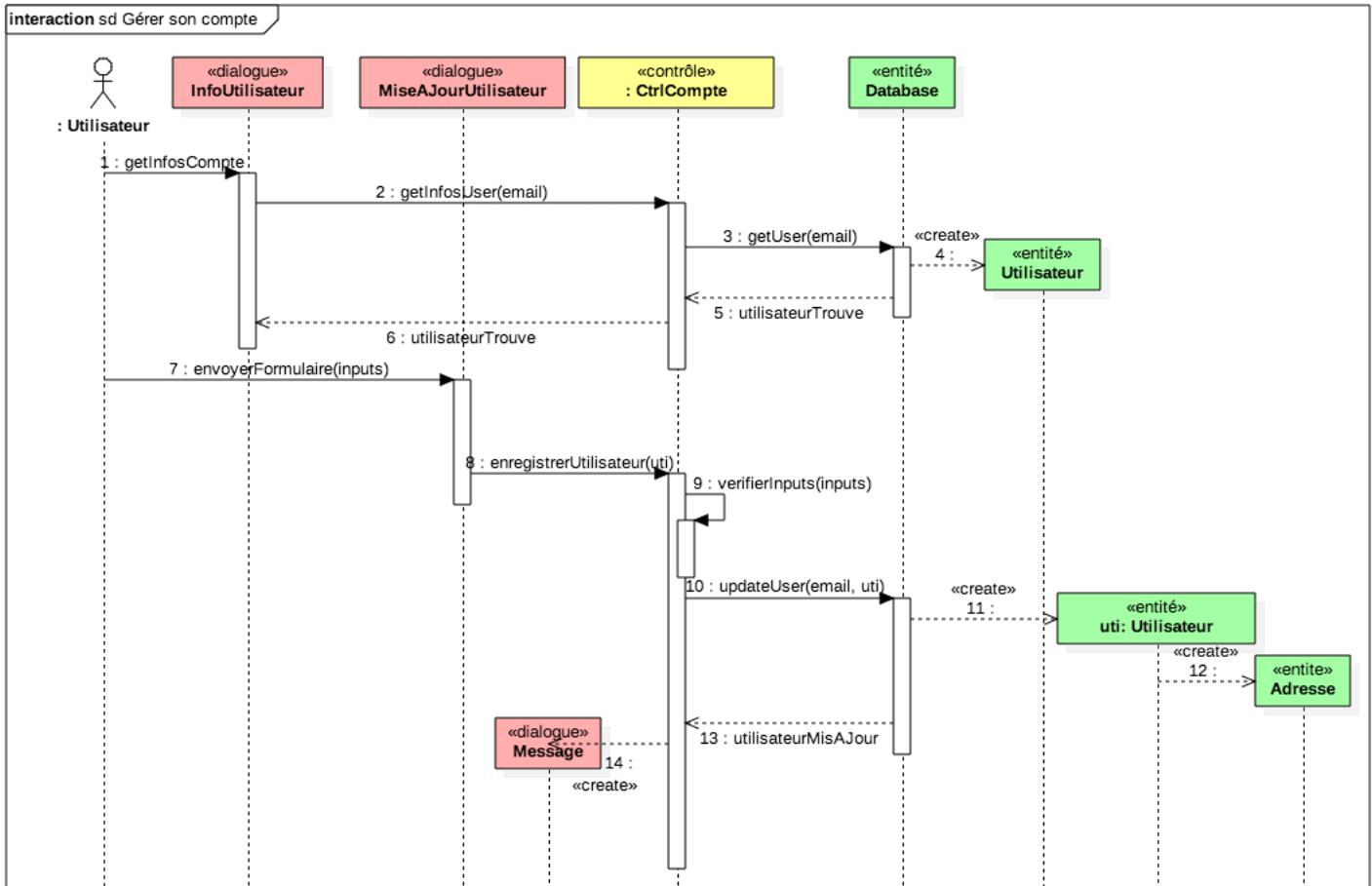


FIGURE 10.1.6 – Diagramme d'interaction "Gérer son compte"

L'utilisateur, en accédant à la page de mise à jour de ses informations, actionne la méthode "getInfosCompte". Le contrôleur va ensuite chercher dans la base de donnée les informations relative à l'utilisateur identifié. Une instance d'utilisateur est créé et contient les informations de l'utilisateur collectées dans la base de données. La méthode updateUser demande une adresse email pour identifier l'utilisateur et crée un nouvel utilisateur qui va mettre à jour avec ses nouveaux champs l'ancien utilisateur. La responsabilité de la création d'une instance d'utilisateur est cette fois-ci laissée à la classe Database.

## Création du gestionnaire de la base de données

Nous pouvons désormais réaliser le CRUD (Create, Read, Update, Delete) pour les utilisateurs. Nous mettons de côté pour le moment le Delete car il n'est pas souhaitable de donner cette possibilité à un utilisateur. Plusieurs méthodes ont été créées. Un choix de conception a été effectué au niveau de l'utilisation de l'e-mail pour retrouver de manière unique un utilisateur. Il s'agit d'une clé primaire qui remplace la clé primaire de la base de données pour le modèle conceptuel (surrogate key). Il est nécessaire de vérifier le mot de passe et la disponibilité d'une adresse e-mail pour qu'un utilisateur s'identifie. Cette action n'a pas fait l'objet d'un UC à cause de sa simplicité.

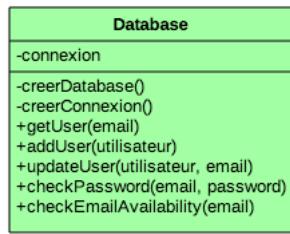


FIGURE 10.1.7 – Gestionnaire de la base de données

## Diagramme de classes

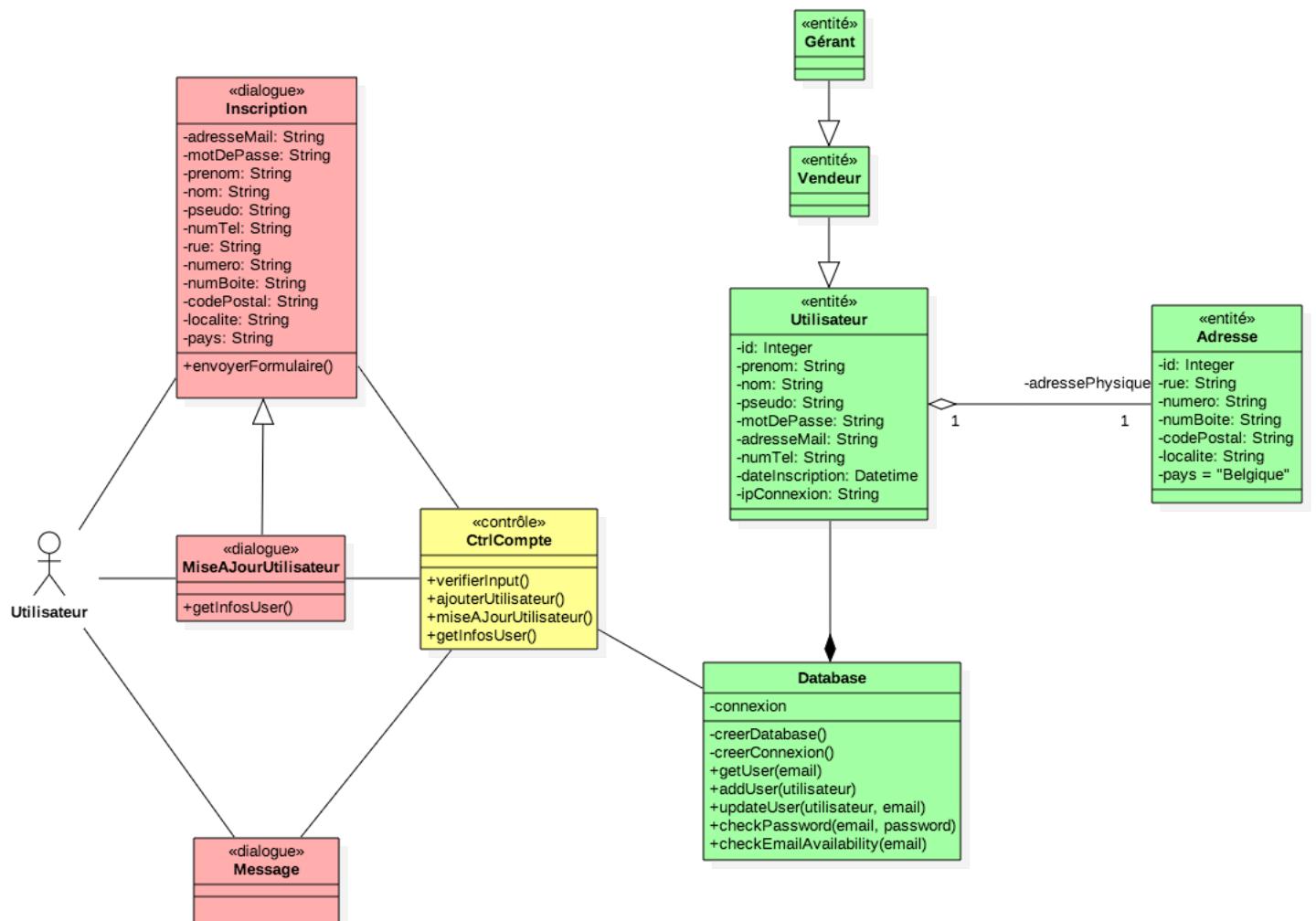


FIGURE 10.1.8 – Diagramme de classes

### 10.1.4 Implémentation

#### Tests unitaires

Pour ce premier UC, il s'agit d'un howto afin de permettre au lecteur de reproduire la démarche pour son propre projet. Il s'agit de travailler selon la méthode Test Driven Development, d'où l'écriture des tests unitaires avant le code proprement dit. Voir appendice A.

**Classe entités** En me demandant comment implémenter cette classe, je me rends compte qu'une relation de composition entre Adresse et Utilisateur ne convient pas. D'abord parce que le couplage est trop fort et empêche la réutilisabilité (cette classe pourra être utilisé pour une autre classe ultérieurement), ensuite parce que le langage PHP ne permet pas de créer une classe interne de manière triviale. Ce surplus de travail n'étant pas nécessaire, je décide de corriger la classe Adresse.

Il reste encore les deux classes à tester : Gérant et Vendeur. Il s'agit de tester cette fois l'héritage. L'opération consiste à vérifier que les types sont identiques.

```
1 public function testInheritance(){
2     $this->assertTrue($this->vendeur instanceof Utilisateur);
3 }
```

Il faut maintenant tester le gestionnaire de la base de données. Une classe de tests est créée et va travailler de la même manière pour les différentes méthodes du gestionnaire de la base de données.

#### Classe du contrôleur

Le première chose à faire est de créer une classe Trait<sup>2</sup> PHP nommé "CtrlCompte.class.php" dans le dossier controllers. Ensuite, on peut créer une classe EnregistrementAction.inc.php dans laquelle les méthodes du modèle relatives à CtrlCompte seront utilisées. On clique droit sur cette dernière classe et on demande à Eclipse PHP de créer une classe de tests. Cette classe de test aura préparé les tests relatifs aux méthodes du Trait CtrlCompte.

---

2. <https://secure.php.net/manual/en/language.oop5.traits.php> Un trait en PHP est une manière commode d'éviter la complexité de l'héritage multiple pour les langages ne comportant que l'héritage simple. Une première tentative pour l'application "Natural Corner" se basait sur une classe abstraite contrôleur à chaque fois héritée par une classe héritée de la classe Action. Cette solution a été élégamment remplacée par un trait.

```

1 <?php
2 include_once(__DIR__.'/../actions/Action.inc.php');
3 /**
4 * @author Daniel
5 * Contrôleur du compte.
6 */
7 trait CtrlCompte{
8     /**
9      * Vérifie la validité des données introduites par l'utilisateur.
10     * @return mixed array
11     */
12    public function verifierInput(){
13        $filters = array(
14            'email' => array(
15                'filter' => FILTER_SANITIZE_EMAIL | FILTER_SANITIZE_MAGIC_QUOTES,
16                'flags' => FILTER_NULL_ON_FAILURE ,
17            ),
18            'pwd' => array(
19                'filter' => FILTER_VALIDATE_REGEXP,
20                'flags' => FILTER_NULL_ON_FAILURE,
21                'options' => array('regexp'=>'^(?=.*\d)(?=.*[a-z])(?=.*[A-Z])[0-9a-zA-Z]{6,}$')
22            ),
23            'pwd2' => array(
24                'filter' => FILTER_VALIDATE_REGEXP,
25                'flags' => FILTER_NULL_ON_FAILURE,
26                'options' => array('regexp'=>'^(?=.*\d)(?=.*[a-z])(?=.*[A-Z])[0-9a-zA-Z]{6,}$')
27            ),
28            'prenom' => array(
29                'filter' => FILTER_VALIDATE_REGEXP | FILTER_SANITIZE_FULL_SPECIAL_CHARS | FILTER_SANITIZE_MAGIC_QUOTES,
30                'flags' => FILTER_NULL_ON_FAILURE ,
31                'options' => array('regexp'=>'^[[0-9a-zA-Z]{3,128}$')
32            ),
33            'nom' => array(
34                'filter' => FILTER_VALIDATE_REGEXP | FILTER_SANITIZE_FULL_SPECIAL_CHARS | FILTER_SANITIZE_MAGIC_QUOTES,
35                'flags' => FILTER_NULL_ON_FAILURE,
36                'options' => array('regexp'=>'^[[0-9a-zA-Z]{3,128}$')
37        );
38    }
39 }

```

FIGURE 10.1.9 – Trait PHP implémentant le contrôleur CtrlCompte

La difficulté ici revient à trouver le moyen de simuler les inputs de l'utilisateur matérialisés par la variable superglobale `$_POST`. Le choix s'est porté sur l'utilisation de la fonction `filter_var_array()`<sup>3</sup>. En suivant cette exemple trouvé sur stackoverflow<sup>4</sup>, j'ai réussi à mettre en place un batterie de tests pour la méthode `validerInputs()`.

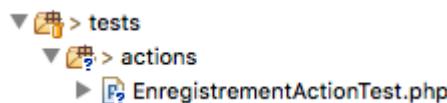


FIGURE 10.1.10 – Répertoires des tests

Les tests se borneront à reproduire les inputs sans rentrer dans les détails puisque les mutateurs de la classe `Utilisateur` ont déjà testés les différentes expressions régulières utilisées dans la méthode `verifierInput()`. Tout le code ne sera malheureusement pas couvert par les tests unitaires car une bonne partie du contrôleur relève de la gestion de la Vue et ne relève pas du domaine des tests unitaires.

3. <https://secure.php.net/manual/fr/function.filter-var-array.php>

4. <https://stackoverflow.com/questions/30123803/setting-post-for-filter-input-arrayinput-post-in-phpunit-test>

```

1 <?php
2
3 require_once '/Users/ivymike/Documents/workspacePHP/NaturalCorner/actions/EnregistrementAction.inc.php';
4
5 /**
6  * Test class for EnregistrementAction.
7  * Generated by PHPUnit on 2016-04-06 at 00:19:26.
8  */
9 class EnregistrementActionTest extends PHPUnit_Framework_TestCase
10 {
11     /**
12      * @var EnregistrementAction
13     */
14     protected $object;
15
16     /**
17      * Sets up the fixture, for example, opens a network connection.
18      * This method is called before a test is executed.
19      */
20     protected function setUp()
21     {
22         $_SERVER['REQUEST_METHOD'] = 'POST';
23         $this->object = new EnregistrementAction();
24     }

```

FIGURE 10.1.11 – Tests unitaires du contrôleur CtrlCompte

A la ligne 22, on peut voir la technique employée pour imiter les inputs de l'utilisateur lors du test unitaire.

```

45
46     /**
47      * @covers EnregistrementAction::verifierInput
48      * @todo Implement testVerifierInput().
49      */
50     public function testVerifierInput()
51     {
52
53         $_POST = array(
54             'email' => 'coucou@coucou.be',
55             'pwd' => 'coucoU1',
56             'pwd2' => 'coucoU1',
57             'prenom' => 'coucou',
58             'nom' => 'coucou',
59             'pseudo' => 'coucou',
60             'adresse' => 'coucou',
61             'poste' => '1000',
62             'localite' => 'coucou'
63         );
64         $data = $this->object->verifierInput();
65         $this->assertEquals('coucou@coucou.be', $data['email']);
66         $this->assertEquals('coucoU1', $data['pwd']);
67         $this->assertEquals('coucoU1', $data['pwd2']);
68         $this->assertEquals('coucou', $data['prenom']);
69         $this->assertEquals('coucou', $data['nom']);
70         $this->assertEquals('coucou', $data['pseudo']);
71         $this->assertEquals('coucou', $data['adresse']);
72         $this->assertEquals('1000', $data['poste']);
73         $this->assertEquals('coucou', $data['localite']);
74     }

```

FIGURE 10.1.12 – Tests unitaires du contrôleur CtrlCompte-méthode testVerifierInput()

A la ligne 53, on initialise et remplit un tableau ayant le nom de la variable superglobale récupérant les requêtes POST du protocole HTML en PHP. Il suffit alors de tester la méthode verifierInput().

## Code

La démarche préconisée dans le livre de référence<sup>5</sup> suggère de mettre en place une conception objet détaillée. Nous passerons plutôt directement à l'écriture du code en reliant au framework mis en place la conception objet préliminaire. Cette façon de procéder évite de rentrer dans des détails d'implémentation sur un schéma UML liés à un langage qui peut lui même varier en fonction d'un framework.

Le framework impose que les différentes classes d'implémentation dérivent des classes mères View et Action. Dès lors, le contrôleur se matérialise par les différentes classe \*Action.inc.php. Cependant, l'héritage multiple n'étant pas possible en PHP, il est compliqué de créer une classe abstraite qui aurait les méthodes du contrôleur. L'idée pour implémenter le contrôleur 'CtrlCompte' de l'analyse est de créer un trait PHP appelé "CtrlCompte" qui comporte les méthodes implémentées et qui sera utilisées par les classes dérivant de la classe Action. Le framework peut ainsi rendre compte du modèle modèle-vue-contrôleur.

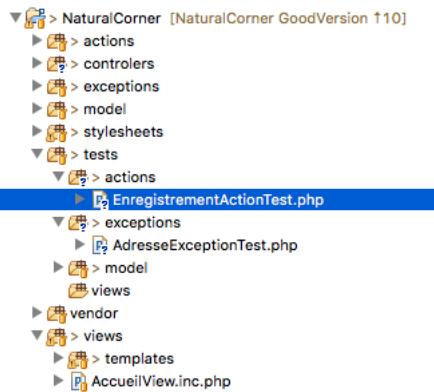


FIGURE 10.1.13 – Hiérarchie des dossiers représentant une conception objet MVC en PHP

Dans la classe du contrôleur EnregistrementAction.inc.php (Voir figure 9.1.20), les données envoyées par l'utilisateur pour s'enregistrer doivent être traitées afin d'assurer l'intégrité des données manipulées par le modèle et la base de données. Il convient de se protéger des certaines failles de sécurité. On utilise traditionnellement différentes techniques comme la fonction magic\_quote ou html\_special\_char pour échapper le code. Cependant, PHP a prévu un moyen sécurisé et efficace pour manier les données de l'utilisateur : la méthode filter\_input.

```
1 $form_data = filter_input_array(INPUT_POST, $filters);
```

Il suffit de transmettre un tableau contenant l'ensemble des filtres sur les inputs de l'utilisateur à la méthode filter\_input\_array() et l'on obtient un procédé propre et net pour manier les données de l'utilisateur. Sur l'exemple choisi (ligne 14 à 16), on applique le filtre FILTER\_SANITIZE\_EMAIL et FILTER\_SANITIZE\_MAGIC\_QUOTES pour l'e-mail indiqué par l'utilisateur. Le flag FILTER\_NULL\_ON\_FAILURE permet d'éviter de répéter les méthodes isset() avec !empty() et de les remplacer par un simple test sur input !==NULL. Pour le mot de passe, une option a été ajoutée, l'expression régulière sur le mot de passe afin de contrôler qu'il y a bien au moins 6 caractères munis d'une majuscule et d'un nombre au minimum. Si un des filtres ne passe pas, la valeur NULL est retournée. Avec ces données, un nouvel utilisateur est instancié :

5. Roques, P., UML2 Modéliser une application web, Eyrolles 4ème édition, 2008, p. 147

```

59 /**
60  * @param array $form_data données introduite au moment de l'inscription.
61  * @param string $message message d'erreur eventuel.
62  * @return bool si l'utilisateur est inséré dans le catalogue.
63 */
64 public function ajouterUtilisateur($form_data, $insertionMessage=''){
65     $estInsere=false;
66     $utilisateur = new Utilisateur($form_data['prenom'], $form_data['nom'], $form_data['pseudo'],
67         password_hash($form_data['pwd'], PASSWORD_BCRYPT, ["cost"=>PASSWORD_BCRYPT_DEFAULT_COST]),
68         $form_data['email'], $form_data['adresse'], $form_data['poste'],
69         $form_data['localite'], new DateTime('NOW'),
70         filter_var(isset($_SERVER['REMOTE_ADDR']) ? $_SERVER['REMOTE_ADDR'] : NULL, FILTER_VALIDATE_IP));
71     try{
72         $estInsere = $this->database->addUser($utilisateur);
73     }catch(EmailAlreadyTakenException $eate){
74         $estInsere=false;
75         $insertionMessage = $eate->getMessage();
76     }
77     return $estInsere;
78 }
```

FIGURE 10.1.14 – Méthode ajouterUtilisateur()

A noter l'utilisation, au lieu du traditionnel md5(), de la méthode password\_hash() (ligne 67) qui, selon la documentation PHP, est plus appropriée pour le “hashage” d'un mot de passe<sup>6</sup>. Son utilisation est très simple et passe par un “salage” du mot de passe nommé cost relatif à une constante PASSWORD\_BCRYPT\_DEFAULT\_COST. Cette constante devra changer dans le temps par la communauté des développeurs de la bibliothèque PHP, ce qui permet ainsi au développeur d'une application de ne pas devoir trop se documenter sur les nouveautés en matière de “salage” de mot de passe<sup>7</sup>.

L'adresse IP est récupérée à l'aide de la variable superglobale \$\_SERVEUR['REMOTE\_ADDR'] (ligne 70). La méthode filter\_var() filtre le résultat obtenu avec la constante FILTER\_VALIDATE\_IP.

La classe du modèle Database.inc.php :

6. Voir <https://secure.php.net/manual/fr/faq.passwords.php>  
 7. <https://secure.php.net/manual/fr/function.password-discretionary{-}{}{}hash.php>

```

266 /**
267 * Met à jour dans la base de donnée un utilisateur.
268 * @param Utilisateur $utilisateurMisAJour -> un utilisateur avec les nouvelles données.
269 * @param string $email -> l'email de l'utilisateur à modifier.
270 * @return boolean indique si la mise à jour est réussie.
271 * @throws EmailAlreadyTakenException si l'email à modifier est déjà pris.
272 */
273 public function updateUser(Utilisateur $utilisateurMisAJour, $email){
274     $estMisAJour=false;
275     $email=trim($email);
276     if($utilisateurMisAJour->getAdresseMail()!=$email)
277         if(!$this->checkEmailAvailability($utilisateurMisAJour->getAdresseMail()))
278             throw new EmailAlreadyTakenException("Cet email existe déjà dans notre base de données.");
279     $this->creerConnexion();
280     $requete = $this->connection->prepare(" UPDATE UTILISATEURS
281                                         SET PRENOM=:prenom, NOM=:nom, PSEUDO=:pseudo,
282                                         PASS=:pass, ADRESSE_MAIL=:adresse_mail,
283                                         ADRESSE_PHYSIQUE=:adresse_physique,
284                                         CODE_POSTAL=:code_postal, LOCALITE=:localite,
285                                         DATE_INSCRIPTION=:date_inscription,
286                                         IP_CONNEXION=:IP_CONNEXION
287                                         WHERE ADRESSE_MAIL=:EMAIL");
288     $estMisAJour = $requete->execute(array(
289         ':prenom'=> $utilisateurMisAJour->getPrenom(),
290         ':nom'=> $utilisateurMisAJour->getNom(),
291         ':pseudo'=> $utilisateurMisAJour->getPseudo(),
292         ':pass'=> $utilisateurMisAJour->getPass() ,
293         ':adresse_mail' => $utilisateurMisAJour->getAdresseMail(),
294         ':adresse_physique' => $utilisateurMisAJour->getAdressePhysique(),
295         ':code_postal' => $utilisateurMisAJour->getCodePostal(),
296         ':localite' => $utilisateurMisAJour->getLocalite(),
297         ':date_inscription' => $utilisateurMisAJour->getDateInscription(),
298         ':IP_CONNEXION' =>$utilisateurMisAJour->getIdConnexion(),
299         ':EMAIL'=>$email
300     ));
301     $requete->closeCursor();
302     return $estMisAJour;
303 }

```

FIGURE 10.1.15 – Méthode updateUser de la classe Database

La documentation de style Javadoc permet de comprendre facilement le code (ligne 266 à 272)<sup>8</sup>. Envoie un booléen et une exception relatif à l'e-mail en cas d'échecs. Le booléen est un choix de type 'procédural' qui garde des avantages. J'ai choisi de lancer une exception sur l'email, qui fait office de clé auxiliaire (surrogate), si celui-ci est déjà pris. La bibliothèque PDO de PHP, qui est par ailleurs orientée objet, s'occupe de l'accès à la base de données. Les requêtes préparées ont pour elles d'offrir une bonne défense à la faille injection SQL. Comme il n'y a pas de ORM utilisé, les requêtes SQL sont à chaque fois pensées et élaborées de la manière la plus judicieuse pour l'application. La requête va faire la mise à jour des données d'un utilisateur.

Le template userinfo.inc.php pour la class de la vue VoirCompteView.inc.php :

<sup>8</sup>. [http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-005-elements-of-software-construction-fall-2011/lecture-notes/MIT6\\_005F11\\_lec03.pdf](http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-005-elements-of-software-construction-fall-2011/lecture-notes/MIT6_005F11_lec03.pdf), consulté le 31/03/2016

```

1<div class="block-center "><br>
2
3<table class="table-responsive col-xs-6 col-sm-6 col-md-6 col-lg-6 col-md-offset-3 col-sm-offset-3">
4    <tr>
5        <td class="label label-success">
6            Prénom
7        </td>
8        <td>
9            <div class="pull-right"><?php echo $this->user->getPrenom(); ?></div>
10       </td>
11    </tr>
12    <tr><td><br></td></tr>

```

FIGURE 10.1.16 – Template userinfos.inc.php de la vue VoirCompteView

Ce template est appelé par la classe VoirCompteView :

```

1<?php
2include_once("views/View.inc.php");
3
4class VoirCompteView extends View {
5
6    /**
7     * Affiche le formulaire de modification de mot de passe.
8
9     * @see View::displayBody()
10    */
11    public function displayBody() {
12        include("views/templates/userinfos.inc.php");
13    }
14}
15

```

FIGURE 10.1.17 – Classe VoirCompteView.inc.php

Cette manière de procéder permet de séparer le code PHP du code HTML au maximum. Seul des valeurs comme \$this->user->getPrenom(); se retrouvent dans le code. A noter l'utilisation de Bootstrap dans les balises HTML : <table class="table-responsive col-xs-6 col-sm-6 col-md-6 col-lg-6 col-md-offset-3 col-sm-offset-3">. signifie qu'il s'agit d'un tableau s'ajustant aux différentes tailles d'écran ou du navigateur selon une logique "grid". Chaque "row" est découpée en 12 cases. Si l'écran est de taille moyenne "md" alors le tableau doit prendre 6 cases après avoir laissé un "offset" de trois cases à gauche class=" col-md-6 col-md-offset-3". Trois cases à droites sont automatiques calculées puisque 12 - 6 - 3 = 3. Cette manière simple de procéder permet de créer un site responsive qui s'adapte à toutes les tailles d'écran.

### 10.1.5 Difficultés rencontrées, réflexions et conclusions provisoires

- La difficulté principale tient à bien comprendre les liens entre le contrôleur, la vue et les templates. Une fois habitué, il devient plus aisée de penser de cette manière. Il est intéressant de constater que ce MVC s'adapte très bien à une analyse UML.
- L'écriture de la classe Utilisateur.class.php m'a donné plusieurs choix d'implémentation. Vais-je laisser le contrôleur vérifier les données entrées par l'utilisateur ou vais-je les vérifier dans les mutateurs des attributs de la classe ? L'idée serait de créer une méthode statique sanitazeData() comme préconisé dans le livre "PHP 5 Social Networking" <sup>9</sup>. Cette méthode serait sans une classe "statique" ou utilitaire dans le contrôleur et je l'utilise directement dans les classes du modèle. J'ai ainsi un contrôle des inputs à la source et me prémunie directement de la faille XSS ou de l'injection SQL. Finalement, en compulsant la documentation PHP et en m'inspirant du site du professeur B. Estellon <sup>10</sup>, j'ai préféré utilisé la méthode filter\_input().

9. PHP 5 Social Networking, Packt Publishing, 2010, p. 37

10. <http://pageperso.lif.univ-mrs.fr/~bertrand.estellon/webCCI/cours/Filtragedesentres.pdf>

- Il fallait configurer plusieurs modules pour faire fonctionner l'ensemble des logiciels. J'ai réussi à intégrer le module PTI<sup>11</sup> à Eclipse, j'étais en mesure d'effectuer des tests unitaires avec PHPUnit<sup>12</sup>. La documentation de PHP Unit est très bien fournie et je me suis formé à réaliser des tests de manière satisfaisante en la lisant. En réalité, la bibliothèque PHP Unit est très semblable dans sa philosophie à JUnit pour le langage Java et tout ce que j'ai appris sera sans nul doute profitable pour un autre langage de programmation.
- J'ai appris qu'il n'y avait pas de surcharge de méthode (overload) en PHP 5, je dois trouver une solution alternative. Soit je crée "à la main" la surcharge grâce à une méthode trouvée ici<sup>13</sup>. Soit je me passe de la surcharge. Je dois donc utiliser un seul constructeur, ce qui est plutôt pénible pour la récupération d'un utilisateur dans la base de données. Le problème vient du fait que je dois connaître le fichier Utilisateur et savoir à l'avance le résultat de la requête, ce qui implique un fort couplage entre la base de données et la classe Utilisateur. Je voudrais éviter ce cas de figure. Première possibilité je créer un pattern DAO. Autre possibilité je mets en oeuvre "l'hydratation des objects" comme indiqué dans ici<sup>14</sup>. Je vais essayer la solution la plus directe en instanciant les objets avec le constructeur le plus évident (chaque attribut privé de la classe reçoit un paramètre lui donnant une nouvelle valeur).
- Je n'arrivais pas à configurer le chemin d'accès dans la fonction require\_once de manière à l'utiliser en chemin relatif au dossier du projet plutôt qu'en chemin absolu sur mon système de fichiers. Après quelques recherches, j'ai trouvé la variable "Magic Constant"<sup>15</sup> \_\_DIR\_\_ qui fournit le chemin absolu du dossier dans lequel s'exécute le script.
- En faisant les tests je découvre qu'un objet de type horodatage "DateTime" ne peut être converti en objet String et j'ajoute le new DateTime qui permet de créer un objet à la volée. Pour l'opération inverse de l'objet vers un un objet String j'ai trouvé sur StackOverFlow une réponse satisfaisante utilisant la méthode format()<sup>16</sup>.
- J'apprends que le test addUser() renvoie faux et, en vérifiant la base de données, je me rends compte qu'il n'y a effectivement aucune insertion. La base de données est créé et la requête aurait dû fonctionner. En fait, je dois essayer de trouver le moyen de voir le résultat de l'exception envoyée par la méthode PDO qui s'occupe de l'insertion. Il faut en fait configurer PDO pour qu'il envoie les exceptions<sup>17</sup>. J'ai maintenant les messages d'erreur pour débuguer le code. J'apprends que j'ai le message SQLState[3D000] "invalid catalog name : 1046 No Database selected". J'apprends qu'en faisait un copier-coller du code je n'ai pas replacé le nom de la base de données à laquelle je me connecte.
- Erreur de conception de la méthode getUser() : je ne peux pas utiliser l'ID de l'utilisateur pour le retrouver dans la grille CRUD. L'id est intrinsèque à la conception de la base de données et l'auto incrémentation ne me permet pas de gérer cela. Je dois en fait envisager une clé secondaire avec laquelle je peux faire une recherche selon la logique métier ( qui ne doit en fait pas interférer avec la logique informatique).
- La configuration du dossier comportant les fichiers .js et .css dans Google App Engine m'a pris beaucoup de temps. Je réfléchis à la manière dont je vais concevoir l'application. Je vais créer une formulaire qui permet de s'inscrire en tant que client à l'application.
- J'ai installé le SDK Facebook pour répondre au besoin de "like" de la page Facebook demandé par le gérant de Natural Corner. L'installation est très facile et je peux déjà utiliser les fonctionnalités du SDK en une vingtaine de minutes.
- En écrivant les setters et getters de la classe Utilisateur, je me suis rendu compte que je peux économiser et rendre plus pratique la méthode getDateInscription() en utilisant ->format('Y-m-d H:i:s').
- J'ai trouvé le moyen de sécurisé l'accès à la base de données. Il est évident que les mots de passe ne peuvent se trouver en dur dans le code, à plus forte raison si celui-ci apparaît sur Github en public. Il s'agit de remplir le fichier app.yaml comme indiqué ici<sup>18</sup>. Je crée des variables d'environnement pour le projet. Le fichier app.yaml ne sera plus "committed" et "pushed" sur Github dorénavant.
- LA SDK Facebook avec PHP demande une fonction de callback contenant l'adresse du site. Etant donné que je travaille en local, je vais déployer l'application sur Google App Engine et effectuer les tests directement dessus.

11. <https://marketplace.eclipse.org/content/pti-php-tool-integration>

12. <https://phpunit.de/manual/current/en/index.html>

13. <https://openclassrooms.com/courses/php5-comment-gerer-l-absence-de-surcharge>

14. <https://openclassrooms.com/courses/programmez-en-orientee-objet-en-php/manipulation-de-donnees-stockees>

15. <https://secure.php.net/manual/en/language.constants.predefined.php>

16. <http://stackoverflow.com/questions/10569053/convert-datetime-to-string-php>

17. <http://php.net/manual/fr/pdo.error-handling.php>

18. <https://gae-php-tips.appspot.com/2013/10/22/getting-started-with-laravel-on-php-for-app-engine/>

Pour ne pas payer une instance de CloudSQL, je vais mettre en commentaire linstanciation de la base de données. LAPI Facebook ne voulait pas fonctionner tant que je n'avais pas ajouter un favicon... Le login avec Facebook fonctionne. Lapi est encore buggué.

- La fonction updateUser me pose problème. Je dois utiliser l'email pour mettre à jour en tant que clé secondaire mais si j'utilise un email déjà connu je ne peux pas mettre à jour. La solution est de voir si l'adresse email précédente est différente de celle de l'objet Utilisateur qui va modifier l'utilisateur dans la base de données. Si ce sont les mêmes adresses, cela signifie que l'utilisateur ne modifie pas son adresse e-mail et donc je ne contrôle pas si cette adresse est déjà prise et je n'envoie pas l'exception EmailAlreadyTaken. Si ces adresses sont différentes, c'est que l'utilisateur veut modifier son adresse e-mail, alors je dois vérifier si cette nouvelle adresse est disponible.
- Petit problème de responsabilité de la méthode qui reçoit une exception. J'ai fait le choix d'initialiser les variables à l'aide des mutateurs dans le constructeur de la classe Utilisateur. Cependant, lorsqu'une exception est levée, ce constructeur se borne à l'attraper et à afficher le message d'erreur. Lors des tests, je n'arrive pas à retrouver ces messages dans la console et à débugguer correctement le code.
- J'ai passé trois heures à chercher pourquoi le retour d'une fonction ne voulait pas retourner un objet d'instance Utilisateur. En fait, j'utilisais "echo \$user" qui utilise la méthode `_toString()`. Je n'avais pas tester en TDD cette fonction et j'en ai payé le prix.
- La méthode employée pour réaliser ce UC va servir à l'ensemble des UC décrits pour la suite de ce travail. Cette méthode s'adapte très bien à une analyse de type "Processus unifié" et permet de livrer rapidement une produit testable et utilisable par le gérant du magasin.

**Conclusion (provisoire)** Le résultat obtenu me semble très encourageant. En voulant connecter différents éléments disparates, j'ai créé un Framework PHP sur le Cloud parfaitement adapté comme backend à une application de type Angularjs ou Android. En effet, si je crée une API se basant sur l'architecture REST, il me sera possible créer un lien avec n'importe quelle application cliente. La conception de l'application alliant l'analyse décrite par des diagrammes UML, le développement conduit par les tests, la flexibilité d'un PaaS et la réutilisabilité du patron de conception MVC est un argument de premier choix pour répondre au problème actuel du développement full-stack.

### 10.1.6 Première livraison du logiciel et feedback du client.

Le 25 janvier 2016 à 15h30, dans les locaux du magasin "Natural Corner", j'ai pu montrer et utiliser cette version bêta de l'application. Le design paru convainquant et j'ai reçu certaines recommandations. Le gérant désirait :

1. Voir figurer l'adresse de l'enseigne sur la page d'accueil,
2. voir figurer l'horaire,
3. voir figurer un lien vers le site page Facebook avec les recettes bio,
4. incorporer un lien vers l'app pour commander les ingrédients en promo.
5. message en dessous du Natural Corner "bio, cosmétique, etc..."
6. produits en promo

Ce point de vue a été consigné et sera pris en compte pour les incrémentations suivantes.

## 10.2 UC "Recherche des produits"

### 10.2.1 Diagramme des classes participantes (DCP)

#### Réflexions sur le UC "Recherche des produits"

Une recherche d'un produit doit fournir rapidement les produits désirés. Les résultats doivent pouvoir être classés comme indiqués dans la spécification du projet. Pour correspondre aux attentes du client, les méthodes de présentation des résultats de la recherche ont été mis en place (`classerParDenomination()`, `classerParPrixUnitaire()`, `classerCategorie()`, `pageSuivante()`, `pagePrecedente()`). Les produits peuvent être mis dans le panier (`mettreDansPanier()`) et on peut accéder à une page de détails du produit. Deux contrôleurs entrent en jeu. Le contrôleur "CtrlRecherche" va gérer la recherche, et le contrôleur "CtrlArticle" va s'occuper de l'opération de sélection de l'article dans le panier. Pour clarifier la situation, on peut s'aider d'un diagramme de communication pour expliciter le cas simple d'une recherche rapide par mot-clé.

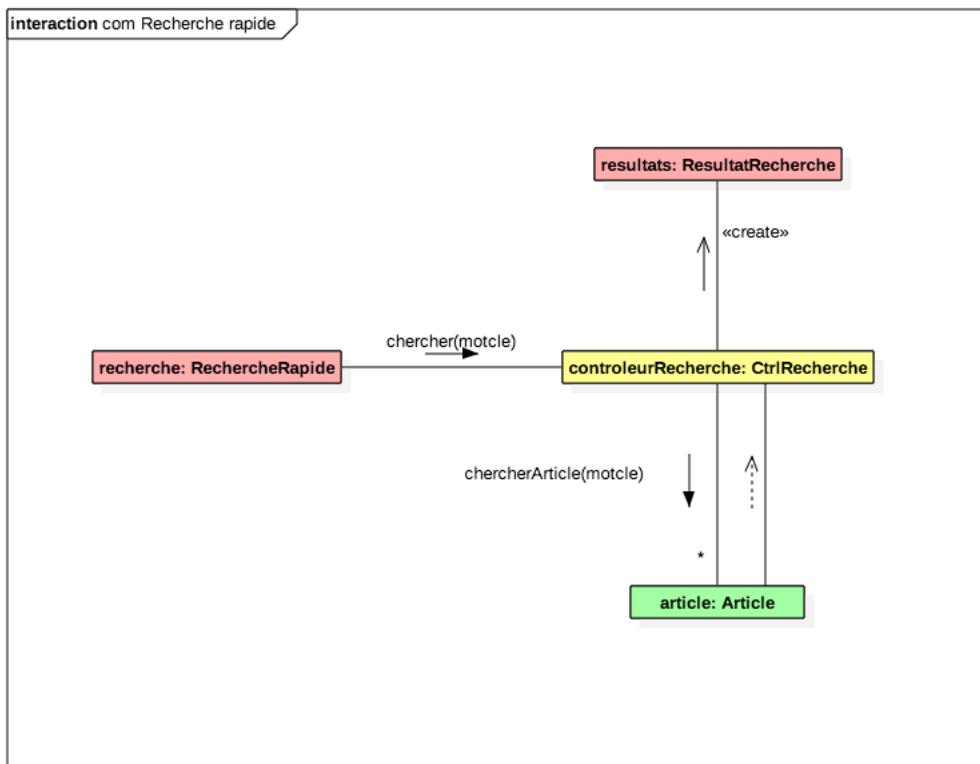


FIGURE 10.2.1 – Diagramme de communication ”Recherche rapide”

La vue demande la recherche de tous les articles correspondant au mot-clé. Le contrôleur interroge la base de données, qui lui renvoie une collection d’articles correspondant au mot-clé. Ensuite, le contrôleur peut envoyer la collection à la vue ResultatRecherche qu’il crée.

Si l’utilisateur demande les détails de l’article, il actionnera un autre processus, impliquant cette fois le contrôleur CtrlArticle. Une fois les détails sur l’article reçu, la vue FicheDetaillee est créée pour montrer à l’utilisateur les détails de l’articles.

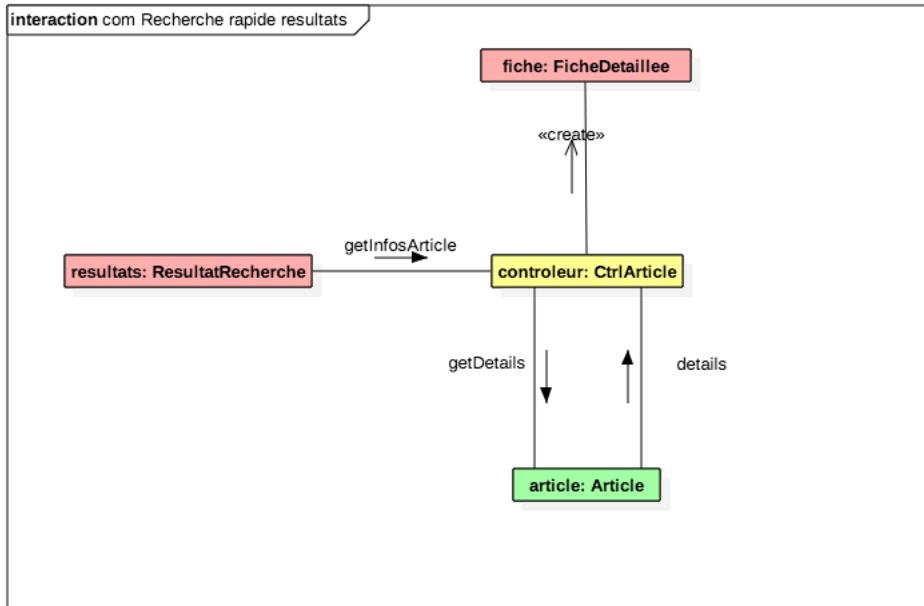


FIGURE 10.2.2 – Diagramme de communication "Recherche Rapide"

### Diagramme des classes participantes

Un premier diagramme rend compte de la recherche, qu'elle soit simple ou détaillée. Ce DCP prend en compte les deux diagrammes précédents. Nous nous approchons de la logique du logiciel petit à petit en prennent en compte différents points de vue.

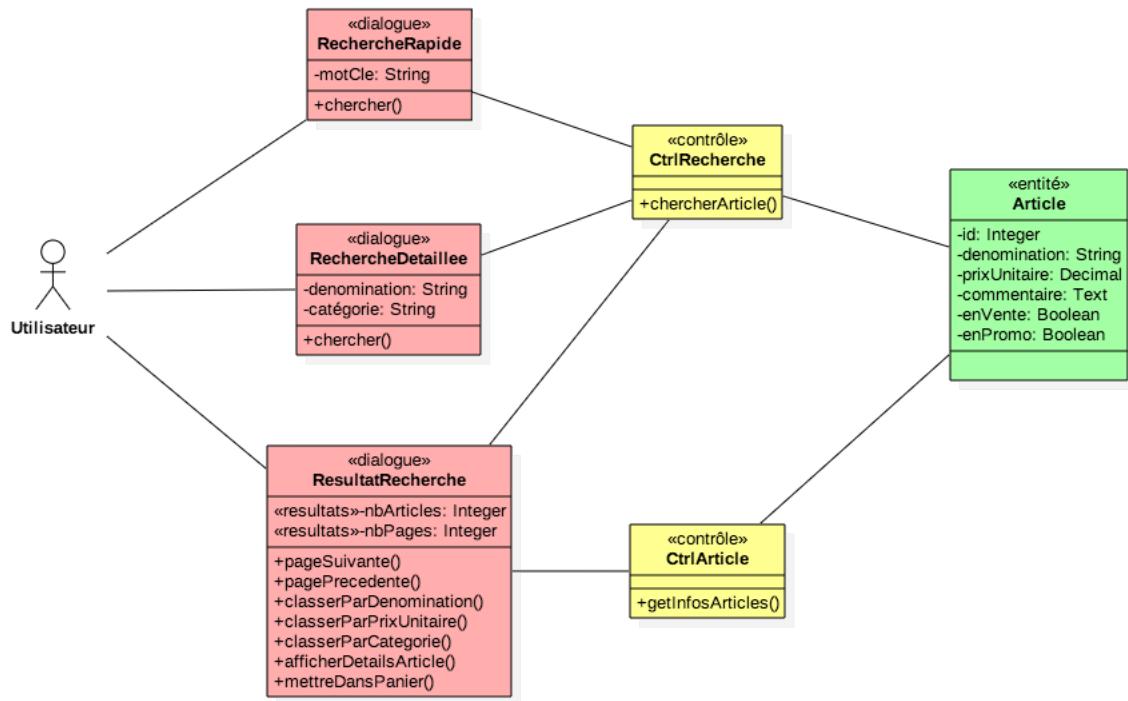


FIGURE 10.2.3 – Diagramme des classes participantes "Recherche des produits"

### 10.2.2 Diagramme de navigation

On peut simplement représenter la navigation de ce UC. Le schéma est suffisamment explicite pour se passer de commentaire. A noter l'historique de la recherche lorsqu'une recherche a échoué et que l'utilisateur veut faire une autre recherche.

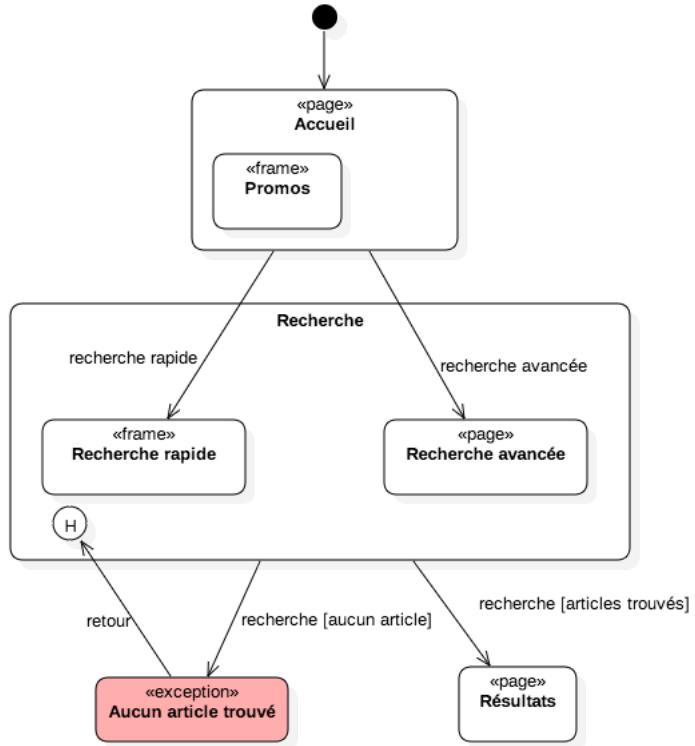


FIGURE 10.2.4 – Diagramme de navigation "Recherche"

### 10.2.3 Conception objet préliminaire

#### Diagrammes d'interactions pour le UC "Recherche des produits"

Les diagrammes d'interaction vont permettre de compléter la vision de l'UC afin d'être exhaustif et de pouvoir passer à l'écriture complet du diagramme de classe. Le dialogue RechercheRapide aura la responsabilité de contrôler la syntaxe du mot-clé. Le gestionnaire de base de données, la classe Database, s'occupe d'aller chercher les articles désirés, étant donné qu'elle est "Information Expert" dans ce cas<sup>19</sup>.

19. Applying UML and Patterns, An introduction to object-oriented analysis and design and unified process, 2001, seconde édition, p. 221

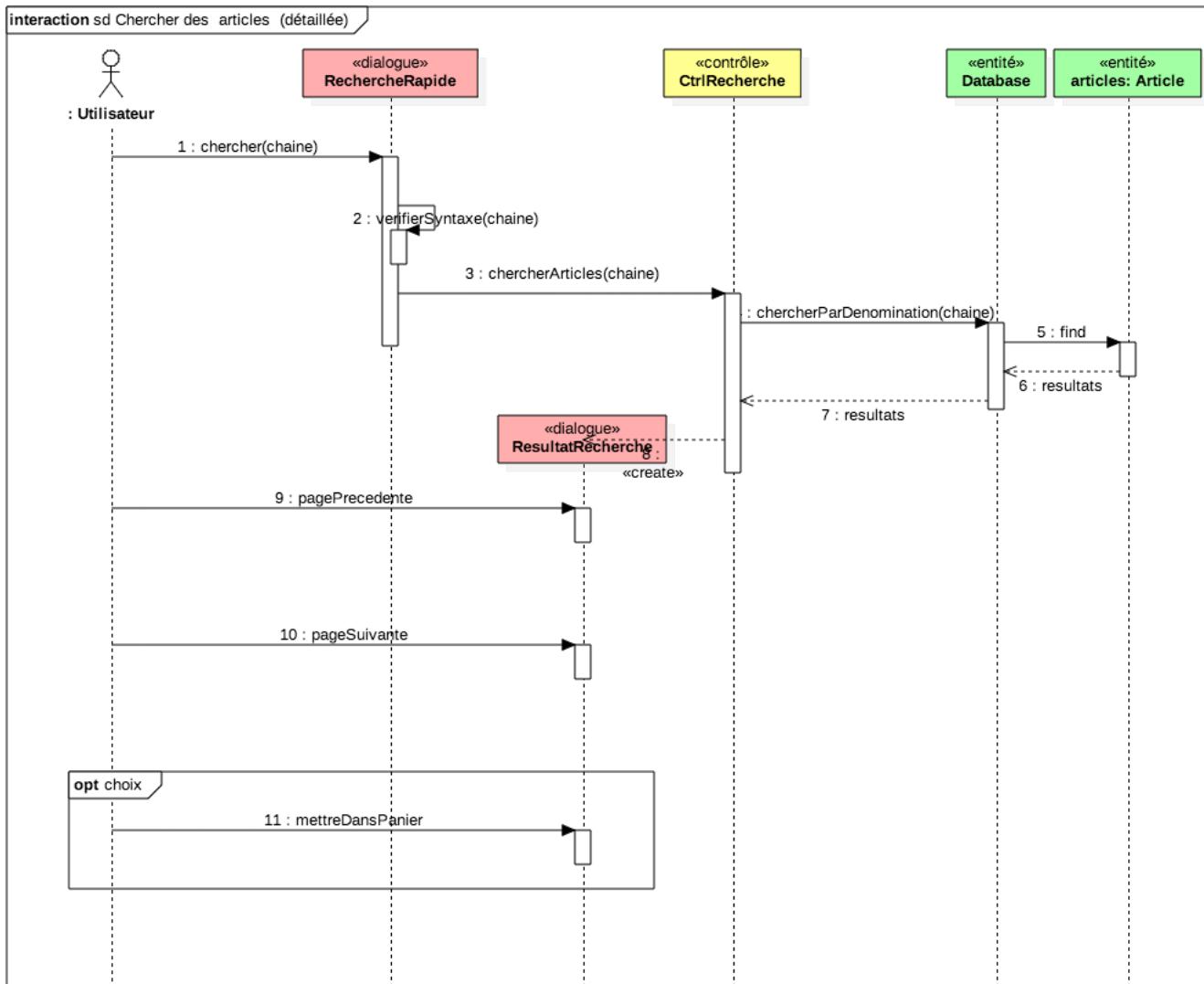


FIGURE 10.2.5 – Diagramme de séquence "Recherche rapide article"

Les nouvelles méthodes liées à la vue vont permettre à l'utilisateur de changer de page lorsque la recherche est trop grande.

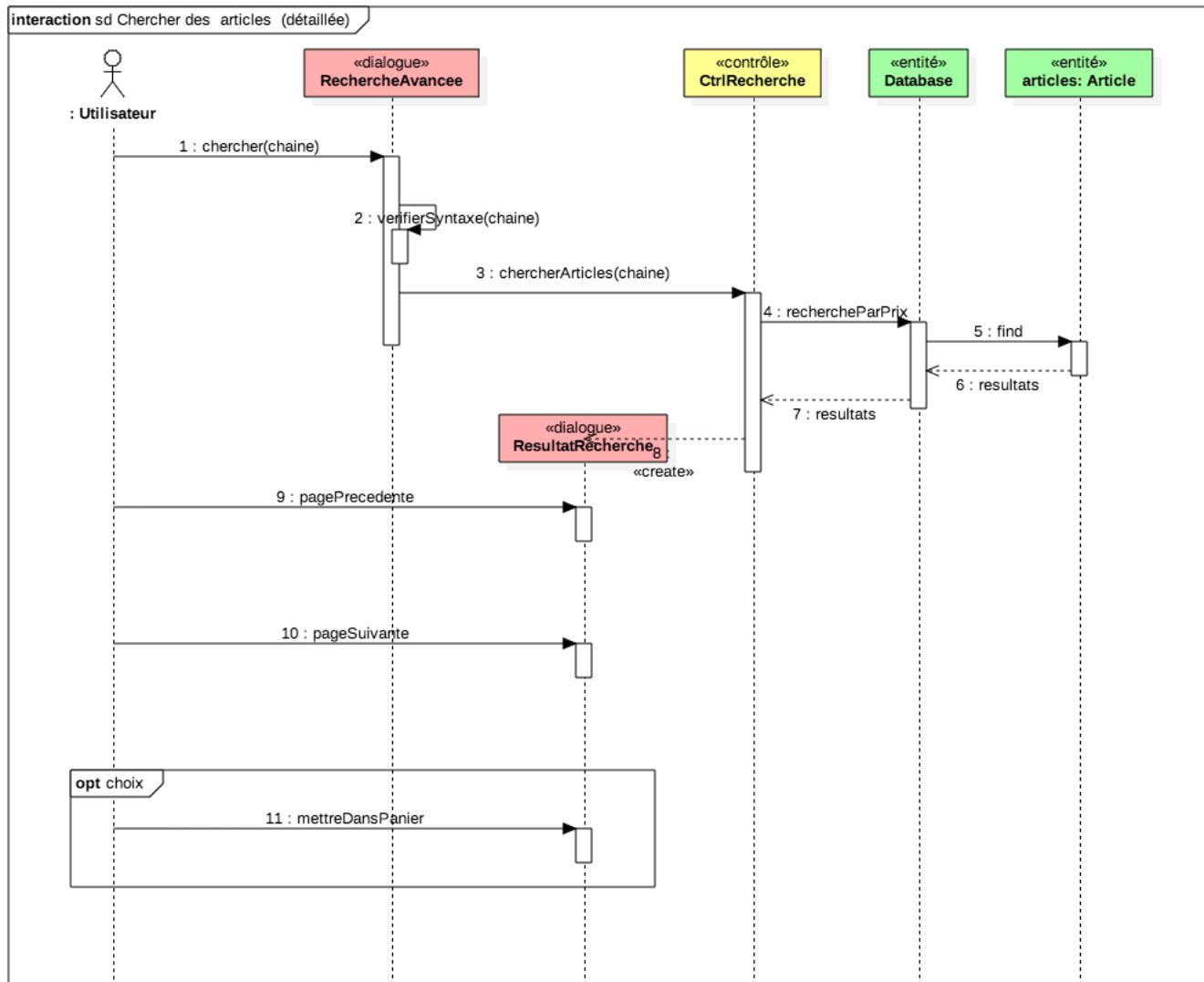


FIGURE 10.2.6 – Diagramme de séquence ”Recherche détaillée par prix”

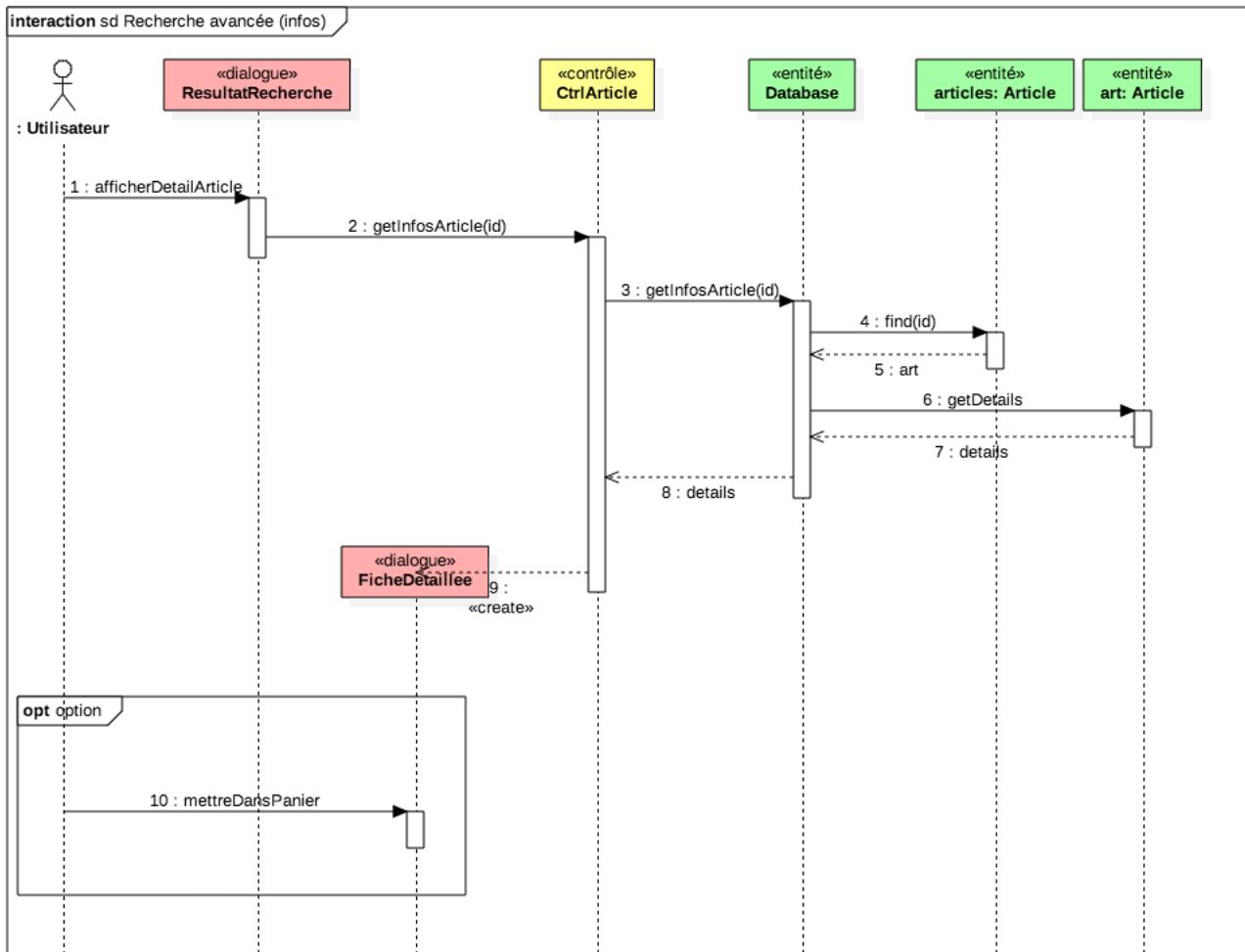


FIGURE 10.2.7 – Diagramme de séquence "Recherche avancée"(infos)

En fait, une recherche rapide est une recherche avancée sur la dénomination. Une recherche avancée est la même sauf qu'il y a plusieurs possibilités de recherche et que la navigation diffère. Nous nous bornerons à étudier la recherche par prix, les autres recherches étant similaires du point de vue de l'interaction des classes.

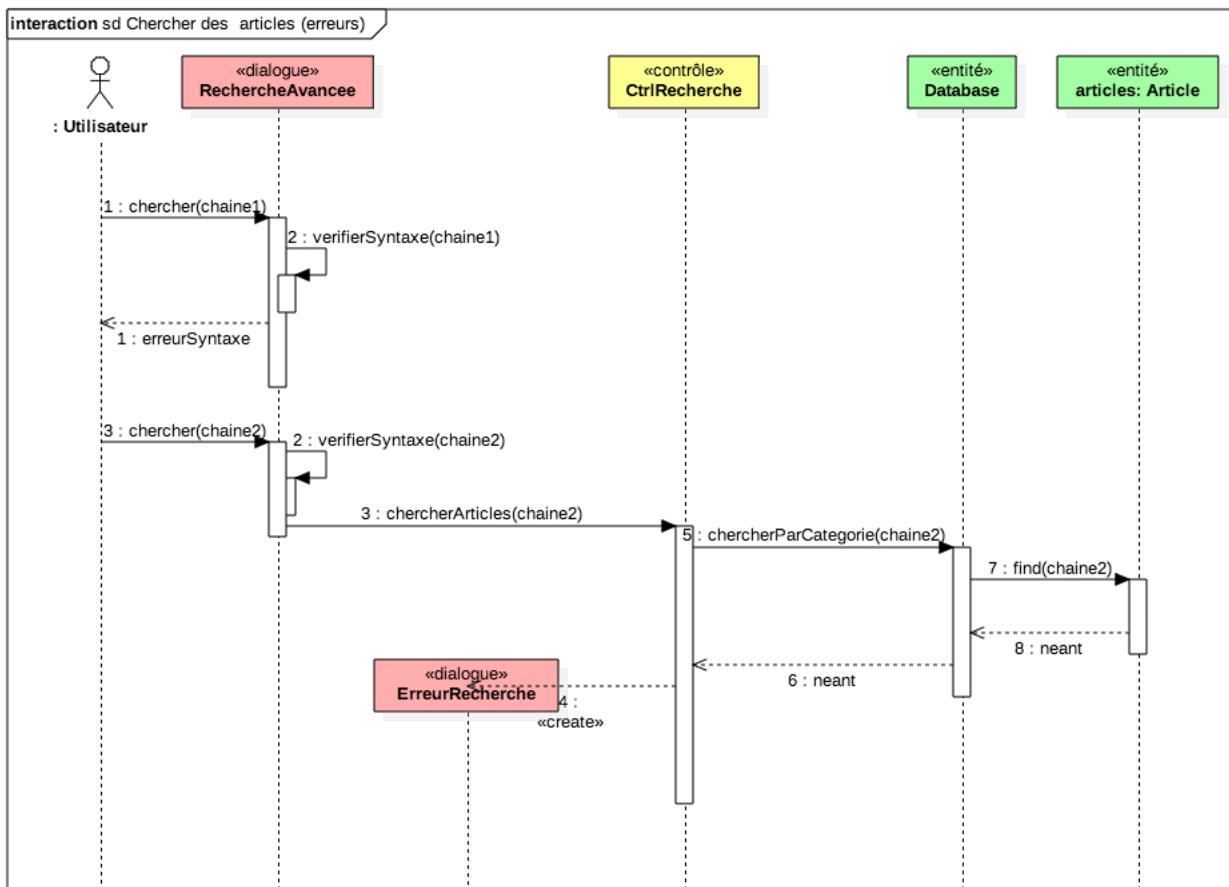


FIGURE 10.2.8 – Diagramme de séquence ”Recherche article”(erreurs)

Ce dernier diagramme s’arrête sur les erreurs qui peuvent advenir pendant la recherche. Il aurait été possible de les faire figurer sur les autres diagrammes mais cela aurait été trop peu lisible.

#### Amélioration du gestionnaire de la base de données

On peut maintenant enrichir la classe Database avec les méthodes découvertes pendant la réflexion sur l’UC ”Recherche article”.



FIGURE 10.2.9 – Gestionnaire de la base de données amélioré

### Diagramme de classes

Ce schéma est la vision orientée objet statique de l'UC "Recherche article". Il reprend l'ensemble des méthodes trouvées et permet de passer à l'écriture du code.

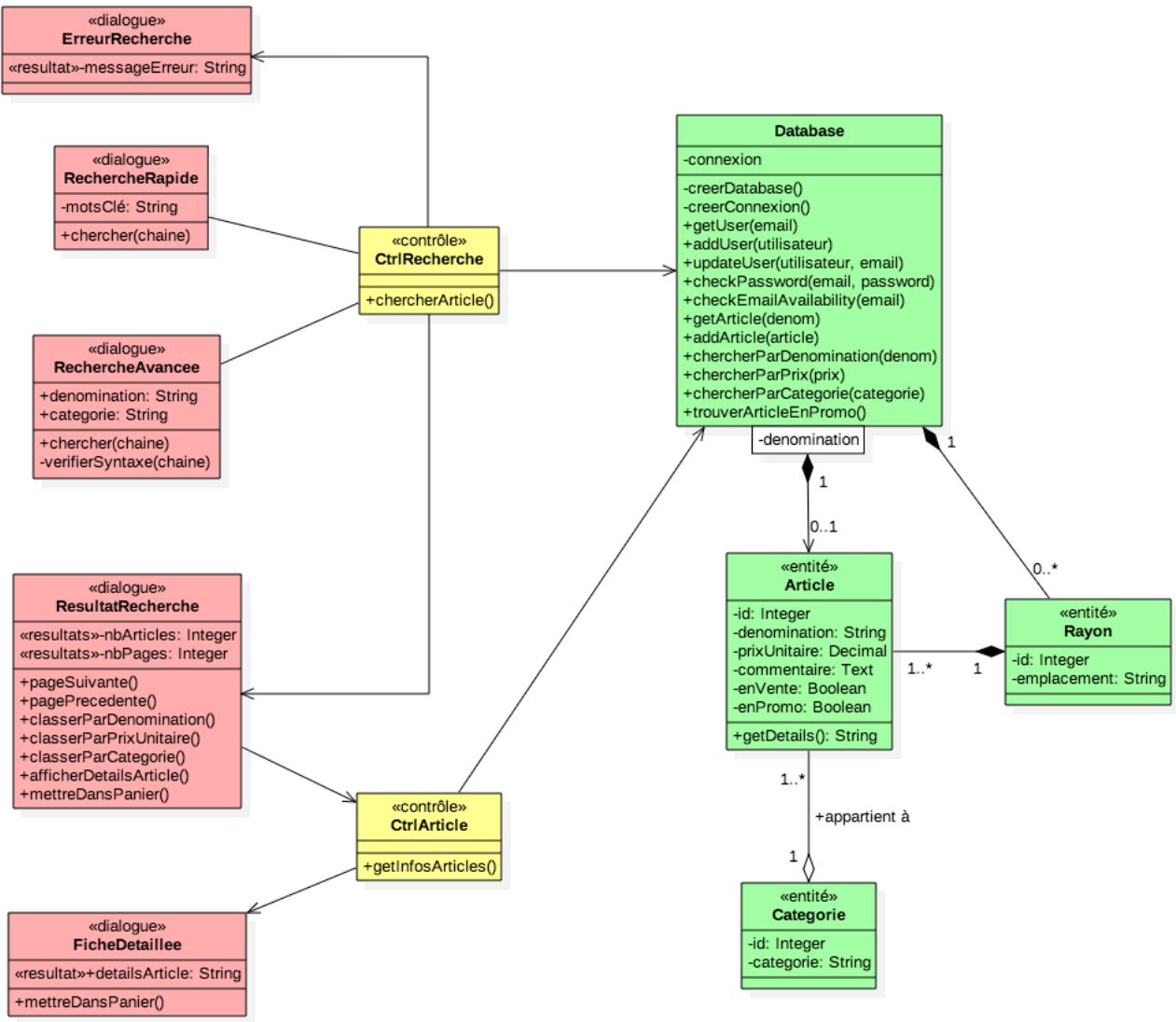


FIGURE 10.2.10 – Diagramme de classes "Recherche article"

#### 10.2.4 Implémentation

##### Tests unitaires

Les tests sont semblables dans la démarche à ceux mis en place pour les UC précédents. Exemple de tests de la classe du modèle Article :

```

72 /**
73  * @covers Article::getDenomination
74  * @todo Implement testGetDenomination().
75 */
76 public function testGetDenomination()
77 {
78     $this->rayon = new Rayon("porte 1");
79     $this->article = new Article("poire", 150, "Poire bio de Wallonie", 1, 1, $this->rayon);
80     $this->assertEquals("poire", $this->article->getDenomination(), "Devrait afficher poire");
81 }
82
83 /**
84  * @covers Article::setDenomination
85  * @todo Implement testSetDenomination().
86 */
87 public function testSetDenomination()
88 {
89     // tests du setter
90     // test de l'exception
91     $this->rayon = new Rayon("porte 1");
92     $this->article = new Article("poire", 150, "Poire bio de Wallonie", 1, 1, $this->rayon);
93     try{
94         $this->article->setDenomination("NouvellePoire");
95     }catch(ArticleException $ue){
96
97         $this->fail( "n'aurait pas dû lancer une exception." .->.$ue);
98     }
99
100    $this->assertEquals($this->article->getDenomination(), "NouvellePoire", "devrait afficher NouvellePoire");
101
102 }

```

FIGURE 10.2.11 – Test unitaire de la méthode getDenomination() et setDenomination()

## Code

**Contrôleur CtrlRecherche** Voici le choix d’implémentation du contrôleur CtrlRecherche. Il correspond à une vision en PHP des diagrammes d’interactions présentés.

```

1  <?php
2  include_once("actions/Action.inc.php");
3  include_once(__DIR__."/../controlers/CtrlRecherche.class.php");
4
5  class RechercheRapideAction extends Action
6  {
7      use CtrlRecherche;
8      private $mot;
9      private $articles;
10
11     /**
12      * @see Action::run()
13      */
14     public function run()
15     {
16         $search_html = filter_input(INPUT_POST, 'article', FILTER_SANITIZE_SPECIAL_CHARS);
17         if($search_html!=null AND $search_html!=false){
18             $this->setMot($search_html);
19             $this->setRecherchesArticles($this->database->trouveArticles($this->getMot()));
20             if(count($this->getRecherchesArticles())==0){
21                 $this->setMessageView("Aucun article ne correspond à votre recherche", "alert-danger");
22             }else{
23                 $this->setView(getViewByName("RechercheRapide"));
24                 $this->getView()->setRecherche($this->getRecherchesArticles());
25             }
26         }else{
27             $this->setView(getViewByName("RechercheRapide"));
28         }
29     }

```

FIGURE 10.2.12 – Implémentation de la classe contrôleur CtrlRecherche

Veuillez noter l'utilisation de la méthode setMessageView pour réaliser la vue MessageError et Message. Rien n'oblige le programmeur à suivre de manière littérale l'exigence de l'analyste pour autant que sa solution suit la vision du logiciel proposée.

### 10.2.5 Difficultés rencontrées, réflexions et conclusions provisoires

- Il existe des règles communément acceptées par la communauté des développeurs pour implémenter un diagramme de classe UML dans un langage orienté objet. La relation de composition entre la classe Article et la classe Rayon se traduit par l'intégration au sein du code de la classe Article d'une classe nichée Rayon<sup>20</sup>. Le langage PHP n'offre pas une telle possibilité d'implémentation mais rien n'empêche d'implémenter à la manière d'une relation d'agrégation ou de relation avec une navigabilité à sens unique ces deux classes. C'est ce dernier choix qui a été réalisé. A la ligne 31, on voit un attribut privé \$\_rayon. Le constructeur à la ligne 46 reçoit en dernier paramètre une instance de la classe Rayon. Il s'agit d'une manière d'implémenter la relation avec navigabilité unique entre deux classes.

20. Charroux B., Osmani A., Thierry-Mieg Y. , UML2 Pratique de la modélisation, Pearson Synthex 3ème édition, 2010, p. 266

```

27    /**
28     *
29     * @var string : le rayon où se trouve l'article.
30     */
31    private $_rayon;
32    /**
33     *
34     * @var String : la catégorie de l'article.
35     */
36    private $_categorie;
37
38    /**
39     * @param string $denomination : Dénomination de l'article.
40     * @param int $prixUnitaire : Prix unitaire de l'article.
41     * @param string $commentaire : Commentaire sur l'article.
42     * @param bool $enVente : Etat de vente de l'article.
43     * @param bool $enPromo : Etat de promotion de l'article.
44     * @param Rayon $rayon : rayon de l'article.
45     */
46    public function __construct($denomination, $prixUnitaire, $commentaire, $enVente, $enPromo, Rayon $rayon) {
47        $this->_id = 0;
48        $this->_denomination = $denomination;
49        $this->_prixUnitaire = $prixUnitaire;
50        $this->_commentaire = $commentaire;
51        $this->_enVente = (int)$enVente;
52        $this->_enPromo= (int)$enPromo;
53        $this->_rayon=$rayon;
54    }

```

FIGURE 10.2.13 – Classe Article

- Le template trouver.inc.php de la vue RechercheRapideView.inc.php fut sans doute le plus compliqué à réaliser. Cela tenait à la subtilité de l'insertion du langage PHP dans le langage HTML de la vue. Il fallait faire en sorte que les articles présentés soient sélectionnables pour remplir le panier (Voir UC Gérer Panier). La notation dans la balise input name="articles['.\$article->getDenomination().']" signifie qu'une collection nommée "article" sera envoyée par une requête POST du protocole HTML au serveur. La dénomination de l'article servira d'identifiant pour le code côté serveur du panier.

```

17 <?php
18 if ($this->getRecherche() != null) {
19     foreach ( $this->getRecherche() as $article ) {
20         if ($article->isVente()) {
21             echo '<tr>';
22             echo '<td>';
23                 <div class="checkbox">
24                     <label>
25                         <input type="checkbox" name="articles['.$article->getDenomination().']" value='.$article->getDenomination().'>
26                     </label>
27                 </div>
28             </td>';
29             echo '<td>';
30                 
31             echo '<td id="idDenomination" name="denomination[' . $article->getDenomination().']">
32                 '.$article->getDenomination().'
33             </td>';
34             echo '<td id="idPrixUnitaire" name="prixUnitaire[' . $article->getDenomination().']">
35                 ' . number_format($article->getPrixUnitaire() / 100, 2) . ' &euro;
36             </td>';
37             echo '<td id="idCommentaire" >' . $article->getCommentaire() . '</td>';
38             echo '<td id="idNbArticles" name="check"><div class="form-group"><label for="nbArticles"></label>
39                 <select class="form-control" name="nbArticles[' . $article->getDenomination() . ']' id="nbArticles">';
40             for($i = 1; $i <= 100; $i ++) {
41                 echo '<option value="' . $i . '">' . $i . '</option>';
42             }
43             echo '</select></div></td>';
44             echo '</tr>';
45         }
46     }
47 }
48 ?>
```

FIGURE 10.2.14 – Template trouver.inc.php

A la ligne 40, une boucle se charge d'itérer de 1 à 100 pour proposer de sélectionner jusqu'à 100 articles identiques.

- Il est accepté dans le domaine de l'informatique de gestion que l'utilisation des types double ou float (en général tout calcul utilisant des virgules flottantes) doit être évitée pour des raisons de précisions et d'accumulation d'erreurs d'arrondis<sup>21</sup>. Java et C# utilisent respectivement la classe BigDecimal et le type decimal pour palier à ce problème. Après avoir vérifier sur le web les solutions proposées, j'ai décidé d'utiliser le type int et de travailler avec des centimes. Ensuite, lors de la présentation des données, la fonction number\_format avec une précision de 2 est utilisée pour manier les valeurs monétaires (ligne 35). Le nombre est divisé par 100 avant d'être traité. La division est entière et la fonction se charge de présenter l'information comme s'il s'agissait d'un nombre à virgule. L'intégrité des données est ainsi garantie.

## 10.3 UC "Gérer son panier"

### 10.3.1 Diagramme des classes participantes (DCP)

#### Réflexions sur le UC

Il s'agit de permettre à l'utilisateur de sélectionner les articles et de les introduire dans son panier comme le ferait un client d'un supermarché. Comme mentionné dans la spécification détaillées des exigences, il faut aussi pouvoir ajouter d'autres articles et pouvoir en retirer avant de passer commande. Nous allons utiliser les entités Panier et LignePanier. Une instance d'une lignePanier contient une certaine quantité d'articles identiques. Nous allons donc créer une contrôleur nommé CtrlPanier qui contiendra les méthodes permettant d'ajouter, supprimer, modifier une ligne. On peut aussi vider le panier instantanément et le recalculer après une modification éventuelle. De plus, il sera possible d'imprimer un devis à l'écran. Le dialogue correspond au contrôleur. La classe GestionPanier contient un attribut "quantité" qui aura une valeur correspondante à la taille de la liste contenant les lignes du panier.

21. <https://stackoverflow.com/questions/3730019/why-not-use-double-or-float-to-represent-currency>

## Diagramme des classes participantes

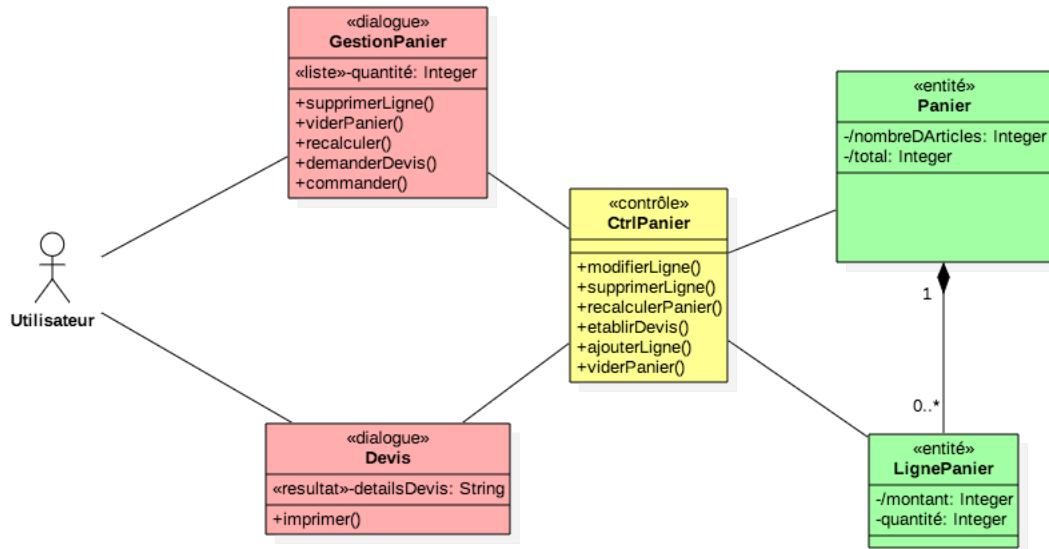


FIGURE 10.3.1 – Diagramme des classes participantes "Gérer son panier"

## 10.3.2 Diagramme de navigation

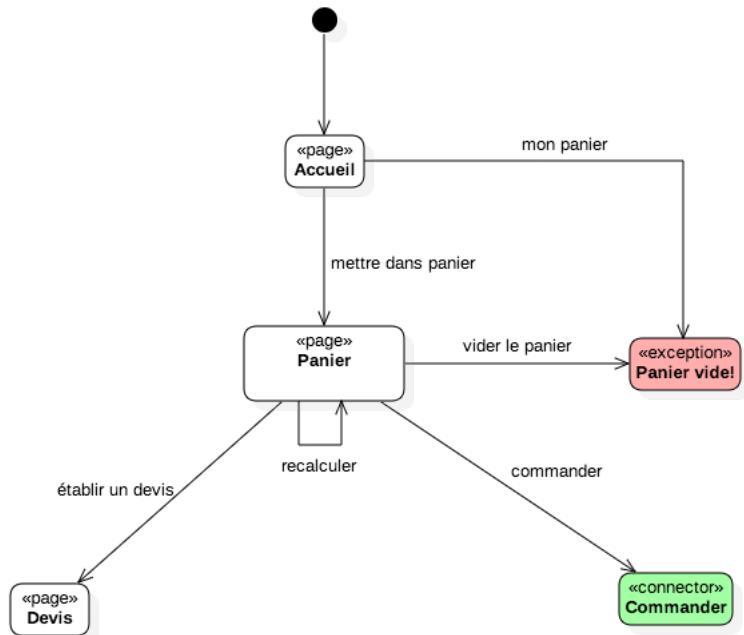


FIGURE 10.3.2 – Diagramme de navigation "Gérer son panier"

### 10.3.3 Conception objet préliminaire

## Diagrammes d'interactions

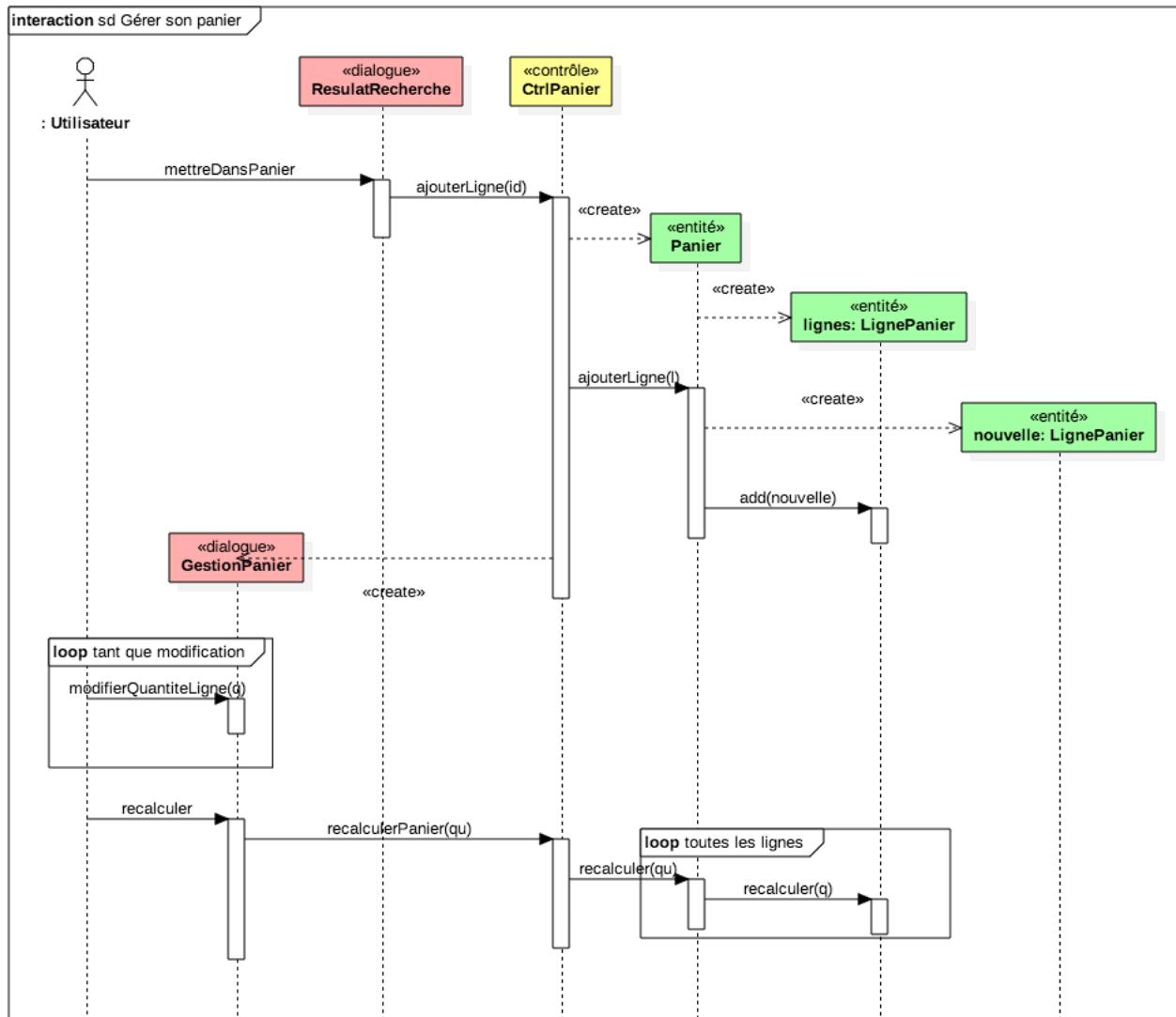


FIGURE 10.3.3 – Diagramme de séquence "Gérer son panier"

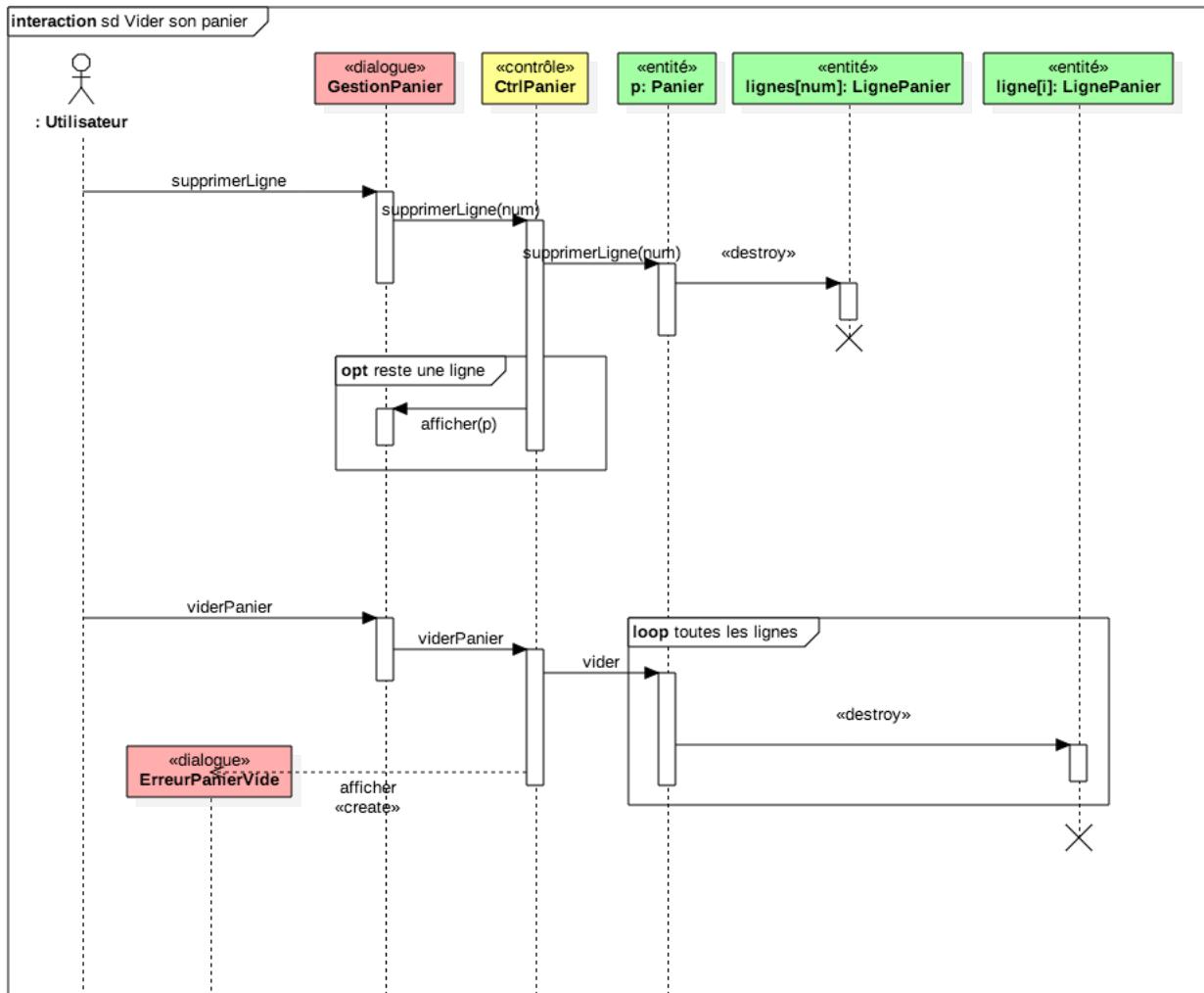


FIGURE 10.3.4 – Diagramme de séquence "Vider son panier"

### 10.3.4 Conception objet détaillée

Diagramme de classes

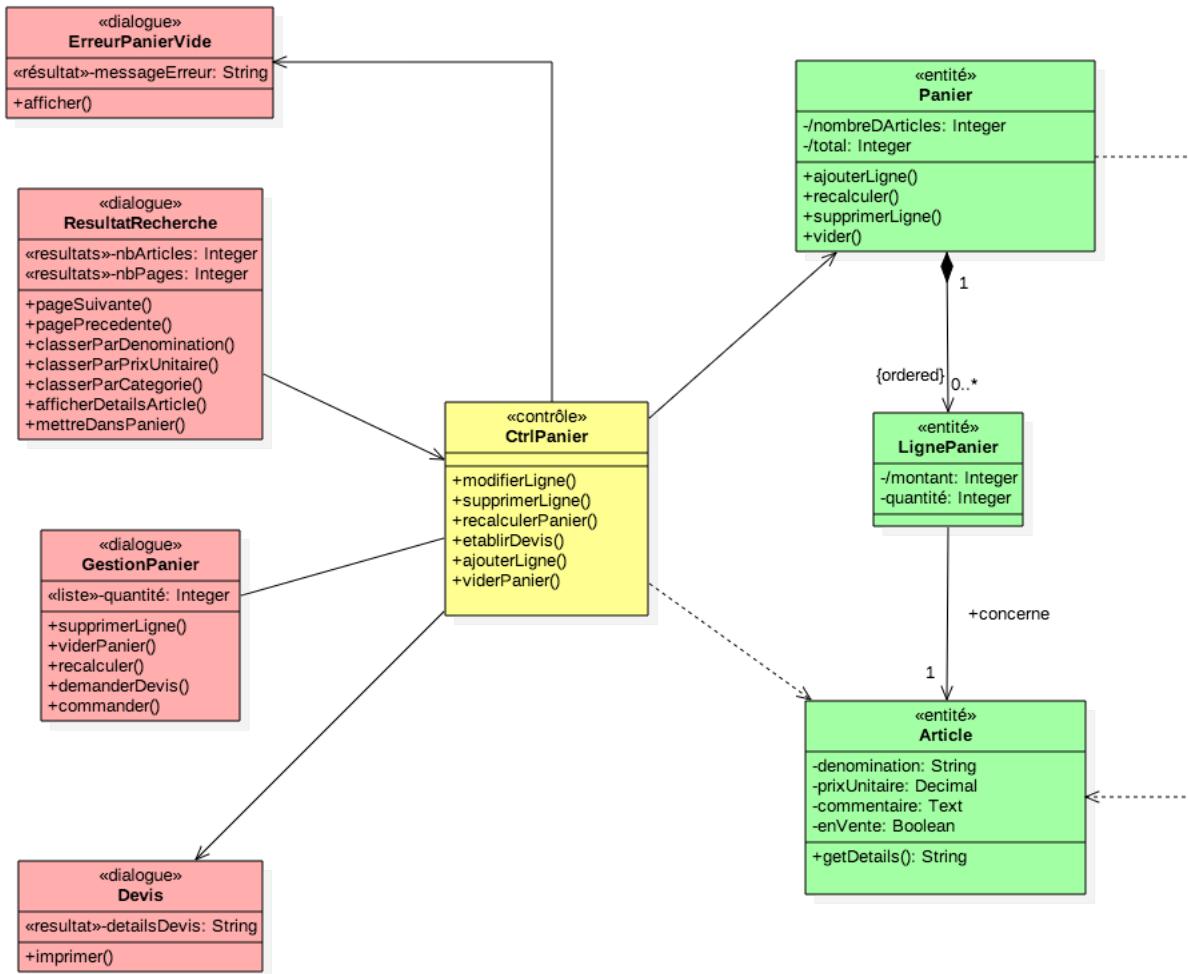


FIGURE 10.3.5 – Diagramme de classe "Gérer son panier"

### 10.3.5 Tests unitaires

```

38  /**
39   * @covers Panier::ajouterLigne()
40   * @covers Panier::getLignePanier()
41   */
42  public function testAjouterLigne()
43  {
44      $rayon = new Rayon("porte 1");
45      $article = new Article("poire", 120, "poire bio de Flandre", 1, 0, $rayon);
46      $ligne = new LignePanier($article, 2);
47      $this->panier->ajouterLigne($ligne);
48      $this->assertEquals($this->panier->getLignePanier(0), $ligne);
49      $ligne=null;
50      $article=null;
51  }
52
53  /**
54   * @covers Panier::retirerLigne()
55   * @covers Panier::ajouterLigne()
56   * @covers Panier::getLignePanier()
57   */
58  public function testRetirerLigne()
59  {
60      $rayon = new Rayon("porte 1");
61      $article = new Article("poire", 120, "poire bio de Flandre", 1, 0, $rayon);
62      $ligne = new LignePanier($article, 2);
63      $this->panier->ajouterLigne($ligne);
64      $this->assertEquals($this->panier->getLignePanier(0), $ligne);
65      $this->panier->retirerLigne(0);
66      $this->assertEquals($this->panier->getLignePanier(0), null);
67  }

```

FIGURE 10.3.6 – Tests unitaires des méthodes AjouterLigne() et RetirerLigne()

Ces tests unitaires correspondent à la classe entité Panier.

### 10.3.6 Code

**Entité** La classe Panier ne présente pas de difficultés particulières. Elle s'implémente naturellement en comportant un attribut de classe nommé `$_lignes` qui est une collection d'objets `LignePanier`.

```

93  /**
94   * Ajouter une ligne au panier.
95   * @param LignePanier $ligne
96   */
97  public function ajouterLigne(LignePanier $ligne) {
98      array_push($this->$_lignes, $ligne);
99      $this->_nbLignes++;
100 }

```

FIGURE 10.3.7 – Méthode ajouterLigne de la classe LignePanier

**Contrôleur** La méthode `ajouterLigne()` de la classe `CtrlAction` :

```

34 }
35 /**
36 * Ajoute une ligne en réaction à l'action de l'utilisateur.
37 * @param $id int : le numéro de la ligne.
38 */
39 public function ajouterLigne($id){
40     foreach($_POST["articles"] as $key=>$value){
41         $this->getPanier()->ajouterLigne(
42             new LignePanier(
43                 $this->database->getArticle($_POST["articles"][$key]) ,
44                 $_POST["nbArticles"][$key]));
45     }

```

FIGURE 10.3.8 – Méthode ajouterLigne() de la classe CtrlPanier

Cette méthode du contrôleur va ajouter une ligne au panier directement à partir de la variable superglobale `$_POST` représentant les articles.

### 10.3.7 Difficultés rencontrées, réflexions et conclusions provisoires

- Le panier doit être enregistré à chaque fois dans la base de données au lieu de la variable `$_POST`. Pour procéder de la sorte, j'ai décidé de créer deux méthodes `setPanierCourant()` prenant en paramètres un panier et un identifiant de session et `getPanierCourant()` récupérant le panier.
- Un choix d'implémentation a dû être opéré au niveau de l'utilisation du langage javascript pour réaliser le panier. Il aurait été beaucoup plus agréable pour l'utilisateur de posséder une panier se mettant à jour sans devoir rafraîchir la page. Cependant, l'idée générale de l'application étant de créer un backend pour une application Android ou AngularJS, il n'a pas été jugé utile d'aller aussi loin pour l'application. On peut à tout moment demander quelle est la valeur d'un panier enregistré dans le back-end à l'aide de la méthode `getJsonData()` :

```

121 /**
122 * @return string : retourne une représentation JSON d'un panier.
123 */
124 function getJsonData() {
125     $var = get_object_vars($this);
126     foreach ( $var as &$value ) {
127         if (is_object($value) && method_exists($value, 'getJsonData')) {
128             $value = $value->getJsonData();
129         }
130     }
131     return $var;
132 }

```

FIGURE 10.3.9 – Méthode GetJsonData() de la classe Panier

Cette méthode récursive va récupérer chaque champs des attributs de l'objet et retrouver la valeur qui lui est associée, tout cela dynamiquement de manière à effectuer une sérialisation sous format JSON de l'objet. Une fois sous cette forme, l'objet, tout en conservant son état, peut être envoyé à une application cliente. La méthode "get\_object\_var()" cherche les propriétés de l'objet donnée, en l'occurrence l'instanciation de la classe possédant la méthode elle-même. La boucle à la ligne 126 va, pour chaque valeur des propriétés, recommencer selon la méthode récursive la récupération des propriétés de chaque propriétés si ces propriétés sont un objet et sont contenu effectivement dans la classe.

## 10.4 UC "Commander"

### 10.4.1 Diagramme des classes participantes (DCP)

#### Réflexions sur le UC

L'utilisateur peut à tout moment effectuer une commande après avoir rempli son panier. Il aura un récapitulatif et ensuite pourra valider sa commande. Il n'aura plus qu'à aller rechercher ce panier chez Natural Corner.

## Diagramme des classes participantes

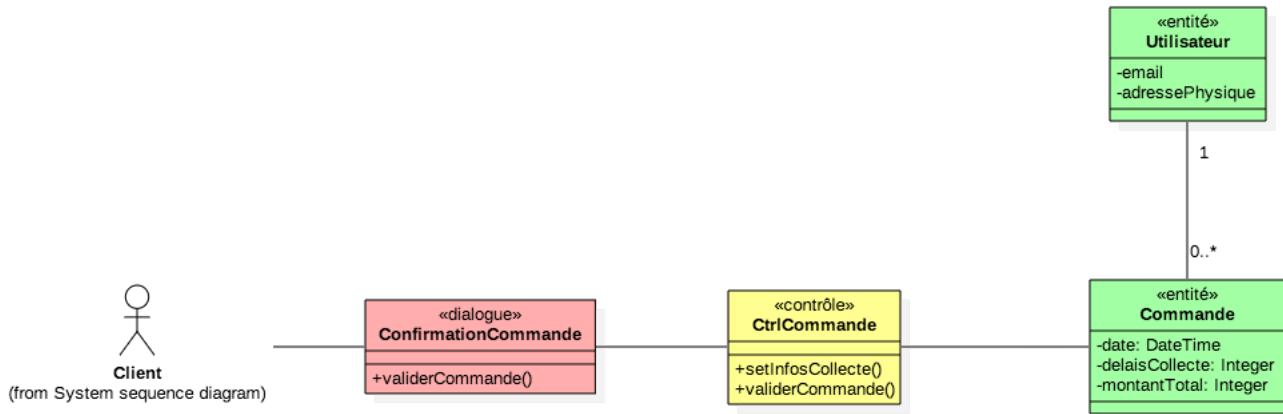


FIGURE 10.4.1 – Diagramme de classes participante ”Commander”

## 10.4.2 Diagramme de navigation

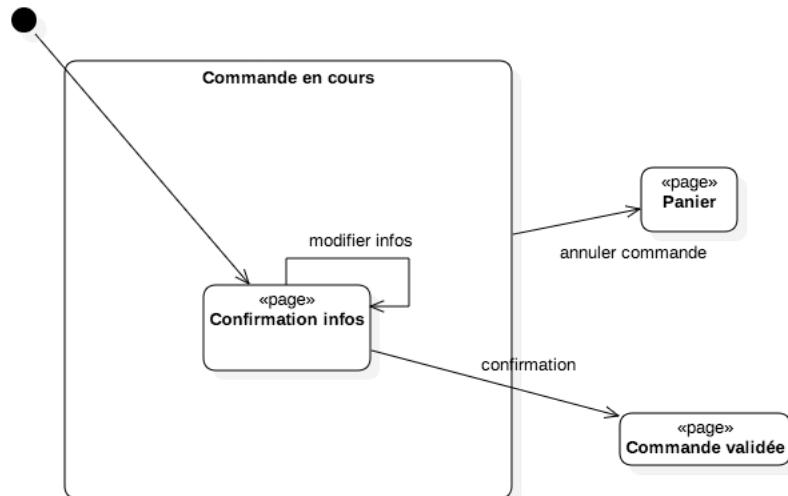


FIGURE 10.4.2 – Diagramme de navigation

### 10.4.3 Conception objet préliminaire

#### Diagrammes d'états

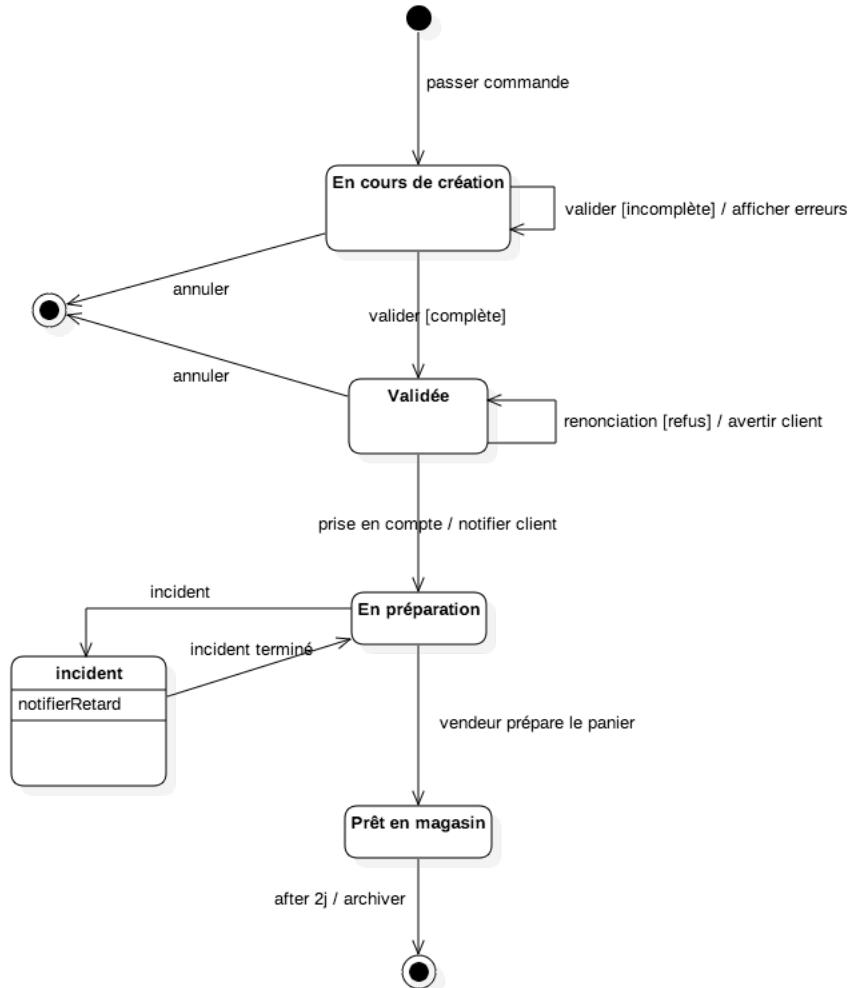


FIGURE 10.4.3 – Diagramme d'état de l'UC "Commander"

#### 10.4.4 Conception objet détaillée

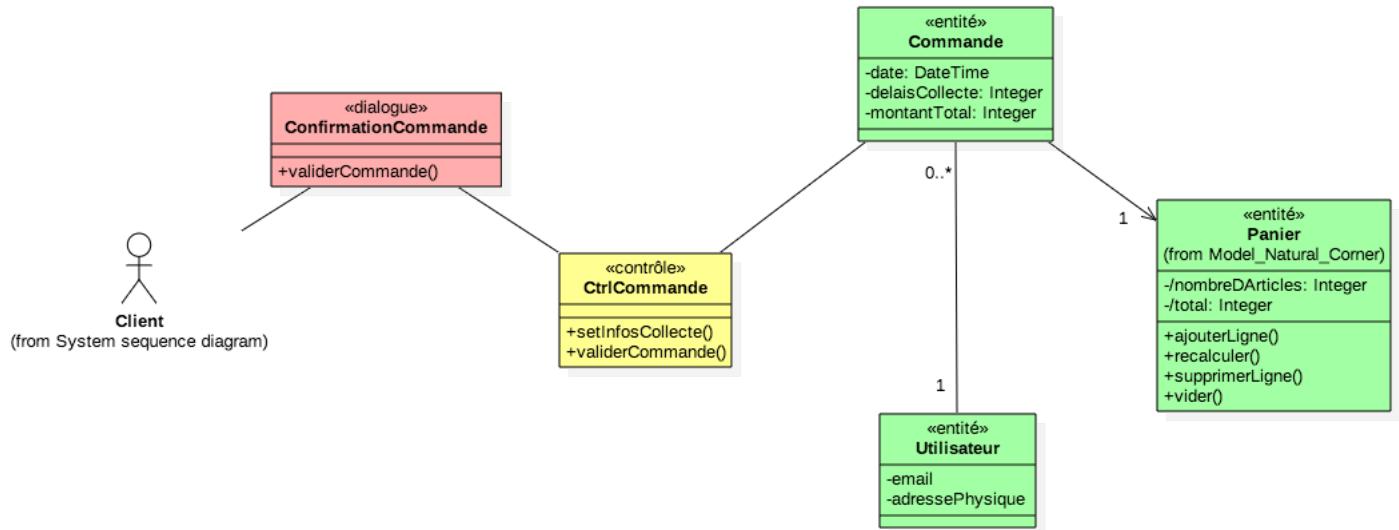


FIGURE 10.4.4 – Diagramme de classes

### 10.5 UC "Maintenir catalogue"

#### 10.5.1 Diagramme des classes participantes (DCP)

##### Réflexions sur le UC

Le gérant doit avoir la possibilité de créer un article, de le modifier et de le supprimer. Il doit de plus être capable de dire s'il est en promo ou disponible. Il doit aussi pouvoir créer un rayon et une catégorie. Pour ce faire, nous allons créer une page qui fera office de control panel dans laquelle nous trouverons une navigation tabulaire permettant de modifier des articles, des rayons et des catégories.

## Diagramme des classes participantes

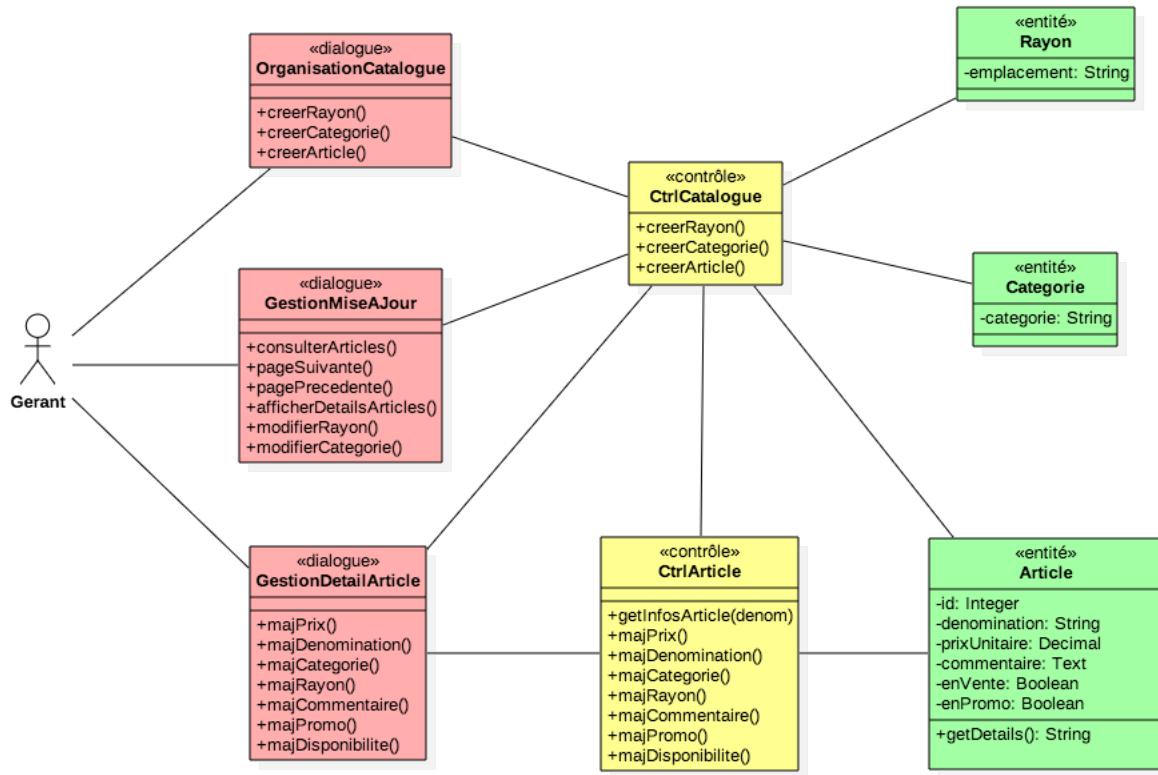


FIGURE 10.5.1 – Diagramme des classes participantes “Maintenir catalogue”

### 10.5.2 Diagramme de navigation

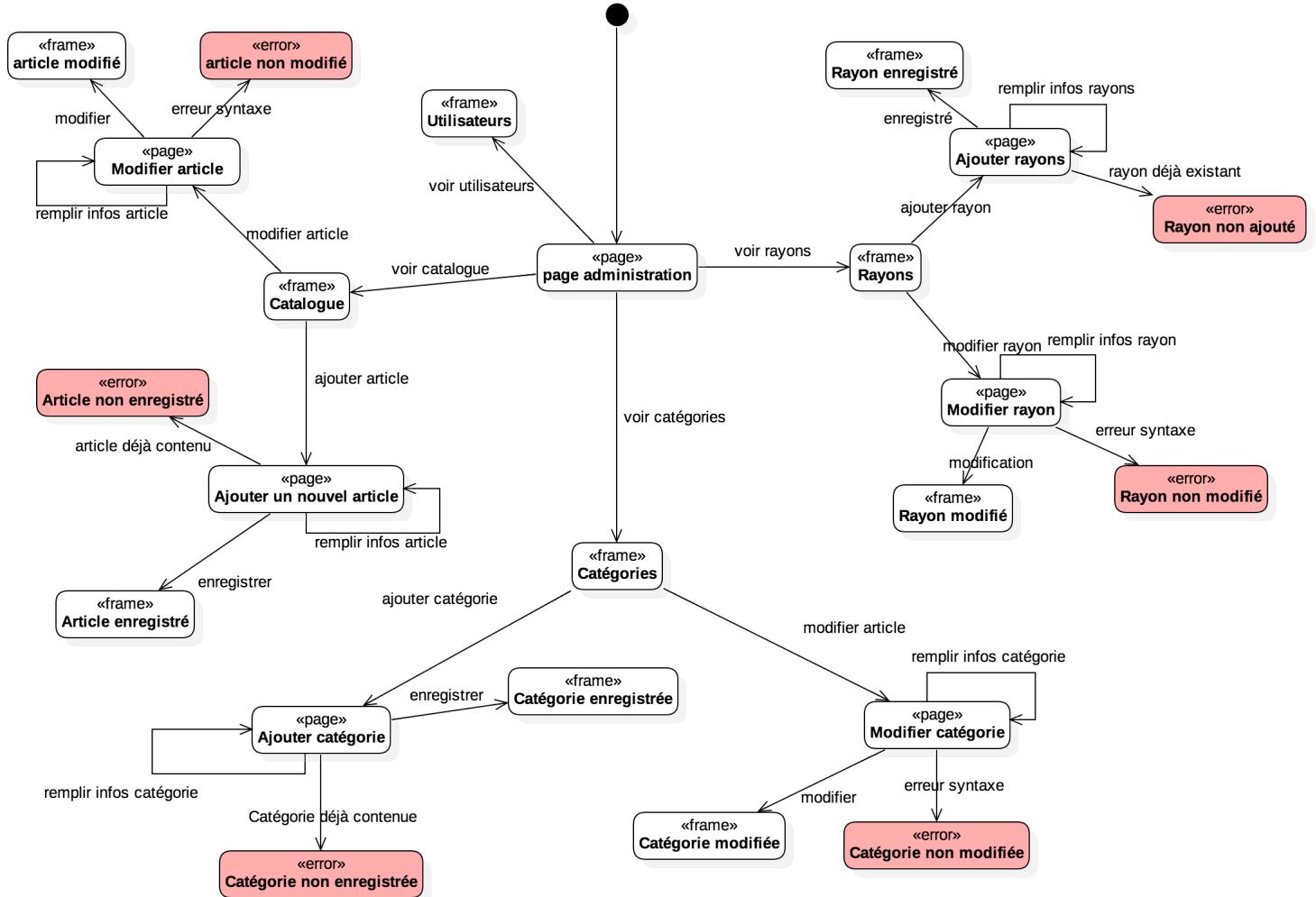


FIGURE 10.5.2 – Diagramme de navigation "Maintenir catalogue"

### 10.5.3 Conception objet préliminaire

#### Diagrammes d'interactions

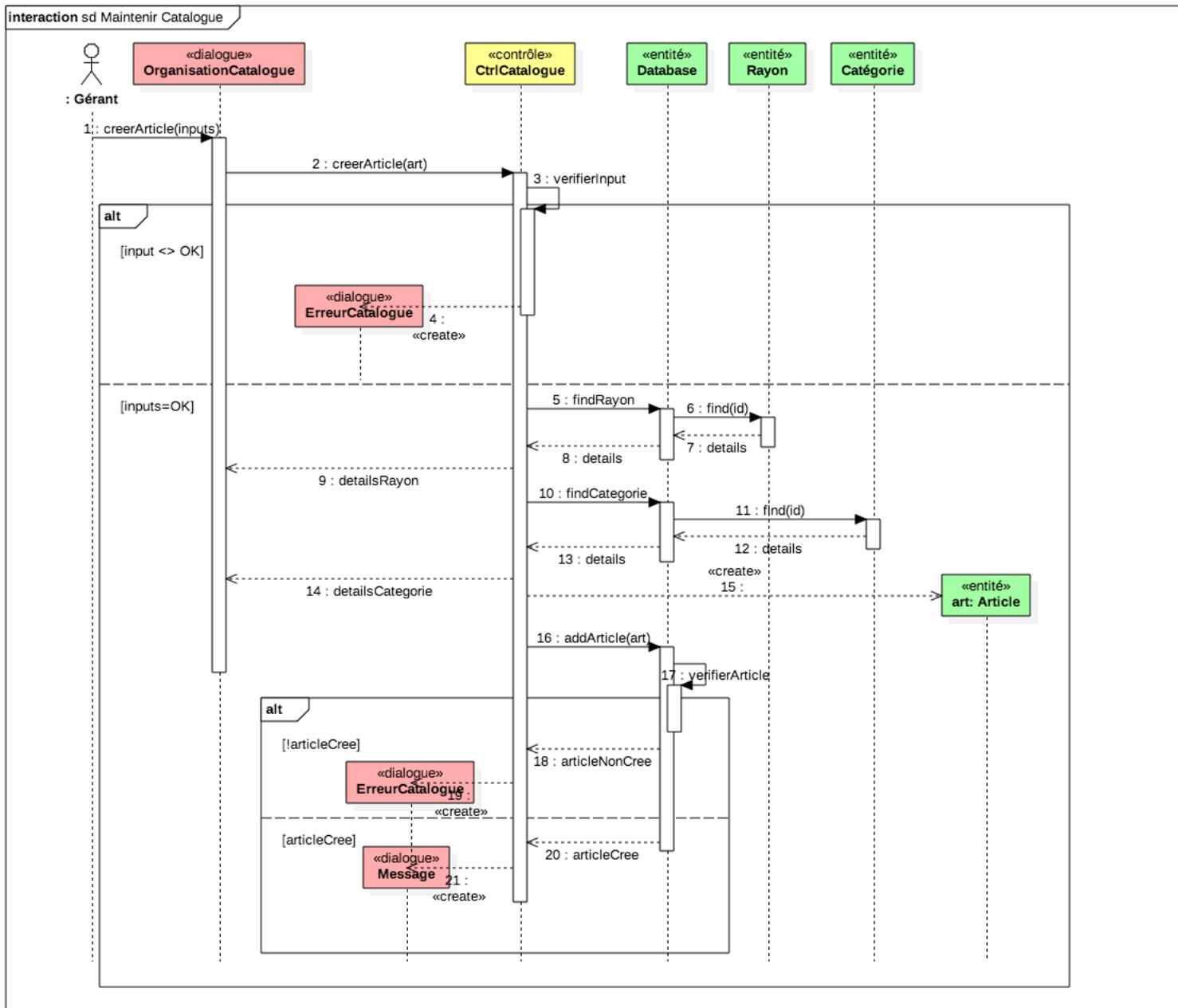


FIGURE 10.5.3 – Diagramme d’interactions ”Maintenir catalogue”

#### Amélioration du gestionnaire de la base de données

Le gestionnaire de la base de données est enrichi des méthodes correspondant au CRUD des articles

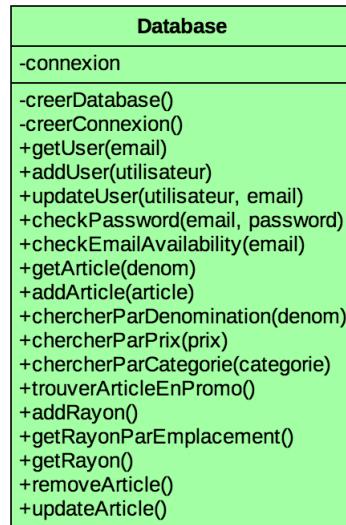


FIGURE 10.5.4 – Gestionnaire de la base de données

#### 10.5.4 Tests unitaires

On peut montrer par exemple les tests concernant le CRUD des articles.

```

420 /**
421 * @depends testCreateDatabase
422 * @covers Database::getArticle()
423 * @covers Database::addArticle()
424 */
425 public function testUpdateArticle() {
426     $test_increment = rand();
427     $emplacement = "porte test ".$test_increment;
428     $rayon = new Rayon($emplacement);
429     self::$bdd->addRayon($rayon);
430     $rayonRecupere = self::$bdd->getRayonParEmplacement($emplacement);
431
432     $article1 = new Article("article1" . ++$test_increment, 100, "commentaire1", 1, 0, $rayonRecupere);
433     $insertionArticle1 = self::$bdd->addArticle($article1);
434     // vérification.
435     $this->assertEquals($article1->getCommentaire(), self::$bdd->getArticle($article1->getDenomination())->getCommentaire());
436     // modification du commentaire.
437     $article1->setCommentaire("nouveauCommentaire1");
438     try {
439         self::$bdd->updateArticle($article1, $article1->getDenomination());
440     } catch ( ArticleException $ae ) {
441         return;
442     }
443
444     $this->assertEquals("nouveauCommentaire1",
445         self::$bdd->getArticle($article1->getDenomination())->getCommentaire(),
446         "Aurait dû afficher nouveauCommentaire1");
447 }
  
```

FIGURE 10.5.5 – Test unitaire de la méthode updateArticle

On crée une valeur aléatoire(ligne 426) pour la concaténer aux chaînes de caractères représentant l'article. A la ligne 439, on retrouve la méthode updateArticle.

### 10.5.5 Code

Nous allons uniquement illustrer le chapitre avec le code du logiciel étant donné que la logique

```

32     if($form_data['denomination']!==null){
33         // un signe ! pour la disponibilité.
34         $rayon = $this->database->getRayon(1);
35         $nouvelArticle = new Article($form_data['denomination'], $form_data['prix'],
36             $form_data['commentaire'], isset($form_data['disponibilite'])? 0:1, 0, $rayon);
37         try{
38             $estInsere = $this->database->addArticle($nouvelArticle);
39         }catch(ArticleException $eate){
40             $estInsere=false;
41             $insertionMessage = $eate->getMessage();
42         }
43         if($estInsere){
44             $this->setView(getViewByName("Catalogue"));
45             $this->setMessageView("Article correctement inséré!", "alert-success");
46         }else{
47             $this->setView(getViewByName("Catalogue"));
48             $this->setMessageView("Article non inséré!", "alert-danger");
49         }
50     }else{
51         $this->setView(getViewByName("Default"));
52         $this->setMessageView("Veuillez introduire une dénomination pour l'article à insérer.", "alert-danger");
53     }
54 }else{
55     $this->setView(getViewByName("Catalogue"));
56 }
```

FIGURE 10.5.6 – Contrôleur CtrlCatalogue

**Le contrôleur CtrlCatalogue** A la ligne 36, l'opérateur ternaire est utilisé pour indiquer à la manière d'un booléan (un integer 0 ou 1 en réalité) si l'article est disponible ou non selon l'input du gérant.

```

464 /**
465 * Ajoute dans la base de donnée un article.
466 * @param Article l'article à insérer.
467 * @return bool indique si l'insertion est réussie.
468 * @throws ArticleException si l'email existe déjà dans la base de données.
469 */
470 public function addArticle(Article $article){
471     $insertionReussie=false;
472     $this->creerConnexion();
473     $this->connection->beginTransaction();
474     $requete = $this->connection->prepare(" INSERT INTO ARTICLES
475                                         DENOMINATION, PRIX_UNITAIRE,
476                                         COMMENTAIRE, EN_VENTE, EN_PROMO, ID_RAYON)
477                                         VALUES(
478                                         :DENOMINATION, :PRIX_UNITAIRE,
479                                         :COMMENTAIRE, :EN_VENTE, :EN_PROMO, :ID_RAYON)");
480     $insertionReussie = $requete->execute(array(
481         ':DENOMINATION'=>$article->getDenomination(),
482         ':PRIX_UNITAIRE'=>$article->getPrixUnitaire(),
483         ':COMMENTAIRE'=>$article->getCommentaire(),
484         ':EN_VENTE'=>$article->isEnVente(),
485         ':EN_PROMO'=>$article->isEnPromo(),
486         ':ID_RAYON'=>$article->getRayon()->getId()
487     ));
488     try{
489         $article->setId($this->connection->lastInsertId());
490     }catch(UtilisateurException $ue){
491         echo $ue->getMessage();
492     }
493     if($insertionReussie)
494         $this->connection->commit();
495     else
496         $this->connection->rollBack();
497     $requete->closeCursor();
498     return $insertionReussie;
499 }

```

FIGURE 10.5.7 – Méthode addArticle de la classe Database

**Méthode addArticle de la classe entité Database** L'article est introduit de manière classique. Aux lignes 473, 494 et 496, les méthodes beginTransaction(), commit() et rollback() sont explicitement appelées pour assurer le caractère ACID de la base de données lorsqu'on veut récupérer l'id de la table Article dans la base de données. Après avoir été rendu persistant, l'objet article reçoit l'id qui sera utile pour l'application (notamment pour la catégorie de l'article). Un objet en PHP étant introduit par référence dans une méthode, l'objet garde le nouvel id après l'enregistrement dans la base de données.

## Chapitre 11

# Création de l'API et utilisation de AngularJS

### 11.1 API REST

L'API REST utilisera le format JSON. On résume généralement dans un tableau les différents verbes du protocole HTTP en relation avec l'adresse URL associée. Voir par exemple le site du framework PHP PhalconPHP<sup>1</sup>. Quatre verbes correspondent aux quatre méthodes HTTP :

- GET pour chercher les données
- POST pour ajouter des données
- PUT pour modifier des données
- DELETE pour supprimer les données

Méthode	URL	Action
GET	/api/articles	Récupère tous les articles
GET	/api/articles/search/pomme	Cherche les articles ayant "pomme" dans leur nom
GET	/api/users/email@mail.com	Cherche un utilisateur dont l'email correspond
GET	/api/articles/3	Récupère un article à partir de la clé primaire
POST	/api/users	Ajoute un nouvel utilisateur
PUT	/api/users/4	Modifie un utilisateur à partir de la clé primaire
DELETE	/api/commandes/5	Détruire une commande à partir de la clé primaire

Chaque classe reçoit la méthode permettant de sérialiser l'état de l'objet instancié par l'application.

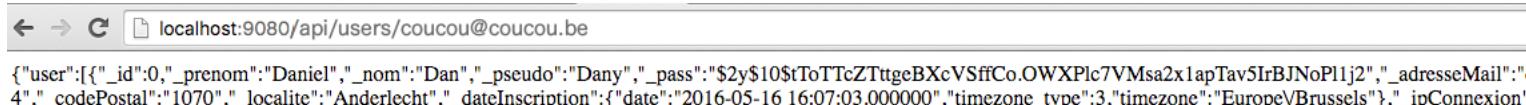


FIGURE 11.1.1 – Récupération d'un objet JSON de type Utilisateur par l'API RESTful

L'application Natural Corner est prête (sous réserve d'un développement ultérieur en accord avec le gérant du magasin) à accueillir une application cliente Android, iOS ou AngularJS.

### 11.2 AngularJS

A titre d'exemple pour montrer le potentiel du back-end créé, voici une solution cliente en AngularJS utilisant l'API REST.

1. <https://docs.phalconphp.com/fr/latest/reference/tutorial-rest.html>

```

124     <script>
125         var val1 = '<?php echo $this->user==null?\'': $this->user->getAdresseMail(); ?>';
126         var app = angular.module('myApp', []);
127         app.controller('customersCtrl', function($scope, $http) {
128             $http.get("<?php echo $_SERVER['PHP_SELF']. '?action=UserJSON'?>",
129                         {params:{'email': val1}}).
130                         then(function(response) {
131                             $scope.myData = response.data.records;
132                         });
133                     });
134     </script>

```

FIGURE 11.2.1 – Code AngularJS du template page.inc.php

A la ligne 128, dans le contrôleur, il y a un appel GET à l'API (l'adresse n'a pas été écrite en utilisant le URL rewriting).

```

3<article>
4    <section ng-app="myApp" ng-controller="customersCtrl" class="col-lg-offset-3 col-lg-8">
5        <form ng-repeat="x in myData" method="post" action="index.php?action=UpdateUser" class="form-horizontal col-lg-10" >
6            <div class="form-group">
7
8                <h3 class="text-center"><span class="label label-success">Modification des informations</span></h3>
9
10
11            </div>
12            <div class="row">
13                <div class="form-group">
14                    <label class="control-label col-lg-4" for="pseudo">Pseudo</label>
15                    <div class="col-lg-6">
16                        <input type="text" name="pseudo" class="form-control" value="{{x._pseudo}}" placeholder=" " pattern="[0-9a-zA-Z]{3,128}" title="alphanumérique ayant au moins trois caractères" >
17                    </div>
18                </div>
19            </div>

```

FIGURE 11.2.2 – Mise en oeuvre de l'API REST avec AngularJS

La ligne 5 contient l'attribut ng-repeat qui est une boucle dans le langage de balise Angularjs. La valeur "x" est l'objet utilisateur et chaque champs de l'objet est représenté par ses noms. Par exemple, à la ligne 15, on voit {{x.\_pseudo}} qui est la valeur du champs nom.

Le lecteur est invité à consulter cette solution sur le Github associé au projet, sur le fork AngularJS : <https://github.com/userdanydan/NaturalCorner/tree/AngularJSVersion>.

Enfin, le code qui permet de créer le flux JSON.

```
1 <?php
2
3 include_once("actions/Action.inc.php");
4
5@ class UserJSONAction extends Action {
6
7@     /**
8@      *
9@      * @see Action::run()
10@     */
11@    public function run() {
12        $this->setView(getViewByName("UserJSON"));
13        if(isset($_GET['email'])){
14            $user = $this->database->getUser($_GET['email']);
15            $record['user'] = array($user->getJsonData());
16            echo json_encode($record);
17        }else{
18            echo '0';
19        }
20    }
21
22 }
```

FIGURE 11.2.3 – classe UserJSON.inc.php

Ce code va simplement aller chercher dans la base de données l'utilisateur entré en paramètre GET par la requête de l'application cliente. A la ligne 16, la méthode json\_encode() renvoie au client l'enregistrement sous forme de format JSON.

# Chapitre 12

## Conclusion

Le développement logiciel traverse une série de transformations depuis quelques années. D'abord l'avènement des technologiques mobiles avec les systèmes d'exploitation Android et iOS. Ensuite l'essor du Cloud computing, technologie initialement lancée par Amazon en 2006<sup>1</sup> pour la communauté des développeurs logiciels, permettant de se passer des infrastructures couteuse d'une société en informatique, enfin, les technologies web se sont nettement améliorées et professionnalisées. Il a été possible grâce à ces avancées récentes de proposer une solution fiable au gérant du magasin Natural Corner. Il va de soi que le projet n'en est qu'à son ébauche et qu'il y a encore beaucoup à faire. L'application frontend basée sur une application Android vient d'être commencée et sera proposée à l'essai lors de la présentation orale de ce présent travail. Son efficacité démontrera par l'exemple la valeur de l'application Natural Corner.

La marché bio est un marché très porteur pour le moment et qui risque bien de prendre de plus grandes proportions dans les années à venir<sup>2</sup>. Il a été particulièrement excitant dans une perspective de carrière de m'intéresser à cette économie à la croisée de l'écologie et du commerce traditionnel, tout en étant résolument 'trendy'. C'est que les nouvelles technologies s'allient parfaitement avec la logique bio. L'idée de travailler en circuit court ne peut trouver meilleur allié objectif dans la commande en ligne des produits frais. Il s'en suit une gestion fine des commandes, une amélioration de la logistique des flux tendus et une réduction des invendus. Cette plongée au coeur du commerce bio de proximité fut une initiation dont je garderai, sans nul doute, une expérience profitable.

Est-il possible de créer en quelque semaines une application "full-stack" ? A mon sens oui, mais cela demande une discipline à deux niveaux. La création de l'application Natural Corner m'a permis de comprendre que, d'abord, l'écriture des tests unitaires sont primordiaux et, ensuite, que l'analyse menant aux diagrammes UML doit être conduite avec rigueur et sérieux. Sans eux, écrire un logiciel revient à vouloir naviguer en pleine nuit sans instruments de géolocalisation et j'ai quelques fois dû revenir sur mes pas pour corriger des erreurs que l'analyse et les tests auraient pu détecter au préalable. Ces petites leçons ont en tout cas permis de me familiariser avec le métier d'analyste-développeur et j'espère que j'aurai la possibilité dans un futur proche d'approfondir ce métier.

Le micro framework doit encore être amélioré. Il le peut en intégrant les API Google App Engine qui regorgent de services très utiles<sup>3</sup> (base données no SQL, email, envois de sms, log par email gmail, cron, taskqueue pour la programmation concurrente, etc...). Une application Android se servant de l'API RESTful est en cours d'élaboration et permettra une expérience pour le client résolument moderne.

Le gérant de Natural Corner effectue pour le moment des démarches pour créer un service de ventes en ligne d'un point de vue logistique. L'avenir nous dira si ce travail aura permis la réalisation du projet initial.

---

1. [https://en.wikipedia.org/wiki/Cloud\\_computing](https://en.wikipedia.org/wiki/Cloud_computing)

2. A en croire les statistiques officiels, l'augmentation est constante depuis une dizaine d'année <http://statbel.fgov.be/fr/statistiques/chiffres/economie/agriculture/biologique/>

3. <https://cloud.google.com/appengine/docs/php/refdocs/namespaces/google>

# Bibliographie

- [1] Charroux B., Osmani A., Thierry-Mieg Y. , UML2 Pratique de la modélisation, Pearson Synthex 3ème édition, 2010
- [2] Gruau C., Conception d'une base de données, Site developpez.com 1ère édition corrigée, 13/7/2006
- [3] Roques, P., UML2 Modéliser une application web, Eyrolles 4ème édition, 2008
- [4] Brouard, F., Bruchez, R., Soutou, C., SQL, Pearson 3ème édition, 2010
- [5] <http://pageperso.lif.univ-mrs.fr/~bertrand.estellon/>, pdf du 1er avril 2014 consulté le 31/3/2016.
- [6] Michael Peacock, PHP 5 Social Networking, Packt Publishing, 2010
- [7] <https://secure.php.net/manual/fr/>
- [8] <http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-005-elements-of-software-construction-fall-2011/lecture-notes>
- [9] Larman, G., Applying UML and Patterns , An introduction to object-oriented analysis and design and unified process, Seconde edition, 2001

## Annexe A

# Dump du code MySQL généré par JMerisse (et corrigé manuellement)

```
#----- # Script MySQL. #-----  
#----- # Table: UTILISATEURS #-----  
CREATE TABLE UTILISATEURS(  
    ID_UTILISATEUR int (11) Auto_increment NOT NULL ,  
    PRENOM Varchar (128) ,  
    NOM Varchar (128) ,  
    PSEUDO Varchar (128) ,  
    PASS Varchar (1028) ,  
    ADRESSE_MAIL Varchar (128) ,  
    DATE_INSCRIPTION Datetime ,  
    IP_CONNEXION Varchar (128) ,  
    RUE Varchar (128) ,  
    NUMERO Varchar (25) ,  
    NUMERO_BOITE Varchar (25) ,  
    CODE_POSTAL Varchar (25) ,  
    LOCALITE Varchar (128) ,  
    PAYS Varchar (25) ,  
    PRIMARY KEY (ID_UTILISATEUR) ,  
    UNIQUE (ADRESSE_MAIL) )ENGINE=InnoDB;  
#----- # Table: COMMANDES #-----  
CREATE TABLE COMMANDES(  
    ID_COMMANDE int (11) Auto_increment NOT NULL ,  
    DATE_COMMANDE Datetime ,  
    ID_UTILISATEUR Int NOT NULL ,  
    ID_PANIER Int NOT NULL ,  
    PRIMARY KEY (ID_COMMANDE) )ENGINE=InnoDB;  
#----- # Table: ARTICLES #-----  
CREATE TABLE ARTICLES(  
    ID_ARTICLES int (11) Auto_increment NOT NULL ,  
    DENOMINATION Varchar (128) ,  
    PRIX_UNITAIRE DECIMAL (15,3) ,  
    COMMENTAIRE Longtext ,  
    EN_VENTE Bool ,  
    EN_PROMO Bool ,  
    ID_RAYONS Int NOT NULL ,  
    PRIMARY KEY (ID_ARTICLES) ,  
    UNIQUE (DENOMINATION) )ENGINE=InnoDB;  
#----- # Table: VENDEURS #-----  
CREATE TABLE VENDEURS(  
    ID_VENDEUR int (11) Auto_increment NOT NULL ,  
    ID_UTILISATEUR Int NOT NULL ,  
    PRIMARY KEY (ID_VENDEUR ,ID_UTILISATEUR) )ENGINE=InnoDB;  
#----- # Table: GERANTS #-----
```

```

CREATE TABLE GERANTS(
    ID_GERANT      int (11) Auto_increment NOT NULL ,
    ID_VENDEUR     Int NOT NULL ,
    ID_UTILISATEUR Int NOT NULL ,
    PRIMARY KEY (ID_GERANT ,ID_VENDEUR ,ID_UTILISATEUR ) )ENGINE=InnoDB;
#----- # Table: RAYONS -----
CREATE TABLE RAYONS(
    ID_RAYONS      int (11) Auto_increment NOT NULL ,
    DENOMINATION   Varchar (128) ,
    PRIMARY KEY (ID_RAYONS ) )ENGINE=InnoDB;
#----- # Table: CATEGORIES -----
CREATE TABLE CATEGORIES(
    ID_CATEGORIE   int (11) Auto_increment NOT NULL ,
    DENOMINATION   Varchar (128) ,
    ID_CATEGORIE_1 Int NOT NULL ,
    PRIMARY KEY (ID_CATEGORIE ) )ENGINE=InnoDB;
#----- # Table: PANIERS -----
CREATE TABLE PANIERS(
    ID_PANIER      int (11) Auto_increment NOT NULL ,
    ID_COMMANDE   Int NOT NULL ,
    PRIMARY KEY (ID_PANIER ) )ENGINE=InnoDB;
#----- # Table: LIGNES_PANIER -----
CREATE TABLE LIGNES_PANIER(
    ID_LIGNE_PAGNIER int (11) Auto_increment NOT NULL ,
    QUANTITE        Int ,
    ID_PANIER       Int NOT NULL ,
    ID_ARTICLES    Int NOT NULL ,
    PRIMARY KEY (ID_LIGNE_PAGNIER ) )ENGINE=InnoDB;
#----- # Table: CARACTERISER -----
CREATE TABLE CARACTERISER(
    ID_ARTICLES   Int NOT NULL ,
    ID_CATEGORIE  Int NOT NULL ,
    PRIMARY KEY (ID_ARTICLES ,ID_CATEGORIE ) )ENGINE=InnoDB;

ALTER TABLE COMMANDES
    ADD CONSTRAINT FK_COMMANDES_ID_UTILISATEUR FOREIGN KEY (ID_UTILISATEUR)
        REFERENCES UTILISATEURS(ID_UTILISATEUR);
ALTER TABLE COMMANDES
    ADD CONSTRAINT FK_COMMANDES_ID_PANIER FOREIGN KEY (ID_PANIER)
        REFERENCES PANIERS(ID_PANIER);
ALTER TABLE ARTICLES
    ADD CONSTRAINT FK_ARTICLES_ID_RAYONS FOREIGN KEY (ID_RAYONS)
        REFERENCES RAYONS(ID_RAYONS);
ALTER TABLE VENDEURS
    ADD CONSTRAINT FK_VENDEURS_ID_UTILISATEUR FOREIGN KEY (ID_UTILISATEUR)
        REFERENCES UTILISATEURS(ID_UTILISATEUR);
ALTER TABLE GERANTS
    ADD CONSTRAINT FK_GERANTS_ID_VENDEUR FOREIGN KEY (ID_VENDEUR)
        REFERENCES VENDEURS(ID_VENDEUR);
ALTER TABLE GERANTS
    ADD CONSTRAINT FK_GERANTS_ID_UTILISATEUR FOREIGN KEY (ID_UTILISATEUR)
        REFERENCES UTILISATEURS(ID_UTILISATEUR);
ALTER TABLE CATEGORIES
    ADD CONSTRAINT FK_CATEGORIES_ID_CATEGORIE_1 FOREIGN KEY (ID_CATEGORIE_1)
        REFERENCES CATEGORIES(ID_CATEGORIE);
ALTER TABLE PANIERS
    ADD CONSTRAINT FK_PANIERS_ID_COMMANDE FOREIGN KEY (ID_COMMANDE)
        REFERENCES COMMANDES(ID_COMMANDE);
ALTER TABLE LIGNES_PANIER
    ADD CONSTRAINT FK_LIGNES_PANIER_ID_PANIER FOREIGN KEY (ID_PANIER)
        REFERENCES PANIERS(ID_PANIER);

```

```
ALTER TABLE LIGNES_PANIER
    ADD CONSTRAINT FK_LIGNES_PANIER_ID_ARTICLES FOREIGN KEY (ID_ARTICLES)
    REFERENCES ARTICLES(ID_ARTICLES);
ALTER TABLE CARACTERISER
    ADD CONSTRAINT FK_CARACTERISER_ID_ARTICLES FOREIGN KEY (ID_ARTICLES)
    REFERENCES ARTICLES(ID_ARTICLES);
ALTER TABLE CARACTERISER
    ADD CONSTRAINT FK_CARACTERISER_ID_CATEGORIE FOREIGN KEY (ID_CATEGORIE)
    REFERENCES CATEGORIES(ID_CATEGORIE);
```

## Annexe B

# Convientions pour les diagrammes de navigation

### Conventions

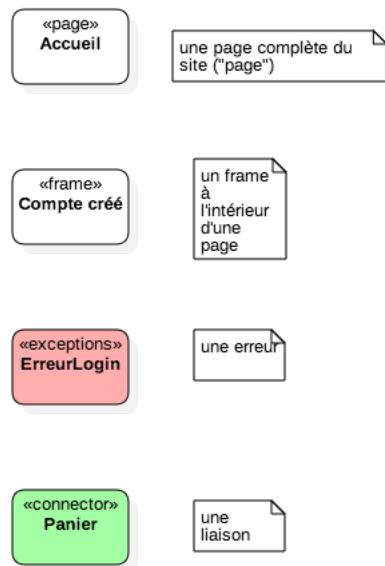


FIGURE B.0.1 – Conventions

## Annexe C

# Créer les tests avec PHPUnit

PHPUnit demande de créer au préalable une classe Adresse dans le modèle.

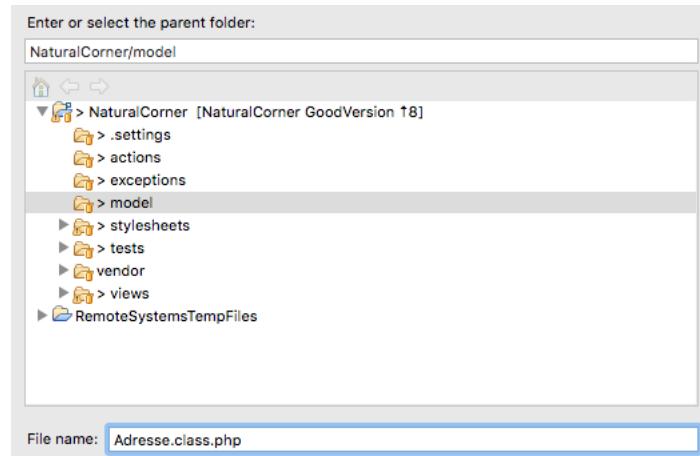


FIGURE C.0.1 – Hiérarchie des fichiers du projet

Ensuite, on clique droit sur le fichier de la classe Adresse.class.php et on demande à Eclipse de créer automatiquement une classe de test.

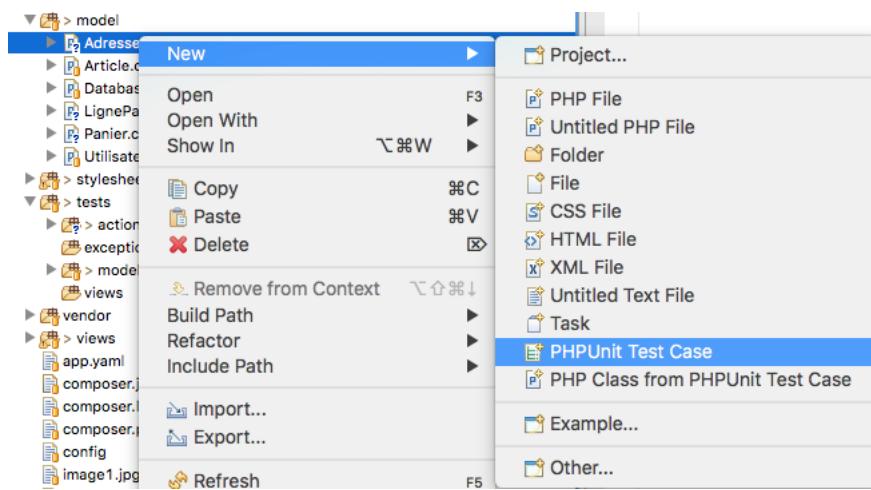


FIGURE C.0.2 – Crédit d'une classe de tests

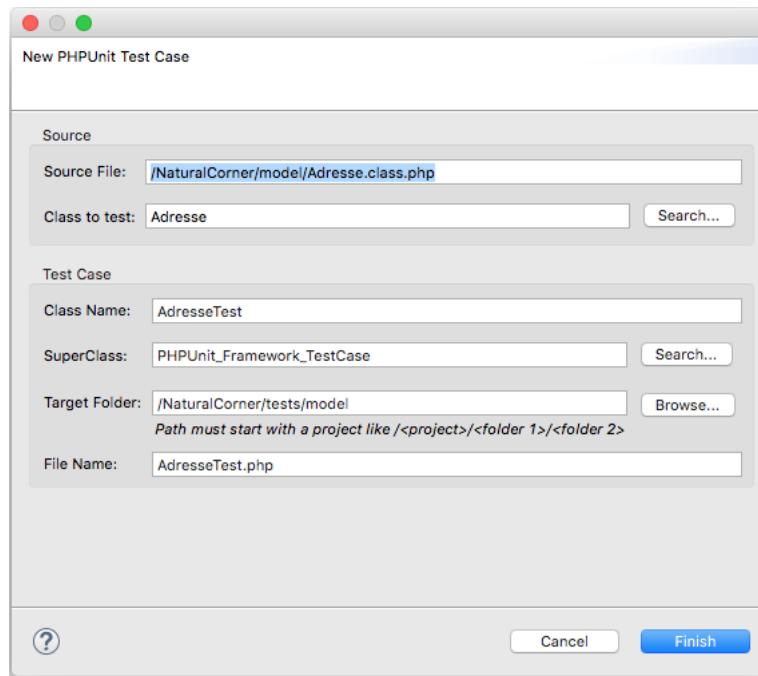


FIGURE C.0.3 – Remplissage automatique des champs pour le test

```

5  /**
6   * Test class for Adresse.
7   * Generated by PHPUnit on 2016-04-01 at 15:47:54.
8   */
9  class AdresseTest extends PHPUnit_Framework_TestCase
10 {
11     /**
12      * @var Adresse
13      */
14     protected $adresse;
15
16     /**
17      * Sets up the fixture, for example, opens a network connection.
18      * This method is called before a test is executed.
19      */
20     protected function setUp()
21     {
22         $this->adresse = new Adresse("rue", "numero", "numBoite", "codePostal", "localite", "pays");
23     }
24
25     /**
26      * Tears down the fixture, for example, closes a network connection.
27      * This method is called after a test is executed.
28      */
29     protected function tearDown()
30     {
31     }
32
33     /**
34      * @covers Adresse::getId()
35      */
36     public function testGetId()
37     {
38         $this->assertEquals($this->adresse->getId(), 0, "devrait afficher 0");
39     }
40 }
```

FIGURE C.0.4 – Création d'une page de tests

On peut commencer à écrire les tests. Pour rappel il s'agit de la méthode Test Drive Development<sup>1</sup> (TDD), qui préconise d'écrire les tests comme si les méthodes avaient déjà été implémentées. Une fois les tests écrits pour les mutateurs et les accesseurs de la classe Adresse, on peut lancer le premier test.

---

1. [http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-005-elements-of-software-construction-fall-2011/lecture-notes/MIT6\\_005F11\\_lec02.pdf](http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-005-elements-of-software-construction-fall-2011/lecture-notes/MIT6_005F11_lec02.pdf)

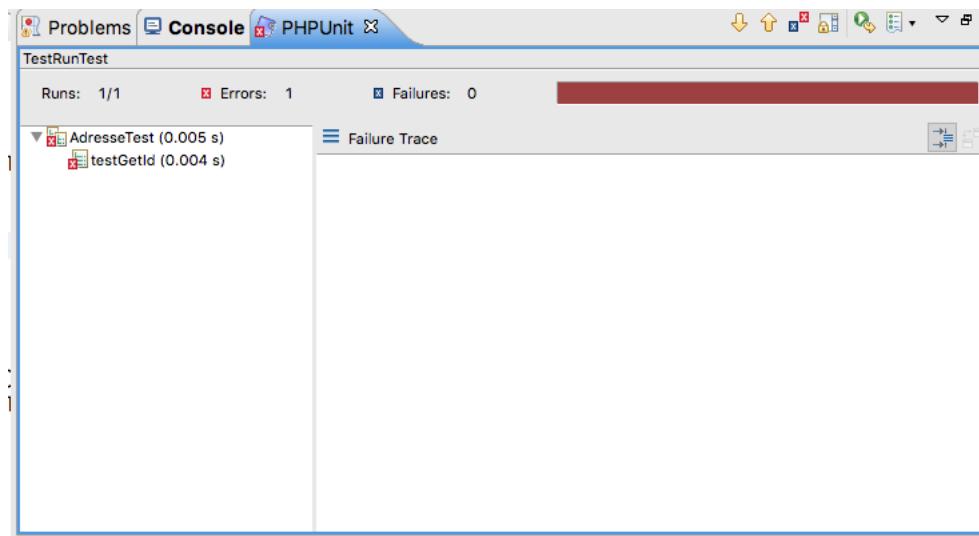


FIGURE C.0.5 – échec du test

La méthode `getId()` de la classe `Adresse` n'ayant pas été implémentée, le test échoue. Nous commençons donc à écrire la classe munie des champs que l'on retrouve sur le diagramme de classes détaillé ainsi que son constructeur. Dans un premier temps, le constructeur est écrit sans aucune mesure de sécurité quant aux données introduites en paramètre, cette responsabilité sera donnée aux setters, qui ne sont pas encore écrits.

```

class Adresse{
    /**
     * id de l'adresse.
     */
    private $_id;
    /**
     * Rue de l'adresse.
     */
    private $_rue;
    /**
     * Numero de l'adresse.
     */
    private $_numero;
    /**
     * Numéro de boite de l'adresse.
     */
    private $_numBoite;
    /**
     * Code postal de l'adresse.
     */
    private $_codePostal;
    /**
     * Localité de l'adresse.
     */
    private $_localite;
    /**
     * Pays de l'adresse.
     */
    private $_pays;

    /**
     * @param string $rue : Rue de l'adresse.
     * @param string $numero : Numero de l'adresse.
     * @param string $numAdresse : Numéro de boite de l'adresse.
     * @param string $codePostal : Code postal de l'adresse.
     * @param string $localite : Localite de l'adresse.
     * @param string $pays : Pays de l'adresse.
     */
    public function __construct($rue, $numero, $numBoite, $codePostal, $localite, $pays) {
        $this->_id = 0;
        $this->_rue = $rue;
        $this->_numero = $numero;
        $this->_numBoite = $numBoite;
        $this->_codePostal = $codePostal;
        $this->_localite = $localite;
    }
}

```

FIGURE C.0.6 – écriture de la classe Adresse

Dans une démarche de test, l'attribut id est initialisé à 0 par le constructeur afin de faciliter la démarche, on pourra ultérieurement, en concordance avec l'implémentation de la base de données, réaliser ce travail.

Il est intéressant de créer une exception pour une classe du modèle à ce niveau de la réalisation du UC. Cela clarifie le déboguage et permet d'identifier plus rapidement les sources du problème. La philosophie des tests unitaires préconise de créer une page de test par classe. Cependant cette classe ne contenant que trois lignes de code, nous passerons outre le précepte.

```

1 <?php
2 class AdresseException extends Exception {
3
4     public function __construct($string, $code=0) {
5         parent::__construct($message, $code);
6     }
7
8 }

```

FIGURE C.0.7 – classe ArticleException

L'écriture du test des méthodes getId() et setId() donne immédiatement lieu à un test, permettant de progresser tranquillement jusqu'à l'écriture complète de la classe Adresse.

```

/**
 *
 * @param int $id : id de l'adresse.
 * @throws AdresseException : si l'id n'est pas un nombre entier positif
 */
public function setId($id) {
    $id = (int) $id;
    if ($id >= 0)
        $this->_id = $id;
    else
        throw new AdresseException("<strong>id invalide : " . $id . "</strong>");
}

/**
 *
 * @return int L'id de l'adresse.
 */
public function getId() {
    return $this->_id;
}

```

FIGURE C.0.8 – implémentation des méthodes de la classe Adresse

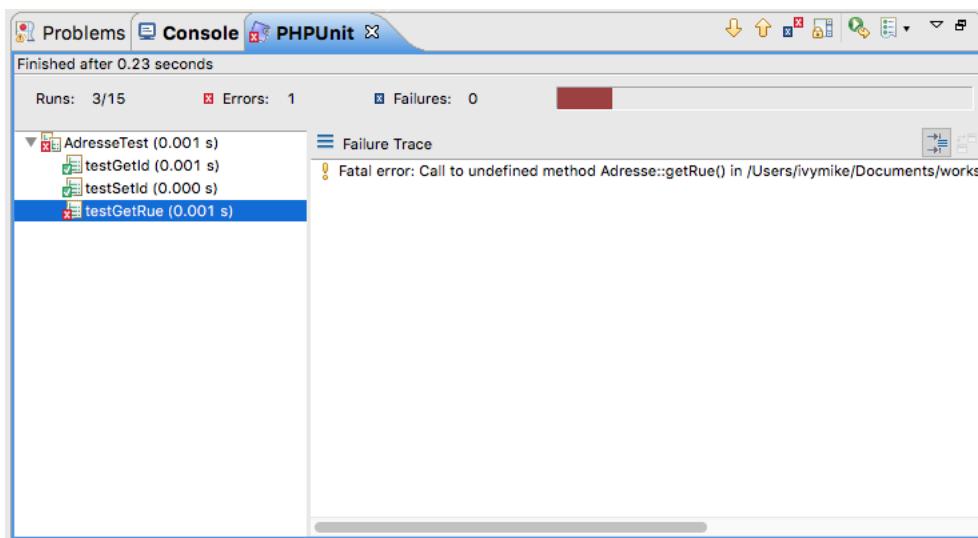


FIGURE C.0.9 – tests des méthodes

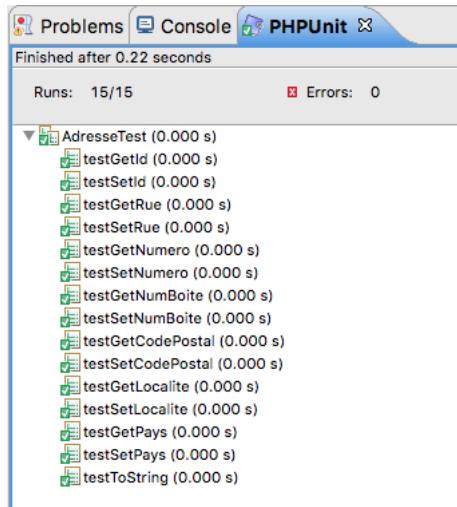


FIGURE C.0.10 – tests réussis

On réitère le procédé avec la classe Utilisateur.

## Annexe D

# Déployer une application en PHP sur Google App Engine

Depuis la console Google App Engine, on choisit "Créer un projet" :

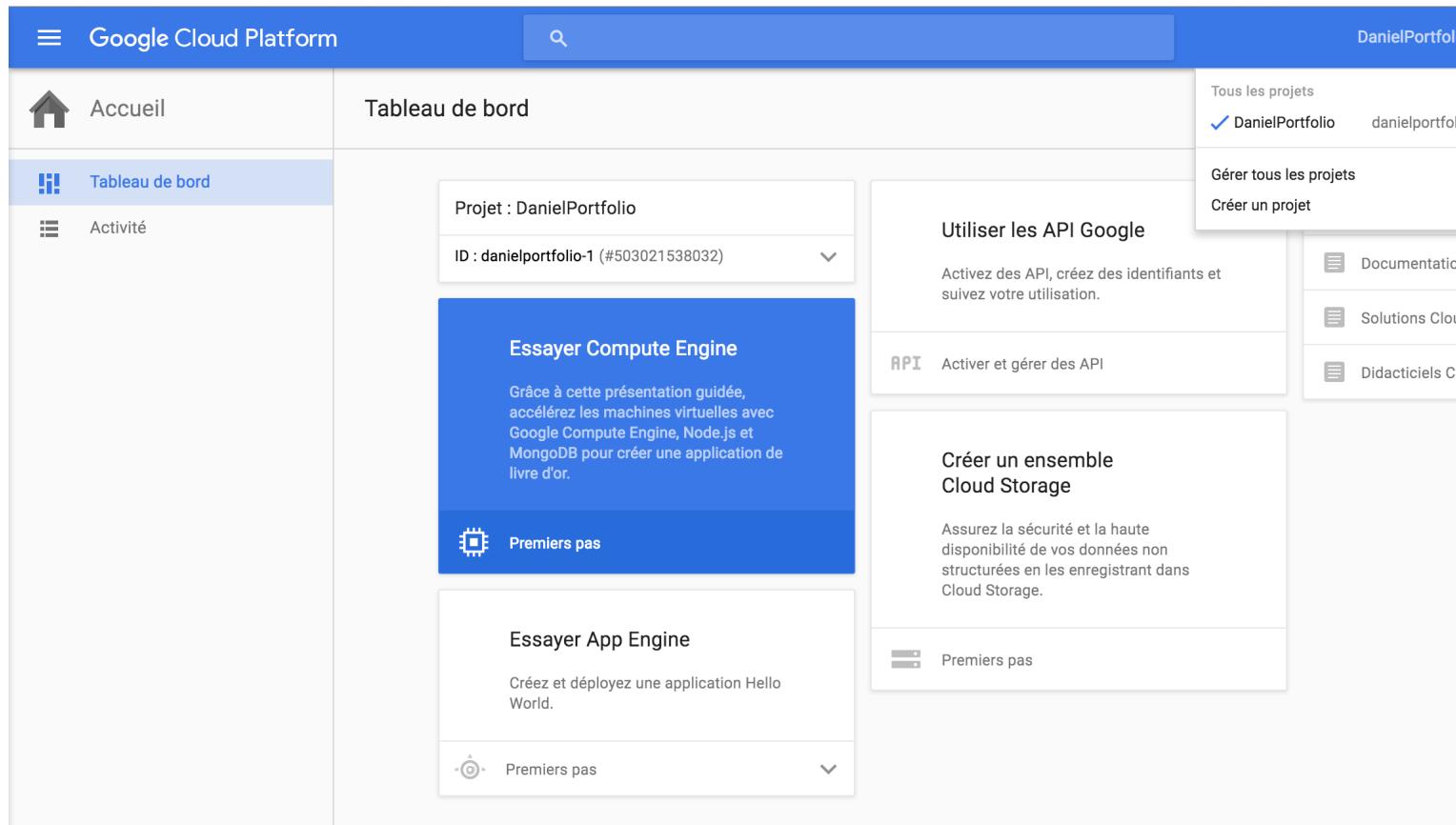


FIGURE D.0.1 – Console Google App Engine

On renseigne le nom du projet.

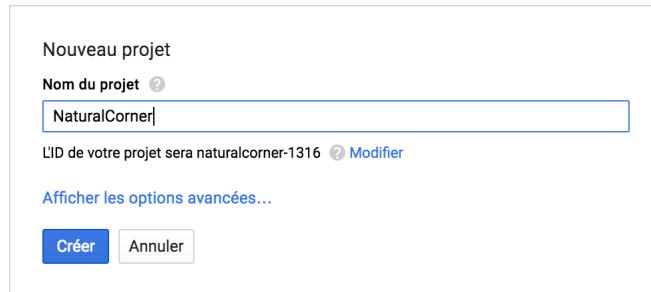


FIGURE D.0.2 – Création du projet

Ensuite, sur l'application GoogleAppEngineLauncher, on crée une nouvelle application :

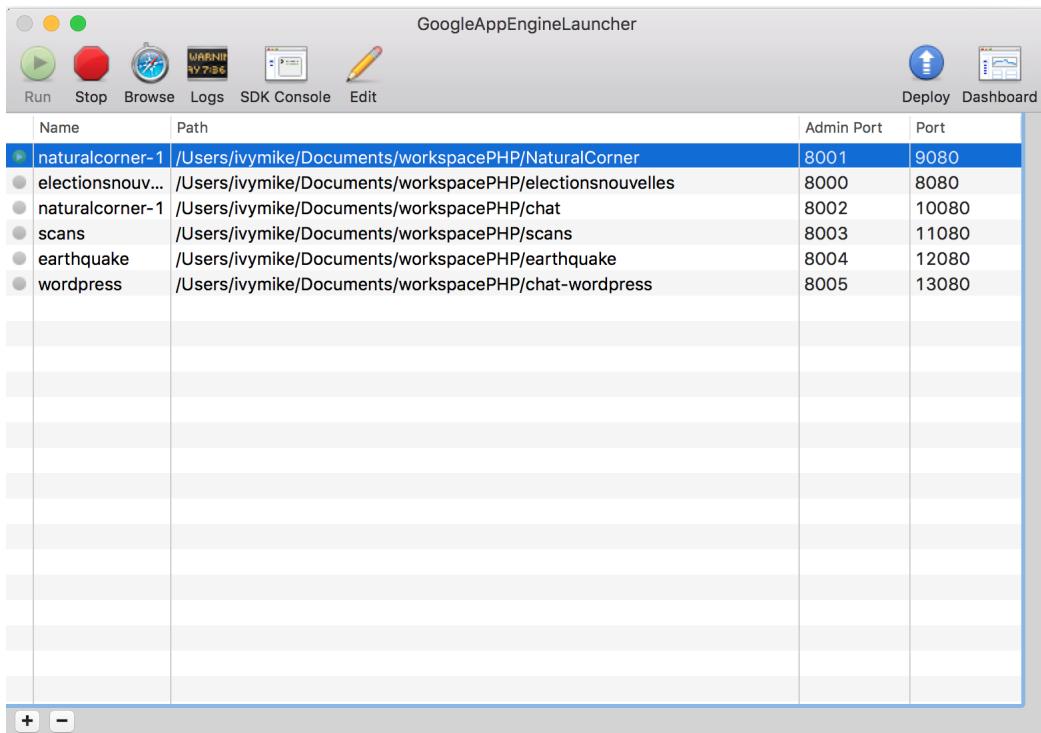


FIGURE D.0.3 – GoogleAppEngineLauncher

Les boutons "Run" et "Deploy" permettent respectivement de lancer l'application en local (exactement comme le ferait un serveur Apache) et de déployer sur le web l'application. Elle reçoit dès lors l'adresse <https://naturalcorner-1.appspot.com>.

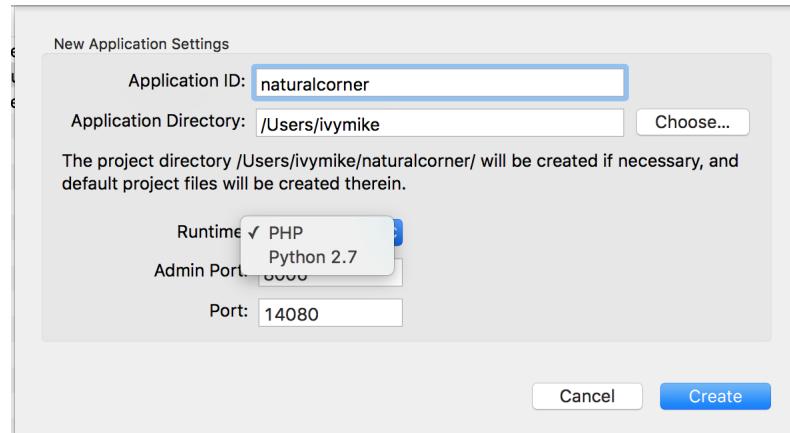


FIGURE D.0.4 – Choix du runtime PHP